SIMH Users' Guide Supplement
20-Jun-2023

COPYRIGHT NOTICE

The following copyright notice applies to the SIMH source, binary, and documentation:

Original code published 1993-2012, written by Robert M Supnik
Copyright © 1993-2012, Robert M Supnik
Copyright © 2019-2023, J. David Bryan

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the names of the authors shall not be used in advertising or otherwise to promote the sale, use, or other dealings in this Software without prior written authorization from the authors.

1 Introduction 4
1.1 Unpacking and Compiling the Simulator 4

2 Simulator Control Program Extensions 5
2.1 Quoted Strings 5
2.2 BREAK Command 5
2.3 REPLY Command 6
2.4 GO UNTIL and RUN UNTIL Commands 7
2.5 GO FOR and RUN FOR Commands 8
2.6 IF Command 8
2.7 Quoting Actions 9
2.8 GOTO Command 10
2.9 CALL and RETURN Commands 11
2.10 DELETE Command 11
2.11 PAUSE Command 12
2.12 ATTACH and DETACH Commands 12
2.13 SET Command 13
2.13.1 SET BINARY 13
2.13.2 SET ENVIRONMENT 13
2.13.3 SET CONSOLE CONCURRENT 14
2.13.4 SET CONSOLE SERIAL 15
2.14 DO Command 15
2.14.1 DO and SET CONSOLE CONCURRENT 16
2.15 ABORT Command 16
2.16 Aborting Command File Execution 17
2.17 FLUSH Command 17
2.18 SHOW Command 17
2.19 Variable Substitution 18
2.20 Global Initialization File 20

3 SCP 4.x-to-3.x Conversion 22
3.1 Unimplemented Commands 22
3.2 SCP 3.x Replacements 23
3.2.1 Quoted Strings 23
3.2.2 EXPECT and SHOW EXPECT 23
3.2.3 SEND and SHOW SEND 24
3.2.4 SLEEP 24
3.2.5 Host File System Commands 24
3.2.6 IF...ELSE 25
3.2.7 SET REMOTE and SHOW REMOTE 25
3.2.8 Miscellaneous Commands 25
3.3 SCP 3.x Differences 25
3.3.1 Initialization File Search Order 25
3.3.2 Serial Port Names in ATTACH and DETACH Commands 25
3.3.3 IF Command 25
3.3.4 HELP Command 26

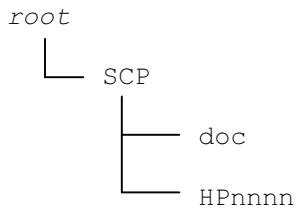
3.3.5 Predefined Substitution Variables 26
3.3.6 Command Option Switches 26

1 Introduction

This manual documents the extensions to the Simulator Control Program (SCP) provided by the HP 2100 and 3000 simulators. It is intended for use in conjunction with the *SIMH Users' Guide* manual, which describes the general commands that may be entered at the SCP prompt.

1.1 Unpacking and Compiling the Simulator

The simulator source distribution consists of a single ZIP archive file. The archive directory structure is:



The ***root*** directory of the distribution is empty but will contain the simulator executable after compilation. The **SCP** subdirectory contains the source files for the Simulator Control Program and makefiles to build the simulator. The **HPnnnn** subdirectory contains the files for the HP2100 or HP3000 simulator. The **doc** subdirectory contains the simulator operation guides in PDF format and the release notes in text format. This directory structure must be preserved when the distribution archive is unpacked.

To build the simulator, first unpack the archive into a blank directory; this becomes the root directory of the simulator. The source files contain Windows line ends (i.e., CR LF pairs). Your host platform may object to this; if so, convert the files to your local line-end convention. The ***unzip*** utility from Info-ZIP will do this automatically if the ***–a*** option switch is specified.

Two makefiles are provided: one for GNU make, and the other for the Microsoft Visual C++ NMAKE utility. Both provide these targets:

- hp2100 — to make the HP 2100 simulator
- hp3000 — to make the HP 3000 simulator
- clean — to remove the simulator object and executable files.

...and both will place the resulting executable file in the root directory. Both are invoked from the root directory.

To compile for Windows and UNIX/Linux systems using the *gcc* or *clang* compilers and GNU make, enter:

```
make -C SCP <target>
```

...for a gcc compilation, or:

```
make -C SCP <target> GCC=clang
```

...for a clang compilation.

To compile for Windows using the Microsoft Visual C++ compiler and NMAKE, enter:

```
nmake /f SCP\makefile.mak <target>
```

The intermediate object files will be placed in the **Release** subdirectory of the root directory.

2 Simulator Control Program Extensions

The HP simulators provide all of the commands described in the *SIMH Users' Guide* manual, in addition to those new or extended commands described below. Additional simulator-specific commands are described in the individual HP Simulator User's Guides for the HP2100 and HP3000 simulators. All commands and command switches are case-insensitive; they are rendered in uppercase in examples below simply for clarity.

2.1 Quoted Strings

Several extended commands take quoted-string parameters, which receive additional processing. A quoted string is a sequence of characters delimited either by quotation marks or apostrophes. The starting and ending quote characters must match. Within a quoted string, the following escape sequences are replaced with the corresponding characters:

Sequence	***Replacement***	***Description***
%%	%	Percent symbol
\r	*CR*	Carriage return (octal 015)
\n	*LF*	Line feed (octal 012)
\"	"	Quotation mark
\'	'	Apostrophe
****	\	Backslash
\nnn	*octal nnn*	Octal character (000-177)

The **\"** and **\'** escapes permit insertion of the string delimiter into the quoted string. For example, these quoted strings are interpreted identically:

```
"Quoth the raven \"Nevermore.\""

'Quoth the raven "Nevermore."'
```

The ***nnn*** escape permits insertion of any character in the octal range 000-177. Exactly three octal digits (0-7) must be present.

2.2 BREAK Command

The existing ***BREAK*** command described in the *Breakpoints* section of the *SIMH Users' Guide* manual is extended to add temporary breakpoint capability. The new ***–T*** breakpoint type switch indicates that the breakpoint will be set temporarily. A temporary breakpoint is removed once it occurs; this is equivalent to setting the (first) breakpoint action to ***NOBREAK***. Without ***–T***, the breakpoint is persistent and will cause a simulation stop each time it occurs.

The existing ***BREAK*** and ***NOBREAK*** commands are also extended to accept character strings as well as memory addresses. String breakpoints cause simulation stops when the character sequences are encountered in the system console output stream; these are similar to stops that occur when CPU execution reaches specified memory addresses. The additional command forms are:

```
BREAK { -T } <quoted-string> { ; <action> ... }
BREAK { -T } <quoted-string> DELAY <interval> { ; <action> ... }
BREAK DELAY <interval>
NOBREAK <quoted-string>
NOBREAK ""
```

The first form sets a breakpoint that stops simulator execution when the content of the quoted string appears in the system console output. By default, the simulator stops immediately after the final character of the quoted string is output. This may be too fast for the interrupted program if subsequent SCP commands affect it (e.g., if entering a tape unit ***ATTACH*** command generates an interrupt that will be processed by the program). In this case, the second form may be used to insert a delay of the specified interval before execution stops. With either form, if the quoted string duplicates an existing breakpoint, that breakpoint will be updated with the specified delay and actions. As with address breakpoints, the ***–T*** switch indicates that the breakpoint is temporary and should be cleared once it occurs.

