

***PERQemu* User's Guide**

For *PERQemu* version 0.7.5

S. Boondoggle
22-Apr-2025

Copyright © 2023–2025
Boondoggle Heavy Industries, Ltd.
1060 West Addison
Chicago, IL 60613
(800) 555-1212

The *PERQemu* software was developed by Josh Dersch, Copyright © 2006-2025.

Many of the dubious modifications or enhancements to the software discursively described here were contributed by S. Boondoggle, Boondoggle Heavy Industries, Ltd.

Accent was a trademark of Carnegie-Mellon University. Accent and many of its subsystems and support programs were originally developed by the CMU Computer Science Department as part of its SPICE Project.

PNX was likely a trademark of International Computers, Ltd.

PERQ, PERQ2, PERQ4, LINQ, and Qnix were trademarks of PERQ Systems Corporation.

The information in this document is subject to change without notice and should not be construed as a commitment by Boondoggle Heavy Industries. The company assumes no responsibility for any errors that may appear in this document.

Boondoggle Heavy Industries will make every effort to keep customers apprised of all documentation changes as quickly as possible. A Reader's Comments card will be distributed with this document to request users' critical evaluation to assist us in preparing future documentation.

Table of Contents

1. Introduction	12
Where to Obtain <i>PERQemu</i>	13
Software Requirements	14
System Requirements	14
Host System Recommendations	15
Roadmap and Release History	17
What's New in <i>PERQemu</i> v0.7.5	17
2. The Emulator	18
Getting Started	18
Running <i>PERQemu</i>	19
The Command Line Interface	20
Getting Help	20
Tab Completion	21
Command Arguments	22
Editing and History	24
The Configurator	25
Choosing a Predefined Configuration	25
Creating Custom Configurations	26
Chassis Types	27
Processor Type	28
Configuring Memory	28
I/O Board Types	29
Display and Tablet Options	30
I/O Options	31
Status of I/O Option Development	31
Configuring Sub-options	32
Serial Ports	33
Network Address	33
Disks, Floppies, and Tapes	33
Checking for Errors	34
Naming and Saving Configurations	35
Working with Storage Devices	36

Supported Devices	36
Unit Numbers	37
The Disks Directory	38
Supported Media Formats	38
Loading, Saving, and Unloading Media	39
Showing Storage Device Status	40
Changing Formats, Making Backups	41
Creating and Formatting New Media	42
Working with Communication Devices	44
Serial Devices	44
The RSX: Pseudo-device	46
“Speech” Output	46
Ethernet	46
Encapsulation Options and Administrative Privileges	48
Network Address Translation and the PERQemu Virtual Hub	48
Current Problems and Limitations	48
Working with Output Devices	50
Canon Laser Printers	50
Canon Runtime Options	51
Multi-page Documents	52
Managing the Virtual Machine	53
Starting and Stopping the PERQ	53
The Diagnostic Display (DDS)	55
Boot Selection	56
Bootstrapping in Depth	57
Automating Actions with Scripts	59
Limitations and Implementation Notes	59
Setting User Preferences	60
Autosave Options	60
Device Assignment Options	60
Performance Options	60
Miscellaneous Settings	62
Debugging Features	62
3. PERQ Operations	63

Virtual Machine Interface	63
Special Keys	63
Mouse Input	64
Button Mapping	65
Floppy Formats	65
Hard Disk Formats	66
Tape Formats	67
Supported Operating Systems	68
POS	69
Availability	69
Configuration	70
Login and User Interface	70
File Transfer via RSX:	72
File Transfer using RS232	73
File Transfer using Floppies	73
RT-11 Floppy Limitations	73
File Transfer using Tapes	74
Laser Printing	74
Logout and Shutdown	74
MPOS	75
Availability	75
Configuration	75
Login and User Interface	75
Logout and Shutdown	75
Accent	76
Availability	76
Configuration	76
Login and User Interface	77
Using the Accent Shell	79
Laser Printing	79
Logout and Shutdown	79
Limitations	80
Interim Ethernet Support	80
Live Ethernet Support	80

PNX	82
Availability	82
Configuration	82
Login and User Interface	83
Logout and Shutdown	85
Future Possibilities	86
More to come...	86
4. Command Reference	87
Top-level Commands	87
about	87
bootchar [<i>char</i>]	87
configure [...]	87
debug [...]	87
go	87
gui	87
help [<i>keyword</i>]	88
history	88
load (floppy harddisk tape) <i>filename</i> [<i>unit</i>]	88
load paper [<i>size</i>]	88
power (on off)	88
quit	88
quit [without save]	88
reset	88
save (floppy harddisk tape) [as <i>filename</i> [<i>format</i>]]	89
save history	89
save screenshot [<i>filename</i>]	89
settings [...]	89
start	89
status	89
stop	89
storage [...]	89
unload floppy	89
unload harddisk [<i>unit</i>]	90
unload tape	90

Configuration Commands	91
assign <i>filename</i> [<i>unit</i>]	91
chassis <i>type</i>	91
check	91
cpu <i>type</i>	91
default	91
description <i>string</i>	91
disable rs232 [<i>port</i>]	91
display (portrait landscape)	91
enable rs232 [<i>port</i>]	92
ethernet address <i>address</i>	92
io board <i>type</i>	92
list	92
load <i>config</i>	92
memory <i>capacity</i>	92
name <i>string</i>	92
option board <i>type</i>	92
option add <i>option</i>	93
option remove <i>option</i>	93
save	93
show [<i>config</i>]	93
tablet (bitpad kriz both none)	93
unassign <i>filename</i>	93
unassign unit <i>unit</i>	93
Storage Commands	94
create <i>driveType filename</i> [<i>overwrite</i>]	94
define <i>driveClass driveType</i>	94
list	94
show <i>driveType</i>	94
status [<i>unit</i>]	94
Debugging Commands	95
clear breakpoint <i>type watchpoint</i>	95
clear interrupt <i>irq</i>	95
clear logging <i>category</i>	95

disable breakpoints	95
disable console logging	95
disable file logging	96
disassemble microcode [<i>address</i>] [<i>length</i>]	96
dump <i>item</i>	96
edit breakpoint <i>type watchpoint</i>	96
enable breakpoints	96
enable console logging	96
enable file logging	96
find memory <i>address value</i>	96
inst	97
jump <i>address</i>	97
load microcode <i>binfile</i>	97
load qcodes <i>codeset</i>	97
raise interrupt <i>irq</i>	97
reset breakpoints <i>type</i>	97
set breakpoint <i>type watchpoint</i>	97
set console loglevel <i>severity</i>	97
set execution mode (asynchronous synchronous)	98
set file loglevel <i>severity</i>	98
set logging <i>category</i>	98
set memory <i>address value</i>	98
set xy register <i>reg value</i>	98
show	98
show breakpoints [<i>type</i>]	98
show cpu registers	98
show cstack	98
show estack	98
show execution mode	99
show memory <i>address [length]</i>	99
show memstate	99
show opfile	99
show variables	99
show xy register [<i>reg</i>]	99

step	99
z80 [...]	99
Z80 Debugging Commands	100
clear breakpoint <i>type watch</i>	100
dump <i>item</i>	100
edit breakpoint <i>type watch</i>	100
inst	100
reset breakpoints <i>type</i>	100
set breakpoint <i>type watch</i>	100
show breakpoints [<i>type</i>]	100
show memory <i>address</i> [<i>length</i>]	101
show registers	101
top	101
Settings Commands	102
assign audio device <i>hostDevice</i>	102
assign ethernet device <i>hostDevice</i>	102
assign rs232 device <i>port hostDevice</i> [<i>baud</i>] [<i>charBits</i>] [<i>parity</i>] [<i>stopBits</i>]	102
autosave (floppy harddisk tape) (yes no maybe)	102
canon output format (raw png tiff jpeg)	102
canon paper size (nocassette a4 b5 uslegal usletter)	103
canon resolution (240 300)	103
default	103
display cursor (crosshairs defaultarrow none)	103
load	103
output directory <i>dir</i>	103
pause on reset (true false)	103
pause when minimized (true false)	103
rate limit <i>option</i>	103
save	104
show	104
show ethernet devices	104
unassign audio device	104
unassign ethernet device	104
unassign rs232 device <i>port</i>	104

Appendix A: A Brief History of PERQ	105
Appendix B: Bibliography	106
POS Documentation	106
Accent Documentation	106
PNX Documentation	107
Additional Documentation	107
Applications	107
Appendix C: Additional Resources	108
<i>PERQdisk</i> : Filesystem Munger Tool	108
<i>Stut</i> : The Streamer Utility	108
Appendix D: Release History	109
What's New in <i>PERQemu</i> v0.6.5	109
What's New in <i>PERQemu</i> v0.5.8	109
What's New in <i>PERQemu</i> v0.5.5	110
What's New in <i>PERQemu</i> v0.5.0	110

List of Tables

Table 1: Chassis types.	27
Table 2: CPU board types.	28
Table 3: I/O board types.	29
Table 4: I/O Option board types.	31
Table 5: I/O sub-options.	32
Table 6: Special key mapping.	63
Table 7: POS hardware requirements.	70

List of Figures

Figure 1: Console window with diagnostic display.	55
Figure 2: Host to PERQ mouse button mapping.	65
Figure 3: Accent startup screen.	77
Figure 4: PNX 1.3 graphical environment.	84

1. Introduction

PERQemu is a software emulator of the venerable PERQ workstations, graphics computers designed and sold in the early 1980s by Three Rivers Computer Corporation (3RCC) of Pittsburgh, PA. It is intended to provide a *nearly* cycle-accurate hardware emulation such that all software that runs on the PERQ hardware can be run unmodified on the emulator. For a brief background on the PERQ's origins, please see Appendix A.

The following hardware is emulated by *PERQemu*:

- A custom bit-slice, microcoded CPU with 20-bit physical addressing
- 4K or 16K of writable control store (48-bit microinstruction words)
- 256KB to 2MB of RAM (in 16-bit words)
- A high resolution bitmapped display at 768 x 1024 pixels (monochrome)
- Custom RasterOp hardware to accelerate bitmap operations

The original PERQ features a standard set of peripherals:

- A 12- or 24MB Shugart SA4000-series hard disk (14" platters)
- A Shugart SA851 8" double-sided floppy drive
- A GPIB interface, typically used to connect a Summagraphics BitPadOne digitizer tablet
- One programmable RS232 port, up to 9600 baud
- A CVSD chip for audio output

As of this writing, all of the standard peripherals except audio output are emulated.

Optional I/O boards can also be fitted, which provide:

- 3Mbit or 10Mbit Ethernet interfaces
- A laser printer interface (Canon LBP-10, or later Canon CX)
- A quarter inch cartridge tape connection (Archive Sidewinder)

At this time, the streaming tape and Canon laser printer I/O options have been fully implemented. 10Mbit Ethernet is available for experimentation and still is in active development; all three interfaces are still undergoing testing and refinement with additional software support coming soon.

The PERQ-2 series, introduced in 1983, extends the original design in a few significant ways and adds a number of additional I/O options:

- A larger landscape display (1280 x 1024, monochrome)
- Faster, higher capacity disk drives (8" Micropolis, or up to two 5.25" MFM/ST-506)
- A faster I/O board with integrated Ethernet, second RS-232 port, real-time clock
- An extended 24-bit CPU with larger memory addressing range

The v0.7.5 release represents significant progress toward incorporating these additional models into the emulator. While there are a few more esoteric hardware options available, they fall outside the scope of what *PERQemu* can realistically emulate. Full support for the PERQ-1 and PERQ-2 enables the preservation of *five* complete operating systems and the vast majority of surviving software.

Where to Obtain *PERQemu*

For the most up-to-date information, source code and binary downloads, please consult the following Github repositories:

<https://github.com/jdersch/PERQemu>

The master branch and definitive source for *PERQemu*;

<https://github.com/skeezicsb/PERQemu>

Experimental branch and source of never ending amusement for the nerdiest or most hardcore PERQ fanatic.

The basic *PERQemu* packages include a number of hard disk images and software to get started. Two additional sources are:

<https://github.com/skeezicsb/PERQmedia>

A library of curated media files (floppy, hard disk and tape images) that will grow over time; additional materials of historical interest (marketing images, clippings, academic papers and more) may be included as well;

<https://bitsavers.org>

The *excellent* Bitsavers archive is a treasure trove of computing history and also features quite a bit of PERQ software and documentation.

Software Requirements

PERQemu is written in C#, targeting the Microsoft .NET Framework v4.8. For Linux and MacOS X this is roughly equivalent to the Mono 6.12.0.x runtime/SDK. In addition, the *SDL2-CS* package provides a wrapper around the SDL2 media library, while the *Z80dotNet* package provides Z80 CPU emulation. *SharpPcap* is used for host Ethernet connectivity. These are bundled with the *PERQemu* binary distribution or are installed via NuGet when building from source.

The emulator should run on any Windows, Mac or Linux environment that can support the appropriate .NET or Mono runtime environment. You may also need to install a current SDL2 package appropriate for your OS (available for download from libsdl.org).

- ❖ *Note:* For MacOS X installations the pre-packaged SDL2 download that installs in `Library/Frameworks` does *not* work with the SDL2-CS package; it has to find the individual SDL2 and SDL2_image libraries. Building from source or installing with a package manager should drop these into `/usr/local/lib` so Mono can find them.

Windows users will need to install WinPcap (or NPcap) to access Ethernet. [*Installation steps, download links TBD*]

The “skeeziCSB/experiments” branch is developed and tested with Xamarin Studio Community 6.3 on MacOS X 10.13 (High Sierra) and verified on Visual Studio for Mac 2019.

This branch has also been tested for compatibility with MS Visual Studio 2022 on Windows 10, with Linux testing on Ubuntu 20.0.4 LTS¹. Cross-platform inconsistencies are fairly minor and mostly related to things like window handling; in general *PERQemu* runs the same on all three platforms.

System Requirements

PERQemu is a nearly cycle-accurate, register-level emulation of a complex microcoded processor *and* a Z80 subsystem – essentially two emulations running side-by-side – driving a full bitmapped graphics display refreshed 60 frames per second. It requires some serious grunt.

¹ *Linux testing is currently performed using the Windows-generated distribution; there is not currently any documented build method using a Linux-hosted .NET / Mono toolchain.*

To gauge how faithfully your computer is able to emulate the PERQ, the title of the display window updates every few seconds to report the frame rate ("fps") and the average cycle time of the PERQ and Z80 processors. At full speed the PERQ will display 60fps, with the CPU executing at 170ns (~5.88Mhz) and the Z80 at 407ns (2.45Mhz for the PERQ-1) or 250ns (4Mhz for the PERQ-2). The emulator is still in active development and performance tuning is an ongoing concern.

Host System Recommendations

Processor: A “reasonably current” multi-core x64 CPU is the baseline requirement. *PERQemu* is sensitive to host memory speed when running asynchronously, but may be CPU-bound when running synchronously. For reference, without rate limiting enabled a “6th gen” Intel i5-6500 @ 3.3Ghz with 2133Mhz memory achieves 80-90fps at ~ 70% system utilization, while a 2008 Mac Pro (8-core 2.8Ghz Xeon, 800Mhz memory) tops out around 35fps. YMMV.

No testing has been done on Apple’s ARM-based silicon or other non-x64 processors. Donations gladly accepted!

Memory, disk space: negligible, by modern standards. The running emulator uses about 100MB of RAM with a typical PERQ-1/Shugart configuration loaded, to 300MB for a large PERQ-2/MFM machine (varies depending on the size of the loaded hard drive). Approximately 200MB of disk space is required for the source tree, or 50MB for the binary distribution with the standard library of bundled disk images.

Display: A typical 1920 x 1080 display is generally adequate to host the PERQ’s displays without clipping or scrolling. *PERQemu* doesn’t currently support a resizable or scrollable window or any full-screen modes. It’s advisable to set the appropriate setting to automatically hide the dock/taskbar in your operating environment so that the bottom of the PERQ window isn’t obscured.

- ❖ *Note:* Currently *PERQemu* expects the graphics window to be large enough to show the 1024-line PERQ display without clipping or scaling. It looks pretty bad, otherwise. The old WinForms version used the mouse wheel or PGUP/PGDN keys to scroll the PERQ’s display on small screens; we may try to bring that back if necessary. There has been no testing on high-DPI monitors.

At full speed the PERQ refreshes its display at 60 frames/second, which should be supported by all modern displays.

Keyboard, mouse: Any standard input devices supported by SDL2 should work. The PERQ does have some special keys that we try to emulate with function keys (but this is not well tested and needs to be more adaptable/configurable). It supports a 3-button mouse or a 4-button “puck” with key modifiers to provide the fourth button if a typical 2-3 button mouse is used; scroll wheels didn't exist in the early 1980s so *PERQemu* doesn't use them.

- ❖ *Note:* *PERQemu* has limited support for non-US/English keyboard input, but the facility only allows for remapping keys in the CLI; currently the emulator (which accesses the keyboard through the SDL2 library) has no internationalization support, as the PERQ itself only came with two keyboard types. You may need to set your system's input locale to US/English (or UK/English?) to generate keyboard input that the PERQ can understand.

Roadmap and Release History

At last a v1.0.0 release comes into view! The primary remaining goals are to realize full 24-bit CPU support and debugging/enhancements to the Ethernet interface so that virtual PERQs (and real ones!) can reliably use the network. Bug fixes, UI improvements and eventually updating the codebase to use a newer .NET toolchain (to support building on Linux, for example) are on the horizon.

In the near term, the focus shifts toward curating and publishing additional hard disk images or software bundles through the *PERQmedia* repository, to supplement the “raw” floppy images published on Bitsavers. There is a lot of interesting software to explore, now that PERQ-2s with larger hard drives can be configured. It will be highly instructive to discover if POS G can be rebuilt from scratch, or if the Accent source base and build environment are complete and can be made self-hosting. It would be beneficial to attempt to recreate as much of the original “camera ready” documentation from the original source material for conversion into PDF for the digital archives.

The *PERQemu* Github repository is the best source for up-to-the-minute updates and status.

What's New in *PERQemu* v0.7.5

The focus of this release is on testing, stability, and bundling additional PERQ-2 hard disk images so that those configurations can be used “out of the box.” This release wraps up the changes since v0.5.0 and marks the completion of *most* major features of the emulator:

- All of the PERQ-2 / EIO changes, Canon laser printer, and other additions since v0.5.0;
- Several new bundled PERQ-2 hard drive images (PNX 2, POS G.6);
- The *final* RasterOp rewrite² to fix an extremely obscure edge case;
- Many small bug fixes, performance/accuracy improvements, and debugging enhancements.

Release notes for prior releases are preserved in Appendix D.

