

# **SIMH Users' Guide, V3.9-0**

## **01-May-2012**

### **COPYRIGHT NOTICE**

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

Original code published in 1993-2012, written by Robert M Supnik  
Copyright (c) 1993-2012, Robert M Supnik

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 ROBERT M SUPNIK 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 name of Robert M Supnik shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from Robert M Supnik.

Introduction .....	3
1 Compiling And Running A Simulator.....	3
1.1 Compiling Under UNIX/Linux/Mac OS-X .....	4
1.2 Compiling Under Windows.....	5
1.2.1 Compiling with Ethernet Support.....	5
1.2.2 Compiling Under MinGW.....	5
1.2.3 Compiling Under Visual C++ .....	5
1.3 Compiling Under OpenVMS.....	6
2 Simulator Conventions.....	6
3 Commands .....	6
3.1 Loading and Saving Programs.....	6
3.2 Saving and Restoring State .....	7
3.3 Resetting Devices.....	7
3.4 Connecting and Disconnecting Devices .....	7
3.5 Examining and Changing State .....	8
3.6 Evaluating Instructions.....	10
3.7 Running A Simulated Program .....	10
3.7.1 Controlling The Simulation Rate.....	10
3.8 Stopping The Simulator .....	11
3.8.1 Simulator Detected Stop Conditions .....	11
3.8.2 User Specified Stop Conditions.....	11
3.8.3 Breakpoints .....	11
3.9 Setting Device Parameters .....	12
3.10 Displaying Parameters and Status.....	12
3.11 Altering The Simulated Configuration .....	13
3.12 Console Options .....	13
3.13 Executing Command Files.....	14
3.13.1 Displaying Arbitrary Text.....	15
3.13.2 Testing Simulator State .....	15
3.14 Executing System Commands.....	15
3.15 Getting Help.....	16
3.16 Controlling Debugging .....	16
3.17 Exiting The Simulator.....	16
Appendix 1: File Representations .....	17
A.1 Hard Disks .....	17
A.2 Floppy Disks.....	17
A.3 Magnetic Tapes.....	17
A.4 Line Printers.....	17
A.5 DECtapes.....	18
Appendix 2: Debug Status .....	19
Revision History (covering Rev 2.0 to Rev 3.5).....	21
Acknowledgements.....	27

# Introduction

This memorandum documents the SIMH simulators. These simulators are freeware; refer to the license terms above for conditions of use. Support is not available. The best way to fix problems or add features is to read and modify the sources yourself. Alternately, you can send Internet mail to simh AT trailing-edge DOT com, but a response is not guaranteed.

The simulators use a common command interface. This memorandum describes the features of the command interface. The details of each simulator are documented in separate, machine-specific memoranda.

## 1 Compiling And Running A Simulator

The simulators have been tested on VAX VMS, Alpha VMS, Alpha UNIX, NetBSD, FreeBSD, OpenBSD, Linux, Solaris, Windows 9x/NT/2000/XP, MacOS X, and OS/2. Porting to other environments will require changes to the operating system dependent code in the SIMH libraries (sim\_fio.c, sim\_timer.c, sim\_console.c, sim\_ether.c, sim\_sock.c).

The simulator sources are provided in a zip archive and are organized hierarchically. Source files for the simulator libraries are in the top level directory; source files for each simulator are in individual subdirectories. Note that the include files in the top level directory are referenced from the subdirectories, without path identifiers. Your build tool needs to search the top level directory for include files not present in the simulator-specific directory, or you will have to copy all files from the subdirectories into the master directory. File manifests for each simulator are given in that simulator's documentation.

The simulators recognize or require a few compile-time #defines:

- The 18b simulators require that the model name be defined as part of the compilation command line (i.e., PDP4 for the PDP-4, PDP7 for the PDP-7, PDP9 for the PDP-9, PDP15 for the PDP-15).
- The PDP-10 and IBM 7094 simulators use 64b integer variables, requiring that USE\_INT64 be defined as part of the compilation command line. Since 64b integer declarations vary, sim\_defs.h has conditional declarations for Windows (\_int64) and Digital UNIX (long). The default is GNU C (long long). If your compiler uses a different convention, you will have to modify sim\_defs.h.
- The PDP-10, PDP-11, and VAX simulators share common peripherals. To distinguish the target system, one of three variables must be defined on the command line: VM\_PDP10 for the PDP-10; VM\_PDP11 for the PDP-11; or VM\_VAX for the VAX.
- The PDP-11, PDP-11, and VAX simulators optionally support Ethernet. To include Ethernet emulation, USE\_NETWORK must be defined as part of the compilation command line. At present, Ethernet support has been tested only on Windows, Linux, NetBSD, OpenBSD, FreeBSD, Solaris, and Alpha VMS, but it should work in any host environment that supports the Pcap library (see the Ethernet readme file).
- The PDP-11 and VAX simulators optionally support disks and serial devices files greater than 2GB. To include large device support, both USE\_INT64 and USE\_ADDR64 must be defined as part of the compilation command line.

- The HP2100 Fast FORTRAN Processor (FFP) and 1000-F CPU options require 64b integer support. Define HAVE\_INT64 (not USE\_INT64) as part of the compilation command line if your host compiler supports 64b integers. On systems without 64b support, the 1000 F-Series CPU will be unavailable, and FFP extended-precision instructions (e.g., XADD) will be disabled; the remainder of the FFP instructions will work normally. There may be some compilation warnings.

To start the simulator, simply type its name. (On VMS the simulators must then be defined as foreign commands in order to be started by name.) The simulator recognizes three command line switches: -q, -v, and -e. If -q is specified, certain informational messages are suppressed. The -v and -e switches pertain only to command files and are described in Section 3.13.

The simulator interprets the arguments on the command line, if any, as the file name and arguments for a DO command:

```
% pdp10 {switches} {<startup file> {arg,arg,...}}
```

If no file is specified on the command line, the simulator looks for a startup file consisting of the simulator name (including its path components) plus the extension .ini. If a startup file is specified, either on the command line or implicitly via the .ini capability, it should contain a series of non-interactive simulator command, one per line. These commands can be used to set up standard parameters, for example, disk sizes.

After initializing its internal structures and processing the startup file (if any), the simulator types out its name and version and then prompts for input with:

```
sim>
```

## 1.1 *Compiling Under UNIX/Linux/Mac OS-X*

The sources originate on a Windows system and have cr-lf at the end of every line. For use on UNIX or Mac, the sources must be converted to UNIX or Mac text conventions. This can be done with the UNZIP utility (unzip -a).

The supplied makefile will compile the simulators for UNIX systems that support the POSIX TERMIOS. The VAX and PDP-11 can be compiled with or without Ethernet support. The makefile will automatically build these simulators with Ethernet support if the necessary libpcap components are available on the system which is doing the building. The recommended libpcap components are those packaged and provided by the host operating system vendor. If your host operating system vendor doesn't distribute libpcap components (all modern ones do), you may be able to use the libpcap which can be downloaded and built from sources from <http://www.tcpdump.org/>.

To compile with or without Ethernet support:

```
gmake {target|ALL|clean}
```

Notes for hand compilation:

- The default UNIX terminal handling model is the POSIX TERMIOS interface, which is supported by Linux, Mac OS/X, and Alpha UNIX. If your UNIX only supports the BSD terminal interface, BSDTTY must be defined as part of the compilation command line.
- The PDP-8, PDP-11, 18b PDP, PDP-10, and Nova simulators use the math library. If your UNIX does not link the math library automatically, you must add -lm to the compilation command line.