If the second form is used, the delay is set temporarily for that breakpoint; it then reverts to the default delay for subsequent breakpoints. If all of the breakpoints for a program require a delay, it may be set as the new default by using the third form; the initial default delay is 0. Combinations of the first three forms may be interspersed as necessary to meet program timing requirements.

Delay intervals may be expressed either directly by a decimal event tick count, or as a *realistic time* value by a decimal count and a time unit. Time units may be ***MICROSECOND[S]***, ***MILLISECOND[S]***, or ***SECOND[S]***; the abbreviations ***USEC[S]*** and ***MSEC[S]*** may be used interchangeably for the first two units, respectively.

The optional action commands are executed when the breakpoint occurs. If the breakpoint is temporary, the actions are executed once; otherwise, they execute each time the breakpoint occurs.

The ***NOBREAK*** command cancels a pending string breakpoint. If the quoted string specifies an existing breakpoint, that breakpoint will be deleted. If the quoted string is empty, then all string breakpoints are deleted. The existing ***NOBREAK ALL*** form cancels all string and address breakpoints.

2.3 REPLY Command

The ***REPLY*** command supplies keyboard input from the system console. It may be used in command files to supply responses to program prompts during long or tedious user interactions (such as system generations). ***REPLY*** sets up the keyboard input but does not resume simulator execution; a subsequent ***GO*** is required to continue simulation. The command forms are:

```
REPLY <quoted-string>
REPLY <quoted-string> DELAY <interval>
REPLY DELAY <interval>
NOREPLY
```

The first form supplies the content of the quoted string to the system console, character by character, as though entered by pressing keys on the keyboard. By default, the first character is supplied to the console device immediately after simulation is resumed with a ***GO*** or ***CONTINUE*** command. This may be too fast for the receiving program, e.g., if it initializes the console device before looking for characters. In this case, the second form may be used to insert a delay of the specified interval before the first character is supplied.

If the second form is used, the delay is set temporarily for that reply; it then reverts to the default delay for subsequent replies. If all of the replies to a program require a delay, it may be set as the new default by using the third form; the initial default is 0. Combinations of the first three forms may be interspersed as necessary to meet program timing requirements.

Delay intervals may be expressed either directly by a decimal event tick count, or as a *realistic time* value by a decimal count and a time unit. Time units may be ***MICROSECOND[S]***, ***MILLISECOND[S]***, or ***SECOND[S]***; the abbreviations ***USEC[S]*** and ***MSEC[S]*** may be used interchangeably for the first two units, respectively.

The ***NOREPLY*** command cancels any pending reply. Specifying a new reply also cancels any existing one, as only a single reply may be active at any given time. Replies are also effectively cancelled when they are consumed.

2.4 GO UNTIL and RUN UNTIL Commands

The existing ***GO*** and ***RUN*** commands are extended with an ***UNTIL*** clause that sets one or more temporary breakpoints. This provides a simplified method of stepping through a program and stopping at a series of addresses. For example, these commands:

```
BREAK 100
GO
NOBREAK
EXAMINE A
BREAK 200
GO
NOBREAK
EXAMINE B
```

...are equivalent to:

```
GO UNTIL 100
EXAMINE A
GO UNTIL 200
EXAMINE B
```

This also provides a concise way of specifying prompt/response pairs when automating replies to program prompts. For example, these commands:

```
BREAK -T "prompt"
GO
REPLY "response"
```

...are equivalent to:

```
GO UNTIL "prompt" ; REPLY "response"
```

The ***GO UNTIL*** and ***REPLY*** commands provide an easy way to automate a series of prompts and responses; for example:

```
GO UNTIL "OPTION? "   ; REPLY "RELOAD\r"
GO UNTIL "CHANGES? "  ; REPLY "YES\r"
GO UNTIL "MEM SIZE? " ; REPLY "128\r"
[...]
```

The extended command forms for both ***GO UNTIL*** and ***RUN UNTIL*** are:

```
GO UNTIL <stop-address> { ; <action> ... }
GO UNTIL <quoted-string> { ; <action> ... }
GO UNTIL <quoted-string> DELAY <interval> { ; <action> ... }

GO <start-address> UNTIL <stop-address> { ; <action> ... }
GO <start-address> UNTIL <quoted-string> { ; <action> ... }
GO <start-address> UNTIL <quoted-string> DELAY <interval> { ; <action> ... }
```

Multiple <stop-address>es, separated by commas, may be specified. For example, ***GO 5 UNTIL 10,20*** sets temporary breakpoints at addresses 10 and 20 and then resumes simulator execution at address 5. As with the ***BREAK*** command, specifying a ***DELAY*** value sets a temporary delay of the specified number of machine instructions before execution stops. If a ***DELAY*** value is not given, the breakpoint uses the default delay set by an earlier ***BREAK DELAY*** command, or a zero delay if the default has not been overridden.

Delay intervals may be expressed either directly by a decimal event tick count, or as a *realistic time* value by a count and a decimal time unit. Time units may be ***MICROSECOND[S]***, ***MILLISECOND[S]***, or ***SECOND[S]***; the abbreviations ***USEC[S]*** and ***MSEC[S]*** may be used interchangeably for the first two units, respectively.

2.5 GO FOR and RUN FOR Commands

The ***GO*** and ***RUN*** commands are extended with a ***FOR*** clause that breaks execution after a specified interval. The command forms are:

```
GO FOR <interval> { ; <action> ... }
GO <start-address> FOR <interval> { ; <action> ... }
```

The interval may be expressed either directly by an event tick count, or as a *realistic time* value by a count and a time unit. Time units may be ***MICROSECOND[S]***, ***MILLISECOND[S]***, or ***SECOND[S]***; the abbreviations ***USEC[S]*** and ***MSEC[S]*** may be used interchangeably for the first two units, respectively. The time delay specified is a *realistic time*, i.e., one that represents wall-clock time when using the original hardware. Although the simulator typically runs an order of magnitude faster than the original hardware did, a given time delay will allow the CPU to execute approximately the same number of machine instructions as would be executed on real hardware during that time.

Timed execution is useful to simulate computer operator response delays. For example, a ***GO FOR 5 SECONDS*** command might be inserted in a command file between a ***BOOT*** command and a ***REPLY*** command. This would allow the operating system enough time to complete its startup process before submitting the first operator command.