² *This statement will not ever come back to haunt me, surely.*

2. The Emulator

Getting Started

Binary distributions of *PERQemu* can be downloaded from the Releases area on Github (see above). The single ZIP archive may be unpacked anywhere you like; there is no need for an installer and the program does not need any special OS privileges to run – unless Ethernet is enabled.

If you prefer to build from source, a clone or download from the Github repository contains everything you need to load the “*PERQemu.sln*” file into Visual Studio. (Please consult the file “*Readme-source.txt*” for more information about the organization of the source tree.) The source builds end up in *bin/Debug* or *bin/Release*, which should be your working directory when running the executable.

When unpacked the *PERQemu* binary release distribution contains several subdirectories:

<i>Conf/</i>	Contains a collection of predefined system configurations as well as device data required for operation. By default, all custom PERQ configurations are also saved to and loaded from this directory.
<i>Disks/</i>	Contains disk images that the emulator can access. Several “stock” hard drive images are included to get you started; additional media images may be added in future releases. All disk images are saved to and loaded from the <i>Disks</i> directory by default.
<i>Output/</i>	When logging debug output to disk is enabled, those files go here by default. Saved screenshots and printed Canon laser printer pages land here too. Output directory is a settable preference.
<i>PROM/</i>	Contains dumps of PERQ ROMs necessary for operation.
<i>Resources/</i>	Contains graphics or other files necessary for operation.

A number of other interesting files and directories are included in the full source distribution, including several “readme” files and additional documentation. (This is all browsable on Github.)

Running *PERQemu*

To launch the program, first set your working directory to where you unpacked the ZIP file. Then just invoke `PERQemu.exe`:

Windows: double-click the `PERQemu.exe` icon. *PERQemu* is a “console application,” so a command window will appear and you’ll be at the command prompt. Or you may also run the executable from the command line.

Unix/Linux/Mac: type “`mono PERQemu.exe`” from the shell prompt in a terminal window. The *PERQemu* command interpreter will announce itself, same as in the Windows version.

At startup you can supply a script to run as an argument. The command syntax is:

```
Usage:  PERQemu [-h] [-v] [-g] [<script>]

      -h      print this help message
      -v      print version information
      -g      start up in GUI mode
script    read startup commands from file
```

The script you provide must be a plain text file, and it may contain any valid command line you would type yourself. (See *Automating Actions with Scripts* later in this chapter for more information.) In the examples which follow, things you type will be indicated in **bold**:

```
[octothorpe:PERQemu/bin/Release] skeezics% mono PERQemu.exe
Initializing, please wait...
Settings reset to defaults.
Default output directory is now 'Output'.
PERQemu v0.7.0 ('As the sparks fly upwards.')
Copyright (c) 2006-2025, J. Dersch (derschjo@gmail.com)
With contributions from S. Boondoggle (skeezicsb@gmail.com)
>
```

- ❖ *Note:* The first time you run *PERQemu*, a settings file is created in your home directory. On Windows this file will be called `PERQemu.cfg`; on the Unix platforms this will be `~/ .PERQemu_cfg`. This is the only file that *PERQemu* will write outside of the folder where you install it.

The Command Line Interface

PERQemu's top-level or main command prompt is a single '>'. The command-line interface (CLI) now organizes the extensive command set into a hierarchical set of "subsystems." The prompt will change according to the current level in the hierarchy, such as:

```
    configure>
or
    settings>
```

Essentially a subsystem is just a convenient way to save typing when executing a number of related commands. To return to the previous level, the "done" command exits the current subsystem.

Getting Help

The CLI provides some on-line help. At the top-level prompt, the "help" command will get you started; this is still a work-in-progress.

- ❖ *Note:* The built-in help is currently hardcoded into the program, but will eventually be expanded to be more complete and/or offer to bring up a web-based help site. Or just be folded into this document?

Additionally, typing "commands" at any prompt will show a summary of all the commands available at that point in the hierarchy. For example:

```
> settings
settings> commands
    assign ethernet device - Map a host network adapter to the PERQ Ethernet device
    assign rs232 device - Map a host device to a PERQ serial port
    autosave floppy - Save modified floppy disks on eject
    autosave harddisk - Save harddisks on shutdown
    autosave tape - Save modified streamer tapes on unload
    canon output format - Set image file format for Canon laser printer output
    ...
    pause on reset - Pause the emulator after a reset
    pause when minimized - Pause the emulator when the display window is minimized
    rate limit - Set rate limit option flags
    rate limit default - Set default rate limits
    save - Save current settings
    show - Show all program settings
    show ethernet devices - List available host Ethernet interfaces
    unassign ethernet device - Unmap a host network adapter (disable PERQ Ethernet)
    unassign rs232 device - Unmap a device from a PERQ serial port
```

You can also type:

> **settings commands**

at the top-level prompt to see the same list. For some subsystems, *PERQemu* keeps track of unsaved changes, updating the prompt with a '*' to show that something has been modified:

```
> settings
settings> autosave floppy yes
Autosave of floppy disks is now Yes.
settings*> save
Settings saved.
settings>
```

Tab Completion

The general form of a command is a series of one or more plain words followed by zero or more arguments. Each command line is executed when RETURN is pressed. At any point, pressing the TAB key will attempt to show the available options or expected argument based on what you've typed so far. For example, at the main prompt pressing TAB shows:

```
> <tab>
Possible completions are:
    about      bootchar    commands    configure...  debug...
    exit       go          gui          help...       inst
    load...    power...   quit         reset         save...
    settings... start      status       step          stop
    storage... unload...
```

Commands that share a common prefix or that lead to subsystems are followed by ellipses (...) to show that additional input will be expected.

The tab completion feature will always try to expand what you've typed when it can:

```
> se<tab>
Possible completions are:
    <CR>      assign    autosave...  commands    default
    display   done      load         output      pause...
    performance save      show         unassign
> settings
```

In the example above, "se" is an unambiguous match for the "settings" command, so *PERQemu* shows you what options are available next, and then expands the partial command for you. Since

settings is also a subsystem, the first option shown is <CR> which means you can press RETURN at this point and the command is complete, or you can continue to type the next command word. If there are multiple possibilities, the CLI expands the input as far as it can then waits for you to type another character to proceed:

```
> st<tab>
Possible completions are:
      status      start      stop      step      storage...
> sta<tab>
Possible completions are:
      status      start
> sta
```

Note that the SPACE character sometimes acts like a TAB too. When it makes sense. Mostly.

Some commands use “noise words”³ to help make the syntax more natural or disambiguate their meaning; *PERQemu* will expand these too:

```
> settings pause <tab>
Possible completions are:
      on      when
> settings pause o<tab>
Possible completions are:
      false      true
> settings pause on reset      ← PERQemu expands both words
> settings pause w<tab>      ← erasing “on reset” and typing
Possible completions are:
      false      true
> settings pause when minimized      ← PERQemu expands both words
```

Command Arguments

Many commands require arguments, while some accept optional arguments. Tab completion can be used to see what is expected next. *PERQemu* will prompt you for the type and offer the name of the argument to be entered:

```
> configure assign<tab>
Possible completions are:
      [string] (file)
```

Here a string argument named “file” is required next.

³ *A paean to the glory days of TOPS-20, for the old timers...*

```
> configure assign g7.prm<tab>
Possible completions are:
    <CR>          [0..255] (unit)
> configure assign g7.prm 1<tab>
Possible completions are:
    <CR>
```

Entering `g7.prm` and pressing TAB shows that `<CR>` may be entered to end the command, in which case default values will be used to fill in the remaining arguments, *or* a numeric value for “unit” may be entered. Typing the value and pressing TAB again now indicates that no more input is required.

- ❖ *Note:* Most command processing is case insensitive; you don’t have to capitalize command words, although sometimes case matters for arguments. For instance, filenames on most Unix hosts *are* case sensitive; see *Working with Storage Devices* later in this chapter for more info.

The most common argument types are:

```
byte      - a positive 8-bit integer value (0..255)
ushort    - a positive 16-bit integer (0..65535)
uint      - a positive 32-bit integer
char      - a single character (generally alphanumeric)
string    - a single word (quoted if the value contains spaces or multiple words!)
```

- ❖ *Note:* String arguments *must* be surrounded by double quotes (“”) if they contain spaces. Pressing TAB while editing a string generally inserts a space instead, until the closing quote is typed (then completion resumes). Pressing RETURN will append a closing quote automatically if needed.
- ❖ *Note:* Currently filenames are entered as plain strings, so you have to type them in full.⁴

Each command will do additional checking of its inputs and the numeric range prompted for may be larger than the range of acceptable values in the given circumstance. In other words, while the `unit` number in the example above may be stored as an 8-bit byte, the limit on the number of storage units you can actually assign is much smaller than 255!

⁴ *Tab completion for filenames is planned for a future release!*

When entering numeric types, you can also supply the value in any of the common bases using a character prefix or common format:

Base	Prefix	Example
Binary	b	b10001100
Octal	o or %	o377 or %177600
Decimal	d (or none)	d12345 or 54321
Hexadecimal	0x, x or \$	0xff, x4eff, \$80000

Some types of arguments do provide tab completion. Boolean values are entered as text and expanded to “true” or “false”, while enumerated types are completed like normal command words.

Editing and History

The CLI allows basic editing of the command line as you type. The following keys do what you generally expect them to do in a DOS-style editor; Unix/Emacs-style control character equivalents work as well, assuming your terminal or console application doesn't intercept them or bind them to another function:

<code>^U</code> or <code>ESCAPE</code>	- erase the entire line
<code>^H</code> or <code>BACKSPACE</code>	- delete character to the left
<code>^D</code> or <code>DELETE</code>	- delete character to the right
<code>^A</code> or <code>HOME</code>	- move cursor to the beginning of the line
<code>^E</code> or <code>END</code>	- move cursor to the end of the line
<code>^B</code> or <code>LEFT</code> (<code>←</code>)	- move cursor left one character
<code>^F</code> or <code>RIGHT</code> (<code>→</code>)	- move cursor right one character

When editing, you may press `RETURN` anywhere on the line to invoke the command. At the end of a line, `RETURN` will also complete any unambiguous final arguments on the line as well:

```
> quit w<return>
> quit without saving
```

PERQemu also keeps a limited command history. The `UP` and `DOWN` arrow keys (or the equivalents `^P` and `^N`) allow you to move forward and backward through the history buffer of previously typed commands; you may then use the editing keys to change the line, or press `RETURN` to execute the same command again.

The Configurator

Beginning with version 0.5.0, *PERQemu* allows dynamic configuration of the PERQ to be emulated. While there aren't a *huge* number of options, previous versions of the program required recompilation to change the size of memory or the CPU type – and very little else was configurable. Now, using the Configuration subsystem, you can specify CPU type, option boards, memory and display size, peripherals to attach and more. The configuration describes a complete “virtual machine” that the emulator can set up and run.

Choosing a Predefined Configuration

The first time you run *PERQemu*, the default configuration is selected. This is the only one built-in to the emulator; all the rest are loaded from the `Conf` directory. Most of the common machine types sold by Three Rivers are provided with the distribution, although there are still a few devices and drivers to be added to the emulator before they can be run.⁵ Currently the default is a PERQ-1A with the POS F.1 image already assigned, so when first installed this version of *PERQemu* runs the same software as prior versions.

To choose a different configuration to run, you might first want to see which ones are available. If you define your own configurations, they'll be stored in the same directory and loaded at startup too. Type:

```
> configure list
```

```
Standard configurations:
```

```
    default - Default configuration
    PERQ1 - Original PERQ w/4K CPU, 256KB
    PERQ1A - Large PERQ-1A w/16K CPU, CIO, 2MB
    PERQ2 - PERQ-2 w/16K CPU, 1MB
    PERQ2-T1 - PERQ-2/T1 w/16K CPU, 1MB
    PERQ2-T2 - PERQ-2/T2 w/16K CPU, 2MB
    PERQ2-T4 - PERQ-2/T4 w/24-bit 16K CPU, 4MB
```

```
Current configuration:
```

```
    default - Default configuration
```

To load one, simply type:

```
> configure
```

← *Let's enter the subsystem*

```
configure> load perq2-t2
```

← *Load one*

```
Configuration 'PERQ2-T2' selected.
```

```
configure> show
```

← *And take a look at the details*

```
Configuration: PERQ2-T2
```

⁵ The file `Docs/PERQ_Chart.png` in the source distribution shows the “official” table of configurations from a scanned dot-matrix printout from 1985. *PERQemu* now supports nearly every option on that chart!

```
Description:  PERQ-2/T2 w/16K CPU, 2MB
Filename:      Conf/perq2-t2.cfg
```

```
-----
Machine type:  PERQ2Tx
CPU type:      PERQ1A
Memory size:   2MB
Display type:  Landscape
Tablet type:   Kriz
IO board:      EIO
```

```
Storage configuration:
-----
```

```
Unit: 0  Type: Floppy   File: <unassigned>
Unit: 1  Type: Disk5Inch File: <unassigned>
Unit: 2  Type: Disk5Inch File: <unassigned>
```

The “`configure show`” command with no argument displays the current configuration. If a name from the list of predefined configurations or a file name is given as an argument, the Configurator will show its details instead.

Note that most reconfiguration must take place while the virtual machine is “powered off.” *PERQemu* will prevent you from changing the configuration while the machine is running if doing so would confuse or crash the PERQ, which like most machines in the age before USB didn’t appreciate having peripherals suddenly appear or disappear at runtime. Operations on removable media such as loading and unloading tapes or floppy disks are allowed, of course.

Creating Custom Configurations

This section is written as a tutorial to explain how and why certain configuration choices are made. For an itemized breakdown of available configuration commands, their syntax and options, see the *Command Reference* chapter later in this document.

The Configurator allows you to modify, name and save your own PERQ configurations. All of the `configure` subsystem’s commands modify the current configuration, so if you want to start from scratch you can use the “`default`” command to begin from a known baseline:

```
> configure default
Setting the machine to defaults.
```

Or you can load an existing configuration that’s closer to what you want as your starting point, then just modify the settings you want to change.

Chassis Types

The basic definition of a PERQ starts with the chassis type, to which boards are added, peripherals attached, and other options specified. The “`configure chassis`” command selects the type.

Chassis Type	Description
PERQ1	The original PERQ cabinet, housing a single 14” Shugart hard drive, an 8” floppy disk, and an IOB or CIO board. Uses the original PERQ-1 style parallel keyboard and GPIB BitPad.
PERQ2	First system to ship with the EIO board. Houses a single 8” Micropolis hard drive and the standard 8” floppy drive. Uses the PERQ-2 style serial keyboard and the Kriz tablet.
PERQ2Tx	This ICL-designed cabinet quickly replaced the early PERQ-2. The “T1” model comes with an 8” Micropolis hard drive; the “T2” model with a 5.25” MFM hard disk in various makes and capacities. The 8” floppy, VT100-style serial keyboard and Kriz tablet are standard. The rare “T4” model is essentially a “T2” fitted with a 24-bit board set.

Table 1: Chassis types.

Starting with *PERQemu* v0.5.8 the first PERQ-2 model (often referred to as “K1” or “Krismas,” sometimes “T0”) can be configured. Extremely rare, this updated model was meant to be smaller and quieter than the original cabinet, but is generally characterized as hotter, more cramped, and harder to work on. By some estimates only a few dozen or so were made, and few references exist in any of the surviving literature and marketing material.

For emulation, selecting the `PERQ2` chassis enforces a limit of *one* 8” Micropolis hard disk and allows both the 4K or 16K CPU. The PERQ-2/T1 is essentially the same machine, but in an ICL-designed cabinet that was common to all of the subsequent PERQ-2 models. As the OS software doesn’t seem to distinguish between the T0 and T1, and *PERQemu* doesn’t yet have a graphical front end that would show you the difference in cabinet types, the `PERQ2` chassis selection represents both models.

The PERQ2Tx designation represents the T2 or T4 models in the ICL-designed cabinet. Selecting this chassis will allow two disk drives but limit the processor selection to the 16K or 24-bit CPUs as discussed in the next section.

Processor Type

The system requires a CPU board. The “`configure cpu`” command allows selection from three types, subject to some limitations based on chassis type:

CPU Type	Description
PERQ1	The basic 4K, 20-bit CPU works with the PERQ-1 and PERQ-2 chassis, providing for a maximum of 1 megaword (2MB) of memory.
PERQ1A	The 16K, 20-bit CPU board is universal, and works with any chassis type. This processor has the widest software compatibility of the available types.
PERQ24	This rare processor is an extension of the PERQ1A. It offers the same 16K of writable control store, but extends the major datapaths and physical address bus to 24 bits for a much higher memory capacity. For emulation purposes this processor will only be supported in the PERQ2Tx chassis.

Table 2: CPU board types.

In PERQ terminology, 4K or 16K refers to the size of the writable control store, a block of fast static RAMs which hold the microinstructions the processor executes. There are subtle differences that the microcoder must be aware of, and some software limitations; these are detailed in the *PERQ Operations* chapter.

The original 4K processor is provided for completeness and historical accuracy. As a general configuration guideline the 16K CPU is the best choice, and is required for later versions of Accent (and possibly PNX).

Configuring Memory

A variety of memory boards exist for the PERQ, differing in total capacity and the type of video display they support. *PERQemu* groups all of them into one “universal” memory board which can drive both display types. Though the 24-bit CPU can address up to 16 megawords (32MB) of RAM, the

practical limit given the DRAM technology available in 1984 is 8MB. While there are some oblique references to an 8MB configuration, it is not yet known if any version of POS, Accent or PNX can actually use that much memory. *PERQemu* caps support at 8MB but this can easily be changed.

Use the “configure memory” command to select how much RAM is available to the virtual machine. With no arguments, a usage message is printed:

```
> configure memory
```

```
Configure the memory capacity, from 256KB to 8MB. The CPU type determines
the maximum memory capacity, which must be a power of two:
```

```
    20-bit CPU: 256, 512, 1024 or 2048 (KB)   or 1, 2 (MB)
```

```
    24-bit CPU: 2048, 4096 or 8192 (KB)       or 2, 4, 8 (MB)
```

```
PERQemu will round up to the nearest legal value.
```

```
> configure memory 2
```

```
Memory size set to 2MB.
```

I/O Board Types

There are two basic PERQ I/O boards, each with two variations. The chassis type selected determines which I/O board may be configured, while the software you intend to run determines which variant is required (detailed in the chapter *PERQ Operations*). The following table explains the differences:

I/O Board	Description
IOB	The original PERQ-1 I/O board. This board contains a 2.45Mhz Z80 and runs the old v8.7 firmware. It contains a Shugart hard disk controller.
CIO	<i>PERQemu</i> uses “CIO” to distinguish an IOB running the new v10.17 Z80 firmware. It also encompasses a very rare ICL-produced variant that can drive either a Shugart hard disk <i>or</i> an 8” Micropolis hard disk.
EIO	The “Ethernet I/O” board, used by all PERQ-2 models. It features a faster 4Mhz Z80, runs v10.17 firmware, has a battery-backed real-time clock chip and a second RS-232 port. The EIO can drive 8” Micropolis <i>or</i> 5.25” MFM hard disks, depending on the chassis type.
NIO	An “Ethernet I/O” board without the Ethernet.