Examples:

- PDP-11 under TERMIOS UNIX:

```
% cc -DVM_PDP11 pdp11_*.c scp.c sim_*.c -lm -o pdp11
```

- PDP-9 under TERMIOS UNIX:

```
% cc -DPDP9 pdp18b_*.c scp.c sim_*.c -lm -o pdp9
```

- PDP-10 under BSD terminal UNIX:

```
% cc -DVM_PDP10 -DUSE_INT64 -DBSDTTY pdp10_*.c scp.c sim_*.c -lm -o pdp10
```

## 1.2 Compiling Under Windows

### 1.2.1 Compiling with Ethernet Support

The Windows-specific Ethernet code uses the WinPCAP 4.x package. This package for Windows simulates the libpcap package that is freely available for Unix systems. Building simulators with built-in Ethernet support can be done if the required WinPcap components are available on your build system at compile time. These components are available by downloading the file:

<https://github.com/downloads/simh/simh/windows-build.zip>. This zip file contains a file called 0ReadMe\_Build\_Dependencies.txt which explains how to locate the unpacked zip file contents to build simulators with Ethernet support.

In order for a simulator (built with Ethernet support) to provide working Ethernet functionality WinPCAP must be installed on the system. In order to install the WinPcap package the following should be performed:

- Download V4.x from <http://www.winpcap.org>.
- Install the package as directed.

### 1.2.2 Compiling Under MinGW

MinGW (Minimalist GNU for Windows) is a free C compiler available from <http://www.mingw.org>. Msys is a minimal set of Unix utilities to support Mingw, also available from <http://www.mingw.org>. The distribution includes a batch file (build\_mingw.bat) that will build all the simulators from source. By default, the PDP-11 and VAX are built with Ethernet support if the necessary Winpcap components are available at build time

### 1.2.3 Compiling Under Visual C++

Visual C++ requires projects to be defined for each executable which is being built. You can define your own project for each simulator you are interested in, or you can use the predefined project definitions for this simulator release.

Each simulator must be organized as a separate Visual C++ project. Starting from an empty console application,

- Add all the files from the simulator file manifest to the project.
- Open the Project Settings (VC++ 98) or Properties (VC++ .NET) dialog box.
- Under C/C++, Category: General, add any required preprocessor definitions (for example, USE\_INT64).
- Under C/C++, Category: Preprocessor, add the top level simulation directory to the Additional Include Directories. For the VAX and PDP-10, you must also add the PDP-11 directory.
- Under Link, add wsock32.lib and winmm.lib at the end of the list of Object/Module Libraries.

- If you are building the PDP-11 and VAX with Ethernet support, you must also add the WinPCAP libraries (packet.lib, wpcap.lib) to the list of Object/Module libraries.

If you are using Visual C++ .NET, you must turn off /Wp64 (warn about potential 64b incompatibilities) and disable Unicode processing. You will also have to turn off warning 4996 (“deprecated” string functions), or lower the warning level to /W1. Otherwise, the compilations will generate a lot of spurious conversion warnings.

## 1.3 Compiling Under OpenVMS

Compiling on OpenVMS requires DEC C. The simulators that require 64b (PDP-10 and VAX) will not compile on OpenVMS/VAX. The SIMH distribution includes an MMS command file `descrip.mms` that will build all the simulators from source. An example of hand compilation:

- PDP-8 under VMS:

```
$ cc scp.c, sim_*.c, [.pdp8]pdp8*.c
$ link/exec=pdp8 scp.obj, sim_*.obj, [.pdp8]pdp8*.obj
```

## 2 Simulator Conventions

A simulator consists of a series of devices, the first of which is always the CPU. A device consists of named registers and one or more numbered units. Registers correspond to device state, units to device address spaces. Thus, the CPU device might have registers like PC, ION, etc, and a unit corresponding to main memory; a disk device might have registers like BUSY, DONE, etc, and units corresponding to individual disk drives. Except for main memory, device address spaces are simulated as unstructured binary disk files in the host file system. The `SHOW CONFIG` command displays the simulator configuration.

A simulator keeps time in terms of arbitrary units, usually one time unit per instruction executed. Simulated events (such as completion of I/O) are scheduled at some number of time units in the future. The simulator executes synchronously, invoking event processors when simulated events are scheduled to occur. Even asynchronous events, like keyboard input, are handled by polling at synchronous intervals. The `SHOW QUEUE` command displays the simulator event queue.

## 3 Commands

Simulator commands consist of a command verb, optional switches, and optional arguments. Switches take the form:

```
-<letter>{<letter>...}
```

Multiple switches may be specified separately or together: `-abcd` and `-a -b -c -d` are treated identically. Verbs, switches, and other input (except for file names) are case insensitive.

Any command beginning with semicolon (;) is considered a comment and ignored.

### 3.1 Loading and Saving Programs

The `LOAD` command (abbreviation `LO`) loads a file in binary loader format:

```
load <filename> {implementation options}
```

The types of formats supported are implementation specific. Options (such as load within range) are also implementation specific.

The `DUMP` command (abbreviation `DU`) dumps memory in binary loader format:

```
dump <filename> {implementation options}
```

The types of formats supported are implementation specific. Options (such as dump within range) are also implementation specific.

## 3.2 Saving and Restoring State

The `SAVE` command (abbreviation `SA`) save the complete state of the simulator to a file. This includes the contents of main memory and all registers, and the I/O connections of devices except network devices (such as Ethernet controllers and terminal multiplexers):

```
save <filename>
```

The `RESTORE` command (abbreviation `REST`, alternately `GET`) restores a previously saved simulator state:

```
restore <filename>
```

Note: `SAVE` file format compresses zeroes to minimize file size.

## 3.3 Resetting Devices

The `RESET` command (abbreviation `RE`) resets a device or the entire simulator to a predefined condition. If switch `-p` is specified, the device is reset to its power-up state:

```
RESET                reset all devices
RESET -p             powerup all devices
RESET ALL            reset all devices
RESET <device>      reset specified device
```

Typically, `RESET` stops any in-progress I/O operation, clears any interrupt request, and returns the device to a quiescent state. It does not clear main memory or affect I/O connections.

## 3.4 Connecting and Disconnecting Devices

Except for main memory and network devices, units are simulated as unstructured binary disk files in the host file system. Before using a simulated unit, the user must specify the file to be accessed by that unit. The `ATTACH` (abbreviation `AT`) command associates a unit and a file:

```
ATTACH <unit> <filename>
```

If the file does not exist, and the `-e` switch was not specified, a new file is created, and an appropriate message is printed. If the `-e` switch was specified, a new file is not created, and an error message is printed.

If the `-r` switch is specified, or the file is write protected, `ATTACH` tries to open the file read only. If the file does not exist, or the unit does not support read only operation, an error occurs. Input-only devices, such as paper-tape readers, and devices with write lock switches, such as disks and tapes, support read only

operation; other devices do not. If a file is attached read only, its contents can be examined but not modified.

