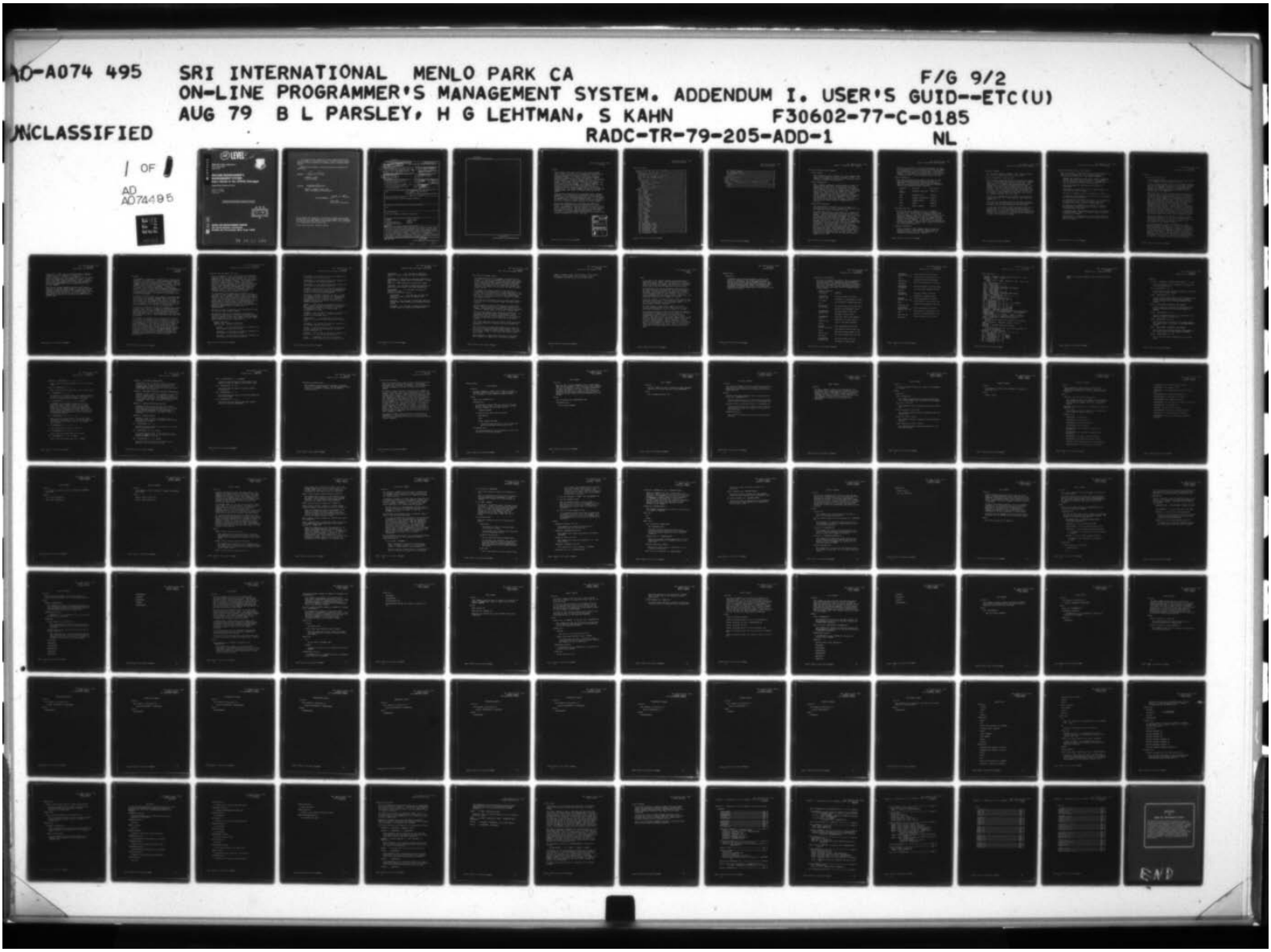
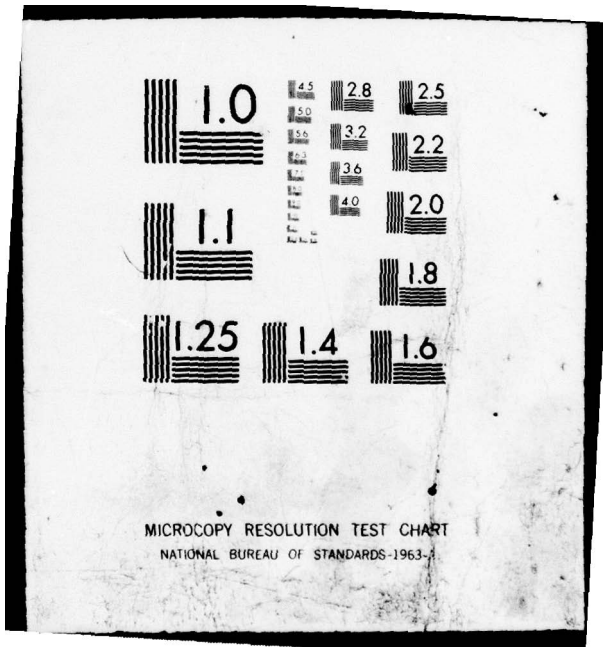
AD-A074 495 SRI INTERNATIONAL MENLO PARK CA F/G 9/2
ON-LINE PROGRAMMER'S MANAGEMENT SYSTEM. ADDENDUM I. USER'S GUID--ETC(U)
AUG 79 B L PARSLEY, H G LEHTMAN, S KAHN F30602-77-C-0185
UNCLASSIFIED RADC-TR-79-205-ADD-1 NL

1 OF 1
AD
AD74495




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(12) LEVEL II

AD A074468

RADC-TR-79-205, Addendum I
Final Technical Report
August 1979

ON-LINE PROGRAMMER'S MANAGEMENT SYSTEM
User's Guide to the JOVIAL Debugger

Augmentation Resources Center

Bruce L. Parsley
Harvey G. Lehtman
Susan Kahn

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-79-205, Addendum I has been reviewed and is approved for publication.

APPROVED: Raymond A. Liuzzi

RAYMOND A. LIUZZI
Project Engineer

APPROVED: Wendall C. Bauman

WENDALL C. BAUMAN, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISIE), Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-79-205, Addendum I	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): ON-LINE PROGRAMMER'S MANAGEMENT SYSTEM User's Guide to the JOVIAL Debugger, Addendum I.	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, Sep 77 — Mar 79
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): Bruce L. Parsley, Harvey G. Lehtman, Susan Kahn	8. CONTRACT OR GRANT NUMBER(s): F30602-77-C-0185
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Augmentation Resources Center, 20705 Valley Green Drive, Cupertino CA 95014 (SRI Int. Menlo Park CA)	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 55811003
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIE), Griffiss AFB NY 13441	12. REPORT DATE: Aug 1979
	13. NUMBER OF PAGES: 96
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Raymond A. Liuzzi (ISIE)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Debugging
On-Line
Software Engineering
Programming Environments
Computers
System Software
JOVIAL
Compilers

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report is composed of studies that have been conducted to develop the NLS system as an on-line programming environment and to provide an on-line JOVIAL interactive debugger with the capabilities to debug JOVIAL language programs. The final report contains several design additions to the NLS system to create an on-line programming environment. A JOVIAL User's Guide prepared in Addendum Technical Report I provides an extensive set of commands for using the JDAD Debugger. Addendum Technical Report II provides a generalized approach to debugging and describes the NLS/NSW Do-All Debugger (DAD).

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

410 281

Preface

This document is the User's Guide to the JOVIAL debugger (JDAD). JDAD is an interactive debugger that can be used to debug JOVIAL programs running on the TENEX or TOPS-20 operating system. It is based on DAD, a multi-language interactive debugger that runs on the TENEX or TOPS-20 operating system. DAD has a modular structure. The Frontend of DAD consists of a command language interpreter and a grammar. The grammar is a data structure that specifies the user interface to the debugger. The command language interpreter follows the grammar and interacts with user. The Backend of DAD consists of three separate modules: a dispatcher module, a language module and an an operating system module. The JOVIAL debugger was made by replacing the language module (LM) module in DAD and modifying the DAD grammar.

JDAD was written specifically for debugging JOVIAL language programs. It can interpret all JOVIAL data types, ordinary JOVIAL tables, the JOVIAL procedure call and return mechanism, the walkback data, and JOVIAL parameter lists. It can find the symbol table in a standard JOVIAL program and consequently knows of all external symbols defined in a JOVIAL program.

