

CDC - ADVANCED SYSTEMS DEVELOPMENT
ERS for CYBIL-CC Interactive Debugger

1
09/29/80
REV: D

ERS
FOR
CYBIL-CC INTERACTIVE DEBUGGER

Submitted: Sherman Eagles-----

Approved: -----

DISCLAIMER:

This document is an internal working paper only. It is subject to change and does not necessarily represent any official intent on the part of CDC.

COMPANY PRIVATE

CDC - ADVANCED SYSTEMS DEVELOPMENT
ERS for CYBIL-CC Interactive Debugger

09/29/80
REV: D

REVISION DEFINITION SHEET

REV	DATE	DESCRIPTION
A	01/05/79	Original.
B	05/07/79	Updated for release
C	01/20/80	Updated for SES release 13
D	09/29/80	Updated to add LINE TRAPS

c 1977, 1978, 1979, 1980
by Control Data Corporation

1.0 SCOPE

1.0 SCOPE

This document describes the external features and characteristics of CYBIL-CC Interactive Debug, a supervisor program running under the NDS operating system. It is primarily directed toward CYBER 180 software development programmers who are assumed to be familiar with CYBIL-CC and NDS.

1.1 APPLICABLE DOCUMENTS

Publication	Number
NDS Reference Manual Volume 1	60445400
NDS Reference Manual Volume 2	60445300
Network Products Interactive Facility Version 1 Reference Manual	60455250
Network Products Interactive Facility Users Guide	60455260
CYBER Interactive Debug Version 1 Reference Manual	60481400
CYBIL Language Specification	ARH2298
CYBIL V2.0 Implementation Dependent Handbook	ARH3078

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

2.0 INTRODUCTION

2.0 INTRODUCTION

CYBIL-CC Interactive Debug (CCDBG) extends the capabilities of CYBER Interactive Debug (CID) to allow symbolic debugging of CYBIL-CC programs. Locations within a CYBIL-CC program can be referenced by variable names and line numbers if the program was compiled with the DEBUG option selected. This option causes CYBIL-CC to produce special tables for CCDBG as part of the object code.

The CID command syntax has been completely changed in CCDBG to make it conform with the NDS/180 System Command Language. Also, many changes have been made to the parameters of most commands to handle CYBIL-CC modules, procedures and stacks instead of Fortran structures. The overall capabilities of CCDBG, however, differ little from those of CID. There are only four new commands: FORWARD, BACKWARD, DISPLAY_DEFAULTS and CHANGE_DEFAULTS. The COMPASS or machine level capability of CID has been maintained, and the overall design of CID has not been changed.

2.1 OVERVIEW

CCDBG is a supervisory program loaded in the user field length to operate on loaded object programs. No special source statements are needed in the program to be debugged; however, a special compiler option must be selected if symbolic debugging is desired. In addition to its symbolic referencing capabilities, CCDBG controls program execution as determined by user-defined breakpoints and traps, allows the user to look at and change memory and registers, and defines a sequence of CCDBG commands and gives them a procedure name so that they will execute each time the name is referenced or on occurrence of a specified breakpoint or trap. Breakpoint, trap and procedure definitions may be saved on a file for use in future sessions, and the session may be checkpointed to be restarted later.

COMPANY PRIVATE

2.0 INTRODUCTION

2.2 TERMINOLOGY

2.2 TERMINOLOGY

Breakpoint	A location within a program where the program's execution may be interrupted by a monitor routine.
Default Module	The name of the CYBIL-CC module that was executing when CCDBG gained control, unless a subsequent CHANGE_DEFAULTS command specified a new module name as the default module. At initial start-up of CCDBG, the default module is the module at which program execution is to begin. Default module is equivalent to HOME PROGRAM in CID.
Default Proc	The CYBIL-CC procedure that was executing when CCDBG gained control, unless a subsequent CHANGE_DEFAULTS command specified a new procedure name as the default proc. At initial start-up of CCDBG, the default proc is undefined.
Entry Point	A special named location within a program which may be accessed by programs compiled separately. If there is an alias, the alias name will be used. Entry points are limited to 7 characters by the CYBER Loader, if they are longer than 7 characters, only the first 7 characters will be used. To reference an entrypoint that contains a character that is not legal in a name, the entrypoint name must be enclosed in single quotes.
Expression	An algorithm used for computing a value. A CCDBG expression may contain symbolic address references, CCDBG variables, numeric literals, and the operators + and -. They may be used to express an address or a value.
Heap	A dynamic storage area explicitly allocated and freed by CYBIL-CC programs.
Interactive	Capable of a two-way back and forth exchange of information.
Interpret	Execution of computer machine instructions by other than direct means. A special routine examines each instruction and simulates its execution.
Map	A list of module and entry point memory addresses, produced when the program is loaded.

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

2.0 INTRODUCTION

2.2 TERMINOLOGY

Module	A compilation unit in CYBIL-CC. This is the alias defined in CYBIL-CC or the first 7 characters of the module name if there is no alias. To reference a module which contains a character that is not legal in a name, the module name must be enclosed in single quotes.
Nested	A procedure that is defined within another procedure.
Overlay	A portion of a program which can share an area of memory with other similar program portions. When access to a particular portion of the program is required, the overlay containing that portion is loaded, overlaying the previous contents of the memory area.
Procedure	A named portion of a program in CYBIL-CC. This is the actual name defined in CYBIL-CC, not an alias.
Program	The completely loaded set of one or more modules.
Separator	A character required between two items so that they may be distinguished. In CCDBG, spaces or commas are used as separators between parameters and between elements in a list, and semi-colons and end-of-line are used as separators between commands.
Stack	A dynamic storage area automatically allocated by CYBIL-CC programs on each procedure call.
Trace	An ordered list of procedures or modules that have executed, showing program flow.

2.3 METALANGUAGE

Throughout this document, whenever a CCDBG command is discussed, the manner of writing that command is illustrated with a uniform system of notation. This notation is called a metalanguage and is not part of CCDBG. Through the use of a metalanguage we are able to provide a brief, but precise, explanation of the general patterns that CCDBG permits. The metalanguage does not describe the meaning of the language elements, merely their structure; i.e., it indicates the order in which the elements may (or must) appear, punctuation that is

2.0 INTRODUCTION

2.3 METALANGUAGE

required, and options that are allowed. These following metalanguage rules apply:

- 1) The symbol ::= is read as "IS DEFINED TO BE".
- 2) Elements enclosed by < > are to be considered a single syntactic unit in relation to surrounding meta symbols.
- 3) Elements enclosed by [] are optional and are to be considered a single syntactic unit in relation to surrounding meta symbols.
- 4) Elements separated by ; are mutually exclusive, and the symbol is read as "OR".
- 5) Elements followed by ... can be repeated.
- 6) <..> will be used to indicate that an ellipsis (two or more periods) is required. In this case, the ellipsis is part of the language.
- 7) <ascii> will be used to indicate that an ascii character is required.
- 8) <sp> will be used to indicate that a space is required.
- 9) The symbol EOL will be used to indicate end of line.

2.4 BASIC_CONCEPTS

2.4.1 INTEGER

<integer> ::= <dec_digit> [<digit>...] [(<base>)]

Integers may be expressed as octal (8), decimal (10), or hexadecimal (16). When the base specification is omitted, decimal (10) is assumed. When a hexadecimal representation is specified, a leading zero may be required to ensure that the constant begins with a decimal digit.

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

2.0 INTRODUCTION

2.4.2 NAME

2.4.2 NAME

<name> ::= <alpha> [<alpha>!<digit>] ...

A name is a string of alphanumeric characters not contained in a comment or constant. The name must be preceded and followed by a delimiter. Any character not allowed inside a name delimits a name. The initial character of a name must not be a digit and the number of characters in a name must not exceed 31.

2.4.3 ADDRESS

<address> ::= <expr>

An address is an expression which when evaluated, provides an 18 bit central memory address or a 24 bit ECS address.

2.4.4 CCDBG VARIABLES

These variables are part of CCDBG and are available to the user. They all have identifiers beginning with #. Ten user variables designated #V1, #V2, ..., #V10 may be changed by the user. The other variables may be accessed by the user, but may not be changed.

#BP	Number of breakpoints currently defined.
#TP	Number of traps currently defined.
#PR	Number of debugger SCL procedures currently defined.
#FL	Central memory field length.
#P	Program address register.
#ERRCODE	Reprive error code.
#CPUERR	Mode error code.

The following CCDBG variables may only be accessed when CCDBG is in interpret mode.

COMPANY PRIVATE

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

2.0 INTRODUCTION

2.4.4 CCDBG VARIABLES

```

#INS          Current instruction as number.
#INSL         Current instruction length (15, 30 or 60)
#PARCEL       Instruction parcel counter.
#OP           OP code of current instruction.
#I            i field of current instruction.
#J            j field of current instruction.
#K            k or K field of current instruction.
#EA           Effective address of current instruction.
#EW           Effective word.
#PC           Previous contents.
#PA           Previous address.

```

2.4.5 EXPRESSION (EXPR)

```

<expr> ::= [<operator>] <operand> [<operator><operand>]...
<operator> ::= <+>!<->
<operand> ::= <constant>!<name>!(<expr>)

```

An expression is an algorithm for computing a value.

2.4.6 SEPARATOR

```

<sep> ::= <sp>!<, >

```

A comma or a space may be used to separate parameters or elements in a list.

2.4.7 COMMENTS

```

<comment> ::= "<ascii> [<ascii>]..."

```

Comments are not interpreted by CCDBG and serve only as

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

2.0 INTRODUCTION

2.4.7 COMMENTS

documentation. A comment acts syntactically the same as a space; i.e., whenever a space is allowed a comment is allowed, and whenever a space is required as a delimiter a comment will serve the same purpose.

2.4.8 COMMAND

`<command> ::= <command name> [<sep><param list>]`

A command consists of the command name followed by any parameters necessary to control its operation.

2.4.9 PARAMETER LISTS

`<param list> ::= <param> [<sep><param>]...`

A parameter list consists of a series of parameters separated by spaces or commas. Each parameter in the list can be specified in one of three formats. The first format consists simply of the parameter name.

`<param name>`

The second format consists of the name followed by parameter text.

`<param name>< = ! <sp>> <param value>`

Both of the above formats are positionally independent; i.e., the order in which they are quoted is unimportant. The third format is positionally dependent and consists simply of parameter text. The positional significance of a parameter is one greater than the previous parameter specified in the list. Omission of a parameter can be indicated by two consecutive commas.

`<param value>`

2.4.10 VALUE LISTS

`<value list> ::= (<value> [<sep><value>]...)`

A value list consists of a series of values separated by

COMPANY PRIVATE

2.0 INTRODUCTION

2.4.10 VALUE LISTS

spaces or commas and enclosed in parentheses.

2.4.11 CONTINUATION

<input line> ::= <text><..><EOL><text>

Commands can be continued by placing an ellipsis at the end of the line. The first character of the continuation line replaces the first character of the ellipsis. The total number of characters must not exceed 150. Due to an implementation language limitation, the maximum length of a line of lower case letters (ASCII 6/12 format) is 120.

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

3.0 GENERAL DESCRIPTION OF CCDBG USE

3.0 GENERAL DESCRIPTION OF CCDBG USE

A CCDBG session must begin with a DEBUG(ON) control statement that places the operating system in CCDBG mode. It remains in CCDBG mode until the control statement DEBUG(OFF) is entered. When the operating system is in CCDBG mode, the CCDBG supervisory module is loaded as part of the system response to a request for the load of an object program. Subsequent execution begins within the CCDBG module, not the user program, and the following message appears at the terminal:

```

CYBIL-CC INTERACTIVE DEBUG
?
```

The question mark is a prompting character issued by CCDBG each time it is waiting for a response from the terminal.

Once the CCDBG header line appears, the user should enter the CCDBG commands that would set breakpoints and traps, specify output options, or preset any data values. A command of EXECUTE or GO starts execution of the object program.