For simulated magnetic tapes, the `ATTACH` command can specify the format of the attached tape image file:

```
ATTACH -f <tape_unit> <format> <filename>
```

The currently supported tape image file formats are:

<code>SIMH</code>	<code>SIMH simulator format</code>
<code>E11</code>	<code>E11 simulator format</code>
<code>TPC</code>	<code>TPC format</code>
<code>P7B</code>	<code>Pierce simulator 7-track format</code>

The tape format can also be set with the `SET` command prior to `ATTACH`:

```
SET <tape_unit> FORMAT=<format>  
ATT <tape_unit> <filename>
```

The format of an attached file can be displayed with the `SHOW` command:

```
SHOW <tape_unit> FORMAT
```

For Telnet-based terminal emulators, the `ATTACH` command associates the master unit with a TCP/IP port:

```
ATTACH <unit> <port>
```

The port is a decimal number between 1 and 65535 that is not used by standard TCP/IP protocols.

For Ethernet emulators, the `ATTACH` command associates the simulated Ethernet with a physical Ethernet device:

```
ATTACH <unit> <physical device name>
```

The `DETACH` (abbreviation `DET`) command breaks the association between a unit and a file, port, or network device:

<code>DETACH ALL</code>	<code>detach all units</code>
<code>DETACH &lt;unit&gt;</code>	<code>detach specified unit</code>

The `EXIT` command performs an automatic `DETACH ALL`.

### ***3.5 Examining and Changing State***

There are four commands to examine and change state:

- `EXAMINE` (abbreviated `E`) examines state
- `DEPOSIT` (abbreviated `D`) changes state
- `IEXAMINE` (interactive examine, abbreviated `IE`) examines state and allows the user to interactively change it
- `IDEPPOSIT` (interactive deposit, abbreviated `ID`) allows the user to interactively change state

All four commands take the form

```
command {modifiers} <object list>
```

Deposit must also include a deposit value at the end of the command.

There are four kinds of modifiers: switches, device/unit name, search specifier, and for EXAMINE, output file. Switches have been described previously. A device/unit name identifies the device and unit whose address space is to be examined or modified. If no device is specified, the CPU (main memory) is selected; if a device but no unit is specified, unit 0 of the device is selected.

The search specifier provides criteria for testing addresses or registers to see if they should be processed. A specifier consists of a logical operator, a relational operator, or both, optionally separated by spaces.

```
{<logical op> <value>} <relational op> <value>
```

where the logical operator is & (and), | (or), or ^ (exclusive or), and the relational operator is = or == (equal), != or != (not equal), >= (greater than or equal), > (greater than), <= (less than or equal), or < (less than). If a logical operator is specified without a relational operator, it is ignored. If a relational operator is specified without a logical operator, no logical operation is performed. All comparisons are unsigned.

The output file modifier redirects command output to a file instead of the console. An output file modifier consists of @ followed by a valid file name.

Modifiers may be specified in any order. If multiple modifiers of the same type are specified, later modifiers override earlier modifiers. Note that if the device/unit name comes after the search specifier, the search values will be interpreted in the radix of the CPU, rather than of the device/unit.

The "object list" consists of one or more of the following, separated by commas:

register	the specified register
register[sub1-sub2]	the specified register array locations, starting at location sub1 up to and including location sub2
register[sub1/length]	the specified register array locations, starting at location sub1 up to but not including sub1+length
register[ALL]	all locations in the specified register array
register1-register2	all the registers starting at register1 up to and including register2
address	the specified location
address1-address2	all locations starting at address1 up to and including address2
address/length	all location starting at address up to but not including address+length
STATE	all registers in the device
ALL	all locations in the unit

Switches can be used to control the format of display information:

-a	display as ASCII
-c	display as character string
-m	display as instruction mnemonics
-o	display as octal
-d	display as decimal
-h	display as hexadecimal

The simulators typically accept symbolic input (see documentation with each simulator).

Examples:

```
ex 1000-1100          examine 1000 to 1100
de PC 1040            set PC to 1040
ie 40-50              interactively examine 40:50
ie >1000 40-50       interactively examine the subset
                      of locations 40:50 that are >1000
ex rx0 50060          examine 50060, RX unit 0
ex rx sbuf[3-6]       examine SBUF[3] to SBUF[6] in RX
de all 0              set main memory to 0
de &77>0 0           set all addresses whose low order
                      bits are non-zero to 0
ex -m @memdump.txt 0-7777  dump memory to file
```

Note: to terminate an interactive command, simply type a bad value (eg, XYZ) when input is requested.

### 3.6 Evaluating Instructions

The `EVAL` command evaluates a symbolic instruction and returns the equivalent numeric value. This is useful for obtaining numeric arguments for a search command:

```
EVAL <expression>
```

### 3.7 Running A Simulated Program

The `RUN` command (abbreviated `RU`) resets all devices, deposits its argument (if given) in the PC, and starts execution. If no argument is given, execution starts at the current PC.

The `GO` command does *not* reset devices, deposits its argument (if given) in the PC, and starts execution. If no argument is given, execution starts at the current PC.

The `CONT` command (abbreviated `CO`) does *not* reset devices and resumes execution at the current PC.

The `STEP` command (abbreviated `S`) resumes execution at the current PC for the number of instructions given by its argument. If no argument is supplied, one instruction is executed.

The `BOOT` command (abbreviated `BO`) resets all devices and bootstraps the device and unit given by its argument. If no unit is supplied, unit 0 is bootstrapped. The specified unit must be attached.

#### 3.7.1 Controlling The Simulation Rate

By default, the simulator runs as fast as possible (although at lower than normal priority) and will consume all available processing resources on the host system. This will raise power consumption (and the operating temperature) of many PC's, and drain the battery of a laptop.

The `SET THROTTLE` command allows the user to reduce the effective execution rate to a specified number of instructions per second, or to a specified percentage of total host computing time:

```
SET THROTTLE xM      set execution rate to x mips
SET THROTTLE xK      set execution rate to x kips
SET THROTTLE x%      limit simulator to x% of host time
```

Throttling is only available on host systems that implement a precision real-time delay function.

The `SET NOTHROTTLE` command turns off throttling. The `SHOW THROTTLE` command shows the current settings for throttling.

Some simulators implement a different form of resource management called idling. Idling suspends simulated execution whenever the program running on the simulator is doing nothing, and runs the simulator at full speed when there is work to do. Throttling and idling are mutually exclusive.

## 3.8 Stopping The Simulator

Programs run until the simulator detects an error or stop condition, or until the user forces a stop condition.

### 3.8.1 Simulator Detected Stop Conditions

These simulator-detected conditions stop simulation:

- HALT instruction. If a HALT instruction is decoded, simulation stops.
- Breakpoint. The simulator may support breakpoints (see below).
- I/O error. If an I/O error occurs during simulation of an I/O operation, and the device stop-on-I/O-error flag is set, simulation usually stops.
- Processor condition. Certain processor conditions can stop simulation; these are described with the individual simulators.

### 3.8.2 User Specified Stop Conditions

Typing the interrupt character stops simulation. The interrupt character is defined by the `WRU` (where are you) console option and is initially set to 005 (^E).

### 3.8.3 Breakpoints

A simulator may offer breakpoint capability. A simulator may define breakpoints of different types, identified by letter (for example, E for execution, R for read, W for write, etc). At the moment, most simulators support only E (execution) breakpoints.