The current implementation of JDAD could be expanded to possess more knowledge about the JOVIAL compiler in the future. For example, it would be possible to include code in JDAD to process the ISD (internal symbol dictionary) produced by the JOVIAL compiler. This would greatly increase JDAD's knowledge of the JOVIAL program being debugged. It would allow JDAD to determine JOVIAL instruction boundaries and have complete knowledge of all data structures.

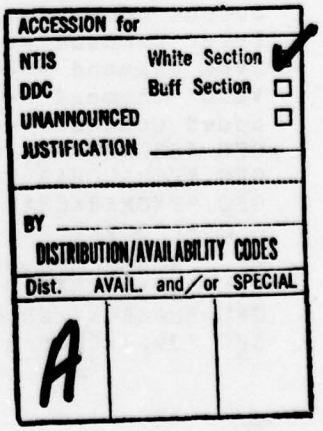


Table of Contents

General Information About Commands

General Format

JDAD commands all have a similar form; most commands begin with a verb followed by a noun or by typed in text. For example, the command verb "Find" may be followed by one of two nouns, "Content" or "Reference".

Command Recognition

JDAD's command recognition mode minimizes the number of characters the user needs to type and echos the full command word as soon as it is recognized. For example, JDAD recognizes the letter "f" as the command word "Find"; as soon as JDAD recognizes the command word, it shows the entire word. Most of the time, JDAD will recognize a command word after the user types the first letter; however, sometimes more than one command word starts with the same letter. JDAD will recognize the most commonly used altenative by its first character. The other alternatives may be specified by typing a space and then the one or two letters needed to disambiguate the conflicts.

Prompts and Noise Words

JDAD uses prompts to indicate to the user what it is expecting the user to type; in general, a prompt is one or more uppercase letters, followed by a colon.

The JDAD herald followed by the prompt for a command word, "JDAD C:", indicates that JDAD is waiting for the next command. JDAD is the debugger herald and "C:" is a prompt. Once the user has typed part of a command, JDAD will respond with the next appropriate prompt. For example, once the "Find" part of the Find command has been shown, the user will see "C:". In this case the "C" stands for "command word" and the user must reply with a command word. For example, the user may type "r" for the noun "References". The other letters are used in prompts include: "T", which means type some text; "OK", which means type <OK>; "OPT", which means type <OPT>; and "RPT", which means type <RC>. (The meanings of these special characters are discussed below.) A slash between letters in a prompt, it means that the user has a choice. For example, "C/OK:" means that the user can type either a command word or <OK>.

After the user types "fr" for "Find References", the user will see the word "to" in parentheses following the command word "References". This is a "noise word". Noise words provide extra information to help the user to understand a command. In this case, "(to)" means that the user now has to specify a value to which references should be found.

Control Characters

Many control characters have special functions in JDAD. Some keyboards have function keys for the control characters. The table below, gives the usual keyboard label for each function key used by JDAD, its function in JDAD and the equivalent control character.

Notation	Function	Equivalent
<BC>	Backspace character	<CTRL-A> or <CTRL-H>
<BW>	Backspace word	<CTRL-W>
<OK>	Command confirmation	<CTRL-D>
<CD>	Command delete	<CTRL-X>
<OPT>	Option	<CTRL-U>
<RC>	Repeat command	<CTRL-B>

Confirming with <OK>

To tell JDAD that a command or part of a command is finished, or to indicate that a typein is complete, the user types <OK>. At the end of a command the user will often be prompted to type <OK> with an "OK:" prompt; in the middle of a command there is a choice of adding another command word or typing <OK> to end the command, the user will see a "C/OK:" prompt. To type <OK> at a typewriter terminal that does not have an OK key, the user presses the key labeled "RETURN". When an <OK> is typed, the user will see an exclamation point (!).

Optional Alternatives

There are places in JDAD commands where the user has optional choices. These optional command paths are accessed by hitting the <OPT> function key (or its equivalent control character, <CTRL-U>.)

Canceling a Command

<CD> is used to cancel a command. <BC> is used to erase only the last command word that has been typed.

Syntactic information: Question Mark

Any time while using JDAD (except in the middle of typing text), a question mark can be used to get a list of command words or of all the things that can be done next. When typed after a "C:" prompt, question mark shows all the command words that JDAD will recognize at that point. For example, if the user types "?" after "JDAD C:", the user will see a list of all the command words that begin JDAD commands. One of these words is "Find"; if the user types "f" to begin a Find command and then types "?" after the "C:" prompt following "Find", the user will see a list of the command words that can follow Find.

In the list of command words, the symbol "<>" preceding a command word means that a space must be typed before that command word.

After the list of command words has been shown, the user can type a character to begin one of them; the command will continue as if the question mark had not been used.

At some steps in commands, JDAD is waiting for the user to type in some text or do something other than begin a command word. In this case, question mark will show what JDAD expects by listing brief instructions that explain the choices. The user can then follow one of the instructions or type <CD> to cancel the command.

Syntax Conventions Used In This Document

With the exception of the formal definition of an address list (which uses a modified BNF), the following syntactical conventions are adhered to in the command summaries:

Command words appear with their first letter in uppercase and the rest of the word in lowercase. When a generic function character (discussed below) is a command word, it will be surrounded by double quotation marks ("...").

Noise words appear as lowercase words enclosed in parentheses.

Alternatives (or paths) for a rule or command appear as a list under the rule or command.

The end of a command or rule is indicated by a colon (:).

An uppercase word preceded by an at sign (@) is a reference to a rule described elsewhere.

An uppercase word preceded by an uparrow (^) is a reference to a selection entity. Selection entity types (text, character, etc.) are listed in a separate section.

An uppercase word preceded by an asterisk (*) refers to that character currently serving the generic function (discussed below) specified.

An uppercase word not preceded by an at sign or up arrow is a Frontend prompt. These are described under the general information on commands.

Angle brackets (<>) are used to inclose single character keystrokes (e.g. <LINEFEED> refers to hitting the linefeed key on a terminal).

Concepts

Entering and Leaving JDAD

The following discussion is relevant to the current release of the debugger and may change in the future.

To use the JOVIAL debugger (JDAD), run the TENEX subsystem JDAD from the debugger directory. The particular debugger directory is dependent on the host on which the user is running. For example, currently on the TYMSHARE host OFFICE-2 the JDAD subsystem can be accessed by typing <SUBSYS>JDAD to the TENEX EXEC.

When JDAD starts, it will do some initialization and then prompt you with the JDAD herald followed by the prompt for a command. JDAD's command language is context dependent, and until you have specified a program for JDAD to debug, only a few global commands will be available. Probably the most useful command at this time is the Debug command in which you specify which program you wish to debug. After specifying a program, the full complement of JDAD's commands will be available. At this time you may set breakpoints where JDAD will suspend execution of programs and await further commands. This allows you to check out your program section by section. Either before starting execution or during breakpoint stops, you may examine and modify the contents of any location in core, execute other instructions, search for references to particular symbols and perform other tasks to aid in the debugging process.

When you are ready to start execution, give the Continue command and execution will start at the program's main entry vector location. (If you do not wish to start at the main entry vector location, you may use some of the sub-commands of the Continue command.)

It is also possible to splice in JDAD after a JOVIAL program has begun execution. To do this, type Control-C. Then type JDAD to the TENEX EXEC. JDAD will respond with the the JDAD herald followed by the prompt for a command. In this case, the full complement of JDAD's command will be available immediately. JDAD will operate in the same manner as if it had been started directly from the TENEX EXEC except that if you give the Continue command with no sub-commands, execution will resume at the instruction that was about to be executed when the Control-C was typed.

To get back to JDAD later (in case you forgot to set any breakpoints, or your program is looping, etc.), use the <CTRL-L> facility. Control-L is a deferred pseudo-interrupt (PSI), which means that you won't actually enter the debugger until the control-L is read. If you wish to enter the debugger immediately, type 2 control-Ls without any intervening typein. To continue execution of what was happening before you re-entered JDAD use the Continue command.

When you are through debugging, you may either enter a control-C or use JDAD's Quit command. If you are through debugging a specific (instance of a) program and wish to debug (a different instance of or) another program, use the Done command which will ask you for a new program to be debugged after removing up the previous program from your address space.