When any of the specified conditions occur, the condition is reported, the execution is suspended, and control passes to the user at the terminal. Diagnostics and trap and breakpoint reports are displayed with a preceding asterisk.

During the time the user has control, CCDBG commands can be entered to display program locations, change location values, set additional breakpoints and traps, and generally explore the behavior of the program. If necessary, the HELP command can be executed to learn about CCDBG commands.

Program execution resumes at the location where it was suspended, or at a user specified location. Any abnormal program abort, as well as normal program termination, returns control to CCDBG.

Debugging of a particular program ends when the QUIT command is entered. This command terminates CCDBG control, and other terminal operations can be performed. If any user program is subsequently referenced in a load request, however, CCDBG is again loaded and gains control.

COMPANY PRIVATE

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

3.0 GENERAL DESCRIPTION OF CCDBG USE

CCDBG features are listed below, along with general information about them.

3.1 PROGRAM_EXECUTION_CONTROL

Program execution halts whenever a user-defined breakpoint or trap condition occurs. Execution can be resumed at the point at which it was interrupted or at any other specified program location within the same procedure.

An interrupt command provides a general stop-on-demand capability that can terminate infinite loops or excessive output.

3.2 CYBIL-CC_SOURCE_SYMBOL_REFERENCE_CAPABILITY

Locations within a CYBIL-CC module can be referenced by variable names and line numbers if the module was compiled with the DEBUG option selected. Fields of records may be referenced using the same notation used in CYBIL-CC (including pointer notation). Array elements may be referenced using a subscript only if the subscript is a constant of the same type as the array index. Variables and names used as constants are not allowed as subscripts.

3.3 MACHINE_LEVEL_DEBUGGING_FEATURES

Several features relating to the hardware instruction set and program registers are available.

Program register values can be examined and changed.

Step mode execution by trapping each instruction prior to execution.

Instruction-oriented trap definitions cause execution in interpret mode. Each time execution stops, the i, j and k operands, and the effective address of the current instruction can be displayed.

After a write or read, the value can be displayed, and the value previously written or the prior contents of the X

COMPANY PRIVATE

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

3.0 GENERAL DESCRIPTION OF CCDBG USE

3.3 MACHINE LEVEL DEBUGGING FEATURES

register before a read can be displayed. The previous contents of the A register can also be displayed.

3.4 OVERLAY_ENVIRONMENT

Program execution can be trapped when particular overlays are loaded. Details of the overlay structure and of the loaded overlays can be indicated. An overlay qualifier can be used in specifying an address or module name.

3.5 DEBUGGER_SCL_PROCEDURE

A debugger SCL procedure consists of a sequence of CCDBG commands and can be given a procedure name, such that all commands in the sequence execute each time the procedure name is referenced.

Similarly, a breakpoint or a trap can be defined with a CCDBG SCL procedure that executes each time the specified event occurs.

3.6 CONDITIONAL_COMMAND_EXECUTION_CAPABILITY

A SKIPIF command within a debugger SCL procedure allows some CCDBG commands to execute only when particular program values or status variables exist.

3.7 VETO_MODE

Veto mode gives the user veto power over each individual command within a debugger SCL procedure that is otherwise executing automatically.

3.8 DEFINITION_FILE_CAPABILITY

Definitions of traps, breakpoints, and debugger SCL procedures can be saved on a file. This feature eliminates the need to re-enter long CCDBG sequences in future sessions.

COMPANY PRIVATE

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

.....

3.0 GENERAL DESCRIPTION OF CCDBG USE

3.9 CHECKPOINT/RESTART CAPABILITY

.....

3.9 CHECKPOINT/RESTART CAPABILITY

A CCDBG session can be checkpointed. The session may later be restarted with the CCDBG environment intact.

3.10 WARNING CAPABILITY

A warning message is displayed prior to the execution of a command that would destroy existing definitions, or that might produce uncertain results.

3.11 CCDBG VARIABLES

Special identifiers can reference CCDBG variables containing information such as current program address, field length, number of breakpoints, traps or debugger SCL procedures defined. Ten scratchpad variables can be fully controlled by the user.

3.12 INFORMATION OUTPUT

Listings that can be obtained during a CCDBG session include:

- Load map information
- Current CCDBG environment
- Currently defined CCDBG SCL procedures

At any time during the CCDBG session, the user can direct a log of the remainder of the session to be written to a file for later printing.

3.13 HELP COMMAND

The HELP command can be used any time during a session to obtain a summary of information about CCDBG features.

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.0 CCDBG_COMMANDS

The following sections describe the features of CCDBG commands. All CCDBG commands adhere to the System Command Language (SCL) definition as specified in the NOS/180 Preliminary ERS. The description of the parameters denotes the positional order of the command parameters.

4.1 BREAKPOINTS AND TRAPS

Breakpoints and traps provide a means for monitoring program execution. When a breakpoint location is reached, execution of the program is suspended, and CCDBG gains control. For a trap, CCDBG gains control when some specific condition occurs. In either case CCDBG commands may be processed, allowing a programmer to look at and change elements of the executing program.

Both traps and breakpoints may be established with "bodies". A body is a sequence of CCDBG commands that are executed automatically when the trap or breakpoint occurs. In this case, the user does not receive notification of the breakpoint or trap, nor does the user receive control unless a PAUSE command is executed as part of the body of the breakpoint or trap.

The fact that a trap or breakpoint is being established with a body is indicated by the presence of the collect parameter on a SET_BREAKPOINT or SET_TRAP command. All subsequent commands, until a collect_end is encountered, constitute the body. These commands are checked for syntax errors when they are entered, but they are not executed until the breakpoint or trap condition occurs.

4.1.1 SET_BREAKPOINT : SB

This command sets a breakpoint in the user's program at a specified location. It may also start the definition of a body (set of CCDBG commands between collect and collect_end).

COMPANY PRIVATE

4.0 CCDBG COMMANDS

4.1.1 SET_BREAKPOINT : SB

```

set_breakpoint      [address_expr]
                    [module=<name>][overlay=(<integer>
                    ,<integer>)] [offset=<integer>]
                    [first=<integer>] [last=<integer>]
                    [step=<integer>] [<collect><eol!;>]
                    [<command_statement><eol!;>]...
                    <collect_end>
    
```

address_expr: The address specified by this parameter is known as the base address and may be specified in any of the following ways:

line : l=<integer>

Where integer is a CYBIL-CC line number generated by the compiler.

entrypoint : e=<name>

Where name is an entry point identifier.

NOTE: automatic variables and parameters are not available until the procedure prolog is completed. This form is included primarily for use with COMPASS entry points.

location : loc=<integer>

Where integer is an absolute address.

If no keyword is specified in the order dependent format, line number will be assumed.

module : m: CYBIL-CC compilation unit to which line or label applies. If it is not specified, the default module is used. If address_expr is not specified, the base address is the beginning of the module.

overlay : ovl: This specifies the overlay number in which the breakpoint is to be set. Default is to use the loaded overlays in the following order: 1) overlay which is currently being executed, 2) main overlay, 3) primary overlay, 4) secondary overlay.

offset : o: A displacement which is added to the base address to form the effective memory address. Default is zero.

first : f: The number of times the breakpoint must be reached before it is honored. Default is 1.

last: The breakpoint will not be honored again after it has been reached this number of times. Default is infinity.

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.1.1 SET_BREAKPOINT : SB

step : s: Frequency parameter. Breakpoint will be honored every step times it is reached. Default is 1.

collect : c: Activates collect mode to establish a body for this breakpoint.

command_statement: CCDBG commands that make up the body of this breakpoint.

collect_end : ce: Terminates collect mode.

Upon establishment of a breakpoint, a number is assigned. This breakpoint number, #n, is useful in referring to breakpoints, and is used in the breakpoint reporting message.

When a breakpoint is reached and the frequency criteria are met, the breakpoint is honored. The message is:

```
* B #n AT location
```

where n is the breakpoint number, and location defines the location as specified in the SET_BREAKPOINT command. After issuing the message, CCDBG is ready to accept new commands.

Examples: SB 10

Sets a breakpoint at line 10 of the default module.

```
SB e=simple offset=10 f=11 c
DR b=4;PA;ce
```

Sets a breakpoint 10 locations after entrypoint simple. It is honored the 11th time it is reached, and each time thereafter. When the breakpoint is honored, register B4 is displayed and a pause is done.

4.1.2 SET_TRAP : ST

This command establishes a trap. It indicates the type of trap to be established and the scope of the trap. If the scope is not specified, the condition is trapped anywhere in the program.

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.1.2 SET_TRAP : ST

```

set_trap : st      type=<trap_type> [scope_expr]
                  [overlay=(<integer>,<integer>)]
                  [module=<name>] [proc=<name>
                  [.<name>],...] [<collect><eol!;>]
                  [<command_statement><eol!;>]...
                  <collect_end>

```

type: Specifies condition which will cause interruption of program execution. Trap_type must be one of the following.

```

OVERLAY : OVL
END : E
ABORT : A
INTERRUPT : INT
INSTRUCTION : I
LINE
RJ
XJ
JUMP : J
READ : R
WRITE : W

```

NOTE: If trap_type is OVERLAY or LINE, the keyword TYPE must be specified, since OVERLAY, OVL and LINE are also used as keywords in this command.

scope_expr: This specifies where the trap is to apply. It is an address range which may be specified by any of the following:

line : l=<integer>[<..><integer>]

Where integer is a CYBIL-CC line number generated by the compiler.

location : loc=<integer>[<..><integer>]

Where integer is an absolute address.

offset o=<integer>[<..><integer>]

Where integer is an address relative to a module.

variable : var=<name>

Where name is the identifier of the variable. This form is used for READ and WRITE type traps. NOTE: Dynamic variables are assigned only when the proc in which they are declared is active. Traps on dynamic variables should not remain established after the procedure completes.

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.1.2 SET_TRAP : ST

If the order dependent format is used and a keyword is not given, line will be assumed.

overlay : ovl: If trap_type is OVERLAY, this specifies which overlay number to trap. For any trap_type other than overlay, it may be necessary to specify which overlay the address range applies to. Default is to use the loaded overlays in the following order: 1) overlay which is currently executing, 2) main overlay, 3) primary overlay, 4) secondary overlay.
NOTE: Overlay numbers default to decimal on input.

module : m: CYBIL-CC compilation unit to which scope_expr applies. If it is not specified, the default module is used.

proc : p: CYBIL-CC procedure name to which variable applies. Nested procedures may be referenced in the format proc1.proc2.proc3, where proc3 is the desired procedure. If proc is not specified, the default proc is used.

collect : c: Activates collect mode to establish a body for this trap.

command_statement: CCDBG commands that make up the body of this trap.

collect_end : ce: Terminates collect mode.

Upon establishment of a trap, a number is assigned. This trap number, #n, is useful in referring to traps and is used in the trap message. A trap remains established for the remainder of the CCDBG session, unless it is redefined or cleared.

When the condition specified in an established trap is encountered, trap action occurs. The message to the user is:

```
* T #n, type trap_message xx location
```

Where n is the trap number, type is the trap_type specified in the SET_TRAP command, trap_message is additional information depending on type, xx is AT or IN, and location identifies where the trap occurred. After issuing the message, CCDBG is ready to accept new commands.

Examples: ST t=jump l=30..70

4.0 CCDBG COMMANDS

4.1.2 SET_TRAP : ST

Sets a trap on jumps between line 30 and line 70 of the current default module.