Associated with a breakpoint are a count and, optionally, one or more actions. Each time the breakpoint is taken, the associated count is decremented. If the count is less than or equal to 0, the breakpoint occurs; otherwise, it is deferred. When the breakpoint occurs, the optional actions are automatically executed.

A breakpoint is set by the `BREAK` command:

```
BREAK {-types} {<addr range>{[count]}, {addr range...}}{;action;action...}
```

If no type is specified, the simulator-specific default breakpoint type (usually E for execution) is used. If no address range is specified, the current PC is used. As with `EXAMINE` and `DEPOSIT`, an address range may be a single address, a range of addresses low-high, or a relative range of address/length. Examples:

```
BREAK                               set E break at current PC
BREAK -e 200                         set E break at 200
BREAK 2000/2[2]                      set E breaks at 2000,2001 with count = 2
```

```

BREAK 100;EX AC;D MQ 0      set E break at 100 with actions EX AC and
                             D MQ 0
BREAK 100;                  delete action on break at 100

```

Currently set breakpoints can be displayed with the `SHOW BREAK` command:

```
SHOW {-types} BREAK {ALL|<addr range>{,<addr range>...}}
```

Locations with breakpoints of the specified type are displayed.

Finally, breakpoints can be cleared by the `NOBREAK` command.

### 3.9 Setting Device Parameters

The `SET` command (abbreviated `SE`) changes the status of one or more device parameters:

```
SET <device> <parameter>{=<value>},{<parameter>{=<value>},...}
```

or one or more unit parameters:

```
SET <unit> <parameter>{=<value>},{<parameter>{=<value>},...}
```

Most parameters are simulator and device specific. Disk drives, for example, can usually be set `WRITEENABLED` or write `LOCKED`; if a device supports multiple drive types, the `SET` command can be used to specify the drive type.

All devices recognize the following parameters:

```

OCT          sets the data radix = 8
DEC          sets the data radix = 10
HEX         sets the data radix = 16

```

### 3.10 Displaying Parameters and Status

The `SHOW` command (abbreviated `SH`) displays the status of one or more device parameters:

```
SHOW {<modifiers> <device> <parameter>{=<value>},
      {<parameter>{=<value>},...}
```

or one or more unit parameters:

```
SHOW {<modifiers> <unit> <parameter>{=<value>},
      {<parameter>{=<value>},...}
```

There are two kinds of modifiers: switches and output file. Switches have been described previously. The output file modifier redirects command output to a file instead of the console. An output file modifier consists of `@` followed by a valid file name.

All devices implement parameters `RADIX` (the display radix), `MODIFIERS` (list of valid modifiers), and `NAMES` (logical name). Other device and unit parameters are implementation-specific.

`SHOW` is also used to display global simulation state:

```
SHOW CONFIGURATION          shows the simulator configuration and
```

SHOW DEVICES	the status of all devices and units
SHOW MODIFIERS	shows the simulator configuration
SHOW NAMES	shows all available modifiers
SHOW QUEUE	show all logical names
SHOW TIME	shows the simulator event queue
SHOW VERSION	shows the elapsed time since the last RUN
SHOW <device>	show the simulator version and options
SHOW <unit>	shows the status of the named device
	shows the status of the named unit

SHOW QUEUE and SHOW TIME display time in simulator-specific units; typically, one time unit represents one instruction execution.

### **3.11 Altering The Simulated Configuration**

In most simulators, the SET <device> DISABLED command removes the specified device from the configuration. A DISABLED device is invisible to running programs. The device can still be RESET, but it cannot be ATTACHED, DETACHED, or BOOTED. SET <device> ENABLED restores a disabled device to a configuration.

Most multi-unit devices allow units to be enabled or disabled:

```
SET <unit> ENABLED
SET <unit> DISABLED
```

When a unit is disabled, it will not be displayed by SHOW DEVICE.

The standard device names can be supplemented with logical names. Logical names must be unique within a simulator (that is, they cannot be the same as an existing device name). To assign a logical name to a device:

```
ASSIGN <device> <log-name>    assign log-name to device
```

To remove a logical name:

```
DEASSIGN <device>            remove logical name
```

To show the current logical name assignment:

```
SHOW <device> NAMES          show logical name, if any
```

To show all logical names:

```
SHOW NAMES
```

### **3.12 Console Options**

Console options are controlled by the SET CONSOLE command.

The console terminal normally runs in the controlling window. Optionally, the console terminal can be connected to a Telnet port. This allows systems to emulate a VT100 using the built-in terminal emulation of the Telnet client.

```
SET CONSOLE TELNET=<port>    connect console terminal to Telnet session
```

```
SET CONSOLE NOTELNET      on port
                           disable console Telnet
```

Output to the console can be logged simultaneously to a file:

```
SET CONSOLE LOG=<filename> log console output to file
SET CONSOLE NOLOG         disable logging
```

The console provides a limited key remapping capability:

```
SET CONSOLE WRU=<value>   interpret ASCII code value as WRU
SET CONSOLE BRK=<value>   interpret ASCII code value as BREAK
                           (0 disables)
SET CONSOLE DEL=<value>   interpret ASCII code value as DELETE
SET CONSOLE PCHAR=<value> bit mask of printable characters in range
                           [31, 0]
```

Values are hexadecimal on hex CPU's, octal on all others.

The `SHOW CONSOLE` command displays the current state of console options:

```
SHOW CONSOLE               show all console options
SHOW CONSOLE TELNET        show console Telnet state
SHOW CONSOLE LOG           show console logging state
SHOW CONSOLE WRU           show value assigned to WRU
SHOW CONSOLE BRK           show value assigned to BREAK
SHOW CONSOLE DEL           show value assigned to DELETE
SHOW CONSOLE PCHAR         show value assigned to PCHAR
```

Both `SET CONSOLE` and `SHOW CONSOLE` accept multiple parameters, separated by commas, e.g.,

```
SET CONSOLE WRU=5,DEL=177  set code values for WRU and DEL
```

### **3.13 Executing Command Files**

The simulator can execute command files with the `DO` command:

```
DO <filename> {arguments...} execute commands in file
```

The `DO` command allows command files to contain substitutable arguments. The string `%n`, where `n` is between 1 and 9, is replaced with argument `n` from the `DO` command line. The string `%0` is replaced with `<filename>`. The sequences `\%` and `\\` are replaced with the literal characters `%` and `\`, respectively. Arguments with spaces can be enclosed in matching single or double quotation marks.

If the switch `-v` is specified, the commands in the file are echoed before they are executed.

If the switch `-e` is specified, command processing (including nested command invocations) will be aborted if a command error is encountered. (Simulation stop never abort processing; use `ASSERT` to catch unexpected stops.) Without the switch, all errors except `ASSERT` failures will be ignored, and command processing will continue.

`DO` commands may be nested up to ten invocations deep.

Several commands are particularly useful within command files. While they may be executed interactively, they have only limited functionality when so used.

### 3.13.1 Displaying Arbitrary Text

The `ECHO` command is a useful way of annotating command files. `ECHO` prints out its argument on the console:

```
ECHO <string>                output string to console
```

If there is no argument, `ECHO` prints a blank line on the console. This may be used to provide spacing in the console display or log.