Processes

The debugger is designed to be a multi-tool, multi-process debugger. This means that a JOVIAL program is allowed to contain any internal process structure it desires, and the debugger is able to debug more than one process. The JDAD commands often use the term "tool" and may require users to specify tools as objects of commands. A tool is a collection of one or more interacting, collaborating processes. Thus a typical JOVIAL program is an example of a tool. Process is being used in the conventional computer science meaning of the word: it has its own virtual programming environment including its own Program Counter and stack environment.

To handle a multiple number of processes the debugger uses the concept of an Internal Debugger Handle (IDH). An IDH is an unique (per debugging session) positive integer. Each process that the debugger knows about is assigned an IDH. A user may always refer to a process by its IDH, and, in some commands, if the process is the top process, the user may also refer to it by the the program name.

A process is assigned an IDH when the debugger first learns of the process. When the debugger is first pointed at a program, it will determine the process structure for that program and assign an IDH for each process. Thereafter, the debugger will monitor the program's execution, and will assign new IDHs to newly created processes at the time they are created.

At any time, the debugger can be pointed at one, and only one, process. This process will be referred to as the current or active target process. This does not mean that the debugger can not know about more than one process, nor that the debugger is not capable of varying the current target process over time. It just means that at any instant, all commands are refering to the current process (with the obvious exception of the Debug command to point at another process). During a debugging session, when a breakpoint is encountered, the process containing the breakpoint will automatically be made the current target process, regardless of which process was current previously.

Character Sets and Generic Functions

Since the debugger on which JDAD is based is designed to support a number of different languages, and since most languages do not use the same character sets, it is not possible for the debugger always to use the same character to mean the same thing in a command. For example, a semi-colon character may be a valid character in an identifier in some languages, and it cannot therefore be used to separate address ranges (discussed below) in an address list. Therefore the debugger has adopted the concept of a generic function and a generic function character (GFC). A GFC is that character which is currently serving a specific generic function.

For documentation and communication purposes, it is convenient to have a generic name to refer to the specific character that is currently serving a particular generic function. Thus, while the specific character may change, it can still be referred to by its generic name. The generic name for a character is the uppercase word of the default generic function symbolic name preceded by an asterisk, e.g. the generic name for the GFC that is currently serving the generic function of an address list delimiter (whose default is a semi-colon) is *SEMICOLONCHAR.

The current values of each GFC can be determined by using the Character (set) Display command.

The symbolic names and the meaning of these generic functions are as follows. The default character used in JDAD for a generic function will appear under the meaning column delimited by a left angle bracket (<) and a right angle bracket followed by a semicolon (>;):

generic function symbolic name	meaning of character
pluschar	<+>; the user is using this character as the arithmetic addition operator
minuschar	<->; the user is using this character as the arithmetic subtraction operator
timeschar	<*>; the user is using this character as the arithmetic multiplication operator

dividechar <'>; the user is using this character as the arithmetic division operator

lparenchar <(>; the user is using this character as the arithmetic left grouping character

rparenchar <)>; the user is using this character as the arithmetic right grouping character

blockchar <&>; the user is using this character as a block delimiter; e.g. the string: string1&string2 should be interpreted as symbol string2 in block string1 if & is the current BLOCKCHAR

escapechar <ALTMODE or ESCAPE>; the user is using this character to mean interpret the next character as a debugger builtin variable; e.g., ESCAPECHAR followed by a 'Q (or 'q) refers to the builtin debugger variable which has the value of the most recently displayed cell

lmchar <%>; the user is using this character to mean interpret the next character(s) as a language module builtin variable or construct; there are no language module builtins in the current implementation of JDAD

commachar <:>; the user is using this character as an address range delimiter to separate the two elements of an address range

semicolonchar <;>; the user is using this character to separate address ranges within address lists

larrowchar <_>; the user is using this character as the debugger assignment character

tabchar <tab>; the user is using this character to mean display the cell addressed by the most recently displayed cell

poundchar <#>; the user is using this character to mean back up to the previous displayed cell

lfchar <LINEFEED>; the user is using this character to mean display the next sequential cell

uparrowchar <^>; the user is using this character to mean display the previous sequential cell

bslashchar <\>; the user is using this character to mean display an address list in string mode; in JDAD this means as a JOVIAL character variable with length

equalchar <=>; the user is using this character to mean display the value of the input address list

excmarkchar <!>; the user is using this character to mean display cells as ascii values

lsquarechar <[>; the user is using this character to mean display an address list numerically

qmarkchar <?>; the user is using this character to mean tell where symbols in an address list are defined

slashchar </>; the user is using this character to mean display an address list symbolically

User Input and Debugger Output

All communication with the debugger is governed by the values of 4 records: the permanent and current input mode records, and the permanent and current output mode records. At the beginning of most commands (exceptions discussed below) the permanent input and output mode records are copied to the current input and output mode records, and thereafter the command is governed by the value of these current records.

For example, all numbers entered by the user will be interpreted as being numbers in the base specified by the current input mode radix, and all numbers displayed to the user will be formatted to conform to the current output mode radix. Two exceptions to this treatment of numbers are discussed below.

The values of the permanent input and output mode records can be displayed via the Typeout (mode) and Input (mode) commands.

Several commands provide for modifying the current input and/or output mode records for a specific instance of a command. These ephemeral values are then lost at the start of the next command except in the cases discussed below.

There are some commands that consist of a single GFC, e.g., the assign command as entered by *LARROWCHAR. These commands will use the current values of the input and output mode commands at their invocation, i.e., the values of these records that were in effect for the previous command.

The current input mode radix and current output mode radix govern the evaluation of numbers with the following two exceptions:

When specifying or viewing these radixes, the radix will always be interpreted as being decimal numbers. Thus one may specify a change to input or output radix of the value 10 by typing 10 even if the current input radix is some other value.

When specifying a JOVIAL table definition in the Define Table command, all numbers will be treated as decimal

numbers. However, when a JOVIAL table or table item is referenced in an address list, all numbers will be interpreted according to the current radix mode.

Frame

On the PDP-10, the JOVIAL procedure calling mechanism is implemented using a stack. Each procedure call causes an entry to be made on the stack. The return from a procedure causes the entry to be removed from the stack. This entry points back to the word following the call instruction.

The information that can be determined from this entry on the stack is called the "frame" for the called procedure. The frame contains the names of the called and calling procedure, the contents of the parameter list, the values of the parameters and the return location. When the calling procedure has been compiled with walkback data, the frame also contains the line number of the call.

A specific register is used by JOVIAL to keep track of the last entry on the stack. JDAD knows which register is used and thus can determine the last procedure called, i. e., the procedure currently executing.

The JDAD concept 'current frame' refers to the most recently displayed frame or the frame on the top of the stack when a breakpoint is hit or a Control-L is typed.

When displaying frames in JDAD via the "Display" command it is important to remember that the first entry on the stack (oldest entry) is the bottom of the stack and the corresponding frame is referred to by the FB address element. The last entry on the stack (newest entry) is the top of the stack and the corresponding frame is referred to by the FT address element. When a parameter or return location is included in an address list, JDAD interprets this to be the parameter or return location for the current frame: the particular frame (first, last, or somewhere in the middle of the stack) which was last accessed by the user.

Address Lists

Discussion

An address list is the basic manner in which a user refers to elements in the current target process. Basically, an address list is composed of one or more address ranges; and an address range consists of one or two address range elements (AREs). The character that terminates an address list, while it may modify the functional use of the address list, is not a part of the address list itself.