Table 3: I/O board types.

The “configure io board” command is used to make your selection:

```
> configure io board cio  
IO Board type CIO selected.
```

When you change the I/O board type, *PERQemu* attempts to remap any assigned media files from the old board selection to the new one. While configuration commands may be given in any order, you may want to assign or load media as the last step; that way the Configurator can verify that the disk controller on your configured I/O board is compatible with the selected disk images.

As of *PERQemu* v0.5.8 all four I/O board types are emulated, enabling most PERQ-1 and PERQ-2 configurations to run. At this time the CIO Micropolis hard disk support is not yet implemented, as a viable OS image has not been found to verify its operation. Currently EIO support for the 24-bit CPU is unverified; the PERQ-2/T4 configuration remains untested.

Display and Tablet Options

PERQemu supports both of the PERQ's monochrome graphics displays: the 768 x 1024 portrait mode display, and the 1280 x 1024 landscape display. The portrait display is compatible with every model and is supported by every PERQ OS. The landscape display was introduced with the PERQ-2 line, but a few extremely rare PERQ-1 landscape memory boards were produced. *PERQemu* allows you to configure either display, and both have been tested successfully!

The “configure display” command does the obvious and expected thing. Refer to the *PERQ Operations* chapter for details about software restrictions, as some older OSes do not have landscape support, and some that do require a separate disk boot be created to enable it.

Two types of digitizing tablet and pointing device are available: the PERQ-1 generally uses a GPIB-connected Summagraphics BitPad One, while the PERQ-2 uses a Three Rivers-produced “Kriz tablet.” Named after its creator J. Stanley Kriz, a brilliant hardware engineer and co-founder of the company, the Kriz tablet comes in portrait and landscape variants. *PERQemu* emulates the appropriate version based on the display selection.

The “configure tablet” command lets you attach one *or both* types to your PERQ. Although the operating system will only use one or the other, later versions of POS allow you to choose which one to activate when you log in. The chapter *PERQ Operations* explores this in depth.

I/O Options

Every PERQ contains two slots in its card cage for optional boards: a CPU Option and an I/O Option. No specific CPU Option board was ever produced, however, so *PERQemu* combines these into one virtual option board. While some of the optional hardware available is not yet implemented in the emulator, the Configurator allows you to see what should be coming in future releases.

To select an option board, use the “configure option board” command to choose which board to install:

Option Board Type	Description
OIO	A multi-function board that contains up to four controllers in any combination: Link, Ether, Canon and Tape.
MLO	The “Multibus-Laser Option” board provides a Multibus-I compatible interface to the PERQ. A Canon interface is built-in. Sub-options include: SMD and Tape.
Ether3	This full-size board provides a 3Mbit Ethernet interface.
None	Removes any configured board and leaves the Option slot empty.

Table 4: I/O Option board types.

Status of I/O Option Development

As of *PERQemu* v0.5.0, only the OIO option board is emulated. This is by far the most common option, as it's the only way to add Ethernet to a PERQ-1. The QIC Tape option was added in v0.4.8; an initial “fake” Ethernet device was added in v0.4.9. A preliminary “real” Ethernet was introduced in v0.5.2, while the Canon interface appeared in v0.5.4.

The MLO was once thought to have been forged from pure *unobtainium*, but several boards have been recovered and source code for the driver appears to be complete. Documentation and schematics exist to some degree. The primary lure of emulating the MLO is to allow PERQ-2 configurations to attach great big SMD hard disks, but verifying emulation accuracy without a working hardware model to compare to will make things difficult.

The “Ether3” board was not officially a product of Three Rivers Computer, as it was mostly used at Carnegie-Mellon University to connect to the early “experimental” Ethernet. (CMU was one of the sites that received a bunch of Xerox Altos with their original Ethernet interfaces in the 1970s.) Only one complete PERQ 3Mbit board is *known* to exist, wire wrapped on a full-size prototyping board. While there are no schematics or documentation for it, software support still exists, so adding a driver to *PERQemu* is a (very) long-term goal.

Configuring Sub-options

The OIO and MLO boards are multifunction boards that come in different varieties. The “configure option add” and “configure option remove” commands allow you to selectively customize these boards, as described in the table below:

Option	Description
Link	Provides a 16-bit DEC-compatible parallel interface that allows one PERQ to connect directly to the control store of another. <i>(In hardware this is a “universal option” that can be installed in the CPU Option slot of any machine.)</i>
Ether	Provides a 10Mbit Ethernet interface for the OIO board when installed in a PERQ-1. Conflicts with the Ethernet interface on the PERQ-2’s EIO board.
Canon	Provides a video bitstream interface to the early Canon laser printers, the LBP-10 (240dpi) or the LBP-CX (300dpi). Can be added to the OIO board, and comes built-in on the MLO board.
Tape	When added to the OIO <i>or</i> MLO board, it provides an interface to the Archive Sidewinder streaming tape drive. <i>(The streamer interface is also a universal option and can occupy the CPU Option slot.)</i>
	Far, distant future: the MLO board should also emulate the Ciprico Tapemaster Multibus controller for connecting a 9-track ½” tape drive.
SMD	When added to the MLO board, it provides a Multibus Storage Module Disk controller (Interphase SMD 2190).

Table 5: I/O sub-options.

Since the `Link` option is probed as part of the boot process, the emulator currently implements enough to let the microcode know that nothing is connected. Unsupported options for the OIO are

quietly ignored. The Configurator will explicitly disallow any configuration choice that would prevent the virtual machine from starting up. It also checks that the options selected are appropriate for the board type configured.

Serial Ports

Every PERQ-1 has a single RS-232 port that runs at up to 9600 baud. The PERQ-2 has two ports and can run them at 19200 baud. These are called RS232-A and RS232-B in most of the operating system literature. The hardware uses the Zilog SIO and CTC chips connected to the Z80 on the I/O board, with software support for programming them directly.

PERQemu emulates these chips at the register level, providing a translation to either a “real” serial device connected to the host computer, *or* a “pseudo device” called RSX: that lets the emulator transfer text files to and from the host by pretending to be a PDP-11. This is explained in detail in the section *Working with Communication Devices* below.

Network Address

Each PERQ can have an Ethernet interface. As an early adopter of Ethernet, most PERQ software was written to display the 48-bit address as three signed 16-bit integers, before the formatting convention of six bytes (octets, in network parlance) displayed in hexadecimal and separated by ‘:’ or ‘-’ became common. Don’t be alarmed if you see a negative number in a PERQ Ethernet address!

PERQemu sets the first three octets to 02:1c:7c (Three Rivers’ prefix assigned by Xerox) and the fourth octet according to the board type. The last two octets are read from the NET PROM on the hardware; they can be configured in the emulator with the “configure ethernet address” command. If set to zero or left unconfigured, a random address is generated at runtime. On PERQ-2 machines, the last two octets of the Ethernet address are also the system serial number that can be read from the backplane.

Disks, Floppies, and Tapes

Based on the chassis type and I/O board selected, the Configurator “knows” what type and how many storage devices can be attached. For practical use, every PERQ must have a bootable hard disk or floppy disk loaded. The configuration of storage devices is explored in the section *Working with Storage Devices* below.

Checking for Errors

To make sure your custom configuration is valid, the “configure check” command can be run at any time. The Configurator will make sure there are no conflicts, alerting you to problems that must be resolved and issuing warnings about unusual or rare configurations that may have limited software support. An example dialog to illustrate the types of messages produced:

```
configure*> show
Configuration: test
Description:   A test system
Filename:      Conf/test.cfg
-----
Machine type:  PERQ2
CPU type:      PERQ1A
Memory size:   4MB
Display type:  Portrait
Tablet type:   BitPad
IO board:      CIO
Option board:  OIO  Options: Link, Ether

Storage configuration:
-----
Unit: 0  Type: Floppy   File: <unassigned>
Unit: 1  Type: Disk8Inch File: <unassigned>

configure*> check
Configuration is not valid:
PERQ1A (16K) CPU supports a maximum of 2MB of memory.
configure*> memory 2
Memory size set to 2MB.
configure*> check
Configuration is not valid:
IO Board type CIO not compatible with PERQ2 chassis.
configure*> io board eio
IO Board type EIO selected.
Note: Updating storage from CIO to EIO.
configure*> check
Configuration is not valid:
OIO Board Ethernet option conflicts with EIO board.
configure*> option remove ether
IO Option 'Ether' deselected.
configure*> check
Configuration is valid, with warnings:
The PERQ won't boot without a hard or floppy disk present.
configure*> done
Note: the configuration has been modified but not yet saved.
Use the 'configure save' command to save your changes.
>
```

Note that the Configurator doesn't impose any particular order in which commands must be given. It may complain when you enter a value that is in conflict but it doesn't prevent you from doing so. When the `power on` command is given, however, the emulator will refuse to start any configuration that doesn't pass the `check` rules.

Naming and Saving Configurations

Finally, a custom configuration must be named before it can be saved. The “`configure name`” command accepts a string which will form the basis of the saved filename. In general, you should

- use a fairly short, simple name without spaces or special characters;
- let the Configurator save the file in the `Conf` directory automatically.

Of course, you may also choose to provide a pathname to store the configuration wherever you please, but at startup *PERQemu* won't know where to find it and you'll have to type the same pathname in full to reload it.

The “`configure description`” command lets you add a one-line “human readable” description for the configuration. This is optional; it's displayed by the “`configure list`” command to show what each file contains. For example:

```
> configure
configure> name s5lisp
configure*> description "PERQ-1 Landscape with Accent S5, Lisp"
configure*> list
Standard configurations:
    default - Default configuration
    fl6dev  - POS F.16 development machine
    ...
Current configuration:
    s5lisp  - PERQ-1 Landscape with Accent S5, Lisp
```

To save the completed configuration, “`configure save`” writes out the file:

```
configure*> save
Configuration saved to 'Conf/s5lisp.cfg'.
configure> done
>
```

That's all there is to it. Easy as pie!

Working with Storage Devices

The PERQ supports a range of storage devices, including floppy disk, hard disk and tape drives. In *PERQemu* these devices are emulated at the block level, stored as “media images” in files on the host computer in a number of different supported formats.

- ❖ **Warning:** *PERQemu* does not automatically save modifications to loaded storage media! All changes are made to an in-memory copy and are lost when the virtual machine shuts down unless they are explicitly saved. New preferences may be set that affects this; see the section *Setting User Preferences* below.

All PERQ operating systems generally require that a hard disk be present, though you can boot and run the machine in a *very* limited way from a floppy disk alone. *PERQemu* includes several hard disk images in the distribution, and the growing library of software will expand as additional disk drive types are added to the emulator.

Supported Devices

Each PERQ can support one type of internal hard disk based on the chassis type and I/O board selected. The Configurator checks that only the correct type of image is attached to the disk controller in the virtual PERQ. Table 1 shows the maximum number and type of supported devices:

	PERQ-1	PERQ-2	PERQemu ⁶
Floppy drives	1	1	2
Internal hard drives			
14" Shugart	1	-	2
8" Micropolis	1	1	2
5.25" MFM	-	2	4
External hard drives			
SMD	-	?	4

⁶ *PERQemu* currently allows only the maximum number of drives that the actual hardware supports, but in future releases (as software and microcode support is enhanced) we may expand the limits. This column represents the theoretical maximums.

	PERQ-1	PERQ-2	PERQemu ⁶
Tape drives			
QIC streamer (¼")	1	1	1
9-track (½")	-	?	1

Table 4: Storage device types and limits

- ❖ *Author's note:* While the 8" floppy drive was always listed as an "option," I've never actually come across a PERQ that didn't have one attached. Because it is essentially free to emulate, all standard configurations in *PERQemu* have a floppy drive attached as unit #0. You are welcome to configure a machine without one, but that option has not been exhaustively tested and may result in boot failures or other problems.

Each PERQ model supports only one type of internal hard drive, based on the chassis type and I/O board type selected – so the PERQ-2 models may contain 8" Micropolis *or* 5.25" MFM drives, but not both. As discussed in the *Configurator* section above, external SMD and tape drives are connected by optional I/O boards and are not yet implemented.

Unit Numbers

Each device in *PERQemu* is assigned a "unit #" to uniquely identify it when more than one of a given type is configured. In this release it's a very straightforward mapping, as the options are quite limited:

Unit 0:	Floppy drive	(all configurations)
Unit 1:	Hard drive	(for PERQ-1 or PERQ-2, the only hard drive)
Unit 2:	2nd hard drive	(only in PERQ-2/Tx models)
Unit 3:	Streamer tape	(optional, when configured)

For compatibility with *PERQemu* versions prior to v0.5.0, the "load floppy" and "load harddisk" commands are provided as shortcuts for loading units 0 or 1, while new command syntax allows specifying the unit number (in multi-drive configurations). Many commands can automatically determine which unit to assign. When configured with an I/O Option board with the streaming tape, the command "load tape" automatically selects unit #3.

The Disks Directory

In the discussion that follows, *PERQemu* assumes that all media files are stored in the `Disks` directory provided in the distribution. This is located in the base directory (where `PERQemu.exe` resides) as discussed in the *Getting Started* section above.

A simple algorithm is used to find files to load, trying each step in turn and returning the first match:

1. Look for the file name exactly as given;
2. If no directory was specified, look in `Disks` for the file name;
3. If the file does not have an extension, retry steps 1 and 2 for each of the known extensions.

While the distribution is currently just a handful of files, an archive of many *hundreds* of floppy, tape and hard disk images is being assembled, so it is recommended that you choose simple names and allow *PERQemu* to apply the correct directory and file extensions when loading or saving media. Future releases may rename `Disks` or add `Floppies` and `Tapes` to the “search list” to keep things manageable.

For now, any hard disk, floppy or tape images you create or download should be dropped into `Disks` so *PERQemu* can find them without a lot of typing!

Supported Media Formats

For *PERQemu* v0.5.0, a new common *PERQmedia* file format has been devised to store *all* of the supported storage devices. While the world may not need Yet Another File Format, the PERQ universe is fairly static and self-contained; translating images into the new format provides both data compression and a standard 32-bit CRC for verifying the integrity of the archive, features which the older formats do not have. Text and graphical labels may be stored in the file as well, which future versions of *PERQemu* can use to aid in the selection of media to load.

While the *PERQmedia* file format will be used by default, *all* of the previous formats may still be used as well, and utilities will be available to transpose between them. *PERQemu* recognizes these file types:

PRQM format: the new unified format; uses the file name extension “.prqm”. Can be used to store any supported PERQ storage device.

PHD format: files ending in “.phd” contain Shugart 12MB or 24MB hard disk images, suitable for use with older versions of *PERQemu*. These drives are only supported on PERQ-1 models.

IMD format: floppy images in IMD format typically have the “.imd” (or DOS “.IMD”) file extension. *PERQemu* can read and write these, and will preserve the media label information when transferred to/from PRQM format.

PFD/raw format: older floppy images might have a “.raw” or “.pfd” extension. These were likely converted from DMK or IMD images into a “raw” format with no metadata. The PFD format was my brief and ill-advised flirtation with adding a small “cookie” to the first sector of a raw floppy image (undetectable by *PERQemu* or the emulated PERQ) that could provide some hints as to format and contents.

❖ *Note:* While this version of *PERQemu* can read both formats, if asked to write a “raw” floppy it will always include the PFD cookie and will append/replace the extension with “.pfd” to more uniquely identify the image.

TAP format: *PERQmedia* now supports the SIMH magtape format for both reading and writing. These files are currently limited to QIC tape formats with fixed 512-byte blocks. The “Stut” utility (available for POS and later versions of Accent) provides a way to dump and restore entire partitions at a time. TAP files end in “.tap”.

Loading, Saving, and Unloading Media

Each configuration optionally stores the names of media files to load at startup, so that specific peripheral or machine configurations may be paired with the necessary software to support them. The “configure assign” command establishes a mapping:

```
> configure assign <file> [unit #]
```

Note that *PERQemu* figures out the appropriate unit number based on the file’s contents; for a dual-drive PERQ-2/Tx configuration, you should specify unit 2 to avoid ambiguity. If left unassigned, media files must be selected at runtime after the configuration is loaded.

The “configure unassign” command removes the association of image files:

```
> configure unassign <file>
    or
> configure unassign unit <unit #>
```

These assignment commands change the mapping of files in the configuration, but don't affect the loaded media if the machine is running.

The top-level `load`, `save` and `unload` commands can also still be used to update a configuration:

```
> load <device> <file> [unit #]
> save <device>
> unload <device>
```

Where <device> is "floppy," "harddisk" or "tape." These commands update the current media assignments in the same way the `assign` command does, but also affect the running machine. Because floppies and tapes are removable media, *PERQemu* allows these commands at any time.⁷ The current configuration record is updated to reflect the status of the drive *and* the emulated drive receives a status change signal.

- ❖ *Note:* You should take care to not `load`, `save` or `unload` removable media while the operating system is trying to access it, of course, as data corruption may occur (just as if you suddenly eject a floppy from a real drive while it is being written). The emulator now overlays small icons of a floppy or tape cartridge on the lower right corner of the display when those drives access their media, as there is no other visual or auditory cue as with the hardware to let you know when the device is busy.

As the internal hard drives in the PERQ are not removable, a `load` or `unload` of a hard disk image is disallowed when the emulator is running; it must be powered down first (see *Managing the Virtual Machine*, below).