### 3.13.2 Testing Simulator State

The `ASSERT` command tests a simulator state condition and halts command file execution if the condition is false:

```
ASSERT {<dev>} <reg>{<logical-op><value>}<conditional-op><value>
```

If `<dev>` is not specified, CPU is assumed. `<reg>` is a register (scalar or subscripted) belonging to the indicated device. The `<conditional-op>` and optional `<logical-op>` are the same as those used for "search specifiers" by the `EXAMINE` and `DEPOSIT` commands (see above). The `<value>`s are expressed in the radix specified for `<reg>`, not in the radix for the device.

If the `<logical-op>` and `<value>` are specified, the target register value is first altered as indicated. The result is then compared to the `<value>` via the `<conditional-op>`. If the result is false, an "Assertion failed" message is printed, and any running command file is aborted. Otherwise, the command has no effect.

For example, a command file might be used to bootstrap an operating system that halts after the initial load from disk. The `ASSERT` command is then used to confirm that the load completed successfully by examining the CPU's "A" register for the expected value:

```
; OS bootstrap command file
;
ATTACH DS0 os.disk
BOOT DS
; A register contains error code; 0 = good boot
ASSERT A=0
ATTACH MT0 sys.tape
ATTACH MT1 user.tape
RUN
```

In the example, if the A register is not 0, the "ASSERT A=0" command will be echoed, the command file will be aborted with an "Assertion failed" message. Otherwise, the command file will continue to bring up the operating system.

## 3.14 Executing System Commands

The simulator can execute operating system commands with the `!` (spawn) command:

```
! <host operating system command>
```

If no operating system command is provided, the simulator attempts to launch the host operating system's command shell.

### **3.15 Getting Help**

The `HELP` command prints out information about a command or about all commands:

```
HELP                print all HELP messages
HELP <command>     print HELP for command
```

### **3.16 Controlling Debugging**

Some simulated devices may provide debug printouts to help in diagnosing complicated problems. Debug output may be sent to a variety of places, or disabled entirely:

```
SET CONSOLE DEBUG=STDOUT    direct debug output to stdout
SET CONSOLE DEBUG=STDERR   direct debug output to stderr
SET CONSOLE DEBUG=LOG      direct debug output to log file
SET CONSOLE DEBUG=<filename> direct debug output to file
SET CONSOLE NODEBUG        disable debug output
```

If debug output is enabled, individual devices can be controlled with the `SET` command. If a device has only a single debug flag:

```
SET <device> DEBUG          enable device debug output
SET <device> NODEBUG        disable device debug output
```

If the device has individual, named debug flags:

```
SET <device> DEBUG          enable all debug flags
SET <device> DEBUG=n1;n2;... enable debug flags n1, n2, ...
SET <device> NODEBUG=n1;n2;... disable debug flags n1, n2, ...
SET <device> NODEBUG        disable all debug flags
```

If debug output is directed to `stdout`, it will be intermixed with normal simulator output.

### **3.17 Exiting The Simulator**

`EXIT` (synonyms `QUIT` and `BYE`) returns control to the operating system.

## Appendix 1: File Representations

All file representations are little-endian. On big-endian hosts, the simulator automatically performs any required byte swapping.

### ***A.1 Hard Disks***

Hard disks are represented as unstructured binary files of 16b data items for the 12b and 16b simulators, of 32b data items for the 18b, 24b, and 32b simulators, and 64b for the 36b simulators.

### ***A.2 Floppy Disks***

Floppy disks are represented as unstructured binary files of 8b data items. They are nearly identical to the floppy disk images for Doug Jones' PDP-8 simulator but lack the initial 256 byte header. A utility for converting between the two formats is easily written.

### ***A.3 Magnetic Tapes***

Magnetic tapes are represented as unstructured binary files of 8b data items. Each record starts with a 32b record header, in little endian format. If the record header is not a special header, it is followed by n 8b bytes of data, followed by a repeat of the 32b record header. A 1 in the high order bit of the record header indicates an error in the record. If the byte count is odd, the record is padded to even length; the pad byte is undefined.

Special record headers occur only once and have no data. The currently defined special headers are:

0x00000000	file mark
0xFFFFFFFF	end of medium
0xFFFFFFFFE	erase gap

Magnetic tapes are endian independent and consistent across simulator families. A magnetic tape produced by the Nova simulator will appear to have its 16b words byte swapped if read by the PDP-11 simulator.

SIMH can read and write E11-format magnetic tape images. E11 format differs from SIMH format only for odd-length records; the data portion of E11 records is not padded with an extra byte.

SIMH can read TPC-format magnetic tape images. TPC format uses a 16b record header, with 0x0000 denoting file mark. The record header is not repeated at the end of the record. Odd-length records are padded with an extra byte.

SIMH can read Pierce-format seven-track magnetic tape images. Pierce format uses only 6 data bits, and one parity bit, in each byte. The high order bit indicates start of record. End of file is indicated by a record of one (occasionally two) bytes consisting of code 017 (octal).

### ***A.4 Line Printers***

Line printer output is represented by an ASCII file of lines separated by the newline character. Overprinting is represented by a line ending in return rather than newline.

## **A.5 DECTapes**

DECTapes are structured as fixed length blocks. PDP-1/4/7/9/15 DECTapes use 578 blocks of 256 32b words. Each 32b word contains 18b (6 lines) of data. PDP-11 DECTapes use 578 blocks of 256 16b words. Each 16b word contains 6 lines of data, with 2b omitted. This is compatible with native PDP-11 DECTape dump facilities, and with John Wilson's PUTR Program. PDP-8 DECTapes use 1474 blocks of 129 16b words. Each 16b word contains 12b (4 lines) of data. PDP-8 OS/8 does not use the 129th word of each block, and OS/8 DECTape dumps contain only 128 words per block. A utility, DTOS8CVT.C, is provided to convert OS/8 DECTape dumps to simulator format.

A known issue in DECTape format is that when a block is recorded in one direction and read in the other, the bits in a word are scrambled (to the complement obverse). The PDP-11 deals with this problem by performing an automatic complement obverse on reverse writes and reads. The other systems leave this problem to software.

The simulator represents this difference as follows. On the PDP-11, all data is represented in normal form. Data reads and writes are not direction sensitive; read all and write all are direction sensitive. Real DECTapes that are read forward will generate images with the correct representation of the data.

On the other systems, forward write creates data in normal form, while reverse write creates data in complement obverse form. Forward read (and read all) performs no transformations, while reverse read (and read all) changes data to the complement obverse. Real DECTapes that are read forward will generate data in normal form for blocks written forward, and complement obverse data for blocks written in reverse, corresponding to the simulator format.