ST t=write var=test_result

Sets a trap on writes into variable test_result. :

ST TYPE = LINE :

This example allows you to step through your program executing one line at a time. :

```
ST TYPE=LINE LINE=120..140 COLLECT
CV #V1 #P-1
DV #V1 F=ADR
COLLECT_END
```

This example sets a trap on the execution of any of the lines specified in the scope range (120-140), and displays the line number that was trapped. :

```
ST type=line c
CV #v1 #p-1
DV #v1 f=adr
GO
ce
```

This example creates a trace, displaying each line number as it is executed. :

4.1.3 DISPLAY_BREAKPOINT : DB

This command lists all breakpoints in a program (or part of a program) or if a specific breakpoint number or a specific location is given the body of that breakpoint is displayed.

```
display_breakpoint : db [address_expr]
                        [b=<integer>[<..><integer>] ;
                        [<integer>[<..><integer>]
                        [<sep><integer>[<..><integer>]]...]]
                        [offset=<integer>]
                        [overlay=(<integer>,<integer>)]
                        [module=<name>]
```

address_expr: Specifies the location or range of locations to display breakpoints from. It must be one of the

09/29/80

REV: D

ERS for CYBIL-CC Interactive Debugger

4.0 CCDBG COMMANDS

4.1.3 DISPLAY_BREAKPOINT : DB

following:

line : l=<integer>[<..><integer>]

Where integer is a CYBIL-CC line number generated by the compiler.

location : loc=<integer>[<..><integer>]

Where integer is an absolute address.

entrypoint : e=<name>

Where name is an entry point identifier.

If the order dependent format is used and a keyword is not given, line will be assumed.

b: The breakpoint number assigned when the breakpoint was established.

offset : o: A displacement that is added to the base address to form the effective memory address. Default is zero.

overlay : ovl: This parameter may be necessary to specify which overlay the address range applies to. Default is to use the loaded overlays in the following order: 1) overlay which is currently executing, 2) main overlay, 3) primary overlay, 4) secondary overlay.

module : m: The CYBIL-CC compilation unit to which the line or offset parameter applies. If module is not specified, the default module is used.

Examples: DB 3

Displays the body of breakpoint number 3.

DB overlay=(1,3)

Lists all the breakpoints in overlay number (1,3).

4.1.4 DISPLAY_TRAP : DT

This command lists traps in all or part of the program; or, if a specific trap number or specific location is given, the body of that trap is listed. If traps from a range of locations are being listed, the user may have all types of traps listed, or may list only those traps of a specified type.

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.1.4 DISPLAY_TRAP : DT

```

display_trap : dt      [type=<trap_type>] [scope_expr]
                       [t=<integer>[<..><integer>] ;
                       (<integer>[<..><integer>]
                       [<sep><integer>[<..><integer>]]...)]
                       [overlay=(<integer>,<integer>)]
                       [module=<name>] [proc=<name>
                       [.<name>]]...]
```

type: Specifies type of trap to be displayed. Trap_type must be one of the following:

```

OVERLAY : OVL
END : E
ABORT : A
INTERRUPT : INT
INSTRUCTION : I
LINE
RJ
XJ
JUMP : J
READ : R
WRITE : W
```

If no type is specified, all types are displayed.

NOTE: If trap_type is OVERLAY or LINE, the keyword TYPE must be specified, since OVERLAY, OVL and LINE are also used as keywords in this command.

scope_expr: Specifies the location or range of locations to display traps from. It must be one of the following:

line : l=<integer>[<..><integer>]
Where integer is a CYBIL-CC line number generated by the compiler.

location : loc=<integer>[<..><integer>]
Where integer is an absolute address.

offset : o=<integer>[<..><integer>]
Where integer is an address relative to a module.

variable : var=<name>
Where name is the identifier of the variable.

If the order dependent format is used and no keyword is given, line will be assumed.

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.1.4 DISPLAY_TRAP : DT

t: The trap number assigned when the trap was established.

overlay : ovl: This parameter may be necessary to specify which overlay the address range applies to. Default is to use the loaded overlays in the following order: 1) overlay which is currently executing, 2) main overlay, 3) primary overlay, 4) secondary overlay.

NOTE: Overlay numbers default to decimal on input.

module : m: The CYBIL-CC compilation unit to which the line or offset parameter applies. If it is not specified, the default module is used.

proc : p: The CYBIL-CC procedure name to which variable applies. If not specified, the default proc is used. Nested procs may be referenced in the format proc1.proc2.proc3, where proc3 is the desired procedure.

Examples: DT m=my_mod

Lists all traps of any type in module my_mod.

DT type=rj

Lists all RJ traps set in the program.

4.1.5 SAVE_BREAKPOINT : SAVEB

This command saves the breakpoint definition on a local file. Either specific breakpoints or all breakpoints in the program or part of the program may be saved. If the breakpoint has a body, it will also be saved. If no qualifying parameters are specified, all breakpoints will be saved.

```
save_breakpoint : saveb      file=<name> [scope_expr]
                             [b=<integer>[<..><integer>] ;
                             [<integer>[<..><integer>]
                             [<sep><integer>[<..><integer>]]...)]
                             [offset=<integer>]
                             [overlay=(<integer>,<integer>)]
                             [module=<name>]
```

file : f: The name of a local file to write breakpoint definitions on.

COMPANY PRIVATE

4.0 CCDBG COMMANDS

4.1.5 SAVE_BREAKPOINT : SAVEB

scope_expr: This specifies the location or range of locations to save breakpoints from. It must be one of the following:

line : l=<integer>[<..><integer>]

Where integer is a CYBIL-CC line number generated by the compiler.

location : loc=<integer>[<..><integer>]

Where integer is an absolute address.

entrypoint : e=<name>

Where name is an entry point identifier.

If the order dependent format is used and no keyword is given, line will be assumed.

b: The breakpoint number assigned when the breakpoint was established.

offset : o: A displacement that is added to the base address to form the effective memory address. Default is zero.

overlay : ovl: This parameter may be necessary to specify which overlay the address range applies to. Default is to use the loaded overlays in the following order: 1) overlay which is currently executing, 2) main overlay, 3) primary overlay, 4) secondary overlay.

module : m: The CYBIL-CC compilation unit to which line or offset apply. If it is not specified, the default module is used.

Examples: SAVEB f=bpfile b=(2,3)

Saves breakpoints number 2 and number 3 on local file BPFIL.

SAVEB, bps, 10..40,,, (1,3), xyz12

Saves on file BPS, all breakpoints in lines 10 to 40 which are in module xyz12 on overlay (1,3).

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.1.6 SAVE_TRAP : SAVET

4.1.6 SAVE_TRAP : SAVET

This command saves the trap definition on a local file. Either specific traps or all traps in the program or part of the program may be saved. All traps may be saved, or just one type of trap may be saved. If a trap has a body, it will also be saved. If no trap number, type, or scope_expr is specified, all traps will be saved.

```
save_trap : savet      file=<name>[type=<trap_type>]
                       [scope_expr]
                       [t=<integer>[<..><integer>] ;
                       (<integer>[<..><integer>]
                       [<sep><integer>[<..><integer>]]...)]
                       [overlay=(<integer>,<integer>)]
                       [module=<name>] [proc=<name>]
                       [.<name>]...]
```

file : f: The local file to write trap definitions to.

type: Specifies the type of trap to be saved. Trap_type must be one of the following.

```
OVERLAY : DVL
END : E
ABORT : A
INTERRUPT : INT
INSTRUCTION : I
LINE
RJ
XJ
JUMP : J
READ : R
WRITE : W
```

If no type is specified, all types are saved.

NOTE: If trap_type is OVERLAY or LINE, the keyword TYPE must be specified, since OVERLAY, DVL and LINE are also used as keywords in this command.

scope_expr: This specifies the location or range of locations to save traps from. It must be one of the following:

```
line : l=<integer>[<..><integer>]
      Where integer is a CYBIL-CC line number generated by
      the compiler.
```

COMPANY PRIVATE

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.1.6 SAVE_TRAP : SAVET

location : loc=<integer>[<..><integer>]

Where integer is an absolute address.

offset : o=<integer>[<..><integer>]

Where integer is an address relative to the beginning of a module.

variable : var=<name>

Where name is the identifier of the variable. This form is used for READ and WRITE traps.

If the order dependent format is used and no keyword is given, line will be assumed.

t: The trap number assigned when the trap was established.

overlay : ovl: This parameter may be necessary to specify which overlay the address range applies to. Default is to use the loaded overlays in the following order: 1) overlay currently executing, 2) main overlay, 3) primary overlay, 4) secondary overlay.

NOTE: Overlay numbers default to decimal on input.

module : m: The CYBIL-CC compilation unit to which the line or offset parameter applies. If it is not specified, the default module is used.

proc : p: The CYBIL-CC procedure name to which var applies. If it is not specified, the default proc is used. Nested procedures may be referenced in the format proc1.proc2.proc3, where proc3 is the desired procedure.

Examples: SAVET write

Save all traps of type write.

SAVET module=l_test

Saves all traps in module l_test.

4.1.7 CLEAR_BREAKPOINT : CB

This command clears specific breakpoints, or all breakpoints in the program or part of the program. If no parameters are specified, a warning message will be issued. If the user accepts

4.0 CCDBG COMMANDS

4.1.7 CLEAR_BREAKPOINT ; CB

the warning, all breakpoints will be cleared.

```
clear_breakpoint ; cb      [scope_expr]
                          [b=<integer>[<..><integer>] ;
                          (<integer>[<..><integer>]
                          [<sep><integer>[<..><integer>]]...)]
                          [offset=<integer>]
                          [overlay=(<integer>,<integer>)]
                          [module=<name>]
```

scope_expr: This specifies the location or range of locations to clear breakpoints from. It must be one of the following:

line ; l=<integer>[<..><integer>]

Where integer is a CYBIL-CC line number generated by the compiler.

location ; loc=<integer>[<..><integer>]

Where integer is an absolute address.

entrypoint ; e=<name>

Where name is an entry point identifier.

If the order dependent format is used and no keyword is given, line will be assumed.

b: The breakpoint number assigned when the breakpoint was established.

offset ; o: A displacement that is added to the base address to form the effective memory address. Default is zero.

overlay ; ovl: This parameter may be necessary to specify which overlay the address range applies to. Default is to use the loaded overlays in the following order: 1) overlay currently executing, 2) main overlay, 3) primary overlay, 4) secondary overlay.

module ; m: The CYBIL-CC compilation unit to which line or offset and proc apply. If it is not specified, the default module is used.

Examples: CB b=1

Clears breakpoint number 1.

CB line=17 offset=4

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.1.7 CLEAR_BREAKPOINT : CB

Clears breakpoint at 4 words past the word containing the first instruction generated by the statement at line 17.

4.1.8 CLEAR_TRAP : CT

This command clears specified traps or all traps in the program or part of the program. All types of traps may be cleared or one type may be specified. If no parameters are specified, a warning message will be issued. If the user accepts the warning, all traps will be cleared.

```
clear_trap : ct    [type=<trap_type>][scope_expr]
                  [t=<integer>[<..><integer>] ;
                  [<integer>[<..><integer>]
                  [<sep><integer>[<..><integer>]]...)]
                  [overlay=(<integer>,<integer>)]
                  [module=<name>] [proc=<name>]
                  [.<name>]...]
```

type: Specifies type of trap to be cleared. Trap_type must be one of the following.

```
OVERLAY : OVL
END : E
ABORT : A
INTERRUPT : INT
INSTRUCTION : I
LINE :
RJ
XJ
JUMP : J
READ : R
WRITE : W
```

If no type is specified, all types are cleared.

NOTE: If trap_type is OVERLAY or LINE, the keyword TYPE must be specified, since OVERLAY, OVL and LINE are also used as keywords in this command.

scope_expr: This specifies the location or range of locations to clear traps from. It must be one of the following:

```
line : l=<integer>[<..><integer>]
      Where integer is a CYBIL-CC line number generated by
```

COMPANY PRIVATE

4.0 CCDBG COMMANDS

4.1.8 CLEAR_TRAP : CT

the compiler.

location : loc=<integer>[<..><integer>]
Where integer is an absolute address.

offset : o=<integer>[<..><integer>]
Where integer is an address relative to a module.

variable : var=<name>
Where name is the identifier of the variable. This form is used for READ and WRITE type traps.