Showing Storage Device Status

When the emulator is running, the "storage status" command prints a one-line status report for each configured drive and its loaded media, or one particular unit # if specified:

```
> storage status
Drive 0 (Floppy) is online, removable, no media loaded
Drive 1 (Disk14Inch) is online, loaded (Disks/fl.phd), writable
```

⁷ PERQmedia will accommodate removable hard disks too; SMD drives that use removable "packs" will be supported in a future release.


```
> storage status 1
Drive 1 (Disk14Inch) is online, type Shugart24
Filename: Disks/newf1.phd (PHDFormat)
Flags:    Loaded, writable.
```

This information is updated in real-time as changes are made, such as loading a floppy or having the PERQ write to the device (setting its “modified” flag). If the media type supports a text label and one is present, it is displayed as well; this is a feature we hope to expand upon in future releases to assist in identifying the contents of media files:

```
> load floppy s5boot
Drive 0 unassigned.
Assigned file 'Disks/s5boot.imd' to drive 0.
Loaded Disks/s5boot.imd.
> storage status 0
Drive 0 (Floppy) is online, type SA851
Filename: Disks/s5boot.imd (IMDFormat)
Flags:    Loaded, removable, bootable, writable.
FS hint:   POS
Label:
-----
621003-00
PQS PERQ SOFTWARE ACCENT S5 NON-SOURCE
Copyright (C) 1984, PERQ Systems Corporation
single density, double sided

Contents:  ACCENT PERQ1 BOOT NS
-----
```

Changing Formats, Making Backups

The save commands shown above always update a file in place, using the same name and format as originally loaded. To save a copy of any loaded disk or tape image, the “save device as” commands allow you to specify a new filename and, optionally, a different file format:

```
> save harddisk as newf1 <tab>
Possible completions are:
    <CR>      imdformat  phdformat  prqformat  rawformat  tapformat
> save harddisk as newf1 prqformat
Updating from Disks/f1.phd
Saved Disks/newf1.prqm.
Drive 1 saved to 'Disks/newf1.prqm'.
```

In the example above, *PERQemu* makes a new copy of the “stock” POS F.1 image and writes a new file in the PRQM format. Because a simple name (“newf1”, rather than a path) was given, the Disks

directory was prepended and the correct file extension was appended. Advantages of the new format are more apparent when debugging output is enabled:

```
Updating from Disks/test.phd
MediaLoader: Compression ratio = 5.40
MediaLoader: Space savings = 81.47%
MediaLoader: Saving computed CRC: df22731c
Saved Disks/test.prgm.
```

- ❖ *Note:* It is recommended that you make a copy of any bundled disk images if you make changes to them and don't want to risk having them overwritten when a new release is installed. This can easily be done using your operating system's native file copy tools outside of *PERQemu*.

Creating and Formatting New Media

The new “storage” subsystem allows the creation of new blank media files in a wide range of predefined types. To view the list of predefined device types, use the “list” command, as shown in the representative output below:

```
> storage list
```

Drive Type	Drive Class	Description
-----	-----	-----
Archive20	TapeQIC	Archive 3020I DC-300XL (20MB) QIC tape
Archive45	TapeQIC	Archive 9045I DC-300XL (45MB) QIC tape
Floppy1024	Floppy	Shugart SA851 DS/DD (1MB) floppy disk
Floppy256	Floppy	Shugart SA851 SS/SD (256KB) floppy disk
Floppy512	Floppy	Shugart SA851 DS/SD (512KB) floppy disk
Shugart12	Disk14Inch	Shugart SA4004 12MB hard disk
Shugart24	Disk14Inch	Shugart SA4008 24MB hard disk
Microp35	Disk8Inch	Micropolis 1202 35MB hard disk
Microp40	Disk5Inch	Micropolis 1304 40MB hard disk
Maxtor71	Disk5Inch	Maxtor XT-1085 71MB hard disk
Maxtor85	Disk5Inch	Maxtor XT-1105 85MB hard disk
Maxtor120	Disk5Inch	Maxtor XT-1140 120MB hard disk
Vertex51	Disk5Inch	Priam/Vertex V150 51MB hard disk
Vertex72	Disk5Inch	Priam/Vertex V170 72MB hard disk
...		

The complete list of known storage devices is loaded by *PERQemu* at startup. This will eventually contain all the drives that have documented software support by one or more PERQ operating systems, as well as a few extra that were era-appropriate.

To create a new type of device, the undocumented “storage define” subsystem is provided. Adventurous users may explore the file “Conf/StorageDevices.db” to see how *PERQemu* creates storage devices, but only the source code describes all of the parameters supplied and their use.

Next, use the “create” command to generate and format a new file. For example, to create a blank “standard” floppy image (double sided, single density):

```
> storage create floppy512 backup
Formatting new drive... done!
Saving the image...
Saved Disks/backup.prqm.
```

The “formatting” applied here is only the writing of empty data blocks; each device will require a specific operation to prepare it for use with the operating system. See the chapter *PERQ Operations* for more information about how each operating system uses storage devices.

This file can then be loaded using the command:

```
> load floppy backup
Assigned file 'Disks/backup.prqm' to drive 0.
```

Note that *PERQemu* always creates new media using the PRQM file format.

Working with Communication Devices

PERQemu allows several “real” devices on the host computer to be mapped to emulated devices in the virtual PERQ. These include serial ports, an audio output device, and an Ethernet adapter; others may be added in future releases. While the emulator provides the PERQ-facing side of each device interface, the C# runtime system or libraries bundled with *PERQemu* pass the data to and from the host, allowing the virtual PERQ to connect in a real way to external systems.

These mappings are established through the Settings subsystem, documented in the *Setting User Preferences* section below. Once a mapping is set up, it can be used by any configuration.⁸ For some devices the association is made explicitly; for others the existence of the mapping allows it to be used implicitly without specific setup through the Configurator.

- ❖ *Note:* When establishing a device mapping, *PERQemu* tries to confirm that the filename you give exists, but doesn't attempt to open it or interact with it in the way the emulator does. Unpredictable or erroneous behavior may result if the mapped host interface is of the wrong type or is not supported by the runtime system.

Serial Devices

The “settings assign rs232 device” command is used to make the connection for serial devices. The syntax is specific to your host operating system, but is generally in the form “COM n ” for Windows, or “/dev/ xyz ” for Unix-based systems:

```
settings> assign rs232 device<tab>
Possible completions are:
    [char] (port)                ← port A for any PERQ; also B on PERQ-2 models
settings*> assign rs232 device a<tab>
Possible completions are:
    [string] (hostDevice)        ← device name is host-specific, may be case-sensitive
settings*> assign rs232 device a /dev/ttyS0
Device '/dev/ttyS0' assigned to serial port A.
```

PERQemu will now display a list of potential serial devices when you press tab, specific to your host system. If your device does not appear in the list, you can override the names given and type in a path. Moreover, you can set the specific port parameters for your device by entering additional optional

⁸ The rationale here is that the host-specific mapping is set up once and persisted with your other preferences, rather than for every configuration that uses the device. However, RS232 port(s) on most “modern” computers require USB-to-serial converters, which often change their ID every damn time you plug them in, which might make this approach less than ideal. I ain't married to it.

parameters for baud rate, character size, parity, and number of stop bits. The defaults are 9600 baud, eight bit characters, no parity, and one stop bit per character.

- ❖ *Note:* Because the PERQ Z80 firmware assumes it is directly controlling the hardware, it issues *numerous* software resets as the machine boots, and it may change the programming in unpredictable ways that can make the C#/Mono runtime system *very* cranky. Now you assign the characteristics of your host device independently and *PERQemu* buffers the serial stream between the host and virtual machine so the PERQ only “sees” the data at the baud rate it can handle.

Once saved, any configuration you load that uses RS232-A will be connected to your specified device. This may, of course, be changed at any time. However, the file is opened when the virtual machine is powered on and any settings change will not take effect until the emulator is restarted. If for any reason you simply wish to disassociate a particular device mapping, the “`settings unassign rs232 device`” command may be used to clear the assignment.

Serial devices require a separate configuration step. The “`configure enable rs232`” command tells the emulator to use the device:

```
> configure enable rs232                ← port A by default
RS-232 port A enabled.
> configure show
...
IO board:      CIO
    RS-232 A:  /dev/ttyS0                ← as configured in the example above
Option board:  OIO  Options: Link, Ether
...
```

The “`configure disable rs232`” command disconnects the selected device for the current configuration. The mapping established in the Settings is preserved, of course.

While there is negligible overhead if the serial port is enabled and not used, you may elect to leave it unconfigured if you don't have an explicit reason to use it. Or you may want to connect the RSX: pseudo-device instead (see below).

Todo: On Linux, you may need to make `/dev/ttyS0` writable, or the serial port will be silently ignored. Figure out if there's a way to make this automatic or at least document it properly.

The RSX: Pseudo-device

The POS operating system offers a little known option to effect text file transfers over the serial port directly from the Shell. Copying a file to `RSX:` initiates a transfer using the DEC RSX11-M “PIP” transfer protocol. Since it’s *unlikely* that you have a PDP-11 running RSX11 attached to your RS232 port, *PERQemu* provides a fake backend that simulates the remote (host-side) part of the protocol.

To enable this, simply specify `RSX:` as the device interface in the Settings `assign rs232 device` command shown above. (This is the same on Windows and Unix hosts.) More information about the use and limitations of this facility is provided in the POS section of the *PERQ Operations* chapter.

“Speech” Output

The PERQ provides a “telephone quality” audio output capability through a Motorola MC3417 Continuously Variable Slope Delta Modulator (CVSD) chip. The output is essentially an 8-16Khz serial data stream pumped through one channel of an SIO chip to a DAC producing a monaural audio signal. No official way to record audio input was included in the hardware; the CPU generates the waveforms and uses `PERQ ⇒ Z80` messages and DMA to pump out the bits.

- ❖ *Note:* *PERQemu* doesn’t yet emulate audio, though the SDL2 library does provide the plumbing required for platform-independent output on the host side. As *cool* as it would be to include this, it’s fairly low on the priority list given its limited PERQ software support (beyond basic system beeps and a few demos) and the complexity of converting CVSD data to a usable modern output format in real time.

Mapping the host audio device may or may not require a manual Settings command. As every PERQ I/O board includes the CVSD chip, it will likely be enabled automatically and not require a separate Configuration command. *TBD*.

Ethernet

PERQ was an early adopter of Ethernet, and there is a decent amount of software that can use it for simple FTP, remote printing, or in the case of Accent, a whole lot more. *PERQemu* now supports the OIO Ethernet (for PERQ-1 configurations) and EIO Ethernet (PERQ-2). As with serial port configuration, *PERQemu* separates the selection of a host Ethernet adapter from the virtual PERQ configuration. This allows you to set the preferred adapter to use for your PC once, then any configuration with the `Ether` option enabled will use it.

The host adapter is specified using the “`settings assign ethernet device`” command. The list of available adapters is determined by your host OS and will vary widely; *PERQemu* provides the list in the form appropriate for your machine. On a Mac or Linux machine this might produce:

```
> settings assign ethernet device<tab>
Possible completions are:
en0          en1          null
```

where the device names are specific to the available interfaces and naming scheme used by your distribution. On Windows 10, this might look like:

```
Possible completions are:
Ethernet      null          VirtualBox Host-Only Network
```

You can also type in a name or identifier if the one you want isn't in the list. This seems to be required on Windows, where the interface name provided by the C# runtime doesn't actually work! Instead, type “`settings show ethernet devices`” to show the SharpPCap view:

```
SharpPcap 4.4.0.0 devices:
0) rpcap://\Device\NPF_{06821D8A-3632-4BAF-9961-95F0B9E7F0F7} - Network adapter
   'WAN Miniport (Network Monitor)' on local host
1) rpcap://\Device\NPF_{7811773C-E0AA-4115-AA7F-1BBC2220FA89} - Network adapter
   'WAN Miniport (IPv6)' on local host
2) rpcap://\Device\NPF_{39EBA7B9-1B38-4CCD-9694-B777F78C84EB} - Network adapter
   'WAN Miniport (IP)' on local host
3) rpcap://\Device\NPF_{843128F1-FA08-4FB6-A314-DD7356D3FFD2} - Network adapter
   'Intel(R) Ethernet Connection (5) I219-LM' on local host
4) rpcap://\Device\NPF_{109C68A9-6929-4864-90AD-7417C7804ED4} - Network adapter
   'VirtualBox Host-Only Ethernet Adapter' on local host
5) rpcap://\Device\NPF_{Loopback} - Network adapter
   'Adapter for loopback traffic capture' on local host
```

Then cut and paste or type the “`rpcap:`” name that matches the device you want to use:

```
> settings assign ethernet device rpcap://\Device\NPF_{843128F1-FA08-4FB6-A314-DD7356D3FFD2}
```

If you think that's ridiculous, you aren't alone.

On all hosts the special name “`null`” is used to select the fake Ethernet driver, which spoofs enough of the hardware operation to let the PERQ think the interface is configured, but no traffic actually flows in or out. This allows Accent to boot and initialize itself properly, and is the default if no selection is made, no appropriate host adapters are found, or the program doesn't have the required privileges to open the device.

Encapsulation Options and Administrative Privileges

Currently the *PERQemu* Ethernet driver runs in “promiscuous mode,” which allows raw access to the low-level Ethernet frames being sent or received on the host’s adapter. The PERQ can use a mix of proprietary “raw” protocols, and has varying degrees of support for Xerox Network Systems (XNS) and PARC Universal Packet (PUP) protocols, as well as early TCP/IP (v4). The emulator attempts to pass this traffic through without interference, except for MAC address translation as described below.

A major caveat to this approach is that to access the host interface in this way *PERQemu* requires Administrator (Windows) or `root` (Mac/Linux) access to open the host device in a mode where it can access the host to send and receive raw packets. If you are understandably skittish about running the program with escalated privileges, the emulator will detect that it cannot open the device and fall back to the `null` Ethernet implementation automatically. Ways to work around this, such as using UDP encapsulation or other ways to sandbox/limit the program’s privileges to achieve the desired functionality are being explored; for now, you can review the source code and decide the risks for yourself.

Network Address Translation and the *PERQemu* Virtual Hub

As mentioned in the *Configuration* section, *PERQemu*’s Ethernet implementation uses the original block of MAC addresses allocated to Three Rivers Computer in 1980. The three leading octets `02:1c:7c` are hardcoded into the low-level firmware and assumed by much of the software that runs on the machine. Rather than force the reassignment of the address on the host network adapter, and to help pre-IEEE 802.3 PERQ software coexist on modern networks, *PERQemu* instead performs a simple address translation on all Ethernet frames sent from or received by the virtual machine. The technical details of this approach are documented extensively in the source/Github distribution in `Docs/Network.txt`.

Currently there is no official command-line interface to interact with or manage the low-level Ethernet or NAT functionality, although there are plans to greatly expand the emulator’s networking features in a future release. For now, the curious can use the “`debug dump ethernet`” command to see the status of the interface. (This is available in Release builds as well.)

Current Problems and Limitations

The PERQ Ethernet interface and software stack was built when 10Mbit/sec, half-duplex operation was state-of-the-art; to say that it *struggles* to keep up with a typical TCP/IP-based home network in the 21st century is an understatement. *PERQemu* attempts to filter and buffer the incoming traffic,

but often the emulated PERQ is just overwhelmed by the volume of traffic. In its current state, POS struggles to accomplish much of anything while Accent is able to hang in much longer and offer some network functionality. But the old software stacks aren't exactly "robust."

As most contemporary PCs have two or more physical ports, it's recommended that you set up an isolated network – ideally with TCP/IP unconfigured on the private host interfaces entirely – so that *PERQemu* virtual machines (along with any real PERQs you own!) can talk to each other without the onslaught of traffic present on a noisy modern 100+ megabit network. Development of tools and techniques to mitigate this in the emulator is ongoing, and is the focus of the next milestone release.

Todo: Updates as testing/feature additions are completed
Provide a set of commands for tuning/monitoring/filtering/resetting the interface
Figure out/document how to potentially integrate Accent nodes into a home network!
Work out IP Encapsulation for ContrAlto interop (once 3Mbit emulation is added!)
Refer to Ops section for OS-specific Ethernet utilities

Working with Output Devices

The PERQ can work with a number of different printers using GPIB and RS-232 connections. The RS-232 driver *can* theoretically be used to drive a host-attached serial printer directly from the virtual machine, but this has not been tested (yet). A future release may include a simulation of some of the 3RCC or ICL supported printers, as GPIB printers cannot (currently) be interfaced to the virtual machine any other way. Chapter 3, *PERQ Operations*, has more details about the available hardcopy options for each operating system.

Canon Laser Printers

The first volume “desktop” laser printer from Canon was adopted very early by Three Rivers, with a custom hardware controller and driver software appearing in 1981⁹. *PERQemu* now emulates both of the supported print engines: the LBP-10 (240dpi), and the popular LBP-CX (300dpi). These are connected via a 37-pin cable and use the Canon “video interface”, where the PERQ renders a bitmap in memory and sends it to the printer as a serial stream of bits. As an historical aside, this same interface is how certain “low cost” models of the Apple LaserWriter II, Sun SPARCprinter, or NeXT Laser Printer would operate years later, until integrated network interfaces and Postscript interpreters became common and much more affordable.

To add the laser printer option to any PERQ, two steps are needed. First, choose the model by specifying its resolution (expressed in dots per inch):

```
> settings canon resolution 240          ← selects the LBP-10
> settings canon resolution 300          ← selects the LBP-CX
```

The default resolution is used for all configurations that include the Canon controller. While this setting may be changed at any time, the virtual machine must be powered off and back on for it to take effect. Second, configure the OIO option board:

```
> configure option board oio              ← if not already specified
> configure option add canon
```

With the Canon option selected, output from the PERQ’s laser printing software will generate full-resolution bitmaps of each printed page. These bitmaps are captured by the Canon driver and written to the default Output directory in common image file formats.

⁹ A print ad from the time says “For hard copy output, you can add our laser printer for just \$18,000.” In *PERQemu*, it’s free!

Canon Runtime Options

The Canon driver offers two options that may be changed at any time, even when the virtual PERQ is running: output format and paper size. If the printer is busy, changing these settings will not take effect until the current print run completes.

Use the command `settings canon output format` to choose PNG or TIFF; JPG is not currently implemented. Additional formats may be offered in a future release. TIFF is the default.

The advantage of PNG is that compression makes the files very small, while uncompressed TIFF images are much larger. However, PNG can only store one image at a time, while TIFF files may store multiple images in a single file. When a multi-page document is printed, the Canon driver queues them in memory until the last page is printed then saves them all at once. The “tumble” utility (see Appendix C for a link) can convert a *PERQemu* TIFF file into a PDF using CCITT Group 4 lossless compression, which brings the file size down considerably.

Both Canon models support four paper cassettes: US Letter, US Legal, A4 and B5. *PERQemu* implements all four, allowing you to select the page size used when saving the output. *However*, there is no support for these in any of the standard PERQ software!¹⁰ As far as the standard microcode is concerned, *all* output is sized to print on US Letter (8.5” x 11”). Limited testing has shown that the output driver also produces reasonably good output if A4 is selected.

PERQemu limits the paper cassette to 100 sheets, just like the hardware does, and counts down with each successfully printed page. When it reaches zero the printer stalls until the paper is replenished. (This helps prevent runaway RAM consumption if a large document is printed all at once, but paper cassette capacity may be increased or made into a runtime parameter if necessary.)

You can set the default paper to be loaded using the `settings canon paper size` command. This size is loaded by default every time the virtual machine is started. To change the paper size when the virtual machine is running, the `load paper` command is available. With no arguments, `load paper` just refills the cassette in the current size; otherwise 100 sheets of the specified size are loaded. The `NoCassette` option behaves just like removing the input tray from a real printer: if a print is in progress it is interrupted and an alert symbol is shown on screen.