Address List Terminators

The user may terminate an address list with a number of different characters, depending on which command is being specified. The terminating character is NOT a part of the address list. The following are the generic characters, with their default character values and their meanings, that may be used to terminate various address lists:

generic character terminator	meaning
*BSLASHCHAR	the user wishes to see the address list displayed in string mode (in JDAD this means as a character variable with size of 1)
*EQUALCHAR	the user wishes to have the value of the input address list displayed to him
*EXCMARKCHAR	the user wishes to see the address list displayed in ascii mode
*LARROWCHAR	after each line of the address list is displayed, the user wishes to assign a new value to the just displayed entity
*LFCHAR	after displaying the current address list, the user wishes to see the cell (or data structure) whose address is one greater than the last displayed cell (or data structure)
*LSQUARECHAR	the user wishes to see the address list displayed in numeric mode

*POUNDCHAR	after displaying the current address list, the user wishes to see the cell (or data structure) that was displayed immediately prior to the last cell (or data structure)
*QMARKCHAR	the user wishes to find out where the symbols in the entered address list are defined
*SLASHCHAR	the user wishes to see the address list displayed in symbolic mode
*TABCHAR	after displaying the current address list, the user wishes to see the cell (or data structure) addressed by the last displayed cell
*UPARROWCHAR	after displaying the current address list, the user wishes to see the cell (or data structure) whose address is one less than the last displayed cell (or data structure)

Formal Definition

```
ADRLIST := ADRRANGE [ *SEMICOLONCHAR ADRLIST ]  / NULL
ADRRANGE := RANGE / BUILTIN
BUILTIN :=
   FRAME / LOCAL / PARAM / PARAMLIST / MEM / PLIST / JFN
   / ERR
ERR := *ESCAPECHAR ('E / 'e)
JFN := AJFN / RJFN
AJFN := *ESCAPECHAR ('J / 'j)
RJFN := AJFN NUMBER [*COMMACHAR AJFN NUMBER]
PLIST := *ESCAPECHAR ('Z / 'z)
MEM := AMEM / RMEM
AMEM := *ESCAPECHAR ('M / 'm)
RMEM := AMEM NUMBER [*COMMACHAR AMEM NUMBER]
PARAM := *ESCAPECHAR ('P / 'p)
PARAMLIST := *ESCAPECHAR ('P / 'p) ('L / 'l)
LOCAL := *ESCAPECHAR ('L / 'l)
FRAME := FSPEC [ *COMMACHAR FSPEC ]
FSPEC := FF / FR / FO / FT / FB / FA
FF := *ESCAPECHAR ('F / 'f)
FR := *ESCAPECHAR ('F / 'f) ('+ / '-) [ NUMBER ]
FO := *ESCAPECHAR ('F / 'f) ('O / 'o)
FT := *ESCAPECHAR ('F / 'f) ('T / 't)
FB := *ESCAPECHAR ('F / 'f) ('B / 'b)
FA := *ESCAPECHAR ('F / 'f) '@ NUMBER
TABLESPEC := SMPLIDENT DIMENSIONLIST
ITEMSPEC := SMPLIDENT *BLOCKCHAR SMPLIDENT DIMENSIONLIST
DIMENSIONLIST := '( (DIMENSIONRANGE / DIMENSIONRANGE ',
DIMENSIONRANGE ) ')
DIMENSIONRANGE := ('* / NUMBER ': NUMBER / NUMBER)
   see the section on JOVIAL Tables for more details
RANGE := EXPRESSION [ *COMMACHAR EXPRESSION ]
EXPRESSION :=
   expressions are defined and discussed in a separate
   section
IDENT := BLCKIDNT / SMPLIDNT / NUMBER / BLTNTRM /
METAIDNT
BLCKIDNT := SMPLIDNT *BLOCKCHAR SMPLIDNT
SMPLIDNT :=
   a string composed of valid identifier characters for
   the current language
METAIDNT := *LMCHAR SMPLIDNT
BLTNTRM := BA / BB / BLN / BPN / BQ / BR / BY
BA := *ESCAPECHAR ('A / 'a)
BB := *ESCAPECHAR ('B / 'b) NUMBER
BLN := *ESCAPECHAR ('L / 'l) NUMBER
BPN := *ESCAPECHAR ('P / 'p) NUMBER
BQ := *ESCAPECHAR ('Q / 'q)
BR := *ESCAPECHAR ('R / 'r)
BY := *ESCAPECHAR ('Y / 'y)
```

NUMBER := a string of digits in the current input mode radix

Semantics

ADRLIST := ADRRANGE [*SEMICOLONCHAR ADRLIST] / NULL

the NULL address list is equivalent to entering the last input address list

ERR := *ESCAPECHAR ('E / 'e)

used to show the last operating system error incurred by the current target process

AJFN := *ESCAPECHAR ('J / 'j)

used to display an indication of the files being used (listed by their JFNs; equivalent to the TENEX FILSTAT command and the TOPS-20 INFORMATION FILES command.)

RJFN := AJFN NUMBER [*COMMACHAR AJFN NUMBER]

used to display an indication of names and statuses of files being used for file numbers NUMBER [to NUMBER]

PLIST := *ESCAPECHAR ('Z / 'z)

used as a shorthand notation to be equivalent to the previously typed in address list

AMEM := *ESCAPECHAR ('M / 'm)

used to show the utilization of the address space of the target process. Equivalent to the TENEX MEMSTAT command.

RMEM := AMEM NUMBER [*COMMACHAR AMEM NUMBER]

used to show the utilization of the address space of the target process for pages NUMBER [to NUMBER]

PARAM := *ESCAPECHAR ('P / 'p)

used to show the formal parameters of the current frame

```
PARAMLIST := *ESCAPECHAR ('P / 'p) ('L / 'l)

   used to show the formal parameter list of the current
   frame

LOCAL := *ESCAPECHAR ('L / 'l)

   used to show the local variables of the current frame

FF := *ESCAPECHAR ('F / 'f)

   FF refers to the current frame.  the current frame is
   the most recently displayed frame or the frame on the
   top of the stack after the debugger is entered

FR := *ESCAPECHAR ('F / 'f) ('+ / '-) [ NUMBER ]

   if NUMBER is not specified it defaults to 1; no
   spaces may precede NUMBER; NUMBER specifies the
   number of frames to move relative to the current
   frame; e.g. if '$ is the current *ESCPAECHAR, and ',
   is the current *COMMACHAR, the FRAME: "$ft, $f-2"
   would display the frame on the top of the stack, and
   the next two frames towards the bottom of the stack
   in the control thread.

FO := *ESCAPECHAR ('F / 'f) ('O / 'o)

   used to show the owner frame of the current frame;
   the owner of a procedure is its caller; the owner of
   a coroutine is the routine that did the openport to
   the coroutine.

FT := *ESCAPECHAR ('F / 'f) ('T / 't)

   used to show the top frame on the stack

FB := *ESCAPECHAR ('F / 'f) ('B / 'b)

   used to show the bottom frame on the stack

FA := *ESCAPECHAR ('F / 'f) '@ NUMBER

   used to show the frame whose mark is NUMBER
```

TABLESPEC := SMPLIDENT DIMENSIONLIST

TABLESPEC is used to show the contents of items in a JOVIAL table. It is possible to show items with specific index values or the entire table. The DIMENSIONLIST is used to indicate which index values of the TABLE are displayed.

ITEMSPEC := SMPLIDENT *BLOCKCHAR SMPLIDENT DIMENSIONLIST

ITEMSPEC is used to show the contents of a particular item in a JOVIAL table. It is possible to show specific index values of the item or all occurrences of the item in the table. The DIMENSIONLIST is used to indicate which index values of the item are displayed.

BLCKIDNT := SMPLIDNT *BLOCKCHAR SMPLIDNT

BLCKIDNT is used to refer to the (local) symbol (specified by the second SMPLIDNT) in the block (or file) specified by the first SMPLIDNT; e.g., if "&" is the current *BLOCKCHAR, then the BLCKIDNT: "fl&sfilev" would refer to the symbol "sfilev" in file "fl".

METAIDNT := *LMCHAR SMPLIDNT

METAIDNT is used to refer to language specific constructs; this notation is not used in the current implementation of JDAD

BA := *ESCAPECHAR ('A / 'a)

this entity has the value of the address of the most recently displayed cell

BB := *ESCAPECHAR ('B / 'b) NUMBER

this entity has the value of the address at which breakpoint NUMBER is set; it has the value of zero if breakpoint NUMBER is not set

BLN := *ESCAPECHAR ('L / 'l) NUMBER

this entity has the value of the address of the NUMBER-th local of the current frame

```
BPN := *ESCAPECHAR ('P / 'p) NUMBER

   this entity has the value of the address of the
   NUMBER-th formal parameter of the current frame

BQ := *ESCAPECHAR ('Q / 'q)

   this entity has the value of the most recently
   displayed cell

BR := *ESCAPECHAR ('R / 'r)

   this entity has the value of the return address for
   the current frame

BY := *ESCAPECHAR ('Y / 'y)

   this entity has the value of the most recently
   completely evaluated EXPRESSION
```

Assigning To Address Lists

Many commands allow the user to assign to an address list as it is being displayed. The specification of new values to be assigned is handled by the @NVLRUL discussed below.

Two Special Characters

There are two characters used by JDAD as pseudo-interrupts (PSI) that need a separate discussion. The specific characters are initialized to <CONTROL-L> and <CONTROL-K>, but may be changed by the user by using the Interrupt command.