If the order dependent format is used and no keyword is given, line will be assumed.

t: The trap number assigned when the trap was established.

overlay : ovl: This parameter may be necessary to specify which overlay the address range applies to. Default is to use the loaded overlays in the following order: 1) overlay currently executing, 2) main overlay, 3) primary overlay, 4) secondary overlay.
NOTE: Overlay numbers default to decimal on input.

module : m: The CYBIL-CC compilation unit to which the line, offset, or proc parameter applies. If it is not specified, the default module is used.

proc : p: The CYBIL-CC procedure name to which variable applies. If it is not specified, the default proc is used. Nested procs may be referenced in the format proc1.proc2.proc3, where proc3 is the desired procedure.

Examples: CT type=overlay

Clears all traps of type overlay in the program.

CT t=instruction loc=1653(8)..1700(8)

Clears any traps of type INSTRUCTION in the address range 1653 to 1700.

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.1.9 SET_INTERPRET ; SI

4.1.9 SET_INTERPRET ; SI

Allows the CCDBG user to explicitly control the use of interpret mode. Interpret mode is turned on by the SET_INTERPRET command.

Interpret mode is also turned on when any trap of types RJ, XJ, JUMP, WRITE, READ, or INSTRUCTION are established.

If a CLEAR_INTERPRET command is subsequently issued, the traps are made inoperative until a SI command is issued, when they again become operative.

4.1.10 CLEAR_INTERPRET ; CI

Clears interpret mode.

4.2 DISPLAYING_AND_CHANGING_PROGRAM_VARIABLES

4.2.1 DISPLAY_VARIABLE ; DV

This command displays the values of CYBIL-CC identifiers in a format corresponding to their program defined type, or in a memory format specified by the user. Fields of records may be referenced using the same notation used in CYBIL-CC (including pointer notation). Array elements may be referenced using a subscript only if the subscript is a constant of the same type as the array index. Variables are not allowed as subscripts.

```
display_variable : dv    [var=<name>]
                        [format=oct ; dec ; hex][module=<name>]
                        [proc=<name>] [.<name>]...
```

variable : var: Name of the program variable to display. It will be displayed in a format corresponding to its program defined type. These types and corresponding formats are:

```
integer - signed decimal integer
character - ascii character enclosed in single
           quotes.
```

COMPANY PRIVATE

4.0 CCDBG COMMANDS

4.2.1 DISPLAY_VARIABLE ; DV

```

ordinal - name composed of ascii characters
boolean - true or false
pointer - octal digits
set - each element is displayed as listed above,
      depending on its defined type.
string - ascii characters enclosed in single
        quotes
sequence - octal digits
array - each item is displayed as above,
        depending on its defined type.
record - each field is displayed as above
        depending on its defined type.

```

If no variable name is specified, a list of variables will be displayed. What variables are displayed depends upon which other parameters are specified. If var is not specified, and proc is, then all variables local to the specified proc are displayed. If only module is specified then only variables at the module level are displayed. If no parameters are specified, local symbols for the current default are displayed. This may be either procedure level or module level depending on the current default value.

format : f: Specifies the format in which to display the variable. One of these three:

```

oct: Octal digits followed by (8)
dec: Signed decimal integer
hex: Hexadecimal digits followed by (16)

```

If this parameter is specified, it overrides the program defined type format.

module : m: The CYBIL-CC compilation unit to which var applies. If it is not specified, the default module is used.

proc : p: The CYBIL-CC procedure name to which var applies. If it is not specified, the default proc is used. Nested procedures may be referenced in the format proc1.proc2.proc3, where proc3 is the desired procedure.

Example: DV var=i proc=calculate_subscript

Displays variable I in format corresponding to its program defined type. Variable I is in proc calculate_subscript.

4.0 CCDBG COMMANDS

4.2.2 CHANGE_VARIABLE : CV

4.2.2 CHANGE_VARIABLE : CV

This command changes the memory locations at the address of the CYBIL-CC identifier. Subscripted references and field references may be made in the same way as in DISPLAY_VARIABLE.

```
change_variable : cv          var=<name> value=<expr>
                             [module=<name>] [proc=<name>]
                             [.<name>]...
```

variable : var: Name of the user variable to change.

value : v: The decimal, octal or ascii value to be stored at the specified variable. The format of the value must match the type of the variable (see list under Display_Variable). For boolean variables, only the values 'true' and 'false' are valid. For a set variable, the value specifies an element of the set that is to be added to or deleted from the set.

module : m: The CYBIL-CC compilation unit to which var applies. If it is not specified, the default module is used.

proc : p: The CYBIL-CC procedure name to which var applies. If it is not specified, the default proc is used. Nested procedures may be referenced in the format proc1.proc2.proc3, where proc3 is the desired procedure.

Example: CV var=x_string value='zero'

Changes variable x_string to the string zero. x_string is in the default module and proc.

4.3 DISPLAYING AND CHANGING MEMORY AND REGISTERS

The DISPLAY_MEMORY, FORWARD and BACKWARD commands maintain a single default memory address and a default format. After being set by a DM command, these values are used by FORWARD and BACKWARD. This allows other commands to be used between any memory display commands without losing the position or format of the memory being displayed. The memory address and format are updated by each FORWARD, BACKWARD or DM command. Specifying a format type on a FORWARD or BACKWARD command means that format will be the default until a new format is specified.

4.0 CCDBG COMMANDS

4.3.1 DISPLAY_MEMORY ; DM

4.3.1 DISPLAY_MEMORY ; DM

This command displays the contents of a specified number of words beginning at a specified address.

```
display_memory : dm      <address_expr>
                        [format=oct ; dec ; adr ; hex]
                        [numlocs=<integer>]
                        [offset=<integer>][module=<name>]
                        [indirect]
```

address_expr: The address specified by this parameter is known as the base address. Any one of the following forms may be used.

location : loc=<integer>
Where integer is an absolute address.

line : l=<integer>
Where integer is a CYBIL-CC line number generated by the compiler.

ecs = <integer>
Where integer is an ECS address.

entrypoint : e=<name>
Where name is an entry point identifier.

If the order dependent format is used and no keyword is given, location will be assumed.

format : f: Format of the memory display. It must be one of the following.

oct: Octal digits followed by (8)
dec: Signed decimal integer
adr: Lower 18 bits are displayed as an address
hex: Hexadecimal digits followed by (16)

Default format is oct.

numlocs : n: This specifies the number of memory locations to be referenced. Default is one.

offset : o: A displacement which is added to the base address to form the effective memory address. Default is zero.

4.0 CCDBG COMMANDS

4.3.1 DISPLAY_MEMORY ; DM

module : m: CYBIL-CC compilation unit to which line or offset applies. If it is not specified, the default module is used.

indirect : i: This specifies that the display is to be at the memory location addressed by the contents of the base address.

Example: DM e=cdta,offset=4,f=oct,n=10

Displays ten words of memory in octal digits beginning four words after entrypoint cdta.

4.3.2 FORWARD ; FW

This command continues the memory display forward from the last DM, FORWARD or BACKWARD command. Displays memory starting with the word following the last word displayed by the previous command.

forward : fw numlocs=<integer>
 [format=oct ; dec ; adr ; hex]

numlocs : n: Specifies the number of memory locations to be referenced.

format : f: Format of the memory display. It must be one of the following:

oct: Octal digits followed by (8)
dec: Signed decimal integer
adr: Displays lower 18 bits as an address
hex: Hexadecimal digits followed by (16)

Default value for format is to continue in the same format as the previous display_memory, forward, or backward command.

Example: FORWARD 6

Displays the next 6 words of memory in the same format as the previous DISPLAY_MEMORY, FORWARD or BACKWARD.

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.3.3 BACKWARD : BW

4.3.3 BACKWARD : BW

This command continues the memory display with the section of memory preceding the current display memory location. It displays memory from the word preceding the first word displayed by the previous command back n locations.

```
backward : bw          numlocs=<integer>
                   [format=oct : dec : adr : hex]
```

numlocs : n: Specifies the number of memory locations to be referenced.

format : f: Format of the memory display. It must be one of the following:

```
oct: Octal digits followed by (8)
dec: Signed decimal integer
adr: Displays lower 18 bits as an address
hex: Hexadecimal digits followed by (16)
```

Default value for format is to continue in the same format as the previous display_memory, forward, or backward command.

Example: BACKWARD 4,oct

Displays the four preceding words of memory in octal format.

4.3.4 CHANGE_MEMORY : CM

This command changes the contents of the specified memory locations.

```
change_memory : cm    <address_expr>
                   [module=<name>] [offset=<integer>]
                   value=<expr> [numlocs=<integer>]
                   [indirect]
```

address_expr: The address specified by this parameter is known as the base address. Any one of the following forms may be used.

```
location : loc=<integer>
```

4.0 CCDBG COMMANDS

4.3.4 CHANGE_MEMORY : CM

Where integer is an absolute address.

line : l=<integer>

Where integer is a CYBIL-CC line number generated by the compiler.

entrypoint : e=<name>

Where name is an entry point identifier.

ecs = <integer>

Where integer is an ECS address.

If the order dependent format is used and no keyword is given, location is assumed.

module : m: The CYBIL-CC compilation unit to which line applies.

offset : o: A displacement which is added to the base address to form the effective memory address. Default is zero.

value : v: The octal, decimal, hexadecimal or ascii value which is to be stored at the memory address specified.

numlocs : n: This specifies the number of memory locations to be changed. Default is one.

indirect : i: This specifies that the location to be changed is the memory location addressed by the contents of the base address.

Example: CM loc=6270(8) v=1493

Stores 1493 (note default base is decimal) in octal address 6270.

4.3.5 DISPLAY_REGISTERS : DR

This command displays the contents of the machine registers.

```
display_registers : dr      [p ; fl ; [a ; b ; x=[<integer>]]]
                          [format=oct ; dec ; adr ; hex]
                          [indirect]
```

p ; fl ; a ; b ; x: This specifies which type of register is to be displayed. The number for a, b, or x, indicates which register number to display. If no register

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.3.5 DISPLAY_REGISTERS : DR

number is specified, all of the specified type will be displayed. If the parameter is omitted, all 24 A, B, and X registers will be displayed.

format : f: Format of the register display. It must be one of the following:

oct: Octal digits followed by (8)
 dec: Signed decimal integer
 adr: Displays the lower 18 bits as an address
 hex: Hexadecimal digits followed by (16)

Default value of format is octal for a, b, and x registers, and address for p and fl registers.

indirect : i: This specifies, if quoted, that the display is to be at the memory location addressed by the contents of the register.

Example: DR b

Displays all b registers in octal format.

4.3.6 CHANGE_REGISTERS : CR

This command changes the contents of the machine registers.

change_registers : cr a : b : x=<integer>
 value=<expr> [indirect]

a : b : x: This specifies which type of register is to be changed. The integer specifies the register number to be changed.

value : v: This is the octal, decimal, hexadecimal or ascii value to be stored in the register.

indirect : i: This specifies, if quoted, that the value is to be stored at the memory location addressed by the contents of the register.

Example: CR x=6 v=1236(8)

Places the value 1236 (octal) in register x6.

4.0 CCDBG COMMANDS

4.4 DEBUGGER SCL PROCEDURES

4.4 DEBUGGER_SCL_PROCEDURES

A debugger SCL procedure is a named sequence of CCDBG commands that is established within the CCDBG environment. Any desired CCDBG command sequence can be established as a debugger SCL procedure. Procedures can be invoked either from within some other procedure or from the terminal when in interactive command input mode, by issuing a READ command referencing the name of the defined procedure.

4.4.1 SET_PROCEDURE : SP

This command establishes a debugger SCL procedure.

```
set_procedure : sp procedure=<name> [[<collect><eol!;>]
                    [<command_statement><eol!;>]...]
                    <collect_end>
```

procedure : pr: The identifier assigned to this sequence of CCDBG commands.

collect : c: Activates collect mode. This parameter is not necessary to establish a procedure, but is included in the syntax for compatibility with SET_TRAP and SET_BREAKPOINT commands.

command_statements: CCDBG commands which make up the body of the procedure. They are checked for syntax but not executed at this time.

collect_end : ce: Terminates collect mode. This parameter is required.

```
Example: SET_PROCEDURE procedure=newid; DR b=5; DV var=x;dv y;
          PA; ce
```

Creates a procedure named newid which contains several commands.