¹⁰ *There are several third-party PERQ applications that can produce Canon output, but they have not been tested yet. This guide will be updated as further tests are done and possible future enhancements extend the flexibility of the Canon output driver.*

Multi-page Documents

Neither the Canon Video Interface Specification nor the PERQ microcode provides any documented, reliable method to determine the end of a page, let alone the end of a complete document. The entire apparatus is based on keeping a steady flow of data from the controller to the print engine to maintain its rated throughput of approximately 8 pages per minute.

Thus, the emulator simulates the timing of the printer's mechanical parts (rotating drum, paper feed, fuser warm-up time, etc.) to determine the start and end of a print cycle. When in "standby mode," a print request sets the machinery in motion, taking 5 seconds to bring the drum and laser up to speed. Once ready, the printer receives and saves each page as it is completed (if PNG output is specified) or adds it to a queue (for TIFF). When the PERQ stops sending data for more than 5.7 seconds, the real printer stops the moving parts and transitions back into its low-power standby mode. At this point *PERQemu* writes the entire queue as a single multi-image file (for TIFF output). If another separate document is started immediately after one completes, the emulator may combine them into one file.

The new `PrinterSpeed` rate limiting option can be turned off to limit the startup delay and reduce the timeout before returning to standby mode. Due to specific handshaking and timing constraints that the microcode expects during the active printing of each page, turning off rate limiting does not measurably decrease the actual time to print (approximately 6 seconds per page), but does save around 10 seconds in startup/standby delays.

The Canon emulation is still undergoing testing and refinement, so it may change prior to the next major release.

Managing the Virtual Machine

A typical session with *PERQemu* involves a few basic steps:

1. Choose a configuration to run;
2. Assign storage devices if necessary;
3. Power on the virtual machine;
4. Interact with the PERQ through the Display window;
5. Power off the virtual machine;
6. Reload the same (or load a different) configuration, rinse and repeat;
7. End the session by quitting the program.

The first three steps can be automated by writing commands into a script, so a predefined configuration may be selected and started up immediately. (See *Automating Actions with Scripts*, later in this chapter.)

When you first start *PERQemu* a default configuration is selected. Your preferences file will remember the last configuration you ran and make that the default on subsequent runs.

Starting and Stopping the PERQ

Once you have selected a valid configuration and assigned at least one bootable storage device, the emulator is started by the “power” command¹¹:

> **power on** ← *loads the machine but does not start it running*

This commands the emulator to assemble the virtual PERQ according to the current configuration. Assuming the configuration is successfully set up, the assigned storage device images are loaded into memory. If any errors occur during startup, the console will display an error message to alert you to the problem. (Refer to the section *Checking for Errors* above for resolving configuration problems.)

Once the configuration is loaded, a large display window titled “PERQ” is created. At this point the virtual PERQ is “warming up,” waiting for a command to start execution:

<p>> start</p>	<p>← <i>starts the CPU executing at ROM address 0x0000</i></p>
--------------------------	--

¹¹ *A graphical interface that was in development featured a front panel that looks like the one used on the PERQ-2 machines, with a power switch, reset switch, and diagnostic display. Hopefully someday that old WinForms code will be rewritten and incorporated into PERQemu.*

You may instead choose to use debugging commands, load custom microcode into the control store and jump to it, single step through the boot ROM or Z80 code one instruction at a time, or interrogate the state of the machine. Some limited configuration changes may also be made. Type “start” at any time to begin (or resume) execution.

As in previous versions of *PERQemu*, this shortcut will `power on` and `start` in one step:

> **go** ← *loads the machine and starts it running*

When it starts running, the PERQ first executes a complex bootstrapping operation to load an operating system. (See *Bootstrapping in Depth* below for more information about how the process works, and how to select an OS to load.)

To interact with the PERQ you must click on the display window. This directs your keystrokes and mouse movements/clicks to the virtual machine. At any time you may click on the console window to issue commands. To pause execution of the PERQ, select the console window and type:

> **stop** ← *pauses execution of the virtual PERQ*

At this point you can use debugger commands, save media, reset the machine, or simply pause execution to give your computer a break and go make a nice cup of tea. To restart from exactly where you left off, just type `start` again and away you go!

To shut down the emulation, use the console command:

> **power off** ← *stops (if running) and shuts down the PERQ*

Or you may simply press the close button on the PERQ's display window.

- ❖ *Warning:* Remember that *PERQemu* does *not* automatically save changes to modified floppies or hard disks unless you've told it to. When you power off the emulated machine without saving first, changes to the in-memory disk images are lost.

If you have requested that *PERQemu* ask for confirmation before unloading disks, a dialog box may appear over the display window giving you the option to save before closing; see the section *Working with Storage Devices*, above. See *Setting User Preferences*, below, for information about changing your autosave settings.

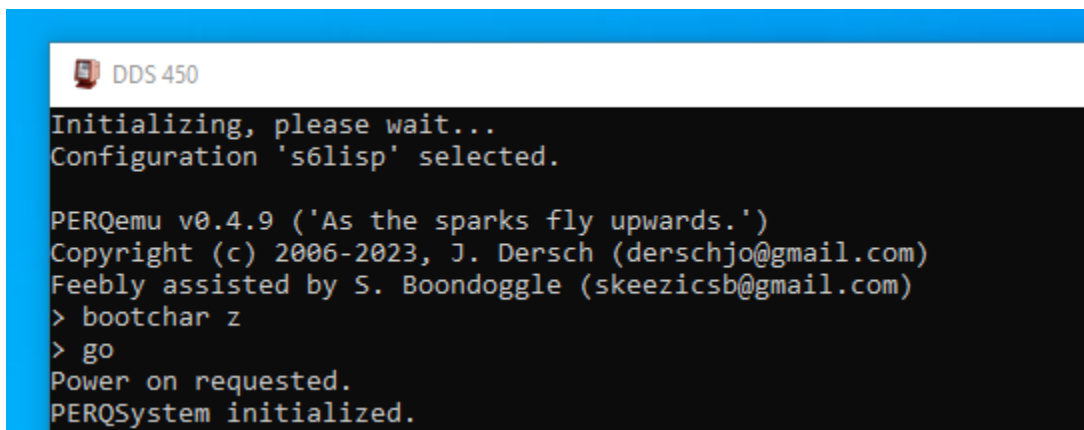
To exit *PERQemu*, as if you'd ever want to do such a thing:

```
> quit                                ← saves/asks if disks are modified, saves settings, exits
> quit without save                  ← exits without saving settings or modified disks!
```

The `quit` commands automatically `stop` the emulator first if it's running.

The Diagnostic Display (DDS)

When first powered on the PERQ it does its best to make you think it's broken, showing no output or flashing strange patterns on the display. Only upon successful loading will the operating system clear the screen and announce itself. PERQ hardware provides a 3-digit numeric display that shows the progress of the boot sequence; *PERQemu* currently emulates this by updating the console window's title bar to read "DDS" followed by a three-digit number:



```
DDS 450
Initializing, please wait...
Configuration 's6lisp' selected.

PERQemu v0.4.9 ('As the sparks fly upwards.')
Copyright (c) 2006-2023, J. Dersch (derschjo@gmail.com)
Feebly assisted by S. Boondoggle (skeezi@csb@gmail.com)
> bootchar z
> go
Power on requested.
PERQSystem initialized.
```

Figure 1: Console window with diagnostic display.

From 000 through 199 (roughly), a common set of diagnostic codes are used; values above 200 are specific to the operating system or diagnostic software being booted. When the boot is complete, the DDS will typically show:

```
PNX:    255
Accent: 400 (1MB of memory) or 450 (2MB)
POS:    999
```

Because *PERQemu* doesn't emulate the 2 minute warm-up time¹² that the old Shugart hard drives required, most hard disk boots will complete in just a few seconds. (Floppy boots, as on the hardware, take considerably longer.) There are several common milestones where it's normal for pauses to occur:

- 010 Basic ROM diagnostics complete; the microcode is attempting to load the next part of the bootstrap
- 030 The more extensive "VFY" microcode tests are running; this is accompanied by memory test patterns appearing on the screen
- 151 The system bootstrap microcode "SYSB" is waiting for a boot character selection

If the OS doesn't come up and your PERQ really is stuck, consult the *Fault Dictionary* in the Bitsavers on-line documentation repository to figure out why. See Appendix B, *Bibliography* for links.

Boot Selection

It is not uncommon for a PERQ hard disk to contain several different operating systems, or different versions of the same OS. Certain types of POS floppies may be bootable as well, to provide a means to install a new operating system or run recovery and repair programs if the hard disk is damaged or corrupted. Each bootable device stores a table of 26 possible "boot letters," or just "boots" for short. The letters a .. z select the hard disk; A .. Z select the floppy disk (if a bootable floppy is loaded).

On the hardware, you select the boot by holding down a key on the keyboard after power on, or after pressing the boot (reset) button. The reliable way to do this is to wait until the VFY memory test begins (when the comb-like patterns first appear on the screen) at DDS 030, releasing the key after DDS 151 comes up. This way works with the emulator too; just remember to click on the display window so the PERQ gets your key presses, not the console!

PERQemu also provides a simpler method (that can be scripted):

- > **bootchar** ← without an argument, shows the current selection
- > **bootchar c** ← sets the boot selection to any alphabetic character c

❖ *Note:* This is one command where the case of the argument is significant!

¹² If you're a real ~~glutton for punishment~~ stickler for accuracy, you can enable the `StartupDelay` as an option!

If the boot character is unset, or if you don't press anything while the machine initializes, the default 'a' boot from the hard disk is automatically selected. *PERQemu* remembers the last selected boot character only for the duration of your session; it is unset when you restart the program.

Bootstrapping in Depth

This section is optional deep background into the periphery of PERQ geekdom that may be skipped by the casual user.

A unique feature of the PERQ is its “soft” architecture: the machine contains almost *no* software!¹³ When first initialized, *PERQemu* loads a very small ROM image into the CPU board, and the I/O board's Z80 starts executing from its onboard ROMs. Everything else – an operating system kernel, language interpreter (which defines the instruction set), and I/O microcode – is loaded from a boot device. As mentioned above, the three phases of the microcode bootstrap are referred to as BOOT, VFY and SYSB (referring to the names typically given to the microcode used to implement them).

Only BOOT is encoded in the ROMs. At power up the ROM overlays the bottom half of the lowest 4K bank of the writable control store. It executes basic diagnostics to verify that enough of the processor is working to continue on to the next phase, reading VFY and SYSB into memory from external storage. The boot microcode tries to load from three¹⁴ possible devices, in sequence:

- Link board: A direct connection from a PDP-11 or another PERQ fitted with a special option board allows microcode to be loaded and executed under the control of a debugger. *PERQemu* currently only emulates enough of the link board to tell the microcode that nothing is plugged in;
- Floppy disk: The PERQ asks the Z80 to try to boot from the floppy drive. This probes to see if a POS file system floppy with a specific “signature” is loaded and online, and if so is selected to provide the next phase of the bootstrap;
- Hard disk: If no boot floppy is present, the hard disk is checked. Every PERQ hard drive has a dedicated boot area set aside (starting at cylinder 0) where the microcode for bootstrapping is written.

¹³ The author wryly acknowledges the irony of this statement. See Appendix A.

¹⁴ Code to support network booting and diskless operation of Accent has been found. It would be really cool to add this in a future release, once Ethernet emulation is complete. Netbooting uses ^A through ^Z for boot character selection, naturally.

If no boot floppy or hard drive is loaded, *PERQemu* (like the actual PERQ hardware) will simply loop forever. The DDS will usually display 010, indicating that the boot could not proceed. It may display 014 if there was an error loading from the device, potentially indicating a corrupted boot track.

VFY and SYSB are usually written into the boot area together; they are loaded as one image. The BOOT code first loads them into memory, then copies them into the high half of the control store. To indicate success, BOOT runs the DDS up to 029 when it hands off to the next stage. Once this is accomplished the ROM is disabled, and the only way to enable it again is by pressing the boot button (in *PERQemu*, using the `reset` command).

VFY's job is to run more extensive diagnostics of the processor and memory to prepare for the final stage of bootstrapping. It advances the DDS to 030, initializes the video hardware (which also performs DRAM refresh), then runs a series of memory tests – the patterns seen on screen are the contents of main memory while the test runs! Upon successful completion, VFY runs the DDS up to 149 then jumps to SYSB.

SYSB announces itself by setting the DDS to 150. It then resets the Z80 and polls the keyboard several times to see if a boot character is pressed, using the default 'a' if nothing is received. SYSB uses the boot character as an index into a table of 26 possible boots from the selected device's Disk Information Block. Two OS-specific microcode binaries with the extensions ".boot" and ".mboot" contain the kernel and I/O microcode, as well as the interpreter which defines the instruction set used by the system. POS and Accent run the "Q-code" interpreter, which supports Pascal, FORTRAN, C and Ada, while PNX runs "C-codes" which are optimized for C and Unix. These are read into main memory, and then SYSB writes them into the control store. The Z80 is then turned off again. (You may notice that the Z80 runs at 0.0ns in the FPS display when it is turned off.)

Finally, SYSB advances the DDS to 198 and jumps to the entry point for the operating system itself. From that point the bootstrapping process is complete, and the selected OS takes over all remaining initialization of devices, filesystems, video display, and so forth. Each operating system can then use the DDS however it pleases; POS shows most of the major steps it takes as it starts up and provides an extensive list of error codes (see the *Fault Dictionary* referenced in Appendix B) while PNX and Accent are quite terse. Accent is unique in that it displays a final code based on the amount of memory detected. And there are some special bootable diagnostics that make extensive use of the DDS to display test results.

"Fascinating," I hear you cry...

Automating Actions with Scripts

PERQemu can read scripts of commands at startup or at any time from the command line. When running the Debug version of the program, scripts may be attached to breakpoints and run automatically when a specific event is triggered.

The format is simple: any plain text file containing valid CLI commands may be read and executed.¹⁵ Note that the commands on each line must be fully specified; the interpreter will not expand or complete partial commands as it does when you type them interactively. To assist in the creation of scripts, a command to save the current history to a text file will be provided.

To run a script from the command prompt, preface it with an '@' character:

```
> @scriptfile
```

where *scriptfile* must be a qualified pathname, according to the host operating system's filesystem conventions. It may be absolute or relative to the directory where you launched *PERQemu*; currently there is no search list or file extension associated with loading scripts.

Limitations and Implementation Notes

Scripts may not be nested. Any line beginning with '@' in a script is ignored.

Blank lines, or comment lines starting with '#' are allowed – they're simply ignored.

PERQemu will always attempt to read the entire script, even when malformed, incomplete or failed commands are encountered.

While you may invoke a script at any point in the command hierarchy, scripts are always executed from the top-level prompt. You are free to navigate the hierarchy within the script, but when completed the current prompt is restored.

¹⁵ *The astute observer will notice that the definitions of storage devices, configuration files, and even the user preferences are all stored in exactly the same format... laziness is a programming virtue after all!*

Setting User Preferences

PERQemu offers a number of customizations that are stored between sessions in a preferences file. These are manipulated using the “`settings`” subsystem. Like other subsystems, type `commands` for a summary of the available commands, `show` to see the current settings, and `done` to return to the CLI top-level.

In the v0.7.5 release most (but not all) of the Settings commands are implemented. Please see chapter four, *Command Reference* for at least a cursory discussion of the current command set.

Autosave Options

As with prior versions of *PERQemu*, changes to disks made while the emulator runs only affect an in-memory copy; these must be saved if you wish to preserve them between sessions. For floppy or QIC tape media, the corresponding “`autosave floppy`” or “`autosave tape`” setting is applied any time a modified disk or cartridge is unloaded. For hard disks, the “`autosave harddisk`” setting is applied when the emulator is shut down with the `power off` or `quit` commands.

Choosing “`yes`” means *PERQemu* will always save modified media; “`no`” means changes are always discarded. “`Maybe`” means that a dialog will be presented only if the media is modified, allowing you to choose on a case-by-case basis. The default setting for all media types is `maybe`.

Device Assignment Options

The Settings “`assign`” and “`unassign`” commands allow you to map or unmap devices on the host for the emulator’s use. These are typically set up once and apply to all PERQ configurations. The default for all currently settable device mappings is unassigned.

See the section *Working with Communication Devices* earlier in this chapter for information about mapping serial devices to the PERQ’s RS-232 ports, or attaching an Ethernet host adapter to the PERQ’s Ethernet interface.

Performance Options

A goal of *PERQemu*’s emulation is to run a virtual PERQ at precisely 100% of the actual hardware’s speed, where the microengine executes one instruction every 170ns (5.88Mhz). Due to the way the PERQ’s video timing is intimately tied to its system clock, accurate CPU emulation leads to a display refresh rate of exactly 60 frames per second. *PERQemu* provides a number of options to affect how the emulator interacts with the host to achieve this target execution rate.

The “`settings rate limit`” command accepts a number of flags:

None	Do not limit performance; run the emulation as fast as the host is able. This clears all the other flags.
Default	Sets the rate limit settings to program defaults.
CPUspeed	Attempt to regulate the CPU to exactly 170ns per microcycle.
DiskSpeed	Accurately reflect disk drive mechanics for seek time, head settling, data transfer rates, etc. ¹⁶
TapeSpeed	Regulate the tape streamer at the hardware's 30 inches/second rate.
StartupDelay	Accurately reflect the disk drive spin-up delay at power on. ¹⁷
PrinterSpeed	Accurately reflect mechanical timing of the Canon laser printer.

The default settings are set for accuracy over speed. This doesn't impact performance much on hosts that can't run the emulation at full speed, but effectively locks in around 60fps on faster machines. While *PERQemu* has been run at an emulation rate in excess of 240fps (for a CPU speed of 23MHz!) with no observed incompatibilities or problems interfacing with peripherals, these purely empirical measurements are no guarantee that faster rates won't expose overruns, errors, or other timing issues.

Limiting hard disk speed with the `DiskSpeed` option is recommended for the Shugart hard disks, as seek timing is actually “baked in” to the Z80's firmware. Turning off the `TapeSpeed` limit allows the streamer to run at the 90 inches/second rate, rather than the 30IPS that the drive is rated for.

The Canon printer interface is driven by a combination of microcode, Pascal and a hardware state machine that requires specific timings to produce a properly formatted image. The `PrinterSpeed` option (on by default) tries to faithfully emulate the 8 pages-per-minute maximum throughput of the print engine. By turning off this option, *PERQemu* can shave a few seconds off of the timings to make print operations run a wee bit faster.