The first of these characters (initialized to <CONTROL-L>) is used to get the user to base command mode in JDAD. For example, a user has inadvertently requested JDAD to display a large number of cells. Upon realizing the mistake, the user may type 2 <CONTROL-L>s to abort output and return to base command mode. Additionally, when tools are executing (i.e. after the user has given the Continue command), if the user wishes to return to JDAD, the user should type one or two <CONTROL-L>s. Since this character is set up as a deferred PSI, it will not take effect until the character is read if only one <CONTROL-L> is typed. If the user wishes immediate action, then two <CONTROL-L>s should be typed. (Note that in the case of aborting JDAD output it may still take a while until the current contents of the output buffers are empty and the user actually is able to enter commands to JDAD.)

The second of these special characters (initialized to <CONTROL-K>) is used to display a short status of tools while they are executing (i.e. after the user has given the Continue command).

Command Summary

Debug Command

Overview

The debug command is used to point JDAD at a target process. Once JDAD is pointed at a target process, the full complement of JDAD commands becomes available.

Syntax

Debug (tool) @TOOLSPEC OK:

TOOLSPEC Rule

If JDAD does not know about any tools yet (as when JDAD is first started, or after the user has given the Done command for all active tools):

^TENEX-FILE-NAME:

If JDAD does know about some processes:

@ACTIVETOOLS:

^IDH:

OPTION ^TENEX-FILE-NAME:

this path allows the user to have one or more parallel processes executing under JDAD

ACTIVETOOLS Rule

the FE maintained rule of the usenames for the tools the user is currently debugging

Done Command

Overview

When the user is done debugging a tool, he/she should issue the Done command. Upon receiving a Done command, JDAD will do whatever cleanup is necessary with respect to JDAD's knowledge of the tool. If the user was debugging only one tool, or three or more tools, then JDAD will ask the user to specify which tool should become the current target tool upon completing the Done command.

Syntax

Done (debugging tool) @ACTIVETOOLS OK:

ACTIVETOOLS Rule

see the Debug command

Quit Command

Overview

The Quit command is used to terminate a JDAD debugging session and to return the user to the TENEX EXEC.

Syntax

Quit (debugging session) OK:

Interrupt Command

Overview

The interrupt command is used to change which characters will serve the two special functions of returning to JDAD's base command mode and of displaying the status of executing tools.

Syntax

Interrupt Executing (programs & abort output character should be) ^ICHARACTER OK:

This path allows the user to specify which character will be used to return the user to base command mode.

Interrupt Status (character should be) ^ICHARACTER OK:

This path allows the user to specify which character will be used to cause the display of the status of executing tools.

ICHARACTER Selector

Any control character not currently serving another function.

Wheel Command

Overview

The Wheel command is used by JDAD implementers and maintainers for the debugging and development of JDAD. Issuing the Wheel command makes available commands not normally available. The Wheel command requires the knowledge of a special password. It is mentioned here only because it may show up in response to a questionmark (?) typed to see the alternatives available.

Status Command

Overview

The Status commands display the status of the debugger to the user.

Syntax

Status OK:

Status Verbose OK:

This command provides more information about each tool being debugged than the default Status command.

Status For (tool) OK:

This command provides information about the current tool.

Status Verbose For (tool) OK:

This command provides verbose information about the current tool.

Status For (tool) ^IDH OK:

This command provides information for the specified process.

Status Verbose For (tool) ^IDH OK:

This command provides verbose information for the specified process.

Comment Command

Overview

This command is used to allow comments to appear on a typescript, etc.

Syntax

Comment ^CTEXT:

Character Command

Overview

These commands are used either to display which characters are serving which generic functions, or to modify which character is to serve a specific generic function.

Syntax

Character (set definitions) Display OK:

This command is used to determine which characters are serving which generic functions. Non-standard definitions will appear first in the resulting display.

Character (set definitions) Use ^FCHARACTER (instead of) @CHARRULE OK:

This command is used to change which character will serve a specific generic function.

CHARRULE Rule

"*PLUSCHAR" (for addition):

"*MINUSCHAR" (for subtraction):

"*TIMESCHAR" (for multiplication):

"*DIVIDECHAR" (for division):

"*LPARENCHAR" (for arithmetic grouping left delimiter):

"*RPARENCHAR" (for arithmetic grouping right delimiter):

"*BLOCKCHAR" (for symbol block delimiter):

"*ESCAPECHAR" (for builtin variable escape):

"*LMCHAR" (for language module escape character):

"*SEMICOLONCHAR" (for address list delimiter):

""*COMMACHAR" (for address range delimiter):

""*EQUALCHAR" (for display value):

""*SLASHCHAR" (for display using permanent typeout mode):

""*LSQUARECHAR" (for display numerically):

""*BSLASHCHAR" (for display as a string):

""*EXCMARKCHAR" (for display in ascii):

""*QMARKCHAR" (for tell where this symbol is defined):

""*LARROWCHAR" (for assignment):

""*LFCHAR" (for move to next address):

""*UPARROWCHAR" (for move to previous address):

""*TABCHAR" (for move to addressed address):

""*POUNDCHAR" (for move to previously displayed address):

Input Command

Overview

This command is used to display or change the permanent input mode.

Syntax

```
Input (mode) Display OK:

Input (mode) @INPTYP OK:
```

Typeout Command

Overview

This command is used to display or change the permanent output mode.

Syntax

Typeout (mode) Display OK:

Typeout (mode) @OUTTYP OK:

Symbol Command

Overview

A process may have more than one symbol table. (For example, if different parts of the address space were compiled and loaded as distinct entities.) The symbol commands allow the user to tell the debugger of the location of the symbol tables. When the debugger, and the appropriate Language Module, is first pointed at a process, the LM will use the default location for finding the symbol table.

The debugger makes its own copy of the process' symbol table. Thus, if a process modifies its symbol table, it is necessary for the user to give a new "Symbol" command. (Ultimately this will be do-able programmatically.) That the debugger uses a copy of the symbol table is desirable in those cases in which code executing code accidentally smashes the symbol table.

If a process contains more that one symbol table then the user can point the debugger to different tables by use of the symbol command and the debugger will copy the symbol table the frist time it is pointed to a new location. However, if a user subsequently points the debugger to a location previously used, the debugger will use its previous copy of the symbol table from that location unless the user specifies that there is a new pointer at the location.

Syntax

Symbol (table) Display (status) OK:

This command will display which symbol tables the debugger knows about, indicate which is the current symbol table, and provide an overview of the current table.

Symbol (table) Display (status) Verbose OK:

This command will display all the information that the Symbol (table) Display (status) command displays. In in addition it will display all the entries in the current symbol table.

Symbol (table) Display (status) Block OK:

This command will display which symbol tables the debugger knows about and will indicate which is the current symbol table. It will also display the boundaries for the symbol table block specified.

Symbol (table) Display (status) Verbose Block OK:

This command will display all the information that the Symbol (table) Display (status) Block command displays. In in addition it will display all the entries in the symbol table block specified.

Symbol (table) Pointer (located at) ^SYMADR OK:

Symbol (table) Pointer (located at) ^SYMADR OPTION (undefined symbol table pointer located at) ^SYMADR OK:

These two commands will point the debugger to the symbol (and undefined symbol) table(s) at the specified location. If the debugger already has a copy of the symbol table at the specified location, it will not copy the process' table.

Symbol (table) Pointer (located at) OPTION (new pointer at) ^SYMADR OK:

Symbol (table) Pointer (located at) OPTION (new pointer at) ^SYMADR OPTION (undefined symbol table pointer located at) ^SYMADR OK:

These two commands will point the debugger to the symbol (and udefined symbol) table(s) at the specified location. This version of the command will force the debugger to make a copy of the specified symbol table(s) regardless of whether or not it already has a copy of the symbol table at the specified location. This is useful if a process has modified its symbol table, or if a process is performing its own swapping in its address space.

Breakpoint Command

Overview

The breakpoint command allows the user to specify that the debugger (conditionally) be entered just prior to the execution of an instruction at a specified address in a target process.

A breakpoint is said to be "hit" when the instruction at the address specified for the breakpoint is about to be executed. After a breakpoint is hit, it either "takes" and the debugger is entered, or it doesn't take and normal execution of the target process continues.

For each case, i.e., the breakpoint taking or not, the user may specify a string that will be fed to the debugger, as if the user typed it, when the breakpoint is hit.