4.4.2 DISPLAY_PROCEDURE : DP

This command displays the commands that make up the specified procedure. If no procedure identifier is specified, the names of

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.4.2 DISPLAY_PROCEDURE : DP

all existing debugger SCL procedures are displayed.

```
display_procedure : dp [procedure=<name> ;
                       (<name>[<sep><name>]...)]
```

procedure : pr: The identifier of the procedure to be displayed, as specified in the SET_PROCEDURE command.

Example: DP procedure=newid

Displays the debugger SCL procedure named newid.

4.4.3 SAVE_PROCEDURE : SAVEP

This command copies specified procedure definitions to a local file. If no procedures are specified, all debugger SCL procedures are copied to the file.

```
save_procedure : savep file=<name> [procedure=<name> ;
                                     (<name>[<sep><name>]...)]
```

file : f: The name of the file the procedure is to be saved on.

procedure : pr: The identifier of the group to be saved.

Example: SAVEP procedure=newid f=saveid

Saves procedure named newid on file saveid.

4.4.4 CLEAR_PROCEDURE : CP

This command removes specified debugger SCL procedure definitions. If no procedure is specified, a warning message is issued, and if the user accepts the warning, all debugger SCL procedures are cleared.

```
clear_procedure : cp [procedure=<name> ;
                     (<name>[<sep><name>]...)]
```

procedure : pr: The identifier of the procedure to clear, as specified on the SET_PROCEDURE command.

Example: CP procedure=newid

4.0 CCDBG COMMANDS

4.4.4 CLEAR_PROCEDURE : CP

Clears debugger SCL procedure newid.

4.5 DEBUGGER_SCL_PROCEDURE_COMMANDS

A procedure is a set of CCDBG commands established as a debugger SCL procedure. Once initiated, the commands in a procedure execute automatically.

Any CCDBG command may be used in a procedure. Most of the following commands, however, are particularly designed for use in a procedure.

4.5.1 PAUSE : PA

This command temporarily suspends the automatic execution of the current CCDBG procedure and enters interactive mode, allowing CCDBG commands to be entered from the terminal.

On the occurrence of the first PAUSE command in a procedure body of a trap or breakpoint, the appropriate trap or breakpoint report is issued prior to entering interactive mode.

```
pause : pa           [text='[<ascii>]...']
```

text : t: Message text to be issued when pause is executed. The text may be any string of characters. It is delimited by a pair of apostrophes ('). Apostrophes within the text are indicated by two consecutive occurrences of the character.

Example: PAUSE text='x3 is new value of I'

4.5.2 MESSAGE : ME

This command issues a designated message to the users terminal during a procedure execution.

```
message : me        [text='[<ascii>]...']
```

text : t: Message text to be issued when the command is executed. The text may be any string of characters. It is delimited by a pair of apostrophes (').

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.5.2 MESSAGE ; ME

Apostrophes within the text are indicated by two consecutive occurrences of the character.

Example: MESSAGE 'Variable XYZ has been modified'

4.5.3 GO

This command causes an exit from the current debugger SCL procedure or interactive mode and a resumption of suspended processing. The current debugger SCL procedure may have been invoked by a read, in which case the processing will be resumed with the statement following the read.

If the GO command is issued from a breakpoint or trap body, program execution is resumed. In this last case, GO is identical to the Execute command. Resumption of program execution can be specified via the optional address parameter.

go [address_expr][offset=<integer>]

address_expr: Address at which execution is to resume. It is not possible to resume execution in a different module or procedure. Address_expr may be any of the following:

line : l=<integer>

Where integer is the CYBIL-CC line number generated by the compiler.

location : loc=<integer>

Where integer is an absolute address.

entrypoint : e = <name>

Where name is an entry point identifier.

If no keyword is given for address_expr, line will be assumed.

offset : o: A displacement added to the base address specified by address_expr.

Example: GO line=42 offset=3

4.0 CCDBG COMMANDS

4.5.4 SKIPIF : S

4.5.4 SKIPIF : S

This command conditionally skips the following command when the specified relation is satisfied.

skipif : s <value_1><relation><value_2>

value_1 : v1: An integer expression. It may include CCDBG variables, but not program variables.

relation: One of the following:

- = Equal
- <> Not equal
- < Less than
- <= Less than or equal
- >= Greater than or equal
- > Greater than

value_2 : v2: An integer expression. It may include CCDBG variables, but not program variables.

Example: SKIPIF #V1 < 20

Skips next command if the value of CCDBG variable #V1 is less than 20.

4.5.5 LABEL : LA

This command defines an identifier for the location in the procedure where it occurs.

Labels are local to the procedure in which they occur. No check is made for duplicate labels.

label : la name=<name>

name : n: Identifier to be used for this location.

Example: LABEL n=b1013

Establishes label B10L3 at the current location in the procedure.

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.5.6 GOTO

4.5.6 GOTO

This command transfers control to the specified label within a procedure. Control can be transferred either forwards or backwards. Search is forward, end around, until the first occurrence of the label is found.

```
goto                label=<name>
```

label : la: Identifier of location to which control is transferred.

Example: GOTO b1013

Transfers control to the location of label B10L3

4.5.7 READ : R

This command is used in three different ways:

- 1) To process CCDBG commands stored or modified by some facility not provided by CCDBG itself (e.g., the editor).
- 2) To reconstitute breakpoint, trap, and debugger SCL procedure definitions previously SAVED on a file.
- 3) To invoke a debugger SCL procedure.

```
read : r            file : procedure = <name>
```

file : f: Name of file containing commands to be executed.

procedure : pr: Name of procedure to be executed.

Examples: READ f=bpfile

Re-establishes breakpoint definitions saved on BPFILe.

READ procedure=newid

Initiates processing of procedure NEWID.

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.6 ADDITIONAL INTERACTIVE MODE COMMANDS

4.6 ADDITIONAL INTERACTIVE MODE COMMANDS

4.6.1 EXECUTE : EX

This command starts or resumes program execution. Program execution is initiated at the next instruction, or at the user specified address. This command differs from the GO command when there are higher levels of CCDBG commands. GO transfers control to the next higher level of CCDBG commands, EXECUTE bypasses all CCDBG commands to resume program execution.

execute : ex [address_expr] [offset=<integer>]

address_expr: An address at which execution is to resume. It is not possible to resume execution in another module or procedure. It may be any of the following:

line : l=<integer>

Where integer is the CYBIL-CC line number generated by the compiler.

location : loc=<integer>

Where integer is an absolute address.

entrypoint : e = <name>

Where name is an entry point identifier.

If no keyword is given for address_expr, line will be assumed.

offset : o: A displacement added to the base address.

Example: EX 146

Resume execution at line 146 relative to the current executing module.

4.6.2 TRACEBACK : TB

This command produces a list of CYBIL-CC procedure names, beginning with the currently executing CYBIL-CC procedure and moving backward through successive levels.

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.6.2 TRACEBACK : TB

```
traceback : tb          [entrypoint=<name>]
```

entrypoint : e: If a non-CYBIL-CC routine has multiple entrypoints, this parameter specifies at which entrypoint to start the traceback.

Example: TB

Lists all calls beginning with the most recent and going backward as far as possible.

4.6.3 MOVE : M

This command moves values from one address range to some other address range or location. The action taken depends on whether either the source or destination parameter is a range specification.

If the source or destination is a range specification, the numlocs parameter is ignored even if present and enough words are moved from the source to fill the destination range. If the source range is smaller than the destination range, the words are moved repeatedly until the destination range is filled. If the source range is larger, only as many words as are needed are moved.

If neither the source nor destination is a range specification, the numlocs parameter determines the number of words to move.

```
move : m                source_expr destination_expr
                        [numlocs=<integer>]
```

source_expr: Address or address range expression from which to move values. Must have one of the following forms:

```
source_line : sl=<integer>[<..><integer>]
  Where integer is a CYBIL-CC line number generated by
  the compiler.
```

```
source_offset : so=<integer>[<..><integer>]
  Where integer is an address relative to a module.
```

```
source_location : sloc=<integer>[<..><integer>]
  Where integer is an absolute address.
```

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.6.3 MOVE : M

If a keyword is not specified, source_location is assumed.

destination_expr: Address or address range expression to which values are to be moved. Must have one of the following forms:

destination_line ; dl=<integer>[<..><integer>]

Where integer is a CYBIL-CC line number generated by the compiler.

destination_offset ; do=<integer>[<..><integer>]

Where integer is an address relative to a module.

destination_location ; dloc=<integer>[<..><integer>]

Where integer is an absolute address.

If a keyword is not specified, destination_location is assumed.

numlocs : n: Number of words to be moved if neither the source nor the destination is an address range. Default is one.

Example: MOVE sloc=112(8)..120(8) dloc=142(8)..150(8)

Moves 7 words beginning at 112(octal) to a 7 word field beginning at location 142(octal), all addresses are absolute.

4.6.4 HELP : H

This command can provide immediate assistance to a terminal user about CCDBG features. It acts as a selectively accessed on-line information summary.

help : h [_{subject} : <command_name>]

<subject>: One of the permissible subject categories.

<command_name>: A CCDBG command identifier or abbreviation.

If no parameter is specified, the subject index is listed. This displays all the permissible subject categories that may be entered as parameters to help.

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.6.4 HELP : H

Example: HELP cmds (lists all CCDBG command identifiers)

4.6.5 SAVE_ALL : SAVEA

This command saves all trap, breakpoint and debugger SCL procedure definitions.

save_all : savea file=<name>

file : f: The name of a local file on which to save the environment.

Example: SAVEA,bigfile

4.6.6 CHECKPOINT : CK

This command allows a user to save the CCDBG information necessary to return to the debug session at a later time.

CHECKPOINT saves the current CCDBG environment (breakpoint, trap, and debugger SCL procedure definitions), and current status (interpret and veto mode settings), output options, default module, default proc, CCDBG variables and tables, and the user program image, on a file.

Restoration of CCDBG to its status, environment, and user program image is done by issuing the system command DEBUG(RESUME, filename).

checkpoint : ck file=<name>

file : f: The name of a local file on which the CCDBG environment will be saved.

Example: CHECKPOINT f=sf

4.6.7 QUIT

This command terminates a CCDBG session.

quit [normal : abort]

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.6.7 QUIT

normal: Terminate normally. This is the default if no parameter is specified.

abort: Causes an abort type of termination to occur. This allows abort processing if CCDBG is being used in a batch job, or from a procedure file.

4.7 CCDBG_ENVIRONMENT_COMMANDS

4.7.1 SET_VETO : SVE

This command provides a method of CCDBG sequence operation that combines the automatic and interactive modes. When veto mode is on, each command in a CCDBG sequence is displayed before it is executed. The user is given temporary control at this point. The user may allow the command to be executed, skip it, or replace it with one or more new commands.

set_veto : sve

4.7.2 CLEAR_VETO : CVE

This command terminates veto mode.

4.7.3 SET_OUTPUT : SO

This command allows the user to control the kinds of CCDBG output that are written to the standard output file.

set_output : so lo=<options>

lo: List options, any of the following:

- e Error messages
- w Warning messages
- d Debug output produced by command execution
- i Informative messages
- r Read command sequence (group or file) when read
- b Body sequence when it occurs (trap or breakpoint)
- t Terminal or standard input file echo

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.7.3 SET_OUTPUT : SO

The default options when the list is omitted are e,w,d,i. If e is omitted, an auxiliary file must be defined with the e option specified.

Example: SO lo=(e,w,i)

4.7.4 SET_AUXILIARY : SAUX

This command allows a user to define an optional auxiliary output file and control the kinds of output that are written to it.

set_auxiliary : saux file=<name> lo=<options>

file : f: Name of the file to be the auxiliary output file.

lo: List options. These define what is to go on the file. Any of the following:

- e Error messages
- w Warning messages
- d Debug output produced by command execution
- i Informative messages
- r Read command sequence (group or file) when read
- b Body sequence when it occurs (trap or breakpoint)
- t Terminal or standard input file echo

Example: SAUX f=pxidaux lo=(d,r)

4.7.5 CLEAR_AUXILIARY : CAUX

Closes the current auxiliary file and clears all the auxiliary options.