The emulator can be quite resource intensive on the host; *PERQemu* offers the “pause when minimized” setting to automatically stop emulation when the Display window is “iconified” and resume it when the application is brought back to the foreground. This *generally* works the same way on Windows, Mac and Linux hosts, where pressing the window's minimize button sends it to the dock or taskbar. The default for this setting is `true`.

¹⁶ Currently *PERQemu* does not enforce transfer rate limits or emulate the memory cycle stealing impact of DMA on CPU performance. “That would require an almost fanatical level of attention to detail,” he said, updating the to-do list.

¹⁷ This was useful as a debugging tool, actually, but waiting two minutes on every cold boot gets old really fast. But I left it in as an option if you want the full experience. Just need to play audio of the distinctive “Shugart rattle and squeal” when the SA4000 fires up...

On the hardware, the “boot button” immediately causes the processor to restart. *PERQemu* will optionally pause the emulation when a `reset` command is typed, allowing additional commands to be issued before manually restarting the processor with `start` or `go`. The default for “pause on reset” is `true`.

Miscellaneous Settings

The “`settings display cursor`” option allows you to specify how the emulator displays the system cursor when the Display window is selected. On some systems the “`crosshairs`” option is less intrusive than the default arrow; or you can turn off the system cursor entirely by choosing “`hidden`”. See chapter three, *PERQ Operations* for more information about how some PERQ operating systems handle mouse tracking.

By default *PERQemu* limits all of the output it generates to the `Output` subdirectory inside the installation directory. The “`settings output directory`” command allows you to specify another destination for trace logs, screenshots, printer output and other potential generated output. Trace logs are output here from Debug builds when file logging is enabled. Screen captures and bitmaps generated by the Canon laser printer are created with file names that should be obvious.

You can reset everything to default values with the “`settings default`” command, or undo changes that have not been saved by typing “`settings load`”. The “`settings save`” command applies any changes you make immediately, although this is automatically done whenever *PERQemu* exits (unless you specifically choose to quit without save).

Debugging Features

TBD. This is pretty “organic” and not well documented. See the *Command Reference* for the basics.

Highlights: logging/tracing, breakpoints, `:variable` syntax, etc. Most of these are implemented and working (well enough, with *plenty* of rough edges), though some are incomplete but aren’t needed in normal day-to-day use. Even in this rough state the emulator has proven invaluable for debugging microcode, though, which ought to appeal to the burgeoning class of fellow PERQ microcoders out there, right guys? Hello? *<tap tap>* Is this thing on?

3. PERQ Operations

Virtual Machine Interface

PERQemu's emulation environment translates keystrokes, mouse movements and other device interactions from the host to the specific protocols expected by the virtual PERQ being emulated. In general, the emulated machine behaves like a modern user would expect; this section describes a few areas that require special attention.

Special Keys

As described in the *Configurator* section of the previous chapter, *PERQemu* supports two keyboard types, based on the model selected. While the emulator hides the implementation details, there are several special keys that may not have direct equivalents or require remapping to work around host OS restrictions on modern keyboards:

PERQ-1	Host	PERQ-2	Host
HELP	F1 or HELP	HELP	HELP
OOPS	F2 or Ctrl-u	OOPS	CLEAR or Ctrl-u
INS	ESC or INSERT	INS/ESC	ESC or INSERT
DEL	DELETE	REJ/DEL	DELETE
LINEFEED	F3 or Ctrl-j	LINEFEED	PAGE DOWN or Ctrl-j
		SETUP	PAGE UP
		NOSCRL	SCROLL LOCK
		BREAK	SYS REQ
		PF1..PF4	F1..F4

Table 6: Special key mapping.

Todo: There *has* to be a way to map and save these in the settings! It's impossible to come up with one default mapping that works with Windows, MacOS and Linux. Even different Apple keyboard layouts assign/reserve different meanings to the function keys. An interactive key editor that saves assignments in the settings (or another) file is required.

The GUI “virtual keyboard” should be recreated in SDL2 as a platform-independent solution.

Mouse Input

Refer to the *Configurator* section in the previous chapter to review selection and configuration of a PERQ pointing device. In the discussion that follows, “mouse” will refer to the host mouse, touchpad or input device, while “puck” will refer to the PERQ's simulated 4-button BitPad puck, or the 3-button Kriz tablet puck (which looks like a mouse).

If that isn't confusing enough, the “system cursor” is the configurable cursor image that follows your mouse as you interact with the emulator, while the PERQ provides its own cursor image based on the specific software being run.

In *PERQemu* the mouse works approximately as you'd expect, though there are functional differences between a digitizer tablet and a modern mouse:

- the PERQ can use absolute tracking, while a mouse only provides relative input;
- a digitizer sends a steady stream of data, while a mouse transmits movements as they happen.

Unlike modern graphical interfaces where the system provides a cursor that is almost always active and visible, some PERQ operating systems leave tracking and using the tablet entirely up to the application being run. When enabled, the cursor may be tracked in *absolute* mode, where the position of the puck or stylus on the tablet maps directly to a coordinate on the screen, or in *relative* mode, where a series of small swipes in one direction moves the cursor relative to its last position (like a modern mouse).

To provide simulated absolute mode positioning, *PERQemu* tracks the position of the cursor based on its location in the Display window, sending coordinates 40-60 times per second based on the tablet type selected. In this mode the host cursor and active “hotspot” of the PERQ's cursor will track together. When you click outside of the Display, the emulated machine sees the cursor at the last position tracked as the mouse leaves the window; it then jumps back to the position where you click to return focus to the Display.

When operating in relative mode, the PERQ detects when the puck is off the tablet and calculates cursor movement based on the start and end points of the next mouse movement. To simulate the “off tablet” behavior, hold down the Alt key (Option key on the Mac) while moving the mouse. You can then reposition the system cursor in the window before releasing the key to start a new swipe.

Button Mapping

Early PERQ operating systems were written to support the 4-button puck or 1-button stylus¹⁸ attached to a Summagraphics BitPad One. When Three Rivers introduced their own Kriz tablets with a 3-button puck, the fourth button (“blue”) had no equivalent, so any function attached to that button had to be revised or a keyboard-initiated alternative provided. *PERQemu* maps the typical modern 3-button mouse in the expected way, but provides a modifier key to produce all of the BitPad buttons as shown in the figure below.

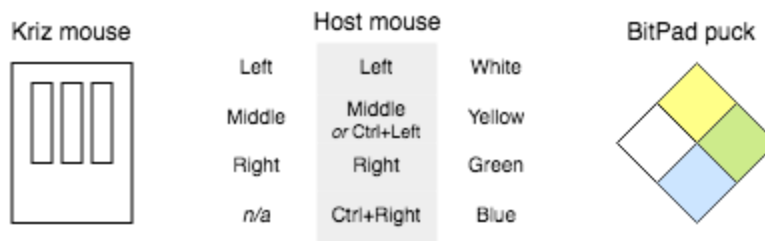


Figure 2: Host to PERQ mouse button mapping.

Floppy Formats

PERQemu doesn’t “know” anything about the contents of a floppy image; it only presents it as a series of blocks to the virtual machine through a simulated floppy controller chip issuing commands to the simulated floppy disk drive. The only time the emulator peeks at the data sectors is to look for the “boot signature,” a specific byte pattern that identifies a bootable diskette.

There are three primary PERQ floppy formats you may encounter:

- DEC RT-11-compatible floppies for data exchange;
- POS filesystem floppies;
- PNX floppies.

The RT-11 data floppies are common to almost all PERQ operating systems and can be used for data interchange. To access these, each OS provides a utility described in the sections below. *PERQdisk*, an

¹⁸ Early on the PERQ supported a 1-button pen-like stylus attached to the BitPad. If anyone out there wants to use a 1-button Apple mouse or a Wacom tablet and stylus, contact the author. It might not be too difficult to add support for emulating the stylus?

external, standalone utility for reading and writing PERQ floppies is now available. It takes advantage of the *PERQmedia* library and can handle all supported floppy image formats; see Appendix C, *Additional Resources* for details.

POS filesystem floppies are designed to act like a small hard disk; they can be mounted, booted from, and interacted with from POS (and MPOS) using all the standard utilities. All bootable PERQ floppies are double sided, single density and contain a POS filesystem.

PNX floppies are *deeply* mysterious. Not much is known about them, but they are not to be trifled with. PNX boot floppies were seduced by the dark side of the format, twisted and *eeevill*; they seem to have a tiny POS partition so the boot ROM/Z80 can initiate the boot load, followed by what is presumably a Unix filesystem used to mount the “miniroot” needed to bootstrap the PNX installation. Other PNX install floppies are also mountable, but it’s unknown how these are created. There may also exist (undocumented) “tar” formatted floppies used for data interchange?

Hard Disk Formats

PERQemu is as agnostic about hard disk formats as it is about floppies; their contents are opaque to the emulator. All PERQ hard disks, however, use a unique sector format, supported by the controller and by *PERQmedia*: each 528-byte sector contains a 16-byte logical header for filesystem metadata, plus a standard 512-byte data block. The emulator does not operate at the physical level, emulating gaps and CRCs and the like, but it is able to flag bad blocks (typically from unreadable or damaged parts of archived physical disks).

POS, MPOS and Accent share a common on-disk format and use the logical header data in a consistent manner. A hard disk can contain several partitions shared by one or more of these compatible operating systems; partitions are generally limited to a maximum of 32,768 blocks (16MB) each. Software limitations are discussed in the original OS documentation. See the *Bibliography* for links.

PNX, however, uses an incompatible on-disk format assumed to be based on the Unix V7 filesystem, though details are not currently known (and no “magic cookie” seems to provide a means to uniquely identify the superblock or format). It is not clear if PNX takes advantage of the logical header.

The *PERQdisk* utility can read POS filesystems from hard disk images; see Appendix C, *Additional Resources* for more information.

Tape Formats

As of v0.4.8, *PERQemu* emulates the one supported streaming tape drive option. The Archive Sidewinder 3020I provides 20MB of storage on DC-300XL 1/4" cartridges (QIC). A library of PERQ tape images is being assembled and should soon be made available for use with the emulator; this allows for much simpler OS installations, for instance, as one tape can take the place of 20 double density floppy disks!

POS and Accent use a program called *Stut* to read and write backups of POS filesystems at the partition level. An external port of *Stut* designed to read and extract files directly on the host is being developed/updated to work with *PERQmedia*; it may be available at the time of the next *PERQemu* release. See Appendix C for more information.

PERQ tape images may be stored in the SIMH magtape format, loaded from ".tap" files. They may also be saved in the native *PERQmedia* ".prqm" format, with labels and other metadata. All streamer tapes use a fixed 512-byte data block.

Todo: Investigate whether SIMH provides any utilities for reading and writing real QIC tapes directly to and from .tap files;
See if the PERQ version of "tar" works with the streamer, or only with 1/2" 9-track magtapes
Is there any streamer support in PNX?
Finish the *Stut* port and start archiving tapes!

Supported Operating Systems

The PERQ can run a number of different operating systems. Efforts are ongoing to preserve old media archives and make them available in a form suitable for use with *PERQemu*. As of version 0.4.2, the following operating systems are known to boot:

- POS versions D.6, F.0 and F.1 (official 3RCC releases);
- POS version F.15 (released by Boondoggle Heavy Industries, Ltd);
- MPOS version E.29 (unreleased by 3RCC);
- Accent S4 (an early version from CMU, unreleased);
- PNX 1.3 (first public release by International Computers Ltd.).

PERQemu v0.5.0 introduces support for the CIO board and “new Z80” ROMs. This allows PERQ-1 configurations to run:

- POS version G releases through G.7 (official 3RCC releases);
- Accent S5 and S6 (Release I and II from 3RCC/CMU);
- Accent S7 (was to be Release III; later available from Accent Systems?).

Versions as of *PERQemu* v0.5.3 also support:

- PNX 2.0 (second release by ICL) for the PERQ-1/CIO.

PERQemu v0.5.8 also supports:

- PNX 2.0 and 3.0 (early access release media) for PERQ-2/EIO, 8” Micropolis disk.

Note that the available media set for PNX 3 is incomplete; only the T1 boot floppy works, so a hard disk installation is not yet possible. Patches to correct the video display list problems in ICL’s non-standard microcode PNX 2 and 3 are applied automatically.

With the *PERQemu* v0.7.0 release, the following OSes also boot but are not yet widely available as hard disk images have not yet been successfully produced:

- POS version G.85 (unreleased, *incredibly* rare);
- PNX 5.02 (final ICL release for the PERQ-2/T2).

In the sections below, some basic information about each tested OS is given to help users new to the PERQ get started exploring the growing library of disk images, demos, games and applications.

POS

The basic PERQ Operating System, or POS, is a simple “no frills” single-process DOS-like OS written in Pascal. It provides the baseline development tools and programming environment for the machine. While relatively unsophisticated by modern standards, it does offer pretty much unrestrained access to the programmer: POS lets you get down to the bare metal and doesn't get in your way.

Availability

There are quite a few complete POS media distributions in the wild and most of them have been archived and made available for use with *PERQemu*. The version naming/numbering scheme is simple, with a letter representing the major release, and a number the minor release. In general, the last version of each major release was the most widely adopted; D.6, F.1 and G.6 were the releases from Three Rivers most likely to be encountered. POS F.15, G.7 and other variants are far less common.

As distributed, *PERQemu* includes a collection of hard disk images that are ready to run. Many are referenced by the predefined configurations, selected as the most representative POS vintage for the given hardware:

d6.prqm	The original <i>PERQemu</i> disk image archived by Josh Dersch. POS D.6 is the default boot; includes Accent S4 as ‘z’ boot.
f0.prqm	A disk image from Skeezics’ PERQ-1. POS F.0 is the default boot; MPOS is ‘q’ boot (more on MPOS below).
f1.prqm	A fresh install of the basic POS F.1 system from floppy. Some additional applications and demos are included.
f15.prqm	An updated, expanded variant of POS F beamed in from an alternate reality. Details and source media available at the <i>PERQmedia</i> GitHub archive.
g6mic.prqm	A new install of POS G.6 for the PERQ-2 (8” Micropolis) with added games, demos, applications and source code! Supports Portrait or Landscape.
g6mfm.prqm	Same POS G.6 as g6mic, but for the PERQ-2/Tx (5.25” Micropolis).
g7.prqm	A new install of the unreleased POS G.7 with a few demos and applications. Also includes Accent S6 as ‘z’ boot.

Unless called out as “mic” or “mfm”, the hard disk images are the Shugart 24MB type, supported by the PERQ-1.¹⁹

¹⁹ As the library grows, the number of disk images bundled with the emulator may be reduced to slim down the basic package, moved instead to the *PERQmedia* archive where they can be downloaded separately.

Configuration

Like most OS environments on the PERQ, the POS taxonomy is split into the “old Z80” and “new Z80” eras. In terms of hardware/emulator support, this means:

Old Z80 ↑	POS D ²⁰ and F releases run only on the PERQ-1 with the IOB, one Shugart hard disk, and the portrait display
	POS D.6 and F.0 releases can only use the BitPad tablet; they don't support the Kriz
	POS F.1 and F.15 can support both tablet types
New Z80 ↓	POS G requires a PERQ-1 with the CIO board or any PERQ-2 configuration
	POS G supports all hard disk types; on the PERQ-2/Tx a second 5.25” hard drive can be provisioned
	POS G also supports both tablets <i>and</i> both display types!
	POS version G does <i>not</i> run on the 24-bit processor ²¹

Table 7: POS hardware requirements.

Login and User Interface

All versions of POS indicate 999 on the DDS when they have successfully initialized. A banner is then printed that shows the available disk partitions, followed by a prompt for the date and time. This can be entered in the requested format, or you can just press RETURN to accept the default (which is the timestamp saved the last time the machine was shut down and the disk saved). For EIO PERQs, the on-board RTC chip may be read to automatically set the time for you!

POS then prompts for a username. In all of the disk images supplied with *PERQemu* the default “guest” account with a blank password is provided. Press RETURN at the login prompt to continue.

Interaction with POS is through a command-line interpreter called the Shell. Commands are not case sensitive (but are shown here in upper case as is the POS tradition); the filesystem is case preserving but is not case sensitive. Here are some basic commands to get you started:

²⁰ POS D.5 has been recovered! No versions of POS prior to this are known to have survived, however.

²¹ POS G.7 and G.85 have been recovered; they might run on the 24-bit CPU but not use its extended features. [To be confirmed.]

?	List the commands defined in your “login profile”
HELP	Show usage for various commands
DIR	Show contents of the current directory. Executable files are suffixed with <code>.RUN</code> and directories are suffixed with <code>.DR</code>
DIRTREE	Display and navigate the filesystem hierarchy graphically
PATH	Change directory. Often aliased to <code>CD</code> , more or less like your Unix or DOS equivalent. Note that the directory delimiter is ‘>’ (this may freak out Unix heads!) PATH <code>foo>bar>baz></code> – change to directory <code>foo>bar>baz</code> PATH <code>..</code> – change to parent directory PATH <code>sys:user></code> – change to root of <code>user</code> partition
SETSEARCH	Add or remove directories from your search path
TYPE	Display a file on the screen, similar to “more” on modern OSes
COMPILE	Run the Pascal compiler. Produces a <code>.SEG</code> file which must be linked
LINK	Run the linker. Produces a <code>.RUN</code> file which may be executed
COPY	Copy a file from one location to another
EDITOR	A fairly simple line-oriented text editor with some mouse support
PEPPER	A more Emacs-like editor, available in several of the POS images

To run a program, just type its name (you can leave off the `.RUN`). Switches are prefaced by ‘/’ as was common to many operating systems of the era. Commands and switches are not generally case-sensitive, and can usually be abbreviated as long as they are not ambiguous. Most commands in POS will prompt you for arguments if you don't provide them, and most commands won't do anything harmful without asking you first.

In general, pressing `Ctrl-C` twice will abort a running program, unless it's hung.

- ❖ *Note:* Control characters on the PERQ are case sensitive! A single `Ctrl-C` is an interrupt; a second `Ctrl-C` signals an abort. A `Ctrl-Shift-C` is used to break out of command files and return you to the Shell. The Pepper editor makes extensive use of shifted control characters to make up for the lack of a “meta” key that Emacs requires. Quirky, no?

The Shell also provides a pop-up menu facility. At the prompt, you can click anywhere on the screen to bring up a menu of programs to run; the current selection is highlighted in reverse video. A small scroll bar at the bottom of the menu lets you move the menu up or down if there are more commands than will fit; the cursor changes to a small scroll icon as you hover over this area. Press and hold the mouse button and move left or right to scroll the menu. To make a selection, click once and the

highlighted command is executed by the Shell; click anywhere outside the menu to abort the selection. Note that switches or arguments cannot be provided when the menu is used.