## Appendix 2: Debug Status

The debug status of each simulated CPU and device is as follows:

System	PDP-8	PDP-11	Nova	PDP-1	18b	PDP
device						
CPU	y	y	y	y	y	
FPU	n	y	-	-	-	h
EIS/CIS	-	h	-	-	-	
console	y	y	y	y	y	
paper tape	y	y	y	y	y	
card reader	-	y	-	-	-	
line printer	y	y	y	h	y	
clock	y	y	y	-	y	
extra terminal	y	y	y	-	y	
hard disk	y	y	y	-	y	
fixed disk	y	y	h	-	y	
floppy disk	y	y	y	-	-	
drum	-	-	-	h	h	
DECTape	y	y	-	h	y	
mag tape	y	y	y	-	h	
system	1401	2100	PDP-10	H316	MicroVAX	3900
device						
CPU	y	y	y	h	y	
FPU	-	y	y	-	y	
EIS/CIS	-	y	y	-	-	
console	h	y	y	h	y	
paper tape	-	y	h	h	-	
card reader	y	-	-	-	y	
line printer	y	y	y	h	y	
clock	-	y	y	h	y	
extra terminal	-	y	y	-	y	
hard disk	h	y	y	h	y	
fixed disk	-	y	-	h	-	
floppy disk	-	-	h	-	y	
drum	-	y	-	-	-	
DECTape	-	-	-	-	-	
mag tape	y	y	y	h	y	
system	GRI	1620	i16	i32	SDS940	
device						
CPU	h	h	d	y	d	
FPU	-	h	d	y	-	
CIS	-	-	-	-	-	
console	h	h	d	y	h	
paper tape	h	h	d	y	h	
card reader	-	h	-	-	-	
line printer	-	h	d	y	h	
clock	h	-	d	y	n	
extra terminal	-	-	h	y	h	
hard disk	-	h	d	y	h	
fixed disk	-	-	-	-	h	
floppy disk	-	-	d	d	-	

drum	-	-	-	-	h
DECTape	-	-	-	-	-
mag tape	-	-	d	y	h

system	LGP-30	1130	780
device			
CPU	h	y	y
FPU	-	-	y
CIS	-	-	y
console	h	y	y
paper tape	h	-	-
card reader	-	y	y
line printer	-	y	y
clock	-	-	y
extra terminal	-	-	y
hard disk	-	y	y
fixed disk	-	-	-
floppy disk	-	-	y
drum	-	-	-
DECTape	-	-	-
mag tape	-	-	y

legend:           y = runs operating system or sample program  
                   d = runs diagnostics  
                   h = runs hand-generated test cases  
                   n = untested  
                   - = not applicable

## Revision History (covering Rev 2.0 to Rev 3.5)

Starting with Rev 2.7, detailed revision histories can be found in file sim\_rev.h.

Rev 3.5, Sep, 05

- Overhauled sources for readability
- Added VAX-11/780

Rev 3.4, May, 05

- Revised memory interaction model

Rev 3.3, Nov, 04

- Added PDP-11/VAX DHQ11 support
- Added PDP-11/VAX TM02/TM03 support
- Added PDP-11 model-specific emulation support
- Added full VAX support
- Replaced SET ONLINE/OFFLINE with SET ENABLED/DISABLED

Rev 3.2, Apr, 04

- Added LGP-30/LGP-21 simulator
- Added global SHOW modifier capability
- Added global SET DEBUG modifier
- Added global SHOW DEBUG,RADIX,MODIFIERS,NAME modifiers
- Added VAX extended physical memory support (Mark Pizzolato)
- Added VAX RXV21 support
- Revised terminal multiplexer library to support variable number of lines per multiplexer
- Added PDP-15 LT19 support (1-16 terminals)

Rev 3.1, Dec, 03

- Added Alpha/VMS, FreeBSD, Mac OS/X Ethernet library support
- Added Eclipse floating point and interval timer support (from Charles Owen)
- Added PDP-1 parallel drum support
- Added PDP-8 TSC8-75 and TD8E support
- Added H316/516 DMA/DMC, magtape, fixed head disk support
- Added PDP-8, PDP-15, 32b Interdata instruction history support

Rev 3.0, May, 03

- Added logical name support
- Added instruction history support
- Added multiple tape format support
- Added 64b address support
- Added PDP-4 EAE support
- Added PDP-15 FP15 and XVM support

Rev 2.10, Nov, 02

- Added Telnet console capability, removed VT emulation
- Added DO with substitutable arguments (from Brian Knittel)
- Added .ini initialization file (from Hans Pufal)
- Added quiet mode (from Brian Knittel)
- Added ! command (from Mark Pizzolato)
- Added Telnet BREAK support (from Mark Pizzolato)
- Added device enable/disable support
- Added optional simulator hooks for input, output, commands

- Added breakpoint actions
- Added magnetic tape simulation library
- Added PDP-11 KW11P programmable clock
- Added PDP-11 RK611/RK06/RK07 disk
- Added PDP-11/VAX TMSCP tape
- Added PDP-11/VAX DELQA Ethernet support (from David Hittner)
- Added PDP-11/PDP-10 RX211/RX02 floppy disk
- Added PDP-11/VAX autoconfiguration support
- Added PDP-10/PDP-11/VAX variable vector support
- Added PDP-1 DECTape
- Added PDP-1, PDP-4 Type 24 serial drum support
- Added PDP-8 RX28 support
- Added PDP-9 RB09 fixed head disk, LP09 line printer
- Added HP2100 12845A line printer
- Added HP2100 13183 magtape support
- Added HP2100 boot ROM support
- Added HP2100 interprocessor link support
- Added IBM 1620
- Added SDS 940
- Added Interdata 16b and 32b systems
- Added 16b DECTape file format support
- Added support for statically buffered devices
- Added magnetic tape end of medium support
- Added 50/60Hz support to line frequency clocks
- Added 7B/8B support to terminals and multiplexers
- Added BREAK support to terminals and multiplexers

#### Rev 2.9, Jan, 02

- Added circular register arrays
- Replaced ENABLE/DISABLE with SET ENABLED/DISABLED
- Replaced LOG/NOLOG with SET LOG/NOLOG
- Generalized the timer calibration package
- Added additional routines to the multiplexer library
- Added SET DISCONNECT, SHOW STATISTICS commands to multiplexers
- Re-implemented PDP-8 TTX as a unified multiplexer
- Implemented a PC queue in most simulators
- Added VAX simulator
- Added GRI-909 simulator
- Added Peter Schorn's MITS 8080/Z80 simulator
- Added Brian Knittel's IBM 1130 simulator
- Added HP2100 DQ, DR, MS, MUX devices
- Added SET VT/NOVT commands

#### Rev 2.8, Dec, 01

- Added DO command
- Added general breakpoint facility
- Added extended SET/SHOW capability
- Replaced ADD/REMOVE with SET ONLINE/OFFLINE
- Added global register name recognition
- Added unit-based register arrays
- Added Charles Owen's System 3 simulator
- Added PDP-11 I/O bus map
- Added PDP-11/VAX RQDX3
- Added PDP-8 RL8A
- Revised 18b PDP interrupt structure
- Revised directory and documentation structure

- Added support for MINGW environment

#### Rev 2.7, Sep, 01

- Added DZ11 (from Thord Nilson and Art Krewat) to PDP-11, PDP-10
- Added additional terminals to PDP-8
- Added TSS/8 packed character format to PDP-8
- Added sim\_sock and sim\_tmrx libraries
- Added sim\_qcount and simulator exit detach all facilities
- Added Macintosh sim\_sock support (from Peter Schorn)
- Added simulator revision level, SHOW version
- Changed int64/uint64 to t\_int64/t\_uint64 for Windows
- Fixed bug in PDP-11 interrupt acknowledge
- Fixed bugs in PDP-11 TS NXM check, boot code, error status; added extended characteristics and status
- Fixed bug in PDP-11 TC stop, stop all functions
- Fixed receive interrupt while disconnected bug in DZ11
- Fixed multi-unit operation bugs, interrupt bugs in PDP-11 RP, PDP-10 RP, PDP-10 TU
- Fixed carrier detect bug in PDP-11, PDP-10 DZ
- Fixed bug in PDP-8 reset routine
- Fixed conditional in PDP-18b CPU
- Fixed SC = 0 bug in PDP-18b EAE
- Fixed bug in PDP-7 LPT
- Upgraded Nova second terminal to use sim\_tmrx
- Upgraded PDP-18b second terminal to use sim\_tmrx
- Upgraded PDP-11 LTC to full KW11-L
- Removed hack multiple console support