4.7.6 DISPLAY_MAP : DMAP

This command displays load map information relating to modules entry points, and overlays.

display_map : dmap [module=<name>]
[overlay=(<integer>,<integer>)]

COMPANY PRIVATE

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.7.6 DISPLAY_MAP : DMAP

module : m: A CYBIL-CC compilation unit. If a module is specified, its origin, length and all entry points contained in it are displayed.

overlay : ovl: If an overlay is specified, the names of all modules in the overlay are listed.

If no parameter is specified in a non_overlay environment, the names of all modules are listed. In an overlay environment, designations of overlays are listed.

Example: DMAP m=adapt

Displays loader information about module adapt.

4.7.7 DISPLAY_DEFAULTS : DD

This command displays the current default value for module and proc.

4.7.8 DISPLAY_STATUS : DS

This command displays information to help the user determine the current state of the debug session.

display_status : ds

The following information is displayed.

The location of the trap or breakpoint currently honored, or the location of user program abort or interrupt to be displayed.

The overlay numbers currently in core and their addresses.

The current terminal output options.

The name of the current auxiliary output file and the current auxiliary output options.

The current state of veto mode (on or off).

The current state of interpret mode (on or off).

COMPANY PRIVATE

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

4.0 CCDBG COMMANDS

4.7.8 DISPLAY_STATUS : DS

The number of breakpoints, traps, and groups currently defined.

4.7.9 CHANGE_DEFAULTS : CD

This command allows the user to change the default values of module and proc to new identifiers.

```
change_defaults : cd    [module=<name>]
                       [proc=<name>] [<.name>]...]
                       [overlay=(<integer>,<integer>)]
```

module : m: This parameter changes the default module from the specified module.

proc : p: This parameter changes the default proc to the specified proc.

overlay : ovl: Overlay which contains the new default module or proc.

Example: CD m=zclmtay

Changes default module to the module with alias zclmtay.

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

5.0 MESSAGES ++++++

5.0 MESSAGES_+++++

5.1 DIAGNOSTIC_MESSAGES

Diagnostic messages issued by CCDBG are listed below. These messages are issues in one of the following forms:

*ERROR - message text
?

*CMD - (command text) *ERROR - message text
?

When in collect mode, some errors, such as those involving invalid syntax, are detected and reported prior to being collected, thus allowing them to be corrected at that time. Other errors are not detected until execution of the command is attempted.

Message	Significance and Action
ADDRESS IN ECS/LCM	GO or EXECUTE has been supplied with a location parameter that is an ECS or LCM address. If supplied, the address must be one in central memory. ACTION: Correct and reenter.
ADDRESS IN UNLOADED OVERLAY	A specified symbolic address implies one contained in an overlay not currently loaded. ACTION: Confine symbolic addresses to those in currently loaded overlays.
ADDRESS OUTSIDE USER AREA	An address reference is to a location in DBUG, beyond the first 100B (approximately) locations, or beyond the user field length. These locations are inaccessible to the user. ACTION: Reenter with an allowable address.

COMPANY PRIVATE

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

5.0 MESSAGES ++++++

5.1 DIAGNOSTIC MESSAGES

```

BAD INDEX      Internal error.
TYPE i
ACTION:  Submit PSR.

BAD
ORDINAL      A variable of type ordinal contains an
VALUE i      undefined value.
ACTION:  None.

BAD SYMBOL    Internal error.
TYPE i
ACTION:  Submit PSR

CANNOT
CHANGE      The CHANGE_VARIABLE command can only change the
VALUE OF    value of simple variables, such as fields of
ENTIRE      records or elements of arrays.
RECORD OR
ARRAY
ACTION:  Reenter with subscript or field notation.

CANNOT
EVALUATE    Internal error.
VARIABLE
TYPE i
ACTION:  Submit PSR.

DEBUG
INTERNAL    An error in Debug is preventing further pro-
ERROR       cessing. Debug must be aborted. The program
            being debugged could have damaged a portion of DEBUG.

ACTION:  Try a new Debug session with all execution
            performed in interpret mode, which protects
            DEBUG. code.

ERROR
MESSAGES    Completing the current SET OUTPUT, SET
MAY NOT BE  AUXILIARY, CLEAR OUTPUT, or CLEAR
SUPPRESSED  AUXILIARY command results in no file being
            designated to receive error messages.

ACTION:  Assign file options consistent with this
            restriction.

```

COMPANY PRIVATE

5.0 MESSAGES ++++++

5.1 DIAGNOSTIC MESSAGES

EXECUTION
ADDRESS
OUTSIDE
CURRENT
PROCEDURE

The address specified is outside the current procedure, and execution of the command would invalidate the stack.

ACTION: Correct and reenter.

INVALID
DESCRIPTOR
WORD FOR
ADAPTABLE
ARRAY

An illegal descriptor word was found for an adaptable array.

ACTION: Use a write trap on the pointer to find the point where the descriptor is destroyed.

INVALID
FIRST,
LAST OR
STEP VALUE

A breakpoint has been supplied with an invalid frequency parameter.

ACTION: Check that all such parameters are positive. Check that LAST is not less than FIRST. Reenter with corrected values. First is limited to 262,143. Step is limited to 4095.

INVALID
ORDINAL
VALUE x

The value input is not a valid value for this variable.

ACTION: Correct value and reenter.

INVALID
PARAMETER
xxxx

The supplied HELP parameter is invalid.

ACTION: Enter HELP for a list of valid parameters. Reenter the HELP command with a valid parameter.

INVALID
VALUE
SPECIFIED
FOR
VARIABLE
OF TYPE x

Value type for a variable or subscript does not match the value input.

ACTION: Verify variable type and reenter in proper format.

5.0 MESSAGES ++++++

5.1 DIAGNOSTIC MESSAGES

INTERNAL Internal error.
ERROR -
STACK
UNDERFLOW

ACTION: Submit PSR.

INTERNAL Too many levels of record and array declarations
ERROR - exist in the variable to allow CCDBG to
SYMBOL TOO display it.
COMPLEX

ACTION: Display portions of the record.

INVALID An overlay trap qualifier other than an overlay
QUALIFIER designation was specified in a SET TRAP,
FOR LIST TRAP, CLEAR TRAP, or SAVE TRAP
OVERLAY command.
TRAPS

ACTION: Correct and reenter.

INVALID A SET_TRAP command has an invalid trap type
TRAP TYPE parameter value.
xxxx

ACTION: Reenter with a valid trap type.

LINE An attempt was made to reference a line number
NUMBERS in an explicitly named or default module
NOT other than a program
AVAILABLE compiled with the DEBUG option.

ACTION: Check the home program.

NO DEFAULT A command has been entered that requires a module
MODULE specification, no module was specified, and no
 default module is established.

ACTION: Define a default module or reenter command
 with module specified.

NO ENTRY A reference has been made to an entry point
POINT xxx name xxx which does not exist; or if an overlay
 qualifier has been supplied, it is not in that
 overlay.

ACTION: Check spelling or overlay qualifier; correct
 and reenter.

5.0 MESSAGES ++++++

5.1 DIAGNOSTIC MESSAGES

NO FILE OR PROCEDURE READ
xxxx

The file or debugger SCL procedure named in a READ parameter does not exist.

ACTION: Check spelling; check to see if the file is logically connected to the job.

NO LABEL
xxxx

A GOTD command has referenced a label which does not exist in the current Debug command sequence.

ACTION: Correct the Debug sequence accordingly.

NO MODULE
xxxx

A reference has been made to a module xxxx which does not exist; or if an overlay qualifier is supplied, the module is not in that overlay. Remember that module is a 1 - 7 character name.

ACTION: Correct and reenter.

NO OVERLAYS

An overlay reference has been made in a nonoverlay environment. This error is detected at collect time if it occurs in a debugger SCL procedure, or if a specific overlay is referenced.

ACTION: Confine Debug commands and address qualifiers to those acceptable in a nonoverlay environment.

NO OVERLAY (xxxx)

The specified overlay does not exist. DISPLAY_MAP indicates all existing overlays.

ACTION: Reenter with the corrected overlay designation.

NO VARIABLE
xxxx

An attempt was made to reference a CYBIL-CC variable xxxx. No such variable exists in the referenced or default module or the variable was not used.

ACTION: Check spelling and the home program.

NON - EXISTENT FIELD xxxx

Field is not defined in current record.

ACTION: Correct and reenter.

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

5.0 MESSAGES ++++++

5.1 DIAGNOSTIC MESSAGES

```

NOT IN          A COLLECT_END command has been entered when
COLLECT        not in collect mode.
MODE -
COLLECT_END
IGNORED

ACTION:  None.

OPTION         An invalid option code was specified in the
CODE MUST     option list of a SET_OUTPUT or SET_AUXILIARY
BE B, D,      command.
E, I, R,
T, OR W

ACTION:  Reenter with all valid option codes.

POINTER       A debug command has attempted to evaluate
VARIABLE      a pointer variable that has not been set.
NOT
INITIALIZED

ACTION:  None.

PROC xxxx     TRACEBACK cannot proceed because the
NOT CALLED    specified procedure has never been called.

ACTION:  Enter another command.

PROC xxxx     Automatic variables are not available
NOT IN        because the procedure is not active.
STACK

ACTION:  None.

PROGRAM       An attempt has been made with either GO or
HAS           EXECUTE to continue program execution from
COMPLETED    the point where program termination has been
              reached.

ACTION:  Reenter specifying some other execution
         addresses, or issue QUIT.

RECURSIVE    The debugger SCL procedure or file named in
READ OF      the current READ parameter is a nested
xxxx        procedure or the current sequence.

ACTION:  Redesign sequence logic to avoid this
         situation.

```

5.0 MESSAGES ++++++

5.1 DIAGNOSTIC MESSAGES

RELATIVE ADDRESS OUTSIDE BLOCK

A module offset is equal to or greater than its length. DISPLAY_MAP gives the program length.

ACTION: Check for the missing octal suffix (8) on the offset value if octal was intended. Correct and reenter.

RESPONSE QUALIFIER MUST BE LINE OR SEQ

In response to an error, warning, veto, or interrupt of a Debug sequence, a response keyword has been followed by text beginning with other than LINE, SEQ or ; (semi-colon).

ACTION: Enter any desired valid response.

SPECIFIED VALUE NOT IN SET

The given value is not in the specified set.

ACTION: Correct and reenter.

STACK POINTER OUTSIDE USER AREA

An invalid stack pointer has been found.

ACTION: None.

SUBSCRIPT REQUIRED AFTER ARRAY xxxx

A field of a record was specified after an array field in a variable entry without specifying a subscript.

ACTION: Correct and reenter.

SUBSCRIPT VALUE i OUT OF RANGE

A subscript value has been specified that is outside the range for the array.

ACTION: Correct and reenter.

SYMBOL NUMBER i NOT FOUND

Internal error.

ACTION: Submit PSR.

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

5.0 MESSAGES ++++++

5.1 DIAGNOSTIC MESSAGES

SYMBOL Internal error.
 WITH
 RELOCATION
 TYPE i
 NOT
 ACCESSABLE

ACTION: Submit PSR.

SYNTAX
 ERROR - Command with keywords added is too long
 COMMAND for buffer.
 TOO LONG

ACTION: Break command up if possible so fewer parameters are needed.

SYNTAX
 ERROR - A DISPLAY_MEMORY command has not been
 DISPLAY_ entered.
 MEMORY
 MUST
 PRECEDE
 FORWARD OR
 BACKWARD

ACTION: Enter a DISPLAY_MEMORY before doing FORWARD or BACKWARD.

SYNTAX
 ERROR - The parameter specified in the message has
 DOUBLY been given more than one value.
 DEFINED
 PARAMETER

ACTION: Correct syntax and reenter.

SYNTAX
 ERROR - An incorrect parameter type was found.
 EXPECTING The message tells what was expected and what
 was found for the specified parameter.