POS doesn't impose much structure on the filesystem; it mostly lumps everything into one big `sys:boot>` directory and lets you sort it out however you like. (In later releases where Accent is bundled, they sometimes used `sys:pos>` instead.) By convention, there are usually games under `sys:user>games>`, demos under `sys:user>demo>`, and user directories are generally kept in `sys:>user>`. Like DOS, there aren't really many filesystem protections; you can scribble files wherever you want! Things are arranged a little bit differently in POS F.15; "type welcome.txt" to learn more when you first log in.

Full POS documentation can be found on Bitsavers in PDF form; see Appendix B for links. The original text versions are also in `sys:user>doc>` in the F.1 hard disk image, or `:boot>docs>` in F.15; documentation included with POS G to be determined as those images are completed.

File Transfer via RSX:

The POS `COPY` command allows you to upload text files from the host into the running virtual machine, or copy files from the running PERQ to the host. This simple transfer method is built-in to nearly every version of POS and follows the same syntax:

```
> copy POSfile ~ RSX:hostFile      ← from the PERQ to the host
> copy RSX:hostFile ~ POSfile      ← from the host to the PERQ
```

There are some notable limitations:

- POS always converts the file names to uppercase; this may not matter if your host uses a case-insensitive filesystem but may cause difficulties when uploading if you don't provide an upper-case filename;
- This is a simulated transfer over the serial port at 9600 baud, so it's not terribly fast! Error checking is minimal, and because it's an ASCII transfer you may have to check the resulting file for missing characters being stripped or line endings being changed.

Note that *PERQemu* does its best to automatically convert Unix-style line endings (LF only) to POS style (CRLF) when uploading, since otherwise the file transfer wouldn't work. When downloading, POS does not strip the carriage returns out so you may need to use an editor or command-line utility such as `dos2unix` to reformat the result.

File Transfer using RS232

POS includes a *very* simple serial communications program called `CHATTER` that has limited file transfer capabilities. To use the serial port you must configure the host side (see the section *Working with Communication Devices* in chapter two) and enable it in your configuration. `CHATTER` should default to RS232-A on all machines, and may allow use of port B on PERQ-2 configurations.

In addition, a slick, full-featured terminal emulator called `JET` exists, and a nifty-but-incomplete version of the popular `Kermit` utility is available as well. These are available in the floppy archive but will hopefully soon be included in the standard distribution as well. Neither has yet been tested extensively with *PERQemu*.

File Transfer using Floppies

The POS `FLOPPY` program is used to create and transfer files from DEC RT-11 formatted floppies. *PERQemu* supports double-sided and double-density operation, with the most typical format storing around 500KB (double-sided, single density, abbreviated “DSSD”). The program itself has extensive on-line help, and even provides pop-up menus for some commands. To prepare a new, blank floppy for use from the emulator, refer to the section *Working with Storage Devices* in the previous chapter.

Once a blank floppy is loaded, you can prepare it for use using `FLOPPY`'s “ZERO” command. This simply recreates the directory structure on the floppy, erasing any files that already exist. You can also use the “FORMAT” command to completely erase the media first, but that's not strictly necessary given that *PERQemu*'s `storage create` command does that step for you – and *much* faster. Floppies created in RT-11 format can be used with external utilities to read and write files outside the emulator, providing a more convenient method for moving binaries or larger files than the `RSX:` method.

RT-11 Floppy Limitations

- For compatibility with RT-11, files are named using the limited Radix-50 character set; you can use upper case only with few special characters in a strict “6.3” naming scheme (think early PC DOS and you won't go far wrong);
- The date format used stores a very limited range and is aggressively *not* Y2K-compliant; when prompted to enter the date when storing files, you can only specify years from 1972 through 1999!
- Files are allocated and stored in contiguous sectors – if the disk becomes fragmented you might have to move files around to coalesce the free space to make room for a larger file. `FLOPPY` can do this with its “COMPRESS” command... *slowly*.

POS “filesystem floppies” use a different approach altogether. These have to be created using the `PARTITION` program, which creates a filesystem on a floppy that looks and acts just like a small, slow hard disk to POS. Filesystem floppies can be made bootable by running `MAKEBOOT`. They must be “mounted” for use (the “`MOUNT FLOPPY`” command), at which point you can use the normal POS file and path syntax for accessing them. You can even run the disk “`SCAVENGER`” program to repair them. Always dismount them with “`DISMOUNT FLOPPY`” when finished before saving or ejecting.

File Transfer using Tapes

POS versions F and G support a utility for backing up and restoring hard disks called `STUT`. This Streamer Utility operates at the partition level, allowing for bulk data transfer for backups (or OS installations) of up to 20MB per cartridge. `STUT` may not be present in some of the bundled hard disk images; source and binary copies are available so it can be added where needed. `STUT` has its own inline help and is fairly simple to use.

Laser Printing

The POS utility `CPRINT` drives the Canon laser printers. The POS F.15 and G.6 hard disk images bundled with *PERQemu* now include this, and source code is available in the floppy archives. Updates to the other pre-built disk images will be included in a future release.

Logout and Shutdown

You can log out of POS by typing “bye” to the Shell. POS will flush any cached disk buffers, record the current time, then log you out. It returns to the Login program to allow you to start a new session. Or, at this point, you can safely shut down the emulator.

The PERQ-1 hardware can do “soft” power! Type “bye off” and POS will tidy up as usual, then power down the machine. (This was *really* cool back in 1981.)

PERQemu handles the power down signal the same as pressing the close button on the Display window. Depending on your `autosave` settings, you may be prompted to save any unsaved changes to your loaded disk(s) before the emulator closes up shop and returns to the CLI.

MPOS

A rare and unreleased POS version E.29 was recovered. Known as MPOS, or the “multi-process” version of POS, it features a more complete window manager and can run multiple processes using a mostly-compatible POS API. We even have source code for this funky thing! Have to play with it some more to really get the hang of how it do what it do, then provide a clean formatted Shugart24 and the floppies!

Availability

In a future release a clean install from scratch onto a freshly formatted hard drive will be bundled (and included in the *PERQmedia* GitHub repo). In the interim, a working copy of MPOS E.29 is provided with a POS F.0 (dump of my own PERQ's hard disk):

```
f0.prqm    MPOS is 'q' boot!
mpos.prqm  A full source install of the rare MPOS E.29 release. [TBD]
```

Configuration

MPOS supports the same hardware as POS F; the default configuration works. It can use 2MB of memory. Kriz tablet, serial ports, and floppy access have not been tested.

Login and User Interface

Most interactions with MPOS are the same as POS D/F, so only the differences need to be highlighted here. The window manager needs some introduction.

Logout and Shutdown

The “bye” command offers most of the same options as POS. You can type `bye off` to log out and power down the PERQ.

Todo: Highlight those differences
 Clean install, new media, uploaded to Github and bundled!
 Window manager!
 Device interaction? Do it do the RSX thang?
 See if the recently found Ethernet code works.

Accent

Accent is a message-based, multi-tasking OS microkernel developed in the early 1980s by Carnegie-Mellon University as part of the *Scientific Personal Integrated Computing Environment* (SPICE) Project. Successor to RIG and precursor of Mach, Accent on the PERQ refers to both its microkernel and the overall operating system environment, consisting of a set of servers designed to be distributed over a network. *PERQemu* can run several versions of this interesting and groundbreaking operating system. Please see Appendix B, *Bibliography*, for links to a wealth of documentation.

Availability

An early release of Accent S4 from CMU is included in the bundled `d6.prqm` disk image. This is an amazing and rare find, as it pre-dates the official releases from Three Rivers/PERQ Systems in late 1984-early 1985. Accent S4 is 'z' boot on this image.

An Accent S5 system, including SPICE Lisp version M2, is being prepared. It should be available and documented here in a future *PERQemu* release.

A fresh installation of Accent S6, paired with POS G.7 and the Accent demo, is now available and included in the standard distribution as `Disks/g7.prqm`. A version of this disk with Spice Lisp M3 is now included as `Disks/s6lisp.prqm` (with a predefined configuration file for convenience in `Conf/s6lisp.cfg`). Accent S6 is 'z' boot on both of these images.

Accent S7, the final unreleased version known to exist, runs on the PERQ-1 as of v0.4.8; in v0.7.x it also runs on the PERQ-2, and will support the 24-bit T4! Disk images will be forthcoming *RSN*.

Configuration

All versions of Accent run best with more memory configured. As *PERQemu* only supports the 20-bit CPUs at this time, that means a maximum of 2MB with a practical minimum of 1MB (though older versions will boot with less, if you insist). On a PERQ-1 24MB Shugart disk with limited swap space, more RAM is better, *especially* if you want to try out the landscape display. With each new release the OS is faster and more reliable, and hardware support improved.

The S4 version of Accent only runs on the PERQ-1/IOB/Shugart configuration. It requires the BitPad (doesn't support the Kriz tablet) and portrait display, but it *can* run with the full 2MB of memory. While S4 *may* support the 4K (PERQ-1) CPU, you should generally configure the 16K (PERQ-1A) CPU. Load the `AccentV4` Q-code set for accurate Q-code disassembly.

A sample dialog for loading and running Accent S4:

```
> configure default
> configure assign d6
> configure memory 2
> debug load qcodes accentv4
> bootchar z
```

Accent S6 runs on the 16K CPU, CIO or EIO, and all disk types; currently only the `g7.prgm` image is bundled. Performance is best with 2MB memory configured. It supports both tablet and display types! Selecting the Kriz tablet gives a slightly more responsive interaction with the window manager, as its binary protocol is more efficient than the ASCII data stream sent by the BitPad. To configure:

```
> configure load perqla
> configure assign g7
> configure memory 2
> configure tablet kriz
> debug load qcodes accentv5
> bootchar z
```

Login and User Interface

Accent's startup is very different from POS. After the usual boot sequence, the DDS will stop after the kernel determines the amount of memory configured (400 for 1MB, 450 for 2MB). The screen is then cleared and a banner printed announcing the OS version and several startup messages. Accent will first prompt you to enter the date and time:

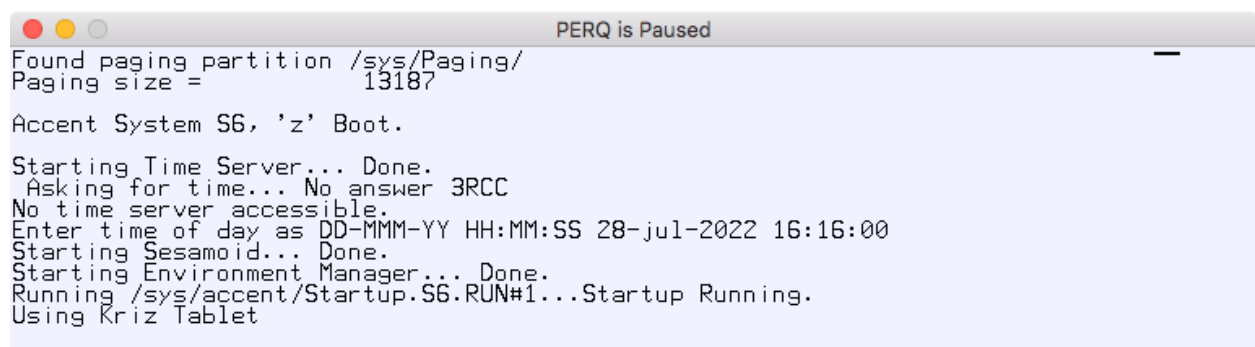


Figure 3: Accent startup screen.

Note that you can enter a 4-digit year; the time routines are Y2K compliant! *However*, you should type carefully when entering date and time, and *only* use the backspace or OOPS key if you make a mistake. One less-than-endearing “quirk” is that a badly formatted date string will just crash the time server and force you to reboot.²²

The interface to Accent is provided by the *Screen Allocation Package Providing Helpful Icons and Rectangular Environments* (SAPPHIRE), an awesome 8-letter acronym and window manager that unifies the native POS-like shell environment, a Common Lisp environment, and a Unix-compatibility layer called Qnix.²³ As it initializes, SAPPHIRE clears the screen to a stippled gray background pattern and draws several windows as it runs through a series of command files to start up the rest of the environment. Once everything is initialized Accent's Shell behaves very much like the POS Shell, with many of the same commands and options. Unlike POS, Accent provides command-line editing, scrollbar, multi-tasking and “real” window management (uncovered windows redraw themselves).

Some hints for getting started with Accent:

- SAPPHIRE tracks the mouse in relative mode; the Alt/Option key is your friend!
- Moving or resizing windows requires a modal type of interaction that may feel a bit clumsy at first to those used to the “direct manipulation” paradigm common today;
- You have to click to select a “listener” window to direct where your typing will go – the listener window has a gray border around it;
- You can use the basic Emacs-like control keys to retrieve previously typed commands and edit them (`^p/^n` for previous/next, `^f/^b` forward/back, etc.);
- You can scroll back to see text that scrolled off the screen (`^V/^v` for backward/forward by roughly a page full of text); as with POS, control characters are case sensitive!
- Interrupting or aborting the current program goes through the window manager, so `^c` or `^C` doesn't work as in POS.

Window manager commands may be invoked by mouse click or the keyboard. Keyboard commands are prefixed by the SETUP key (`Ctrl-Del` on PERQ-1), then a letter. For example, “`Ctrl-Del h`” brings up a command summary. There are also pop-up menus, and the icons change depending on

²² Most Accent machines were connected to Ethernet networks where a time server would provide the time, eliminating this manual step. It does not appear that Accent will read the current date/time from the RTC on PERQ-2/EIO boards, unfortunately.

²³ Work is ongoing to make these additional Accent environments available.

what SAPPHIRE wants you to do (make a window, move a window, etc). To interrupt a program, use “Ctrl-Del c”, or “Ctrl-Del k” for added *oomph*.

The icons in S6 are the set provided by Three Rivers/PERQ Systems, and may not match some of the CMU documentation, but their meanings should be intuitive. Like the POS D.6 installation on the “d6” disk image, someone replaced the default S4 cursor with a silly Opus icon; clearly a *Bloom County* fan used that machine once. In early Accent, the default cursor was an image of a sapphire ring.

Using the Accent Shell

One important change from S4 to S6 is the transition from POS-style syntax to a much more Unix-like approach. In S6 the form changes, but the underlying structure of the filesystem remains compatible:

```
S4:    dev:partition>dir>file
S6:    /dev/partition/dir/file
```

In addition, S6 uses the single hyphen (‘-’) to introduce command switches:

```
S4:    dir /fast /sort=size
S6:    dir -fast -sort=size
```

Type “help” at the shell prompt for information about Accent’s command syntax. Please also refer to Appendix B, *Bibliography* for links to more Accent documentation.

Laser Printing

Accent also uses the CPrint software, but it is not currently bundled.

Logout and Shutdown

There is no “clean” way to logout of Accent S4, so the best way to make sure you flush buffers to disk (if you want to save your work) is to type “trap” at the Shell. This will bring up the kernel debugger which will scribble in reverse video over the top of the screen. From there type ‘y’ to exit to the pager process, and at that point there’s no going back; switch to the *PERQemu* command line and type stop to pause execution, save disks, then power off or reset as desired.

In Accent S6 a “bye” command is provided. If you plan to save your work, it’s important to run this before you save the hard disk; Accent S6 is much more aggressive with caching disk blocks in memory

to improve performance, and the `bye` command makes sure modified data is flushed before shutdown. It also prints a message to tell you when it's safe to turn off or reboot the PERQ.

Limitations

Accent S4 gets *very* cranky if you try to interact with the window manager before it finishes initializing. When booting S4, let it complete *all* of its command files before switching windows or trying to start up additional shells or programs. S6 is a bit more robust, but the same advice applies.

Accent does not support the `RSX`: serial pseudo-device.

Accent S6 *does* support the streamer utility, `STUT`. More testing is needed to document its use and interoperability with POS.

Interim Ethernet Support

Designed as a distributed, networked OS, all versions of Accent require an Ethernet interface to be present. A “fake” Ethernet driver (added in v0.4.9) allows the Accent Net/Message Server to properly initialize. This means that S6 can launch its `FloppyServer` and `RS232AServer` processes, making it possible to load data from floppies and access the serial port.

All PERQ-2 configurations use the EIO board and contain a built-in Ethernet interface. To run Accent on a PERQ-1, add the following commands to your configuration:

```
> configure option board oio  
> configure option add ether
```

If these files exist:

```
/sys/accent/10MhzMsgServer.S6.DIS  
/sys/accent/10MhzNetServer.S6.DIS
```

they should be renamed to have the `.RUN` extension (and you'll need to `reset` the PERQ to launch them). The `g7.prqm` disk image bundled with v0.4.9 has been corrected so that the Net/Message Server starts up at boot time. The older S4 installation included on the `d6.prqm` disk image has also been updated in v0.7.5 to launch its Net/Message Servers, though it is still a bit “touchy.” (Hopefully improved Ethernet support over time will enhance the Accent experience across all versions.)

Live Ethernet Support

In *PERQemu* v0.5.5 the OIO Ethernet emulation was introduced; EIO Ethernet followed in v0.5.8. This means that Accent can now communicate with other emulated PERQs to do remote filesystem access, authentication, time service, remote execution, a messaging/"chat" service, FTP, and more. Later versions of S6 and S7 even incorporate early TCP/IPv4 support; experimental features like network booting, live process migration between nodes, and even distributed demo programs wait to be explored.

As discussed in the earlier *Working with Communications Devices* section, even a relatively quiet switched Ethernet connection on a modern host generates a constant stream of traffic that easily overwhelms the PERQ's 10Mbit, half-duplex interface. To help alleviate this and give Accent a fighting chance to operate on busy networks, *PERQemu* provides filtering at the host interface to throw away IPv6, spanning tree, or other types of packets that the PERQ can't do anything with. It is hoped that a platform-independent way of using Win/NetPcap filters to limit the traffic flow at the host interface can be devised, but the current advice is to assign a private host interface to the emulator and hook up virtual PERQs on a small hub or switch with *no* other hosts attached. Many Macs and PCs built in the last 10 years have dual Ethernet jacks; obviously those extra, unused interfaces are meant to be used by *PERQemu*. (Even with TCP/IP turned off on a Mac OS X host, it seems the kernel – running Mach! – still sends traffic on unconfigured Ethernets. Sigh.)

In summary: There are tantalizing hints at what Accent can do once the emulator's Ethernet interface is more fully debugged and functional!

Todo: Get S5 and Lisp M2 booting! See if MatchMaker will run too
 Install the C environment for S6, and Qnix/Spoonix if available
 We might have an S3 image!? And S7!
 Ethernet, Ethernet, Ethernet!
 Holy crap, the Interactive Debugger runs on Accent! Centipede too!