The decision as to whether or not a breakpoint takes is based on the following algorithm:

If a user has specified a procedure to be called when a breakpoint is hit, this procedure is called and returns one of three values: take the breakpoint, don't take the breakpoint, or base the decision on the proceed mode and counter. If this procedure returns the third value, or if no procedure was specified, then the breakpoint will take if the proceed mode is normal or automatic or if the proceed mode is count and this breakpoint has been hit count times already without taking. (The ability to specify such a procedure is not currently implemented.)

Every breakpoint that is set, i.e., for which an address has been specified, has the following attributes associated with it:

a) its number, ^BTNUMBER

When a breakpoint is first set, the user can request a specific number, or let the debugger assign an unused number for the breakpoint.

The user uses this number when he or she wishes to modify or examine the status of the breakpoint.

b) its address, ^BTADDRESS

This is the address at which the breakpoint is set.

Note that specifying an address for a breakpoint that is already set is equivalent to first clearing that breakpoint and then setting the address.

c) its name, ^BRNAME

If and when a breakpoint takes, its name will be displayed. A name is simply a string (including the null string) used for information purposes only. If a user is debugging more than one process, he or she may choose to name the breakpoints set in each process with the appropriate process name. Names need not be unique.

d) its proceed mode

Every set breakpoint has one of three proceed modes:

Normal mode

Set either by default or by specifying a proceed count of zero.

In this mode, the breakpoint will take each time the breakpoint is hit.

Automatic proceed mode

Set by specifying proceed automatically.

In this mode, the breakpoint will take each time the breakpoint is hit and then the breakpoint will be continued automatically, after processing its take command string if one exists.

Count mode

Set by specifying a non-zero proceed count.

In this mode, the breakpoint will not take until the breakpoint has been hit count plus one times. If a no take command string exists, then the count times this breakpoint is hit before it takes, the no take command string will be executed.

e) its call procedure, ^PNAME - NOT IMPLEMENTED YET

f) its take command string

When a breakpoint takes, if this string is non-null it will be fed to the debugger as if the user entered it on his or her terminal prior to accepting input from the user or automatically continuing.

g) and its no take command string

If a breakpoint doesn't take, and if this string is non-null it will be fed to the debugger as if the user entered it on his or her terminal when the breakpoint is hit and prior to continuing the breakpoint.

Syntax

Breakpoint Display (all) OK:

This command will display the status of all breakpoints that are currently set.

Breakpoint ^BTNUMBER Display OK:

This command will display the status of breakpoint ^BTNUMBER.

Breakpoint Clear (all) OK:

This command will clear all breakpoints, i.e. make them not set.

Breakpoint ^BTNUMBER Clear OK:

This command will clear breakpoint ^BTNUMBER.

Breakpoint Set (at) ^BTADDRESS @BOPT:

Breakpoint ^BTNUMBER Set (at) ^BTADDRESS @BOPT:

These two commands will set a breakpoint at the specified address, and will set any of the attributes specified. If ^BTNUMBER is not specified, then the debugger will assign a number for this breakpoint. If ^BTNUMBER is specified and it refers to a breakpoint that is already set, then that breakpoint will be cleared first, and then set at the new address with any attributes specified in this instance of the command.

Breakpoint ^BTNUMBER @BOPT1:

This command allows the user to modify the attributes of breakpoint ^BTNUMBER.

BOPT Rule

OK:

@BOPT1:

BOPT1 Rule

Call (procedure) ^PNAME @BOPT:

NOT IMPLEMENTED YET.

This rule is used to specify the name of a procedure that will get called when a breakpoint is hit to determine whether or not to take the breakpoint.

Proceed Count (=) ^PNUMBER @BOPT:

This rule is used to place a breakpoint in either normal proceed mode (if ^PNUMBER is zero) or in count mode.

Proceed Automatically @BOPT:

This rule is used to place a breakpoint in the automatic proceed mode.

Name (for this breakpoint is) ^BRNAME @BOPT:

This rule is used to specify the name for a breakpoint.

Break (commands are) ^BRKCMNDS @BOPT:

This rule is used to specify the take command string that gets executed when a breakpoint takes.

No (break commands are) ^BRKCMNDS @BOPT:

This rule is used to specify the no take command string that will get executed if a breakpoint is hit but doesn't take.

Continue Command

Overview

The continue commands allow the user to continue the execution of the process(es) that were executing before entering the debugger (regardless of whether the debugger was entered via a (nested) EXEC DEBUG command or by the taking of a (nested) breakpoint), or to modify the address at which a process will have its execution resumed when execution is ultimately continued, and optionally to modify the speed with which execution will proceed.

Syntax

Continue OK:

This command will continue whatever was going on before the debugger was entered.

Continue OPTION (address for this process is) ^CNADDRESS OK:

This command will change the address at which the current target process will resume execution when it is ultimately continued.

Continue At ^CNADDRESS OK:

This command will change the resume address of the current target process and then continue what was going on before the debugger was entered.

Continue At ^CNADDRESS @CNSPEED OK:

This command will change the resume address of the current target process and then continue what was going on before the debugger was entered, at the specified execution speed.

Continue @CNSPEED OK:

This command will continue what was going on before the debugger was entered, but at the newly specified speed.

CNSPEED Rule

Normal (speed):

For (one) @SPDRULE:

Free Command

Overview

Several debugger operations require the use of free cells in the address space of the target process (e.g. breakpoint continuing, executing an instruction on the behalf of the target process). This command allows the user to specify where the debugger should get the cells it requires.

Currently the debugger requires 4 cells to implement breakpoints. It is expected that when instruction execution on the behalf of the target process, specifically procedure calls, is implemented, the debugger will require as many as 2-3 dozen cells (depending on how many parameters are being passed);

The cells that the debugger is currently using can be determined via the Verbose form of the Status command.

Syntax

Free (core available at) ^FCADR OK:

Define Command

Overview

This is the command to give the debugger the definition of a JOVIAL table.

Any number specified in the table and item definition textual strings, such as item size or table dimensions, are treated as decimal values; this is independent of the current or permanent input mode radix for the debugger.

Syntax

Define Table (table definition) ^TABLEDEFINITION (at location) ^DADDRESSLIST @ITMRUL

TABLEDEFINITION

The text to define the table is similar to the format of the table declaration in the JOVIAL language. The format of this text is given below.

TABLEDEFINITION := *TABLENAME DIMENSIONSPEC* [STRUCTURE] [PACKING] ';

TABLENAME := a valid JOVIAL table name

DIMENSIONSPEC := a valid JOVIAL dimension list

STRUCTURE := a valid JOVIAL structure specifier

If STRUCTURE is not specified, serial structure is assumed.

PACKING := a valid JOVIAL packing specifier

If PACKING is not specified, no packing is assumed.

ITMRUL Rule

(item definition / OK if done)

^ITEMDEFINITION

OK

This rule allows the user to type in a definition for each item in the table. When all items have been specified, the user types OK to terminate the rule.

ITEMDEFINITION

This is a textual string that defines an item. It is similar to the format of the table item declaration in the JOVIAL language. The format of this text is given below.

ITEMDEFINITION := TABLEITEMNAME ITEMSPEC [PACKING]
';

TABLEITEMNAME := a valid JOVIAL table item name

ITEMSPEC := a valid JOVIAL item description

Currently for floating point items, only single precision floating point values with default accuracy has been implemented.

PACKING := a valid JOVIAL packing specifier

If PACKING is not specified, whatever packing has been specified for the table is assumed.

Display Command

Overview

This is the basic command for displaying entities (cells, state information, etc.) in the target process.

Syntax

Display ^DADDRESSLIST:

This command will display the specified address list in the mode specified by the ^DADDRESSLIST terminator (and for certain values of this terminator will let the user modify the displayed address list).

DADDRESSLIST Selector

a ^DADDRESSLIST is an ^ADDRESSLIST that is terminated by either an OK or @DTERM

DTERM Rule

OPTION (typeout mode) @OUTTYP OK:

This terminator will cause the specified address list to be displayed in the output mode specified by @OUTTYP.

OPTION (typeout mode) @OUTTYP OPTION (and assign to address list) OK:

This terminator will cause the specified address list to be displayed in the output mode specified by @OUTTYP, and will allow the user to modify the displayed cells as they are being displayed.