ACTION: Correct syntax and reenter.

SYNTAX
 ERROR - Either the address parameters are not
 IMPROPERLY allowed together (ie, proc and location)
 QUALIFIED or another parameter is needed with
 ADDRESS them (ie, module must be supplied
 when proc and overlay are).

ACTION: Check syntax, correct and reenter.

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

5.0 MESSAGES ++++++

5.1 DIAGNOSTIC MESSAGES

SYNTAX The syntax described by the message is illegal.
 ERROR -
 INVALID

ACTION: Correct syntax and reenter.

SYNTAX An arithmetic operation has caused the integer
 ERROR - to become too large.
 INTEGER
 OVERFLOW

ACTION: Check numeric parameters. If problem persists, submit PSR.

SYNTAX The value entered is too large.
 ERROR -
 INTEGER
 TOO LARGE

ACTION: Correct and reenter.

SYNTAX The specified parameter must be entered for
 ERROR - this command.
 REQUIRED
 PARAMETER
 MISSING

ACTION: Reenter command specifying the required parameter.

SYNTAX String exceeds maximum allowable length.
 ERROR -
 STRING
 OVERFLOW

ACTION: Correct and reenter.

SYNTAX CCDBG internal tables are filled.
 ERROR -
 TABLE
 OVERFLOW

ACTION: Submit PSR.

SYNTAX Not enough of the specified item were input.
 ERROR -
 TOO FEW

ACTION: Correct and reenter.

SYNTAX Too many of the specified item were input.
 ERROR -
 TOO MANY

ACTION: Correct and reenter.

COMPANY PRIVATE

CDC - ADVANCED SYSTEMS DEVELOPMENT

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

5.0 MESSAGES ++++++

5.1 DIAGNOSTIC MESSAGES

```

SYNTAX      Parenthesis or brackets are not matched.
ERROR -
UNBALANCED

ACTION:    Correct and reenter.

SYNTAX      The specified keyword is not valid for
ERROR -    this command.
UNKNOWN
KEYWORD

ACTION:    Check syntax, correct and reenter.

SYNTAX      Parameter value is not within
ERROR -    allowable range.
VALUE OUT
OF RANGE

ACTION:    Check syntax, correct and reenter.

SYNTAX      A value range was specified for a
ERROR -    parameter that cannot have a range.
VALUE
RANGE NOT
ALLOWED

ACTION:    Check syntax, correct and reenter.

TOO MANY    The number of breakpoints has reached the maxi-
BREAKPOINTS mum allowed.

ACTION:    One or more existing breakpoints must be
           cleared before any new ones can be set.

TOO MANY    The number of debugger SCL procedures has
PROCEDURES reached the maximum allowed.

ACTION:    One or more existing groups must be cleared
           before any new ones can be set.

TOO MANY    The number of nested debugger SCL procedures has
NESTED     reached the maximum allowed.  A READ or PAUSE
COMMAND    command is not allowed until the current
SEQUENCES  procedure is terminated and the previous
           procedure is resumed.

ACTION:    Enter GO to resume the previous sequence
           immediately.

```

COMPANY PRIVATE

5.0 MESSAGES ++++++

5.1 DIAGNOSTIC MESSAGES

TOO MANY
TRACE
LEVELS

The TRACEBACK output has reached its built-in feasibility limit. Program logic flow could have errors.

ACTION: Correct and reenter.

TOO MANY
TRAPS

The number of traps has reached the maximum allowed.

ACTION: Clear one or more existing traps before setting new ones.

UNKNOWN
COMMAND

The command text does not contain a syntactically recognizable command name. If this error occurs in a debugger SCL procedure, it is detected at collect time. HELP CMDS list all valid command names.

ACTION: Check spelling. Reenter a valid command.

VARIABLE
NAMES NOT
AVAILABLE

The referenced variable in an explicitly named or default module must be in a CYBIL-CC program compiled with the DEBUG option.

ACTION: Check the home program.

xxxx IS
NOT AN
ARRAY
VARIABLE

A subscript is included in a variable declaration for a variable that is not type array.

ACTION: Correct and reenter.

xxxx IS
NOT A
POINTER
VARIABLE

A pointer symbol (^) has been used in a variable declaration for a variable that is not type pointer.

ACTION: Correct and reenter.

xxxx IS
NOT A
RECORD OR
ARRAY OF
RECORDS

A field name has been specified for a variable that cannot have fields.

ACTION: Correct and reenter.

5.0 MESSAGES ++++++

5.2 WARNING MESSAGES

5.2 WARNING_MESSAGES

Warning messages issued by CCDBG are listed below. These messages have one of the following forms:

*WARN - message text
OK?

*CMD - (command text) *WARN - message text
OK?

These messages indicate by their wording the action that is taken if the user responds with a positive acknowledgement (YES, ACCEPT, or OK).

Warning messages can be suppressed by issuing a SET OUTPUT command that does not include the W parameter in its option list. The action indicated in the message automatically occurs.

09/29/80

ERS for CYBIL-CC Interactive Debugger

REV: D

5.0 MESSAGES ++++++

5.2 WARNING MESSAGES
.....

Message	Significance
ADAPTABLE STRING LENGTH WILL INCREASE FROM i TO i	This CHANGE_VARIABLE command will lengthen the adaptable string variable, and possibly overwrite other data.
ADDRESS RANGE WILL BE TRUNCATED	An address range for CHANGE, DISPLAY, or MOVE extends beyond the user field length or into DEBUG, beyond the first 100 (approximately) locations.
ALL WILL BE CLEARED	A CLEAR_TRAP, CLEAR_BREAKPOINT, or CLEAR_PROCEDURE command has been issued with no parameters.
BREAKPOINT WILL BE SET AT ENTRY POINT	This warning is issued only if the specified address is an entry point, but was not specified as such in a SET_BREAKPOINT command.
EXISTING AUXILIARY FILE WILL BE CLOSED	A SET_AUXILIARY command has been issued which specifies a file name different from that of the existing auxiliary file.
EXISTING BREAKPOINT WILL BE REDEFINED	An attempt is being made to set a breakpoint where one already exists. A positive acknowledgement causes the new definition to override the old one.
EXISTING CHECKPOINT FILE WILL BE OVERWRITTEN	A CHECKPOINT command has been issued giving an existing file name.
PROCEDURE xxx WILL BE REDEFINED	The name supplied in a SET_PROCEDURE command is that of a currently existing PROCEDURE. A positive acknowledgement causes the new definition to override the old one.
LINE STARTS IN PARCEL 3. BKPT SET IN NEXT WORD.	One 15 bit instruction for this line of the program will be executed before breakpoint is honored.
LINE n NOT EXECUTABLE - LINE m WILL BE USED	The specified line number is not executable or is nonexistent. A positive acknowledgement causes line m to be used instead.

5.0 MESSAGES ++++++

5.2 WARNING MESSAGES

```

PERMANENT CHECKPOINT : A CHECKPOINT command has been issued
FILE WILL BE         : while a permanent suspend file exists
RETURNED             : without write or modify access.
                     :
TRAP #n, type,       : A pending SET_TRAP command has a scope
qualifiers, WILL     : which overlays the scope of an existing
BE CLEARED           : trap of the same type. A positive
                     : acknowledgement CLEARS trap #n.

```

5.2.1 INFORMATIVE MESSAGES

Informative messages issued by CCDBG are listed below. These messages have the form:

message text

Informative messages indicate the following: changes in the status of CCDBG, changes in the status of commands that process a list, commands which confirm specific actions taken, when a list element cannot be processed, and when there is no action to be taken.

After the informative message is issued, CCDBG does not pause for a response, except when the message announces the start or resumption of a CCDBG session. Any remaining elements in a list are processed after reporting a list element that cannot be processed.

5.0 MESSAGES ++++++

5.2.1 INFORMATIVE MESSAGES

Message	Significance
CYBIL-CC INTERACTIVE DEBUG	After the program to be debugged has been loaded, this message is issued when CCDBG receives control. An initial set of traps and/or breakpoints should be established at this point before starting program execution.
CYBIL-CC INTERACTIVE DEBUG RESTARTED	A CCDBG session has been resumed from the point where it was checkpointed. The system command statement DEBUG (RESUME,filename) has been entered following the issue of a CHECKPOINT command.
CYBIL MODULE NEXT, STACK POINTER ASSUMED IN REG B2	Previous modules in a traceback have not been CYBIL-CC modules. Continued traceback is possible only if register B2 still contains a valid stack pointer.
DEBUG ABORTED	This message is issued in response to QUIT ABORT; it appears in the dayfile as well.
DEBUG TERMINATED	This message is issued in response to QUIT or QUIT NORMAL; it appears in the dayfile as well.
END COLLECT	Sufficient collect_end commands have been encountered to reduce the collect level to zero, thus ending collect mode. Interactive command mode is resumed; entered commands are immediately executed.
IN COLLECT MODE (,LEVEL n)	This message occurs when the user receives CCDBG control in collect mode; this was not the case when the user last had control. Any subsequent commands entered will no longer be executed immediately, but will be checked for syntax and collected into a debugger SCL procedure for future execution. Level n is included in the message if a nested collect is in effect (n is greater than one). To end collect mode, n collect_end commands are required.

5.0 MESSAGES ++++++

5.2.1 INFORMATIVE MESSAGES

```

INTERRUPT IGNORED      : CCDBG was already in interactive command
                        : mode when a terminal interrupt occurred.
                        : Since the purpose of a terminal interrupt
                        : is to place CCDBG in interactive mode,
                        : the interrupt is ignored.

INTERRUPTED            : A terminal interrupt has occurred while a
                        : debugger SCL procedure or a command which
                        : takes a list as a parameter was
                        : executing.

INTERPRET MODE         : As a result of clearing one or more
TURNED OFF             : traps, no traps remain that require
                        : interpret mode to be on. Subsequent
                        : program execution will be by direct
                        : execution of the machine instructions.

INTERPRET MODE         : A SET_TRAP command has been issued with
TURNED ON              : a trap type that requires interpret mode
                        : of program execution, and, currently,
                        : interpret mode is off. Subsequent
                        : program execution will be by interpreting
                        : all machine instructions.

NO BREAKPOINT xxxx    : A request has been made to DISPLAY,
                        : CLEAR, or SAVE a breakpoint at location
                        : xxxx. No such breakpoint exists. Any
                        : remaining list elements are processed.

NO BREAKPOINT #n      : A request has been made to DISPLAY,
                        : CLEAR, or SAVE a breakpoint #n which does
                        : not exist. Any remaining list elements
                        : are processed.

NO BREAKPOINTS        : There are no breakpoints to DISPLAY,
                        : CLEAR, or SAVE.

```

5.0 MESSAGES ++++++

5.2.1 INFORMATIVE MESSAGES

Message	Significance
NO PROCEDURE xxxx	A request has been made to DISPLAY, CLEAR, or SAVE a debugger SCL procedure xxxx which does not exist. Any remaining list elements are processed.
NO PROCEDURES	There are no debugger SCL procedures to DISPLAY, CLEAR, or SAVE.
NO xxxx TRAP yyyy	A request has been made to DISPLAY, CLEAR, or SAVE a user-defined trap of type xxxx with scope yyyy. No such trap exists. Any remaining list elements are processed.
NO TRAP #n	A request has been made to DISPLAY, CLEAR, or SAVE a user-defined trap #n which does not exist. Any remaining list elements are processed.
NO TRAPS	There are no user-defined traps to DISPLAY, CLEAR, or SAVE. Note that the three default traps: END, ABORT, and INTERRUPT are never displayed, cleared or saved.
PAUSE IGNORED FROM TERMINAL	This message results from entering PAUSE while in interactive (non-collect) mode.
TIME LIMIT	A time limit interrupt has occurred while either the program or a CCDBG sequence was executing. A small amount of time is left, sufficient to do a SAVE_ALL and QUIT. To continue the session, enter CHECKPOINT, followed by a QUIT and DEBUG(RESUME).
TRAP NUMBER IGNORED IN THIS CONTEXT	A trap number has been specified as a list element in DISPLAY_TRAP, CLEAR_TRAP, or SAVE_TRAP command of a form for which trap numbers are not allowed. Any remaining list elements are processed.