2.6 IF Command

The new ***IF*** command permits conditional execution of SCP commands. The command form is:

```
IF { -I } { -E } <comparative-expression> <action> { ; <action> ... }
```

The <comparative-expression> forms are:

```
<Boolean-expression>
<Boolean-expression> <logical> <comparative-expression>
```

The <Boolean-expression> forms are:

```
<quoted-string> <equality> <quoted-string>
<quoted-string> IN <quoted-string> { , <quoted-string> ...}
<quoted-string> NOT IN <quoted-string> { , <quoted-string> ...}
EXIST <quoted-string>
NOT EXIST <quoted-string>
```

The equality and logical operators are described below:

Type	*Operator*	*Description*
equality	==	Equal to
	!=	Not equal to
logical	&&	And
	||	Or

The ***IN*** operation returns *true* if the first quoted string is equal to any of the listed quoted strings, and the ***NOT IN*** operation returns *true* if the first quoted string is not equal to any of the listed strings. For example, these command fragments:

```
IF "%1" IN "CPU","DISC","TAPE" ...
IF "%2" NOT IN "YES","NO","MAYBE" ...
```

...are equivalent to these more verbose forms:

```
IF "%1" == "CPU" || "%1" == "DISC" || "%1" == "TAPE" ...
IF "%2" != "YES" && "%2" != "NO" && "%2" != "MAYBE" ...
```

...respectively.

The ***EXIST*** operation returns *true* if the file specified by the quoted string exists. The ***NOT EXIST*** operation returns *true* if the file does not exist.

If the comparative expression is true, the associated actions are executed. If the expression is false, the actions have no effect. Adding the ***–I*** switch causes the string comparisons to be made case-insensitively. Adding the ***–E*** switch causes the quoted strings to be scanned for escape sequence replacements; without the switch, escapes are treated as ordinary characters. Comparisons are always textual; so, for example, "3" is not equal to "03". Evaluation is strictly from left to right; embedded parentheses to change the evaluation order are not accepted.

2.7 Quoting Actions

The ***BREAK***, ***GO/RUN UNTIL***, and ***IF*** commands each accept a list of action commands to be performed if the condition is true or when the breakpoint occurs. If one of the action commands is itself one of these three, then the actions may not be grouped with the correct actor. Consider, for example, a conditional command in a ***DO*** command file that is intended to display a register value when a breakpoint is reached:

```
IF "%1" == "TEST" BREAK 100 ; EXAMINE A
GO
```

Assuming the test succeeds, the problem is that the ***EXAMINE*** command could be associated either with the ***BREAK*** command or the ***IF*** command. If grouped with ***BREAK***, it would be executed when the breakpoint occurred and would display:

```
sim> BREAK 100 ; EXAMINE A
sim> GO

Breakpoint, P: 00100 (NOP)
sim> EXAMINE A
A:      000000
```

If grouped with ***IF***, as indeed it will be, it would be executed immediately when the command was entered, and the display would be:

```
sim> BREAK 100
sim> EXAMINE A
A:      000000
sim> GO
```

Consider also this command:

```
IF "%1" == "TEST" GO UNTIL "READY?" ; REPLY "YES\r"
```

This will not provide a response to the prompt. In fact, the reply will never be executed, as the ***GO*** command clears any pending actions before resuming execution.

Another problem is that commands containing semicolons as parameter separators cannot be used as breakpoint actions. For example, this command:

```
BREAK 100 ; SET CPU DEBUG=INSTR;DATA
```

...will fail when ***DATA*** is executed as the second action.

To resolve these ambiguities, the commands take action command lists as quoted strings, which are then grouped as single commands. Using the above examples, the addition of surrounding quotes:

```
IF "%1" == "TEST" "BREAK 100 ; EXAMINE A"
IF "%1" == "TEST" 'GO UNTIL "READY?" ; REPLY "YES\r"'
BREAK 100 ; 'SET CPU DEBUG=INSTR;DATA'
```

...causes ***EXAMINE*** to be the action command for ***BREAK***, and ***REPLY*** to be the action command for ***GO UNTIL***, instead of both being the second action command for ***IF***, and causes the ***INSTR*** and ***DATA*** trace options to be set, instead of failing with an **Unknown command** error.

2.8 GOTO Command

The new ***GOTO*** command is used within command files to transfer execution to a labeled statement. The syntax is:

```
GOTO <label>
```

The label is a unique identifier that must appear, preceded by a colon (:), on a line by itself, somewhere within the current command file. When the command is encountered, a case-sensitive search for the label begins at the first line of the command file and extends through the file until the label is found; if it is not present, an **Invalid argument** error is displayed.

The command may be used with the ***IF*** command and parameter substitution to provide conditional command file behavior; for example:

```
IF "%1" == "REALISTIC" GOTO real

SET TTY FASTTIME
SET PTR FASTTIME
GOTO done

:real
SET TTY REALTIME
SET PTR REALTIME

:done
```

Computed GOTOs are also possible; for example:

```
IF "%1" IN "A","B","C" GOTO %1

ECHO Invalid option %1 was supplied.
GOTO done

:A
[...]
GOTO done

:B
[...]
GOTO done

:C
[...]
GOTO done

:done
```

In this case, the ***IF*** command is used to guard against specifying an illegal option and attempting to go to an undefined label.

If the ***GOTO*** command is entered interactively at the SCP prompt, it is rejected with a **Command not allowed** error.

2.9 CALL and RETURN Commands

The new ***CALL*** command is used within command files to transfer execution to a labeled subroutine. The syntax is:

```
CALL <label> { <param> ... }
```

The command saves the current location in the command file and then performs a ***GOTO <label>*** to transfer control to the start of the subroutine. Within the subroutine, the substitution variables ***%1*** through ***%9*** correspond to the ***CALL*** parameters; if fewer than nine parameters are passed, the corresponding substitution variables are null. The value of the substitution variable ***%0*** is the name of the file containing the command.

The subroutine returns to the line after the originating ***CALL*** command by executing the new ***RETURN*** command. The syntax is:

```
RETURN
```

Executing a ***RETURN*** without a preceding ***CALL*** exits the command file as if a ***GOTO*** to a label immediately preceding the end of the file were executed.

2.10 DELETE Command

The new ***DELETE*** command removes the host file specified in the command. The syntax is:

```
DELETE <filename>
```

The command provides a platform-independent way to delete files from a command file, for instance to delete temporary files that were created by an ***ATTACH*** command:

```
ATTACH -N MS0 scratch.tape
[...]
DETACH MS0
DELETE scratch.tape
```

It is functionally identical to executing the host platform's file deletion command (e.g., ***! rm scratch.tape***).

2.11 PAUSE Command

The new ***PAUSE*** command provides a platform-independent way to suspend command file execution for a user-specified time interval. The syntax is:

```
PAUSE <time> <units>
```

The time is a decimal integer specifying the multiple of time units to wait. The time unit may be ***MICROSECOND[S]***, ***MILLISECOND[S]***, or ***SECOND[S]***; the abbreviations ***USEC[S]*** and ***MSEC[S]*** may be used interchangeably for the first two units, respectively. Pauses shorter than one millisecond are rejected. A command specifying a long pause, such as ***PAUSE 30 SECONDS***, may be aborted by entering CTRL+E; SCP responds with **Command not completed** if the full pause time did not elapse.