*SLASHCHAR:

*BSLASHCHAR:

*LSQUARECHAR:

*EXCMARKCHAR:

*QMARKCHAR:

*EQUALCHAR:

*LARROWCHAR:

*TABCHAR:

*POUNDCHAR:

*LFCHAR:

*UPARROWCHAR:

Find Command

Overview

The find commands allow the user to display, and optionally assign to, those cells in an address list that meet certain content requirements. The user may specify a mask to select those bits in a cell that he or she is interested in checking against similar bits in the value that he or she has specified. In fact, each cell in the address list is logically ANDed with the mask and the result is then compared with the AND of the mask and the specified search value.

The mask used in a reference search is one that will select the address field of a cell. The mask used for content and not content searches is the debugger default mask, unless the user specifies a mask for this instance of the command. The default debugger mask can be displayed and modified via the Mask command. It is initially set to select all bits in a cell.

A reference and a content search will display, and optionally allow the user to assign to, those cells in the address list for which the above mentioned compare was equal. A not content search will display, and optionally assign to, those cells that compare unequally.

All displayed cells will be displayed in the current output mode unless the user specified ^FADDRESSLIST terminator modifies the display.

The user may optionally specify an input mode that will be used to evaluate the specified search value, ^FVALUE.

Syntax

Find References (to) @FSPEC (in address list) ^FADDRESSLIST:

This command will display, in the output mode specified by ^FADDRESSLIST terminator, (and, if this terminator dictates it, assign to) those cells in the specified address list whose address field is equal to the specified ^FVALUE.

Find Content @FSPEC (masked by) @MSPEC (in address list) ^FADDRESSLIST:

This command will display, in the output mode specified by ^FADDRESSLIST terminator, (and, if this terminator dictates it, assign to) those cells in the specified address list whose selected bits, as specified by @MSPEC, are equal to the corresponding bits in the specified ^FVALUE.

Find Not (content) @FSPEC (masked by) @MSPEC (in address list) ^FADDRESSLIST:

This command will display, in the output mode specified by ^FADDRESSLIST terminator, (and, if this terminator dictates it, assign to) those cells in the specified address list whose selected bits, as specified my @MSPEC, are not equal to the corresponding bits in the specified ^FVALUE.

FSPEC Rule

^FVALUE:

the search value

OPTION (input mode) @INPTYP (value) ^FVALUE:

This rule allows the user to specify a current input mode that will be used to evaluate ^FVALUE and ^MVALUE

MSPEC Rule

OK:

Use the default debugger mask

^MVALUE:

The mask to be used for this instance of the find command.

FADDRESSLIST Selector

a ^FADDRESSLIST is an ^ADDRESSLIST that is terminated either with an OK or @FTERM

FTERM Rule

*SLASHCHAR:

*EXCMARKCHAR:

OPTION @OUTTYP OK:

OPTION @OUTTYP OPTION (and assign to address list)
OK:

Mask Command

Overview

This command allows the user to examine or to modify the default debugger mask, which is used by the Find and Memory commands.

Syntax

```
Mask Display OK:

Mask Set (to) ^MVALUE OK:

Mask Set (to) OPTION (input mode) @INPTYP (mask value)
^MVALUE OK:
```

Memory Command

Overview

The memory commands allow the user to set (selected bits) in all cells in the specified address list to the specified value.

If the user does not specify to use a mask, then all bits in the pertinent cells will be affected. If the user specifies to use a mask, then he or she may use either the default debugger mask, or may specify a mask for this instance of the command.

If a mask is used then only those bits selected by the mask will be set, and they will be set to the corresponding bits in the specified ^MNVALUE.

Syntax

Memory (set to) @MNSPEC (in address list) ^MADDRESSLIST:

This command will set the selected bits in the cells in the specified address list to the corresponding bits in the specified ^MNVALUE.

MNSPEC Rule

^MNVALUE:

the value to set the selected bits to

OPTION (input mode) @INPTYP (value) ^MVALUE:

this path allows the user to specify a current input mode that will be used to evaluate ^MNVALUE, and ^MVALUE (if one is specified)

MADDRESSLIST Selector

a ^MADDRESSLIST is an ^ADDRESSLIST is terminated by either an OK or @MTERM

MTERM Rule

OPTION (masked by) OK:

this path indicates to use the default debugger mask to select bits in the address list for modification

OPTION (masked by) ^MVALUE OK:

this path allows the user to specify a mask to use to select bits in the address list to be modified

Output Command

Overview

The output commands give the user the capability to multiplex output to his or her terminal and/or to a sequential text file. If output is currently going only to a file, the user will not have the ability to modify cells in an address list unless the Type command is used. If output is going only to a terminal, the user can force output to a file by use of the Print command. When the user first specifies a file, output will be sent to both the file and the terminal. When specifying a file, the user can either specify a new file, or an old file to which the output should be appended.

Syntax

Output (printing) Display:

Output (printing) Append (to file) ^OLDFILELINK OK:

Output (printing) To (file) ^NEWFILELINK OK:

Output (printing) Off OK:

Output (printing) Both (to file and terminal) OK:

Output (printing) Solely (to) File (and not to terminal) OK:

Output (printing) Solely (to) Terminal (and not to file) OK:

Print Command

Overview

This command is used to display the specified address list on the specified file. If there is already a specified output file, then this is the one that will be used, and the user will not be asked to specify a file. (Note that when using this command, it is not possible to modify the cells as they are being displayed since they will be displayed on an output file and not on the user's terminal. Among other things, this may be useful for "core dumps".)

Syntax

Print ^PADDRESSLIST:

This command will display the specified address list on the current output file in the mode specified by ^PADDRESSLIST terminator.

Print (on file) ^NEWFILELINK ^PADDRESSLIST:

This command will display the specified address list on the specified output file in the mode specified by ^PADDRESSLIST terminator.

PADDRESSLIST Selector

a ^PADDRESSLIST is an ^ADDRESLIST terminated by either an OK or @PTERM

PTERM Rule

OPTION (typeout mode) @OUTTYP OK:

*SLASHCHAR:

*BSLASHCHAR:

*LSQUARECHAR:

*EXCMARKCHAR:

*QMARKCHAR:

*EQUALCHAR:

*TABCHAR:

*POUNDCHAR:

*LFCHAR:

*UPARROWCHAR:

Type Command

Overview

This command is used to display the specified address list (in the specified mode) on the user's terminal regardless of his output file settings.

Syntax

Type ^DADDRESSLIST:

(See the Display command.)

Value Command

Overview

This command is equivalent to:

Display ^ADDRESSLIST *EQUALCHAR:

Syntax

Value (of) ^VADDRESSLIST:

VADDRESSLIST SELECTOR

a ^VADDRESSLIST is an ^ADDRESSLIST terminated by either an OK or @VTERM

VTERM Rule

*EQUALCHAR:

Speed Command

Overview

The speed command allows the user to modify the execution speed of the current process. The execution speed can be modified so that the process will execute in a single step mode (a single machine or language instruction at a time); and/or to treat an entire called procedure as if it were one instruction; and/or to execute until a branch or transfer instruction is encountered; and/or to continue automatically after entering the debugger and notifying the user because one of the above conditions has been met.

Syntax

Speed (of execution) Normal OK:

This command resets the execution speed for the current process back to normal speed.

Speed (of execution) Single @SPDRULE:

This command allows the user to modify the execution of the current process.

*BSLASHCHAR Command

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Z *BSLASHCHAR:
```

Syntax

```
"*BSLASHCHAR":
```

*EQUALCHAR Command

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Z *EQUALCHAR:
```

Syntax

```
"*EQUALCHAR":
```

*EXCMARKCHAR Command

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Z *EXCMARKCHAR:
```

Syntax

```
"*EXCMARKCHAR":
```

*LSQUARECHAR Command

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Z *LSQUARECHAR:
```

Syntax

```
"*LSQUARECHAR":
```

*QMARKCHAR Command

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Z *QMARKCHAR:
```

Syntax

```
"*QMARKCHAR":
```

*SLASHCHAR Command

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Z *SLASHCHAR:
```

Syntax

```
"*SLASHCHAR":
```

*LARROWCHAR Command

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Z *LARROWCHAR:
```

Syntax

```
"*LARROWCHAR":
```

*UPARROWCHAR Command

Overview

This command is equivalent to:

```
Display *ESCAPECHAR A - 1:
```

Syntax

```
"*UPARROWCHAR":
```

*LFCHAR Command

Overview

This command is equivalent to:

```
Display *ESCAPECHAR A + 1:
```

Syntax

```
"*LFCHAR":
```

*TABCHAR Command

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Q:
```

Syntax

```
"*TABCHAR":
```

*POUNDCHAR Command

Overview

This command is the inverse for the last Tab, Linefeed, Uparrow, or Pound command.