5.0 MESSAGES ++++++

5.2.1 INFORMATIVE MESSAGES

```

USER PROGRAM          : A program interrupt was detected while
INTERRUPT PENDING    : execution of a CYBIL-CC statement
                     : compiled with the DEBUG parameter was in
                     : progress. Unless a second interrupt is
                     : issued, the interrupt is delayed until
                     : the execution of the current statement is
                     : completed. However, in the interim, a
                     : breakpoint or trap with no body or a
                     : PAUSE statement has been encountered.
                     : This message cautions that control is to
                     : be given to the user by some means other
                     : than the result of the terminal input.
                     : The terminal interrupt is acknowledged.
                     : Control is regained by entering GO or
                     : EXECUTE, causing the CYBIL-CC statement
                     : to complete its execution.

USER RECOVER ROUTINE : CCDBG issues this message after the user
COMPLETED, x        : program has completed its recover
REQUESTED            : routine by making an ABORT or ENDRUN
                     : request.

VARIABLE NAMES NOT  : Either program xxx is not a CYBIL-CC
AVAILABLE FOR xxx   : program, or it was not compiled with the
                     : DEBUG option explicitly specified.

xxx TREATED AS ;    : A collect was found following a command
                     : other than SET_TRAP, SET_BREAKPOINT, or
                     : SET_PROCEDURE; or a COLLECT_END was found
                     : after a statement while not in collect
                     : mode.

```

6.0 ALPHABETICAL COMMAND SUMMARY

6.0 ALPHABETICAL_COMMAND_SUMMARY

```

backward : bw          numlocs=<integer>
                       [format=oct ; dec ; adr ; hex]

change_defaults : cd   [module=<name>]
                       [proc=<name> [.<name>]...]
                       [overlay=(<integer>,<integer>)]

change_memory : cm     <address_expr>[module=<name>]
                       [offset=<integer>] value=<expr>
                       [numlocs=<integer>] [indirect]

change_registers : cr  a ; b ; x=<integer>
                       value=<expr> [indirect]

change_variable : cv   var=<name> value=<expr>
                       [module=<name>][proc=<name>
                       [.<name>]...]

checkpoint : ck        file=<name>

clear_auxiliary : caux

clear_breakpoint : cb  [scope_expr]
                       [b=<integer>[<..><integer>]:
                       (<integer>[<..><integer>]
                       [<sep><integer>[<..><integer>]]...)]
                       [offset=<integer>]
                       [overlay=(<integer>,<integer>)]
                       [module=<name>]

clear_interpret : ci

clear_procedure : cp   [procedure=<name> ;
                       (<name>[<sep><name>]...)]

clear_trap : ct        [type=<trap_type>][scope_expr]
                       [t=<integer>[<..><integer>]:
                       (<integer>[<..><integer>]
                       [<sep><integer>[<..><integer>]]...)]
                       [overlay=(<integer>,<integer>)]
                       [module=<name>] [proc=<name>]

```

6.0 ALPHABETICAL COMMAND SUMMARY

[.<name>]...]

clear_veto : cve

display_breakpoint : db [address_expr]
 [b=<integer>[<..><integer>];
 (<integer>[<..><integer>
 [<sep><integer>[<..><integer>]]...)]
 [offset=<integer>]
 [overlay=(<integer>,<integer>)]
 [module=<name>]

display_default : dd

display_map : dmap [module=<name>]
 [overlay=(<integer>,<integer>)]

display_memory : dm <address_expr>
 [format=oct : dec : adr : hex]
 [numlocs=<integer>]
 [offset=<integer>][module=<name>]
 [indirect]

display_procedure : dp [procedure=<name> ;
 [<name>[<sep><name>]...]]

display_registers : dr [p : fl : [a : b : x=[<integer>]]]
 [format=oct : dec : adr : hex]
 [indirect]

display_status : ds

display_trap : dt [type=<trap_type>] [scope_expr]
 [t=<integer>[<..><integer>];
 (<integer>[<..><integer>
 [<sep><integer>[<..><integer>]]...)]
 [overlay=<integer>,<integer>]
 [module=<name>] [proc=<name>]
 [.<name>]...]

display_variable : dv [var=<name>]
 [format=oct : dec : hex]
 [module=<name>] [proc=<name>]
 [.<name>]...]

execute : ex [address_expr] [offset=<integer>]

6.0 ALPHABETICAL COMMAND SUMMARY

```

forward : fw          numlocs=<integer>
                    [format=oct : dec : adr : hex]

go                  [address_expr] [offset=<integer>]

goto               label=<name>

help : h           [<subject> : <command_name>]

label : la         name=<name>

message : me       [text='[<ascii>]...']

move : m           source_expr destination_expr
                    [numlocs=<integer>]

pause : pa        [text='[<ascii>]...']

quit               [normal : abort]

read : r           file : procedure = <name>

save_all : savea   file=<name>

save_breakpoint : saveb file=<name>[scope_expr]
                    [b=<integer>[<..><integer>]:
                    (<integer>[<..><integer>]
                    [<sep><integer>[<..><integer>]]...)]
                    [offset=<integer>]
                    [overlay=(<integer>,<integer>)]
                    [module=<name>]

save_procedure : savep file=<name>
                    [procedure=<name> :
                    (<name>[<sep><name>]...)]

save_trap : savet  file=<name> [type=<trap_type>]
                    [scope_expr]
                    [t=<integer>[<..><integer>] :
                    (<integer>[<..><integer>]
                    [<sep><integer>[<..><integer>]]...)]
                    [overlay=(<integer>,<integer>)]
                    [module=<name>] [proc=<name>]
                    [.<name>]...

set_auxiliary : saux file=<name> lo=<options>

```

6.0 ALPHABETICAL COMMAND SUMMARY

```

set_breakpoint : sb      [address_expr]
                        [module=<name>][overlay=(<integer>
                        ,<integer>)] [offset=<integer>]
                        [first=<integer>] [last=<integer>]
                        [step=<integer>] [<collect><eol!;>]
                        [<command_statement><eol!;>]...
                        <collect_end>

```

set_interpret : si

set_output : so io=<options>

```

set_procedure : sp      procedure=<name> [<collect><eol!;>]
                        [<command_statement><eol!;>]...
                        <collect_end>

```

```

set_trap : st           type=<trap_type> [scope_expr]
                        [overlay=(<integer>,<integer>)]
                        [module=<name>] [proc=<name>]
                        [.<name>]... [<collect><eol!;>]
                        [<command_statement><eol!;>]...
                        <collect_end>

```

set_veto : sve

skipif : s <value_1><relation><value_2>

traceback : tb [entrypoint=<name>]

Table of Contents

1.0 SCOPE	1-1
1.1 APPLICABLE DOCUMENTS	1-1
2.0 INTRODUCTION	2-1
2.1 OVERVIEW	2-1
2.2 TERMINOLOGY	2-2
2.3 METALANGUAGE	2-3
2.4 BASIC CONCEPTS	2-4
2.4.1 INTEGER	2-4
2.4.2 NAME	2-5
2.4.3 ADDRESS	2-5
2.4.4 CCDBG VARIABLES	2-5
2.4.5 EXPRESSION (EXPR)	2-6
2.4.6 SEPARATOR	2-6
2.4.7 COMMENTS	2-6
2.4.8 COMMAND	2-7
2.4.9 PARAMETER LISTS	2-7
2.4.10 VALUE LISTS	2-7
2.4.11 CONTINUATION	2-8
3.0 GENERAL DESCRIPTION OF CCDBG USE	3-1
3.1 PROGRAM EXECUTION CONTROL	3-2
3.2 CYBIL-CC SOURCE SYMBOL REFERENCE CAPABILITY	3-2
3.3 MACHINE LEVEL DEBUGGING FEATURES	3-2
3.4 OVERLAY ENVIRONMENT	3-3
3.5 DEBUGGER SCL PROCEDURE	3-3
3.6 CONDITIONAL COMMAND EXECUTION CAPABILITY	3-3
3.7 VETO MODE	3-3
3.8 DEFINITION FILE CAPABILITY	3-3
3.9 CHECKPOINT/RESTART CAPABILITY	3-4
3.10 WARNING CAPABILITY	3-4
3.11 CCDBG VARIABLES	3-4
3.12 INFORMATION OUTPUT	3-4
3.13 HELP COMMAND	3-4
4.0 CCDBG COMMANDS	4-1
4.1 BREAKPOINTS AND TRAPS	4-1
4.1.1 SET_BREAKPOINT ; SB	4-1
4.1.2 SET_TRAP ; ST	4-3
4.1.3 DISPLAY_BREAKPOINT ; DB	4-6
4.1.4 DISPLAY_TRAP ; DT	4-7
4.1.5 SAVE_BREAKPOINT ; SAVEB	4-9
4.1.6 SAVE_TRAP ; SAVET	4-11
4.1.7 CLEAR_BREAKPOINT ; CB	4-12
4.1.8 CLEAR_TRAP ; CT	4-14
4.1.9 SET_INTERPRET ; SI	4-16
4.1.10 CLEAR_INTERPRET ; CI	4-16
4.2 DISPLAYING AND CHANGING PROGRAM VARIABLES	4-16
4.2.1 DISPLAY_VARIABLE ; DV	4-16

4.2.2 CHANGE_VARIABLE ; CV	4-18
4.3 DISPLAYING AND CHANGING MEMORY AND REGISTERS	4-18
4.3.1 DISPLAY_MEMORY ; DM	4-19
4.3.2 FORWARD ; FW	4-20
4.3.3 BACKWARD ; BW	4-21
4.3.4 CHANGE_MEMORY ; CM	4-21
4.3.5 DISPLAY_REGISTERS ; DR	4-22
4.3.6 CHANGE_REGISTERS ; CR	4-23
4.4 DEBUGGER SCL PROCEDURES	4-24
4.4.1 SET_PROCEDURE ; SP	4-24
4.4.2 DISPLAY_PROCEDURE ; DP	4-24
4.4.3 SAVE_PROCEDURE ; SAVEP	4-25
4.4.4 CLEAR_PROCEDURE ; CP	4-25
4.5 DEBUGGER SCL PROCEDURE COMMANDS	4-26
4.5.1 PAUSE ; PA	4-26
4.5.2 MESSAGE ; ME	4-26
4.5.3 GO	4-27
4.5.4 SKIPIF ; S	4-28
4.5.5 LABEL ; LA	4-28
4.5.6 GOTD	4-29
4.5.7 READ ; R	4-29
4.6 ADDITIONAL INTERACTIVE MODE COMMANDS	4-30
4.6.1 EXECUTE ; EX	4-30
4.6.2 TRACEBACK ; TB	4-30
4.6.3 MOVE ; M	4-31
4.6.4 HELP ; H	4-32
4.6.5 SAVE_ALL ; SAVEA	4-33
4.6.6 CHECKPOINT ; CK	4-33
4.6.7 QUIT	4-33
4.7 CCDBG ENVIRONMENT COMMANDS	4-34
4.7.1 SET_VETO ; SVE	4-34
4.7.2 CLEAR_VETO ; CVE	4-34
4.7.3 SET_OUTPUT ; SO	4-34
4.7.4 SET_AUXILIARY ; SAUX	4-35
4.7.5 CLEAR_AUXILIARY ; CAUX	4-35
4.7.6 DISPLAY_MAP ; DMAP	4-35
4.7.7 DISPLAY_DEFAULTS ; DD	4-36
4.7.8 DISPLAY_STATUS ; DS	4-36
4.7.9 CHANGE_DEFAULTS ; CD	4-37
5.0 MESSAGES ++++++	5-1
5.1 DIAGNOSTIC MESSAGES	5-1
5.2 WARNING MESSAGES	5-12
5.2.1 INFORMATIVE MESSAGES	5-14
6.0 ALPHABETICAL COMMAND SUMMARY	6-1