2.12 ATTACH and DETACH Commands

The existing ***ATTACH*** command is extended to permit terminal multiplexer lines to be connected to serial ports as well as Telnet listening ports. The syntax is:

```
ATTACH <unit> <port-name>{;<rate>-<size><parity><stopbits>}
```

The unit name indicates the multiplexer line to attach; for example, *MUX2* would indicate that multiplexer line 2 is to be connected to a serial port. The port name is the host-specific serial port name, e.g., *COM1* or */dev/ttyS0*.

An optional serial port configuration string may be supplied after the host name. The required values are:

- ***rate*** is the baud rate in bits per second.
- ***size*** is the character size in bits including the parity bit, if designated.
- ***parity*** designates the parity to use: *N* (no), *E* (even), *O* (odd), *M* (mark), or *S* (space).
- ***stopbits*** is the number of stop bits (*1*, *1.5*, or *2*).

If the port configuration string is omitted, the default configuration specified by the host system for that port is used. If the system default cannot be obtained, the command is rejected and reports **Too few arguments**. This indicates that the ***ATTACH*** command must be reentered with the port configuration string added.

The existing ***DETACH*** command is extended to permit serial ports to be disconnected. The syntax is:

```
DETACH <unit>
```

...where the specified terminal multiplexer unit is currently attached to a serial port. Detaching a serial connection has no effect on any Telnet connections that may be active.

2.13 SET Command

Two new ***SET*** command options have been added, and the existing ***SET CONSOLE*** command has been extended.

2.13.1 SET BINARY

The new ***SET <device> <radix>*** command is enhanced to add setting the default output radix to base 2. The command forms are now:

```
SET <device> BINARY
SET <device> OCTAL
SET <device> DECIMAL
SET <device> HEX
```

For example, the ***SET CPU BINARY*** command changes the default display of examined memory locations to base 2.

2.13.2 SET ENVIRONMENT

The new ***SET ENVIRONMENT*** command creates a user-defined variable and sets its value. The command forms are:

```
SET ENVIRONMENT <variable>=<value>
SET ENVIRONMENT <quoted-string> { , <quoted-string> ...}
```

The first form creates the variable and adds it to the host environment with the string value. Subsequently, it may be referenced in other commands by bracketing the variable name with percent signs. For example:

```
SET ENVIRONMENT A=Hello
ECHO %A%
GO UNTIL "%A%"
```

The ***ECHO*** command will print **Hello** on the simulation console, and the ***GO UNTIL*** command will execute machine instructions until the string **Hello** is output to the system console.

The second form consists of quoted variable assignments, separated by commas. This permits several variables to be set to their corresponding values in a single command. The last assignment in the list may be unquoted, if desired, with the value consisting of all characters remaining in the command. For example:

```
SET ENVIRONMENT 'B=YES','C=NO',D=MAYBE,E=UNSURE
```

...sets variable B to "YES", C to "NO", and D to "MAYBE,E=UNSURE".

If the assignment is not a quoted string, then escape sequences are not decoded. So the commands:

```
SET ENVIRONMENT 'F=\060'
SET ENVIRONMENT G=\060
```

...result in variable F having the value "0" and G having the value "\060". However, predefined and user-defined variables may be used unquoted, and the values will be those at the time the command is entered. This command may be abbreviated as ***SET ENV***.

Note also that Windows and UNIX/Linux systems treat variable name case differently. On Windows, letter case is preserved but not significant, whereas on UNIX/Linux systems, case is significant. Using the first example, an ***ECHO %a%*** command will print **Hello** on Windows but nothing on UNIX/Linux.

2.13.3 SET CONSOLE CONCURRENT

The new ***SET CONSOLE CONCURRENT*** command establishes a mode that permits SCP commands to be entered via the simulation console without stopping simulator execution. A typical example where this is helpful is mounting a new magnetic tape reel on a tape drive.

Normally, mounting a new tape image requires stopping simulator execution with CTRL+E to obtain the SCP command prompt, entering an ***ATTACH*** command to specify the tape image to mount, and entering a ***GO*** command to continue execution. For example:

```
19:44/#S1/14/LOGON FOR: OPERATOR.SYS,OPERATOR ON LDEV #20
HP3000 / MPE V  E.01.00 (BASE E.01.00).  MON, APR 29, 1991,  7:44 PM
:[CTRL+E]

Simulation stopped, P: 071144 (PAUS 0)
sim> ATTACH MS0 backup.tape
sim> GO

19:44/10/Vol (unlabelled) mounted on LDEV# 7
:
```

While the simulation is stopped, the target operating system's time-of-day clock is stopped, so that each time the simulation is stopped to enter SCP commands, the clock loses time. Moreover, all terminal multiplexer activity is stopped as well, so that remote users will have their sessions freeze until simulation is resumed.

However, if concurrent mode is enabled, then entering CTRL+E displays a special SCP prompt (***scp>***) *without* stopping simulator execution. While the ***scp>*** prompt is displayed and command characters are being entered, the simulator continues to execute CPU instructions. When the ENTER key is pressed, simulation pauses just long enough to perform the command before an implicit ***GO*** resumes instruction execution. For example:

```
:[CTRL+E]
scp> ATTACH MS0 backup.tape

19:49/10/Vol (unlabelled) mounted on LDEV# 7

:
```

If the system clock is set for calibrated timing, this small loss of time will be corrected automatically, and the clock will continue to track wall-clock time.

The list of commands that may be entered at the ***scp>*** prompt is limited to those that do not interfere with simulator execution or pause for input. Specifically, entering any of the commands listed below will result in a **Command not allowed** error, and the command will be ignored.

Restricted Commands		
IEXAMINE	**CONTINUE**	**GET**
IDEPOSIT	**BOOT**	**LOAD**
RUN	**SAVE**	**DUMP**
GO	**RESTORE**	**!**
STEP		

Limited editing is provided at the ***scp>*** prompt. Pressing BACKSPACE deletes the last character entered, and pressing ESCAPE clears all characters. Pressing ENTER with no characters present exits command mode and returns keyboard control to the system console.

Pressing CTRL+E while at the ***scp>*** prompt stops simulation:

```
:[CTRL+E]
scp> [CTRL+E]

Simulation stopped, P: 071144 (PAUS 0)
sim>
```

That is, pressing CTRL+E twice in concurrent mode is equivalent to pressing it once in non-concurrent mode.

Concurrent mode is enabled by default. If the previous, non-concurrent mode behavior is desired, a ***SET CONSOLE NOCONCURRENT*** command may be placed in the global startup file; see the *Global Initialization File* section below for details.