Syntax

"*POUNDCHAR":

Common Rules

BASE Rule

Decimal:

Octal:

Hex:

Binary:

INPTYP Rule

Ascii:

Bit:

Bytes (with bytesize of) ^BSVALUE:

Floating (point numbers):

Rad50:

Radix ^RXVALUE:

Radix @BASE:

Sixbit:

Symbolic:

OUTTYP Rule

Addresses (as) Absolute (values):

Addresses (as) Symbolic (values):

Array:

Ascii:

Bit:

Bytes (with bytesize of) ^BSVALUE:

Character: (with size) ^CSVALUE

Floating (point numbers):

Numeric:

Rad50:

Radix ^RXVALUE:

Radix @BASE:

Sixbit:

Symbolic:

NVLRUL Rule

CD:

abort the display of, and assignment to, this address list

OK:

accept the displayed value of this entity

^NVALUE OK:

replace the value of the displayed entity with ^NVALUE, which will be interpreted according to the current input mode

OPTION (input mode) @INPTYP (new value) ^NVALUE OK:

replace the value of the displayed entity with ^NVALUE, which will be interpreted according to the specified input mode

@MOVRUL:

^NVALUE @MOVRUL:

OPTION (input mode) @INPTYP (new value) ^NVALUE @MOVRUL:

the above 3 paths allow the user to terminate the (optionally) newly specified value (for the displayed entity) with the @MOVRUL paths. When the display of, and assignment to, the specified address list is

finished, the last specified @MOVRUL path will take effect as if the user had given the GFC command corresponding the the @MOVRUL path

MOVRUL Rule

*TABCHAR:

*POUNDCHAR:

*LFCHAR:

*UPARROWCHAR:

SPDRULE Rule

In the following, the ordering of @SPDPROC, @SPDEXEC, and @SPDCONT, is not important, and 0, 1, 2, or all 3 of the rules may appear.

@SPDTYPE OK:

@SPDTYPE @SPDPROC OK:

@SPDTYPE @SPDEXEC OK:

@SPDTYPE @SPDCONT OK:

@SPDTYPE @SPDPROC @SPDEXEC OK:

@SPDTYPE @SPDPROC @SPDCONT OK:

@SPDTYPE @SPDEXEC @SPDCONT OK:

@SPDTYPE @SPDPROC @SPDEXEC @SPDCONT OK:

SPDCONT Rule

Proceed (automatically after each instruction):

This rule allows the user to continue automatically after entering the debugger because some single stepping mode condition has been met.

SPDEXEC Rule

Execute (until branch point or transfer instruction):

This rule allows the user to have execution of the process continue until a branch or transfer instruction is encountered.

SPDPROC Rule

Treat (called procedures as one instruction):

This rule allows the user to treat a called procedure as one instruction rather than as a series of instructions.

SPDTYPE Rule

Language (instruction):

This rule means to deal with instructions at the high level language level as opposed to at the machine level.

Machine (instruction):

This rule means to deal with instructions at the machine level as opposed to at the high level language level.

Selectors

In the following discussion an expression is really a text selector that conforms to the rules for expression generation for the current language being used by the debugger.

ADDRESSLIST Selector

text that conforms to the formal definition of an address list (see above)

BRKCMNDS Selector

any text

BRNAME Selector

any text

BSVALUE Selector

a number in the current input mode radix

BTADDRESS Selector

an expression that evaluates to an address

BTNUMBER Selector

a number in the current input mode radix

CNADDRESS Selector

an expression that evaluates to an address

CSVALUE Selector

a number in the current input mode radix

CTEXT Selector

any text

FCADR Selector

a number in the current input mode radix

FCHARACTER Selector

a single non-alphanumeric character

FVALUE Selector

an expression

IDH Selector

a number in the current input mode radix

MNVALUE Selector

an expression

MVALUE Selector

an expression

NEWFILELINK Selector

an new file name string

NVALUE Selector

any text

OLDFILELINK Selector

an old (pre-existing) file name string

PNAME Selector

an expression that evaluates to an address

PNUMBER Selector

a number in the current input mode radix

RXVALUE Selector

a base ten number

SYMADR Selector

a number in the current input mode radix

TENEX-FILE-NAME Selector

a full TENEX file name

Expression Evaluation

User input expressions are evaluated in a left to right order with no precedence of operators. LPARENCHARs and RPARENCHARs (the default values of which are "<" and ">", respectively) can be used, to any practical depth, to group items to modify the default evaluation.

If a value is not found for a specified symbol, then it is looked up in the builtin opcode table, if and only if it is the first symbol in the expression. If it is still not found an error is generated.

Spaces may be used freely in an expression to increase legibility. However, two identifiers separated only by spaces will be assumed to be separated by a PLUSCHAR.

The formal definition and semantics of an expression follows:

EXPRESSION := HALFWORD / ASSEMBLY / ARITH

HALFWORD := EXPRESSION ",," EXPRESSION

Each expression will be evaluated and the resulting right-halves of each expression will comprise the left- and right- halves of the final expression.

ASSEMBLY := OPCODE [REG-EXP ',] ['@] [ADR-EXP] ['(INDX-EXP ')]

This corresponds to the assembly language syntax for a PDP-10. Note that each field (except the opcode field) may itself be an expression.

OPCODE := an identifier in the builtin opcode table

REG-EXP := EXPRESSION

This expression will be evaluated and then the least significant four bits will be placed in the register field of the enclosing expression being evaluated.

ADR-EXP := EXPRESSION

This expression will be evaluated and then the least significant eighteen bits will be placed in the address field of the enclosing expression being evaluated.

INDX-EXP := EXPRESSION

This expression will be evaluated and then the least significant four bits will be placed in the index register field of the enclosing expression being evaluated.

```
ARITH      := TERM / TERM ADD-OP TERM
```

Basically, this is a sum of products (or if you rather a product of sums).

```
TERM       := IDENT / TERM MUL-OP TERM / LPARENCHAR TERM
RPARENCHAR

ADD-OP     := PLUSCHAR / MINUSCHAR / one or more spaces

MUL-OP     := TIMESCHAR / DIVIDECHAR
```

JOVIAL Tables

JDAD supports the JOVIAL table data structure. The Display command can be used to display and modify data in JOVIAL tables.

The Define command is used to give JDAD the description and location of any JOVIAL table. The debugger has an internal data structure to store JOVIAL table definitions. Each Define command creates an entry for one table and all the items in the table in this structure. Once an entry for a JOVIAL table has been made in this structure, the table and/or items within the table can be used in address lists in the Display command. A formal description of the table and item definition text is given in the section on the Define command. The definition includes the name of the table, the name of each item in the table, the dimensions of the table, the structure (parallel or serial), the packing parameter and the location of the table.

When JDAD is parsing an address list, the occurrence of a left parenthesis indicates to JDAD that a table or table item is being referenced. This left parenthesis is the left delimiter of the dimension list for the table or table item. A table is referenced by the table name given in the Define command. An item is referenced by the name of the table, followed by the *BLOCKCHAR character and the item name from the Define command. The formal definition for a table and table item address specification is given under Address Lists -- Formal Definition in the Concepts section.

In the formal definition the DIMENSIONRANGE is defined as follows:

```
DIMENSIONRANGE := ('* / NUMBER ': NUMBER / NUMBER)
```

An asterisk in this context means to display all indices for that dimension. If the DIMENSIONLIST consists of a single asterisk, all indices of all dimensions are displayed. If a dimension is specified with a single number, only that specific index value of the dimension will be displayed. If a range of numbers is specified for a dimension (i.e. NUMBER ': NUMBER), the index values for the first number through the second will be displayed.

Currently only Ordinary Tables are supported in the JOVIAL debugger.

Single Stepping

If the user specifies an execution speed other than normal speed (via the speed or continue commands), and specifies single language instruction mode, then JDAD should find the JOVIAL instruction boundaries and single step at those boundaries.

In order to do this, a JOVIAL program would have to be compiled with the ISD (internal symbol dictionary) switch on and JDAD would need to interpret the ISD. JDAD does not currently include code to interpret the ISD.

Thus, if you are single language stepping, a break will occur before every assembly language instruction.

Appendix I - Alphabetical List of Commands, Rules, and Selectors

Commands

Rules

Selectors

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.