#### Rev 2.6b, Aug, 01

- Added H316/516 simulator
- Added Macintosh support from Louis Chrétien, Peter Schorn, and Ben Supnik
- Added bad block table option to PDP-11 RL, RP
- Removed register in declarations
- Fixed bugs found by Peter Schorn
  - endian error in PDP-10, PDP-11 RP
  - space reverse error in PDP-11 TS
  - symbolic input in 1401
- Fixed bug in PDP-1 RIM loader found by Derek Peschel
- Fixed bug in Nova fixed head disk

#### Rev 2.6a, Jun, 01

- Added PDP-9, PDP-15 API option
- Added PDP-9, PDP-15 second terminal
- Added PDP-10 option for TOPS-20 V4.1 bug fix
- Added PDP-10 FE CTRL-C option for Windows
- Added console logging
- Added multiple console support
- Added comment recognition
- Increased size of string buffers for long path names
- Fixed bug in big-endian I/O found by Dave Conroy
- Fixed DEctape reset in PDP-8, PDP-11, PDP-9/15
- Fixed RIM loader PC handling in PDP-9/15
- Fixed indirect pointers in PDP-10 paging
- Fixed SSC handling in PDP-10 TM02/TU45
- Fixed JMS to non-existent memory in PDP-8
- Fixed error handling on command file

#### Rev 2.6, May, 01

- Added ENABLE/DISABLE devices
- Added SHOW DEVICES
- Added examination/modification of register arrays
- Added PDP-10 simulator
- Added clock autocalibration to SCP, Nova, PDP-8, PDP-11, PDP-18b
- Added PDP-8, PDP-11, PDP-9/15 DECTape
- Added PDP-8 DF32
- Added 4k Disk Monitor boot to PDP-8 RF08 and DF32
- Added PDP-4/7 funny format loader support
- Added extension handling to the PDP-8 and -9/15 loaders
- Added PDP-11 TS11/TSV05
- Added integer interval timer to SCP
- Added filename argument to LOAD/DUMP
- Revised magnetic tape and DECTape bootstraps to rewind before first instruction
- Fixed 3 cycle data break sequence in PDP-8 RF
- Fixed 3 cycle data break sequence in 18b PDP LP, MT, RF
- Fixed CS1.TRE write, CS2.MXF,UPE write, and CS2.UAI in PDP-11 RP
- Fixed 4M memory size definition in PDP-11
- Fixed attach bug in RESTORE
- Fixed detach bug for buffered devices
- Updated copyright notices, fixed comments

#### Rev 2.5a, Dec, 00

- Added CMD flop to HP paper tape and line printer
- Added status input for HP paper tape punch and TTY
- Added Charles Owen's 1401 mag tape boot routine
- Added Bruce Ray's Nova plotter and second terminal modules
- Added Charles Owen's Eclipse CPU support
- Added PDP-9/PDP-15 RIM/BIN loader support
- Added PDP-9/PDP-15 extend/bank initial state registers
- Added PDP-9/PDP-15 half/full duplex support
- Moved software documentation to a separate file
- Fixed SCP handling of devices without units
- Fixed FLG, FBF initialization in many HP peripherals
- Fixed 1401 bugs found by Charles Owen
  - 4, 7 char NOPs are legal
  - 1 char B is chained BCE
  - MCE moves whole character, not digit, after first
- Fixed Nova bugs found by Bruce Ray
  - traps implemented on Nova 3 as well as Nova 4
  - DIV and DIVS 0/0 set carry
  - RETN sets SP from FP at outset
  - IORST does not clear carry
  - Nova 4 implements two undocumented instructions
- Fixed bugs in 18b PDP's
  - XCT indirect address calculation
  - missing index instructions in PDP-15
  - bank mode handling in PDP-15

#### Rev 2.5, Nov, 00

- Removed Digital and Compaq from copyrights, as authorized by Compaq Sr VP Bill Strecker
- Revised save/restore format for 64b simulators
- Added examine to file

- Added unsigned integer data types to sim\_defs
- Added Nova 3 and Nova 4 instructions to Nova CPU
- Added HP2100
- Fixed indirect loop through autoinc/dec in Nova CPU
- Fixed MDV enabled test in Nova CPU

#### Rev 2.4, Jan, 99

- Placed all sources under X11-like open source license
- Added DUMP command, revised sim\_load interface
- Added SHOW MODIFIERS command
- Revised magnetic tape format to include record error flag
- Fixed 64b problems in SCP
- Fixed big endian problem in PDP-11 bad block routine
- Fixed interrupt on error bug in PDP-11 RP/RM disks
- Fixed ROL/ROR inversion in PDP-11 symbolic routines

#### Rev 2.3d, Sep, 98

- Added BeOS support
- Added radix commands and switches
- Added PDP-11 CIS support
- Added RT11 V5.3 to distribution kits
- Fixed "shift 32" bugs in SCP, PDP-11 floating point
- Fixed bug in PDP-11 paper tape reader
- Fixed bug in ^D handling

#### Rev 2.3c, May, 98

- Fixed bug in PDP-11 DIV overflow check
- Fixed bugs in PDP-11 magnetic tape bootstrap
- Fixed bug in PDP-11 magnetic tape unit select
- Replaced UNIX V7 disk images

#### Rev 2.3b, May, 98

- Added switch recognition to all simulator commands
- Added RIM loader to PDP-8 paper tape reader and loader
- Added second block bootstrap to PDP-11 magnetic tape
- Fixed bug in PDP-8 RF bootstrap
- Fixed bug in PDP-11 symbolic display
- Fixed bugs in PDP-11 floating point (LDEXP, STEXP, MODf, STCfi, overflow handling)

#### Rev 2.3a, Nov, 97

- Added search capability
- Added bad block table command to PDP-11 disks
- Added bootstrap to PDP-11 magnetic tape
- Added additional Nova moving head disks
- Added RT-11 sample software
- Fixed bugs in PDP-11 RM/RP disks
- Fixed bugs in Nova moving head disks
- Fixed endian dependence in 18b PDP RIM loader

#### Rev 2.3, Mar, 97

- Added PDP-11 RP
- Added PDP-1
- Changed UNIX terminal I/O to TERMIOS
- Changed magnetic tape format to double ended
- Changed PDP-8 current page mnemonic from T to C
- Added endian independent I/O routines

- Added precise integer data types
- Fixed bug in sim\_poll\_kbd
- Fixed bug in PDP-8 binary loader
- Fixed bugs in TM11 magnetic tape
- Fixed bug in RX11 bootstrap
- Fixed bug in 18b PDP ADD
- Fixed bug in 18b PDP paper tape reader
- Fixed bug in PDP-4 console
- Fixed bug in PDP-4, 7 line printer