2.13.4 SET CONSOLE SERIAL

The new ***SET CONSOLE SERIAL*** command provides an alternate connection for the system console when it is separated from the simulation console. It complements the existing ***SET CONSOLE TELNET*** command.

For convenience and by default, the system console is connected to the simulation console, so that SCP and HP operating system commands may be entered from the same window. However, the system console may be separated from the simulation console by using the ***SET CONSOLE TELNET=<port>*** or ***SET CONSOLE SERIAL=<port>*** command. This leaves the simulation console at the initiating window and moves the system console to a Telnet or serial port, respectively, allowing the use of an HP terminal or terminal emulator. Entering the existing ***SET CONSOLE NOTELNET*** or new ***SET CONSOLE NOSERIAL*** command will rejoin the consoles.

The command syntax is:

```
SET CONSOLE SERIAL=<port-name>{;<rate>-<size><parity><stopbits>}
```

The parameters to this command are identical to those specified for a serial ***ATTACH*** command, as described above.

2.14 DO Command

The sense of the existing ***–E*** switch of the ***DO*** command has been reversed. In the absence of ***–E***, command file execution will now abort if a command returns an error. For example, a command file containing an ***ATTACH –E*** command specifying a file that does not exist will fail with a **File open error** message, and command file execution will stop. If the ***DO*** command specifies ***–E***, however, then the command will fail but command file execution will continue. With this reversal, an explicit override is now required if a command file is to continue regardless of errors occurring.

The ***DO*** command is also extended to accept a new ***–A*** switch. In the absence of ***–A***, the **Breakpoint** and **Step expired** messages that normally result from ***BREAK***, ***GO UNTIL***, and ***STEP*** command completions are suppressed, as are the informational messages from ***ATTACH*** and ***DETACH***, such as **Creating new file**. This provides a cleaner console display log when automated prompts and responses are used. For example, if a command file contains:

```
GO UNTIL "Memory size? " ; ATTACH -N MS0 new.tape ; REPLY "1024\r" ; GO
```

...then running the command file with ***DO*** would display:

```
Memory size? 1024
```

...whereas using ***DO –A*** would display:

```
Memory size?
Breakpoint, P: 37305 (CLF 10)
MS: creating new file
1024
```

The new ***–A*** switch and the existing ***–E*** and ***–V*** switches now propagate to nested ***DO*** command files. For example, invoking a top-level command file with ***DO –V*** will verbosely list not only that file's commands but also the commands within any ***DO*** files invoked therein.

In addition, the ***DO*** command is extended to retry a failed command file open by appending the *.sim* extension to the filename and trying again. That is, a ***DO file*** command will first try to open the command file named ***file*** and then, if that fails, to open the file named ***file.sim***.

2.14.1 DO and SET CONSOLE CONCURRENT

When a ***DO*** command is entered at the concurrent-mode ***scp>*** prompt, it behaves slightly differently from other concurrent-mode commands. First, restricted commands within the ***DO*** file are allowed, so a command file may contain ***GO UNTIL*** and ***REPLY*** commands, for example. However, if the file contains a command such as ***IEXAMINE*** that pauses for user input, the target operating system's time-of-day clock will lose time. Second, when the command file completes, simulation is resumed automatically, so a final ***CONTINUE*** or ***GO*** command need not and should not be present.

A ***DO*** file that must run under both concurrent and non-concurrent modes may use the ***%SIM_RUNNING%*** substitution variable to determine the current mode of execution. See the *Variable Substitution* section below for details.

2.15 ABORT Command

The new ***ABORT*** command may be used within command files to stop execution and return to the SCP prompt. In a nested command-file execution, ***ABORT*** terminates the current command file and all nested invocations. By contrast, the ***RETURN*** command ends the current command file but continues execution of the invoking command file. The syntax is:

```
ABORT
```

If concurrent mode is enabled for the console, ***ABORT*** may be entered at the ***scp>*** prompt to stop simulation and abort command file execution. If simulation execution was initiated from a command file, the simulator will stop, **Command file execution aborted** will be printed, and control will return to the SCP prompt. If execution was initiated from the command line, ***ABORT*** is equivalent to entering CTRL+E to stop simulator execution.

ABORT is useful if a sequence of prompt/response pairs in a command file is not executing properly. Consider a command file containing this sequence of commands:

```
GO UNTIL "OPTION? "   ; REPLY "RELOD\r"
GO UNTIL "CHANGES? "  ; REPLY "YES\r"
GO UNTIL "MEM SIZE? " ; REPLY "128\r"
GO UNTIL "SEG SIZE? " ; REPLY "8192\r"
[...]
```

When the command file replies with the misspelled `RELOAD` option, the executing program prints an error message and repeats the `OPTION?` prompt. The command file, however, is now looking for the `CHANGES?` prompt and so does not supply the `YES` reply. At this point, entering CTRL+E stops simulated execution, but the command file then resumes execution with the next prompt and reply. This continues until enough CTRL+Es are entered to exhaust the set of prompts and responses in the command file.

This may be avoided by entering ***ABORT*** in concurrent mode after the first CTRL+E. This stops the simulator, as well as command file execution at that point.

2.16 Aborting Command File Execution

Improper use of the ***GOTO*** command can lead to command file infinite loops. To abort command file execution in such cases, enter CTRL+C (*not* CTRL+E). For example, executing this command file:

```
:label
GOTO label
```

...with a ***DO*** command will loop forever. Entering CTRL+C will stop execution with **Command file execution aborted**, and control will return to the SCP prompt.

CTRL+C is used because an infinite loop may contain commands that respond to CTRL+E. Using CTRL+C ensures that command file execution, and not an individual command, is aborted.

2.17 FLUSH Command

To improve performance, all simulator log files and most attached device files are buffered. When a simulation stop occurs, buffered files are flushed to the host system disc before returning to the SCP prompt to permit external examination. With concurrent mode enabled by default, most simulation stops are avoided, so files will be only partially written when examined externally. In particular, trace and console logs will be incomplete unless the simulator is stopped and restarted, which defeats the purpose of enabling concurrent mode.

To avoid this, the new ***FLUSH*** command may be entered at the concurrent-mode prompt to force physical disc writes for active files. The command form is:

```
FLUSH
```

For files that are open, the command flushes the console and debug log files, the files attached to all of the units of all devices, and log files associated with terminal multiplexer lines. Flushing does not disturb any of the affected files; it merely ensures that all writes are physically completed and the associated file contents are current.

FLUSH commands are accepted after a simulation stop. However, they have no effect, as the stop caused all files to be flushed automatically.

2.18 SHOW Command

The existing ***SHOW BREAK*** command is enhanced to display string breakpoints as well as address breakpoints. String breakpoints are identified by the console output unit name in the display. For example:

```
sim> BREAK -T "hello" DELAY 6
sim> BREAK -S 00100
sim> SHOW BREAK
TTY1:   T "hello" delay 6
100:    S
sim>
```

The new ***SHOW REPLY*** command displays any pending reply. The reply is displayed with the console input unit name. For example:

```
sim> REPLY "hi" DELAY 100
sim> SHOW REPLY
TTY0:   "hi" delay 100
sim>
```

The new ***SHOW DELAYS*** command displays the current default break and reply delays:

```
sim> SHOW DELAYS
Break delay = 1000
Reply delay = 0
sim>
```

The delays are expressed as event tick counts and will be zero unless they have been changed with ***BREAK DELAY*** or ***REPLY DELAY*** commands, respectively.

The existing ***SHOW CONSOLE*** command is enhanced to display the concurrent mode setting. If the console is connected to a serial port, the ***SHOW CONSOLE*** and ***SHOW CONSOLE SERIAL*** commands display the serial connection status.

2.19 Variable Substitution

Each command line is scanned for predefined and user-defined variables, which are replaced with their corresponding values. Variables are bracketed by percent signs (%). The predefined variables are:

Variable Name	*Description*
DATE_YYYY	The current year, 0000-9999
DATE_YY	The current year, 00-99
DATE_MM	The current month, 01-12
DATE_MMM	The current month, Jan-Dec
DATE_DD	The current day of the month, 01-31
DATE_JJJ	The current (Julian) day of the year, 001-366
DATE_RRRR	A year with the same calendar days as the current year, rescaled to 1972-1999
DATE_RR	A year with the same calendar days as the current year, rescaled to 72-99
TIME_HH	The current hour, 00-23
TIME_MM	The current minute, 00-59
TIME_SS	The current second, 00-59
SIM_MAJOR	The simulator major version number
SIM_NAME	The name of the simulator
SIM_EXEC	The path and name of the simulator executable file
SIM_RUNNING	A non-zero value if the simulator is executing instructions

The date and time variables are useful when setting the system clock after operating system startup. For example:

```
GO UNTIL "DATE?" ; REPLY "%DATE_MM%/%DATE_DD%/%DATE_YY%\r" ; GO
```

If the target operating system is not year-2000 compliant, the ***%DATE_RRRR%*** variable will give a year between 1972 and 1999 that has the same calendar days as the current year. For example, if the current date is Monday, February 29, 2016, ***%DATE_RRRR%*** will yield 1988, as February 29, 1988 is also a Monday.

The ***%SIM_MAJOR%*** variable may be used to ensure that the commands in a ***DO*** file are available with the current version. For example:

```
IF "%SIM_MAJOR%" == "3" GOTO ok
ECHO This command file requires SCP version 3.
ABORT

:ok
```

The ***%SIM_NAME%*** variable contains the name of the simulator as it appears in the ***SHOW VERSION*** command. It may be used to condition parts of a generic ***DO*** command file for specific simulators.

The ***%SIM_EXEC%*** variable contains the path and name of the executable file used to start the simulator. It may be used to start a second copy of the simulator, e.g., to start the I/O processor instance for the dual-CPU HP 2000 Time-Shared BASIC system:

```
; Start the IOP.

! %SIM_EXEC% iop.sim
```

The ***%SIM_RUNNING%*** variable may be used within ***DO*** command files that are invoked both when the simulator is stopped and when it is executing in concurrent mode. If ***%SIM_RUNNING%*** has the value 0, a ***CONTINUE*** or ***GO*** command is required to resume simulator execution. If the value is 1, the simulator is already running, and the ***CONTINUE*** or ***GO*** command must be omitted.

In addition to the predefined variables, environment variables, including those established by the ***SET ENVIRONMENT*** command, may be specified for substitution. If an undefined variable is specified, the resulting substitution value is a null string. For example:

```
sim> ECHO Hel%UNDEFINED%lo
Hello
```

For commands that take action lists, such as ***BREAK*** and ***IF***, variables in the action list are substituted when the command is entered, as well as when the action occurs. So, for example:

```
sim> SET ENV REG=A
sim> BREAK 10 ; EXAMINE %REG%
sim> SET ENV REG=B
sim> GO

Breakpoint, P: 00010 (NOP)
sim> EXAMINE A
A:      000000
sim>
```

Register A is displayed because the variable substitution was performed when the ***BREAK*** command was entered. To defer evaluation until the action occurs, surround the variable with double percent signs:

```
sim> SET ENV REG=A
sim> BREAK 10 ; EXAMINE %%REG%%
sim> SET ENV REG=B
sim> GO

Breakpoint, P: 00010 (NOP)
sim> EXAMINE %REG%
B:      000000
sim>
```

BREAK command entry processing substitutes single percent signs for the double percent signs before the action is stored. When the action is executed, the resulting percent-enclosed variable is substituted.

2.20 Global Initialization File

During simulator startup, a new global initialization command file named *simh.ini* is executed if it is present. The commands in this file are executed before those in the simulator-specific initialization file (e.g., *hp3000.ini*) or the startup file specified on the command line that invokes the simulator. Commands in this file may be used to change global simulator behavior; for example:

```
SET CONSOLE NOCONCURRENT
SET CONSOLE DEL=177
```

The search for this file begins in the current directory. If the file is not found there, then the directory specified by the *HOME* environment variable is searched (or, if *HOME* is not defined, then the directory specified by the *USERPROFILE* variable is searched).

If a startup filename and parameters were passed on the simulator invocation command line, then they will be available via the standard substitution variables ***%1*** through ***%9***, where ***%1*** will be the startup filename, ***%2*** will be its first parameter, etc. If ***%1*** is null, then no startup filename was passed, and the simulator-specific initialization file will be executed after the global file completes.

Commands within the global file may be qualified for a specific simulator by using the ***IF*** command with the ***%SIM_NAME%*** substitution variable as a condition. For example:

```
IF "%SIM_NAME%" == "HP 3000" SET CPU STOP=LOOP
IF "%SIM_NAME%" == "HP 2100" SET CPU STOP=UNSC
```

Normally, the simulator-specific initialization file would be a better location for such commands. However, note that this file is not executed if a command file is specified when invoking the simulator. Placing the commands in the global file ensures that they are executed regardless of the manner in which the simulator is started.