#### Rev 2.2d, Dec, 96

- Added ADD/REMOVE commands
- Added unit enable/disable support to device simulators
- Added features for IBM 1401 project
- Added switch recognition for symbolic input
- Fixed bug in variable length IEXAMINE
- Fixed LCD bug in RX8E
- Initial changes for Win32
- Added IBM 1401

#### Rev 2.2b, Apr, 96

- Added PDP-11 dynamic memory size support

#### Rev 2.2a, Feb, 96

- New endian independent magnetic tape format

#### Rev 2.2 Jan, 96

- Added register buffers for save/restore
- Added 18b PDP's
- Guaranteed TTI, CLK times are non-zero
- Fixed breakpoint/RUN interaction bug
- Fixed magnetic tape backspace to EOF bug
- Fixed ISZ/DCA inversion in PDP-8 symbol table
- Fixed sixbit conversion in PDP-8 examine/deposit
- Fixed origin increment bug in PDP-11 binary loader
- Fixed GCC longjmp optimization bug in PDP-11 CPU
- Fixed unit number calculation bug in SCP and in Nova, PDP-11, 18b PDP moving head disks

#### Rev 2.1 Dec, 95

- Fixed PTR bug (setting done on EOF) in PDP-8, Nova
- Fixed RX bug (setting error on INIT if drive 1 is not attached) in PDP-8, PDP-11
- Fixed RF treatment of photocell flag in PDP-8
- Fixed autosize bug (always chose smallest disk if new file) in PDP-11, Nova
- Fixed not attached bug (reported as not attachable) in most mass storage devices
- Fixed Nova boot ROMs
- Fixed bug in RESTORE (didn't requeue if delay = 0)
- Fixed bug in RESTORE (clobbered device position)
- Declared static constant arrays as static const
- Added PDP-8, Nova magnetic tape simulators
- Added Dasher mode to Nova terminal simulator
- Added LINUX support

#### Rev 2.0 May, 95

- Added symbolic assembly/disassembly

# Acknowledgements

SIMH would not have been possible without help from around the world. I would like to acknowledge the help of the following people, all of whom donated their time and talent to this "computer archaeology" project:

Bill Ackerman	PDP-1 consulting
Alan Bawden	ITS consulting
Winfried Bergmann	Linux port testing
Phil Budne	Solaris port testing
Max Burnet	PDP information, documentation, and software
Robert Alan Byer	VMS socket support and testing
James Carpenter	LINUX port testing
Chip Charlot	PDP-11 RT-11, RSTS/E, RSX-11M legal permissions
Louis Chrétien	Macintosh porting
Dave Conroy	HP 21xx documentation, PDP-10, PDP-18b debugging
L Peter Deutsch	PDP-1 LISP software
Ethan Dicks	PDP-11 2.9 BSD debugging
John Dundas	PDP-11 CPU debugging, programmable clock simulator
Jonathan Engdahl	PDP-11 device debugging
Carl Friend	Nova and Interdata documentation, and RDOS software
Megan Gentry	PDP-11 integer debugging
Dave Gesswein	PDP-8 and PDP-9.15 documentation, PDP-8 DECtape, disk, and paper-tape images, PDP-9/15 DECtape images
Dick Greeley	PDP-8 OS/8 and PDP-10 TOPS-10/20 legal permissions
Gordon Greene	PDP-1 LISP machine readable source
Lynne Grettum	PDP-11 RT-11, RSTS/E, RSX-11M legal permissions
Franc Grootjen	PDP-11 2.11 BSD debugging
Doug Gwyn	Portability debugging
Kevin Handy	TS11/TSV05 documentation, make file
Ken Harrenstein	KLH PDP-10 simulator
Bill Haygood	PDP-8 information, simulator, and software
Wolfgang Helbig	DZ11 implementation
Mark Hittinger	PDP-10 debugging
Dave Hittner	SCP debugging, DEQNA emulator and Ethernet library
Sellam Ismail	GRI-909 documentation
Jay Jaeger	IBM 1401 consulting
Doug Jones	PDP-8 information, simulator, and software
Brian Knittel	IBM 1130 simulator, SCP extensions for GUI support
Al Kossow	HP 21xx, Varian 620, TI 990, Interdata, DEC documentation and software
Arthur Krewat	DZ11 changes for the PDP-10
Mirian Crzig Lennox	ITS and DZ11 debugging
Don Lewine	Nova documentation and legal permissions
Tim Litt	PDP-10 hardware documentation and schematics, tape images, and software sources
Tim Markson	DZ11 debugging
Bill McDermith	HP 2100 debugging, 12565A simulator
Scott McGregor	PDP-11 UNIX legal permissions
Jeff Moffatt	HP 2100 information, documentation, and software
Alec Muffett	Solaris port testing
Terry Newton	HP 21MX debugging
Thord Nilson	DZ11 implementation
Charles Owen	Nova moving head disk debugging, Altair simulator, Eclipse simulator, IBM System 3 simulator, IBM 1401 diagnostics, debugging, and magtape boot

Sergio Pedraja	MINGW environment debugging
Derek Peschel	PDP-10 debugging
Paul Pierce	IBM 1401 diagnostics, media recovery
Mark Pizzolato	SCP, Ethernet, and VAX simulator improvements
Hans Pufal	PDP-10 debugging, PDP-15 bootstrap, DOS-15 recovery, DOS-15 documentation, PDP-9 restoration
Bruce Ray	Software, documentation, bug fixes, and new devices for the Nova, OS/2 porting
Craig St Clair	DEC documentation
Richard Schedler	Public repository maintenance
Peter Schorn	Macintosh porting
Stephen Schultz	PDP-11 2.11 BSD debugging
Olaf Seibert	NetBSD port testing
Brian & Barry Silverman	PDP-1 simulator and software
Tim Shoppa	Nova documentation, RDOS software, PDP-10 and PDP-11 software archive, hosting for SIMH site
Van Snyder	IBM 1401 zero footprint bootstraps
Michael Somos	PDP-1 debugging
Hans-Michael Stahl	OS/2 port testing, TERMIOS implementation
Tim Stark	TS10 PDP-10 simulator
Larry Stewart	Initial suggestion for the project
Bill Strecker	Permission to revert copyrights
Chris Suddick	PDP-11 floating point debugging
Ben Supnik	Macintosh timing routine
Bob Supnik	SIMH simulators
Ben Thomas	VMS character-by-character I/O routines
Warren Toomey	PDP-11 UNIX software
Deb Toivonen	DEC documentation
Mike Umbricht	DEC documentation, H316 documentation and schematics
Leendert Van Doorn	PDP-11 UNIX V6 debugging, TERMIOS implementation
Fred Van Kempen	Ethernet code, RK611 emulator, PDP-11 debugging, VAX/Ultrix debugging
Holger Veit	OS/2 socket support
David Waks	PDP-8 ESI-X and PDP-7 SIM8 software
Tom West	Nova documentation
Adrian Wise	H316 simulator, documentation, and software
John Wilson	PDP-11 simulator and software
Joe Young	RP debugging on Ultrix-11 and BSD

In addition, the following companies have graciously licensed their software at no cost for hobbyist use:

Data General Corporation  
 Digital Equipment Corporation  
 Compaq Computer Corporation  
 Mentec Corporation  
 The Santa Cruz Operation  
 Caldera Corporation  
 Hewlett-Packard Corporation