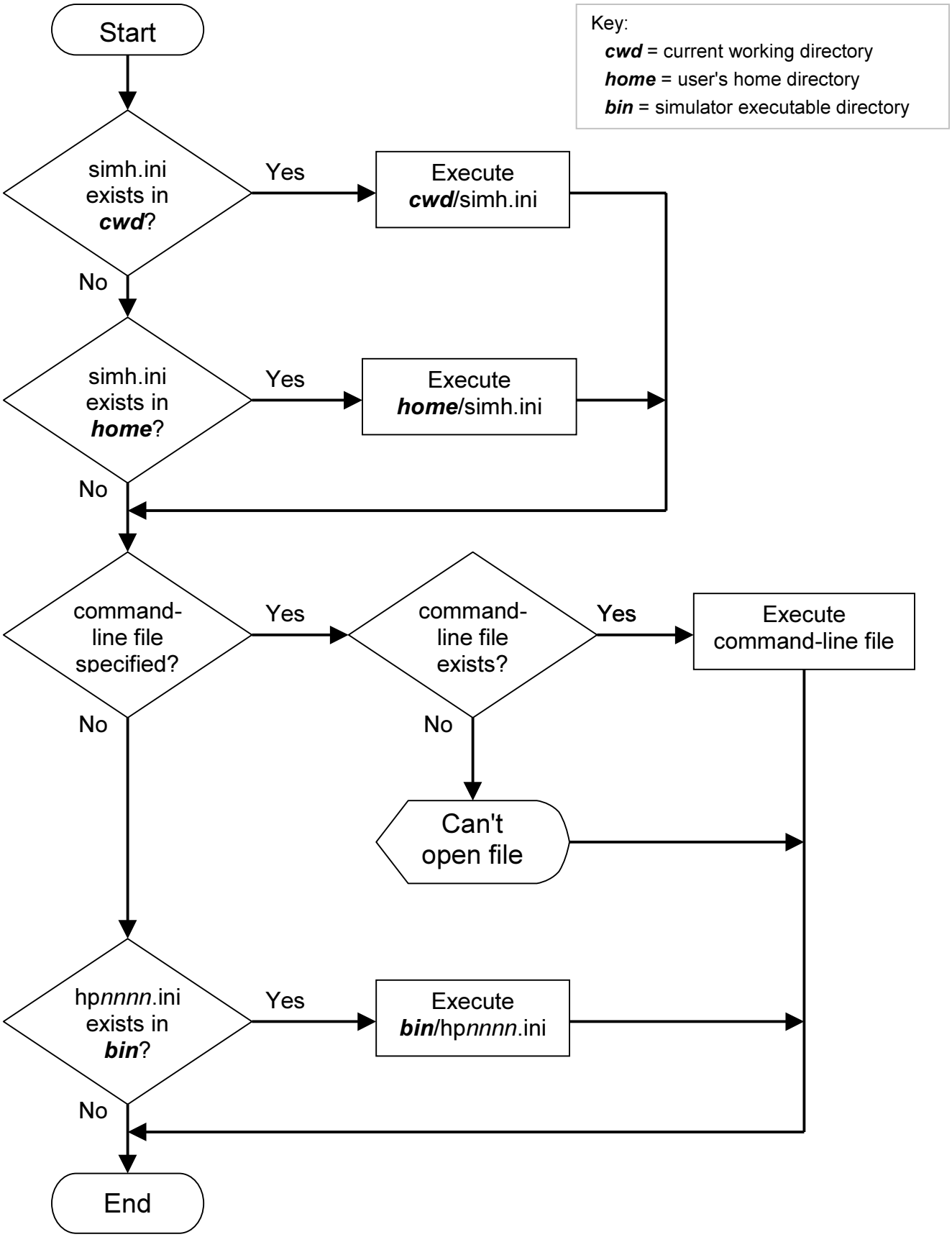
To summarize, at simulator startup:

1. If the *simh.ini* global initialization file exists in the current working directory or the directory indicated by the *HOME* (or *USERPROFILE*) environment variable, it is executed.

2. If a filename was specified on the invocation command line, then if the file exists, it is executed; otherwise, the **Can't open file** message is printed. If no filename was specified, then if the simulator-specific initialization file exists in the directory containing the simulator executable, it is executed.

This logic is expressed in the flowchart on the following page.

As consequences of the search order, an empty *simh.ini* file in the working directory will override one in the *HOME* directory, and specifying a command file of *NUL* (or */dev/null*, etc.) when invoking the simulator will override execution of the simulator-specific initialization file.



Startup Command File Execution Sequence

3 SCP 4.x-to-3.x Conversion

Simulated machine instruction execution operates identically, regardless of the SCP version. However, command files that were written for SCP 4.x may have to be altered to run on SCP 3.x. The latter omits some 4.x commands and uses different syntax for others. These considerations are described below.

3.1 Unimplemented Commands

The following SCP 4.x commands are not implemented (although some have SCP 3.x replacements; see the next section for details):

Command Name(s)	*Description*
EXPECT	Stop execution when specific output is seen
SEND	Insert characters into the input stream
CD	Change the working directory
PWD	Display the working directory
DIR, LS	List the files in the working directory
TYPE, CAT	Display the contents of a file
RM	Delete a file
COPY, CP	Copy a file
SHIFT	Shift the command file parameters
ON	Configure error trapping
PROCEED, IGNORE	Continue command file execution after trapping
ELSE	Execute commands for a false IF conditional
ECHOF	Display text within a quoted string
SLEEP	Suspend simulator execution for a time interval

The following 4.x ***SET*** command options are not implemented:

Option Name(s)	*Description*
REMOTE	Configure a remote console
DEFAULT	Set the current working directory
CLOCK	Set clock calibration parameters
ON, NOON	Enable or disable error trapping
VERIFY, NOVERIFY, VERBOSE, NOVERBOSE	Enable or disable command file execution display
MESSAGE, NOMESSAGE	Enable or disable command file execution error display
QUIET, NOQUIET	Enable or disable miscellaneous information messages
PROMPT	Change the SCP prompt

The following 4.x ***SHOW*** command options are not implemented:

Option Name(s)	*Description*
FEATURES	Display the simulator device feature descriptions
REMOTE	Display the remote console configuration
DEFAULT	Display the current working directory
ON	Display enabled error trapping actions
SERIAL	Display the available host serial ports
MULTIPLEXER	Display open multiplexer device information
EXPECT	Display pending EXPECT commands
SEND	Display pending SEND commands

3.2 SCP 3.x Replacements

Several of the SCP 4.x commands and ***SET/SHOW*** command options are replaced by equivalent or alternate 3.x commands. These are described below.

3.2.1 Quoted Strings

Within quoted strings, the unsupported ***\f***, ***\t***, ***\v***, ***\b***, ***\e***, and ***\?*** escapes, as well as ***\xnn*** hexadecimal escapes are replaced with their equivalent ***\nnn*** octal escapes.

3.2.2 EXPECT and SHOW EXPECT

The ***EXPECT*** command is replaced with string breakpoints, and the ***HALTAFTER=*** option is replaced with ***DELAY***. For example, these commands are equivalent:

```
[4.x] EXPECT HALTAFTER=5000 "Memory size?"

[3.x] BREAK -T "Memory size?" DELAY 5000
```

Breaking on output to terminal multiplexer lines or after specified match counts is not currently supported. Multiple concurrent breakpoints for the same string are not supported, so it is not possible to queue up a series of prompts and responses and then have them trigger in sequence. Instead, each prompt and response pair must be interleaved with simulator execution.

It is possible to queue up a sequence of unique prompts, but it is not recommended. Each output character must be compared to all existing string breakpoint match strings, so having multiple existing breakpoints when only one will be triggered is inefficient. A series of ***GO UNTIL*** and ***REPLY*** commands will execute more quickly than the same number of queued ***BREAK –T*** commands. The use of queued breakpoints should be reserved for those situations where an extra prompt might appear. For example, the HP 2000F system will ask for confirmation of the system startup date if the last shutdown was more than four days in the past. In this case, a queued response is appropriate:

```
GO UNTIL "DATE?  " ; REPLY "%DATE_JJJ%/%DATE_YY%\r"
GO UNTIL "TIME?  " ; REPLY "%TIME_HH%%TIME_MM%\r"
BREAK -T "ARE YOU SURE THAT'S TODAY'S DATE? " ; REPLY "YES\r" ; GO
GO UNTIL "READY\r\n"
[...]
```

All string breakpoints are evaluated independently, so if ***BREAK "HE"*** and ***BREAK "HELLO"*** commands have been entered, both breakpoints will trigger in succession if `Hello` is output on the console.

The ***SHOW EXPECT*** command is replaced with ***SHOW BREAK***, which has been enhanced to display pending string breakpoints in addition to address breakpoints, and ***SHOW DELAYS***, which displays the default break delay.

3.2.3 SEND and SHOW SEND

The ***SEND*** command is replaced with ***REPLY***, and the ***AFTER=*** option is replaced with ***DELAY***. For example, these commands are equivalent:

```
[4.x] SEND AFTER=5000 "1024\r"

[3.x] REPLY "1024\r" DELAY 5000
```

Supplying responses to terminal multiplexer lines, specifying time delays between supplied characters with ***DELAY=***, and specifying response delays in microseconds with the ***–T*** switch are not supported. Response delays in microseconds may be specified by including the keyword ***MICROSECONDS*** or ***USECS*** after the ***DELAY*** value, but the delay is a *realistic time* (i.e., time as seen by the program), rather than the *calibrated time* (time as seen by the user) provided by the ***–T*** switch. The default ***REPLY*** delay is zero rather than 1000 machine instructions.

Each ***REPLY*** command replaces the prior one, rather than appending to the prior one as ***SEND*** does. So these commands:

```
SEND "YES\r"
SEND "NO\r"
```

...must be replaced with the single command ***REPLY "YES\rNO\r"***. Simply substituting ***REPLY*** for ***SEND*** above results in ***"NO\r"*** as the only pending reply.

The ***SHOW SEND*** command is replaced with ***SHOW REPLY***, which displays pending replies, and ***SHOW DELAYS***, which displays the default reply delay.

3.2.4 SLEEP

The ***SLEEP*** command is replaced by ***PAUSE***. The ***PAUSE*** time must be an integer rather than a floating-point value, but ***PAUSE*** allows specification of millisecond values. Units of minutes, hours, and days are not supported, but times in these ranges may be specified as the equivalent number of seconds. For example, these commands are equivalent:

```
[4.x] SLEEP 2D

[3.x] PAUSE 172800 SECONDS

[4.x] SLEEP 0.5S

[3.x] PAUSE 500 MILLISECONDS (or MSEC or MS)
```

A ***PAUSE*** may be aborted by entering CTRL+E, while a ***SLEEP*** is aborted by CTRL+C.

3.2.5 Host File System Commands

The ***CD***, ***PWD***, ***DIR***, ***LS***, ***TYPE***, ***CAT***, ***COPY***, and ***CP*** commands may be replaced with host-specific spawn commands. For example, the ***! dir*** command is equivalent to the ***DIR*** command. The supported ***DELETE*** command is equivalent to ***RM***.

3.2.6 IF...ELSE

The effect of ***ELSE*** may be emulated by a second ***IF*** statement testing the opposite condition; for example:

```
IF "%DATE_MM%" == "1" ECHO It is January
ELSE ECHO It is not January
```

...may be replaced with:

```
IF "%DATE_MM%" == "1" ECHO It is January
IF "%DATE_MM%" != "1" ECHO It is not January
```

An ***ELSE*** covering multiple statements may be replaced with a ***GOTO*** to route control to the statements for the false conditions.

3.2.7 SET REMOTE and SHOW REMOTE

Concurrent mode, enabled by the ***SET CONSOLE CONCURRENT*** for the simulation console, is similar to the Single Command Mode of the ***SET REMOTE*** command, in that SCP commands may be entered without stopping simulator execution. However, concurrent mode does not support redirecting the simulation console to a Telnet connection (note that ***SET CONSOLE TELNET*** redirects the *system console*, not the simulation console). SCP 4.x Multiple Command Mode and Master Mode have no SCP 3.x analogs.

The 3.x ***SHOW CONSOLE CONCURRENT*** command is analogous to the 4.x ***SHOW REMOTE*** command.

3.2.8 Miscellaneous Commands

The ***SHIFT***, ***ON***, ***PROCEED***, and ***IGNORE*** commands have no corresponding SCP 3.x analogs.

The ***ECHO*** command may substitute for the ***ECHOF*** command in most cases.

3.3 SCP 3.x Differences

Some SCP commands common to both 3.x and 4.x differ in their implementations. The authoritative sources for 3.x command behaviors are this manual and the *SIMH Users' Guide* manual. Prominent differences are noted below.

3.3.1 Initialization File Search Order

The order in which the search for the global *simh.ini* file proceeds differs from 3.x to 4.x. The 3.x order is the current directory, then either the *HOME* directory or the *USERPROFILE* directory. The 4.x order is either the *HOME* directory or the *HOMEDRIVE*/*HOMEPATH* directory, then the current directory.

3.3.2 Serial Port Names in ATTACH and DETACH Commands

In addition to host-specific serial port names (such as COM1), SCP 4.x accepts equivalent internally generated names of the form ***serN***, where ***N*** is a sequence number identifying one of the host-specific ports. SCP 3.x accepts only those port names provided by the host.

3.3.3 IF Command

SCP 4.x requires parentheses around comparative expressions containing logical ***AND*** and ***OR*** operators. SCP 3.x does not recognize parentheses; including them will result in an **Invalid argument** error.

3.3.4 HELP Command

The SCP 4.x ***HELP*** command provides extensive descriptions of commands and their parameters, often containing information that is not present in the *SIMH Users' Guide, V4.0* manual. In contrast, SCP 3.x provides only one-line command summaries. Thorough command and parameter descriptions are provided in the applicable 3.x users' guides; the online help is intended only as a quick reference.

3.3.5 Predefined Substitution Variables

The set of predefined substitution variables is smaller in SCP 3.x vs. 4.x. In addition, the ***DATE_19XX_YYYY*** and ***DATE_19XX_YY*** variables that provide the current year rescaled to the 20th century are named ***DATE_RRRR*** and ***DATE_RR***, respectively, in 3.x.

3.3.6 Command Option Switches

Option switches for common commands that are present in 4.x but are not documented here or in the *SIMH Users' Guide* manual are not supported. Unsupported switches are ignored.