PNX

PNX, another unfortunately named PERQ operating system, is an early Unix V7/System III mashup that includes an in-kernel window manager – possibly one of the earliest Unix variants to do so? This was primarily run on PERQs in the UK, where PNX was developed by ICL. Because the on-disk format is based on the Unix filesystem, it cannot co-reside on a disk with POS or Accent (which share a common underlying filesystem layout).

PNX also uses its own language interpreter, running an instruction set more favorable to C than the original PERQ Q-codes. However, *PERQemu's* debugger cannot accurately disassemble PNX C-codes, since we don't currently have access to any PNX source code or much documentation.

Availability

A Shugart 24MB disk image of PNX 1 from the *PERQmedia* Github repository has been converted to the new (compact) PRQM disk format and bundled with *PERQemu* for v0.5.0; in v0.7.5 PNX 2 has been bundled as well:

<code>pnx1.prqm</code>	A full installation of PNX 1.3
<code>pnx2.prqm</code>	A mostly-complete installation of PNX 2.0 (PERQ-1/CIO)
<code>pnx2mic.prqm</code>	An installation of PNX 2.0 for PERQ-2/EIO (8" Micropolis)

As with Accent, PNX became more capable and complete with each release, so there is great interest in bringing the new versions up on the emulator. Versions 3 and 5 will *likely* require the EIO/PERQ-2, but more research is needed to understand their hardware requirements. Currently we do not have access to PNX 4, if that even existed.

Configuration

The early 1.3 release of PNX has the same restrictions as early POS:

- Requires the old Z80 configuration: PERQ-1, IOB, Shugart hard drive, BitPad tablet, portrait display;
- It supports a maximum of 1MB of memory. If you configure 2MB, PNX will fail to initialize and will drop into its debugger, display an "@" prompt and only respond to a few cryptic (and undocumented) commands.

PNX 2.0 now boots on the emulator as well, with a slightly different set of constraints:

- Requires the new Z80: PERQ-1/CIO/Shugart, or PERQ-2/EIO/Micropolis;
- Portrait display only?
- Kriz tablet support available, but requires manually updating `/dev/tablet`;
- The full 1 megaword (2MB) memory option is supported;
- This version has a number of improvements over the 1.x release but is missing some components (manual pages, notably).

There is currently no extra software for these early versions of PNX besides the very basic install images. However, PNX 5.02 does include some demos/apps, *and* there is some exciting news about a possible software archive unearthed in the UK; watch this space!

Login and User Interface

PNX 1.3 clears the screen once the DDS reaches 255 and prints a banner to announce itself:

```
mem = 844288    M=1.4 B=1.1 K=1.3

Microcode Version 1.4
Bootstrap Version 1.1
Kernel Version    1.3

single user
login: <^Z>
```

At this point you may enter single-user mode by entering a login and password (default user `root`, default password `root`), or you can press `^Z` to switch to multi-user mode. PNX 2 skips single user mode and directly proceeds to run `fsck`. Next, both versions prompt you to enter the date and time:

```
type date in form : 4 nov 5:20 or nov 4/5:20 or 11040520 etc.
use two digit numbers (or field separators) in day, hours, mins order

set the date [dd mmm hh mm] : 12 oct 1:03
Wed Oct 12 01:03:00 GMT 1983

are date and time correct ? y
```

Note that *it's always 1983 in PNX land!* In PNX 2, it is (ominously) always 1984.

Once entered and confirmed, the screen is cleared again and the normal multi-user prompt is displayed:

```
mem = 844288    M=1.4 B=1.1 K=1.3

International Computers Limited
PNX Operating System
Release 1

login: root
password: root    ← The password is not displayed
```

Once logged in you are presented with the familiar ‘#’ prompt (as the root user). To start the PNX window manager, type “winit”. The figure below shows a basic display:

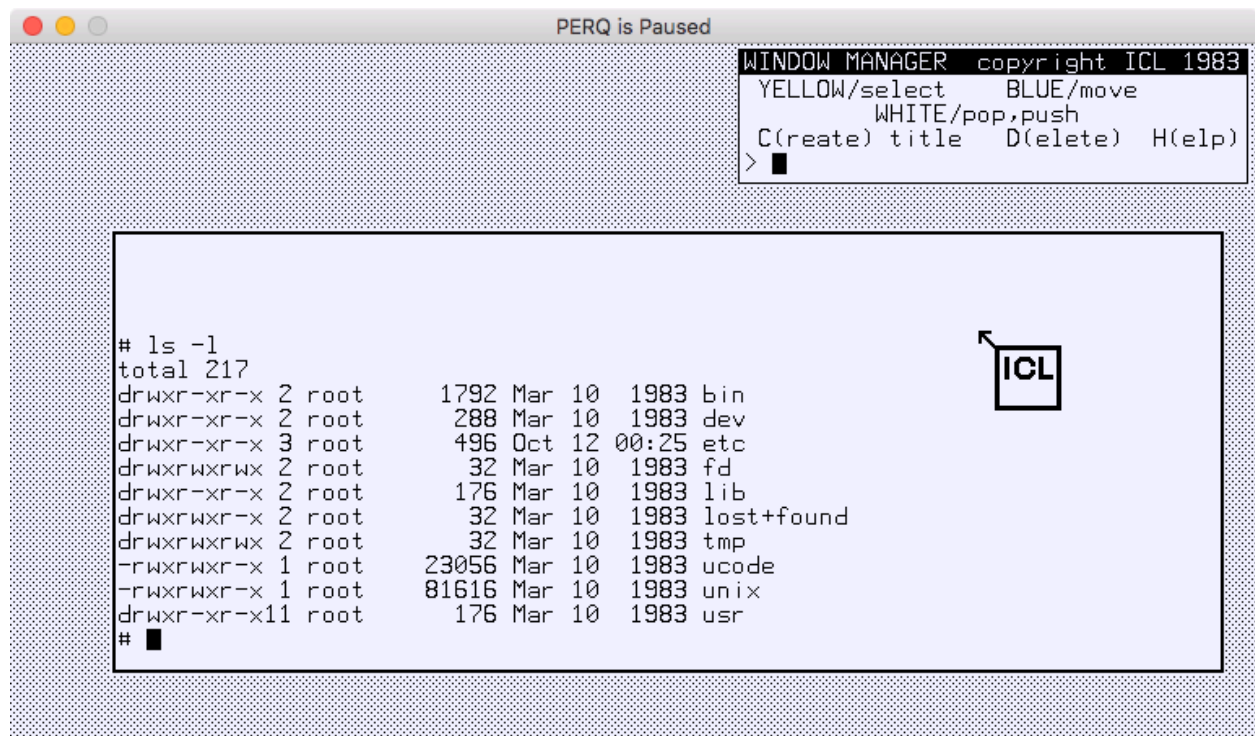


Figure 4: PNX 1.3 graphical environment.

Note that your typing is edited in a small text area at the bottom of the screen! When you press RETURN the input line is sent to the selected window (and echoed there). Use the middle button on your mouse to select the active window; its title bar is then highlighted in bold.

You may also find that output to the window pauses when it's about to scroll off the screen; there seems to be an implicit "more" mode that (unprompted) waits for you to press the SPACE bar to continue.

- ❖ *Author's note:* I haven't explored this version much, and have only a rudimentary understanding of the window manager's operation. Some basic hints are provided here, but hopefully someone with more PNX 1 experience will chime in with additional pointers to documentation or tutorials!

Logout and Shutdown

PNX 1 doesn't provide a standard Unix `halt` or `shutdown` command. You can close a window by pressing ^Z at the prompt; the window manager will shut down when the last window is closed, returning you to the basic blank, non-windowed screen. From there, "`kill -1 1`" will signal the `init` process to drop you back to single user mode. Then you can "`sync; sync`", wait a few seconds, and stop the emulator.

In PNX 2 (and beyond) there's a "`bye`" command that works sort of like its POS equivalent. Note that it doesn't play around; type it (as `root`) and you'll soon see "PNX CLOSED DOWN" and the virtual machine must be reset or powered off.

Todo: Play with the window manager more and document start-up, interaction Applications! Hope to find some, someday...
 Compile Mario Wolczko's Smalltalk for PNX!
 Would *really* love to find a listing of the C-codes somewhere
 T1/EIO/PNX 3 boot floppy works! The rest are all missing trk0 and must be reimaged :-(
 Ultimately aim for T2/EIO/PNX 5/landscape/dual drives to see how it *really* performs :-)

Future Possibilities

Two additional PERQ operating systems exist, but they are exceedingly rare and have not yet been tested. However, thanks to recent efforts, media archives now exist (or may, soon):

- HCR Unix for the PERQ;
- PERQ FLEX.

An early attempt to bring Unix to the PERQ, a set of HCR Unity floppies has been recovered. It is believed that this was a port of Xenix, but it was abandoned and never shipped as a product. There are no installation instructions and it's not clear whether the set is complete. Chances are slim to none that we'll ever get this running on the emulator, but it is an interesting historical find.

PERQ FLEX is an OS produced in the UK by the *Royal Signals & Radar Establishment*. Based on a custom instruction set optimized for Algol-68, efforts are underway to recover several disk images and preserve what promises to be a unique bit of computing history. If successful, this could provide a real 8" Micropolis hard disk image of a PERQ FLEX system.

Of course, if these operating systems are able to be recovered and run under *PERQemu* this document will be updated accordingly!

More to come...

Demos and games!!

Applications

Dev tools and languages

4. Command Reference

Below is an itemized description of the command syntax for *PERQemu* v0.7.5. Command arguments are illustrated using the following typographic conventions:

<i>argument</i>	a required argument supplied by the user
(this that)	a required argument from a list of specific choices
[<i>argument</i>]	an optional, user-supplied argument
[word...]	an optional word (or words) that extend or modify the command
[...]	a subsystem prefix; may be typed by itself or with additional arguments

Type “commands” at any prompt to see a summary for the current subsystem. Type “done” to return to the previous level in the hierarchy.

Top-level Commands

about

Print the program's startup banner with version, copyright and contact information.

bootchar [*char*]

With no argument, show the current boot character (if set). With a character argument, set the desired boot character to *char* (case is significant). If unset, the PERQ defaults to ‘a’ boot.

configure [...]

Prefix for configuration commands. Typed by itself, enters the Configuration subsystem.

debug [...]

Prefix for debugging commands. Typed by itself, enters the Debugging subsystem.

go

A shortcut for the sequence “power on” followed by “start”.

gui

Not (yet) implemented.

help [keyword]

Show the (currently limited) on-line help. With no *keyword*, prints a general help message. Still a work-in-progress and likely to change in the next release of *PERQemu*.

history

Show the last 99 commands entered.

load (floppy | harddisk | tape) filename [unit]

Load the supplied *filename* as the current floppy, hard disk or streamer tape using the default unit numbers. If the emulator is running, removable media is brought online immediately. Updates the current configuration. A unit number may be specified when loading a hard disk in a PERQ-2/Tx system. See also: *Working with Storage Devices* in chapter two.

load paper [size]

If the Canon laser printer option is configured, reload the paper cassette. If *size* is not specified, the current cassette is refilled, otherwise 100 crisp new sheets of the specified type are loaded. If NoCassette is chosen, the paper tray is immediately *removed*, potentially interrupting any printing currently in progress. See also: *Working with Output Devices* in chapter two.

power (on | off)

Use “power on” to validate the current configuration, load the assigned media and start the emulator if it isn’t already running. Type “power off” to shut down the emulator, optionally giving the opportunity to save modified media (according to your autosave preference settings).

quit**quit [without save]**

Exit *PERQemu*. If the emulation is running, a power off is done implicitly. By itself “quit” offers the chance to save any modified media (if configured to do so by Settings) and updates any unsaved Settings changes. If “without save” is specified, *PERQemu* shuts down and exits immediately; unsaved changes are lost.

reset

If the emulator is running, “reset” simulates the press of the PERQ’s boot button to trigger a hardware reset of the virtual machine. The PERQ will then restart execution or pause, depending on user preference. See also: Settings “pause on reset” option.

save (floppy | hddisk | tape) [as filename [format]]

Commit changes to a modified media image using the same format and filename it was loaded from. Optionally “as *filename*” creates a copy of a loaded image and updates the current configuration with the new filename. The copy is created in the same format as the original image, or can be transcribed into *format* if specified. See also: *Working with Storage Devices* in chapter two.

save history

Not yet implemented.

save screenshot [filename]

Take an immediate screenshot of the current display, optionally saving it as *filename*. If no name is given, an obvious name is generated and the file saved to the default output directory. The screen is captured at its native resolution (100 dpi, 1 bpp). You can have any output format you like, so long as it's PNG. See also: Settings “default output directory” option.

settings [...]

Prefix for user preference commands. Typed by itself, enters the Settings subsystem.

start

Start or restart emulation when paused. The machine must already be powered on.

status

Print a summary of the emulator's current state. This is currently a rough debugging command and its output will be enhanced or modified in a future release.

stop

Pause emulation when running. Pressing ^C in the console window or function key F8 when the Display window has focus will also pause the emulator.

storage [...]

Prefix for storage commands. Typed by itself, enters the Storage subsystem.

unload floppy

If the emulator is running, this commands the PERQ to unload any loaded floppy media (giving the opportunity to save changes, if modified). Also updates the current configuration.

unload harddisk [*unit*]

Remove a hard disk from the current configuration. If *unit* is not specified, unload the default unit #1. Fixed hard disks cannot be unloaded when the emulator is running; hard disks with removable “packs” are not yet supported. Updates the configuration.

unload tape

If the QIC Tape option is configured and loaded, this commands the PERQ to unload the cartridge. The default (only) unit #3 is selected. Updates the current configuration.

Configuration Commands

From the top level menu, “configure” enters the Configuration subsystem. See *The Configurator* in chapter two for information about creating, saving and loading configurations.

assign filename [unit]

In configuration mode, *filename* is attached to the appropriate unit number based on the type of the device contained in the file. Optionally the *unit* number may be specified, useful to disambiguate which hard drive to assign in dual-drive PERQ-2/Tx configurations.

chassis type

Select the basic PERQ model from a list of supported *types*.

check

Validate the current configuration to make sure the machine is a valid PERQ that can be emulated. *PERQemu* will print warnings about (but allow) rare or experimental selections that may not have adequate software support, or indicate errors where configuration options are in conflict or are unsupported and will fail to boot.

cpu type

Select the processor to emulate from the list of supported *types*.

default

Reset the current configuration to the built-in default.

description string

Set an optional one-line textual description of the current configuration. The parameter *string* must be surrounded in quotes (“”) if it contains spaces.

disable rs232 [port]

Disable the use of a host RS-232 device. If *port* is not specified, ‘a’ is assumed.

display (portrait | landscape)

Select the format of the PERQ’s display. See also: the chapter *PERQ Operations* for restrictions on operating system support for the landscape display.

enable rs232 [port]

Enable the use of a host RS-232 device. If *port* is not specified, 'a' is assumed. See also: Settings "assign rs232 device" command; the section *Working with Communication Devices* describes how the emulator supports serial ports.

ethernet address address

Set the low word (last two octets) or "node number" of the PERQ's Ethernet address. May be any 16-bit value, but (by convention, and not enforced initially) the value should be in the range 1..65534, reserving the all-zeros and all-ones addresses. If set to zero, *PERQemu* will generate a random address.

io board type

Select an I/O board for the PERQ. The *type* specified must be compatible with the chassis. See the section *The Configurator* for details.

list

Print a list of the configurations available in the `Conf` directory.

load config

Load a configuration named *config*. *PERQemu* presents a selection of named configurations from a scan of the `Conf` directory at startup. You can also type a full pathname to load a *config* from another location.

memory capacity

Configure the PERQ's memory capacity in kilobytes (256, 512, 1024, etc.) or in megabytes (1, 2, 4, 8). *PERQemu* will round up the given argument to the nearest power of two, which must be valid for the configured CPU type. If *capacity* is not given, a helpful usage message is printed.

name string

Name the current configuration. The *string* specified should be a short, unique designation suitable for use as a filename. *PERQemu* will prepend `Conf/` and add the ".`cfg`" extension automatically. See also: the "description" command above; the section *The Configurator* for details of how *PERQemu* saves and loads configuration files.

option board type

Select an optional I/O board for the PERQ. If *type* is "none" the I/O Option slot is left empty.

option add *option***option remove *option***

For some I/O Option boards, specific peripheral controllers may be independently enabled or disabled. Currently *PERQemu* supports a subset of options. See also: chapter two, *Configuring Sub-options*.

save

Save the current configuration if it has been modified. It must have an assigned name; see the “name” command, above.

show [*config*]

With no arguments, print a formatted description of the current configuration. If specified, *config* may be selected from the list of known configurations, or a filename; *PERQemu* will attempt to print the contents without disturbing the current configuration.

tablet (bitpad | kriz | both | none)

Select the type of digitizing tablet to attach to the PERQ. Choosing “both” attaches both the serial Kriz tablet and the GPIB BitPad to the PERQ, but the specific operating system loaded will decide (or allow you to specify) which one it uses to track the mouse. You can also choose “none” but it isn’t advised, unless greatly reduced functionality or waiting through endless I/O timeouts is your particular kink. See also: *The Configurator* section for more information about pointing devices; the chapter *PERQ Operations* for information about operating system support for the different tablet types.

unassign *filename***unassign unit *unit***

Remove a media file from the configuration. The specified *filename* or the specific *unit* is unassigned.

Storage Commands

The “storage” subsystem provides commands that let you create blank media, define new types, or query the status of currently loaded storage devices. The default database of known PERQ media types is quite extensive, so some of the more esoteric commands are not (yet) documented. The most useful ones for end users are listed below.

create *driveType* *filename* [overwrite]

Create a new blank, formatted floppy, disk or tape image of *driveType*, selected from the given list of defined media types. A *filename* must be specified; if just a “simple” name is given, *PERQemu* automatically supplies the default `Disks` directory and adds the appropriate file extension. By default *PERQemu* will not clobber an existing file, but will if *overwrite* is set to `true`.

This command always creates new media files using the PRQM formatter, in a not-so-subtle attempt to drive adoption of the new format.

define *driveClass* *driveType*

Define a new storage device in the given *driveClass* with the name *driveType*. This subsystem allows for the dynamic creation of new geometries, performance data, and media types at runtime. It is not intended for general use.

list

List known storage devices.

show *driveType*

Show detailed device specifications for *driveType*.

status [*unit*]

Show detailed status of a loaded storage device (when the machine is running). If *unit* is not specified, a summary of all currently attached drives is printed.

Debugging Commands

The “debug” prefix provides a rich set of debugging commands. The Release build of *PERQemu* provides a subset of the total command set, mostly limited to interrogating the state of the emulator and the virtual machine. The Debug build of the program provides a much wider range of tools, including some that can modify the running state, but runs *much* more slowly.

For convenience, the Debugging subsystem provides the following execution commands:

go reset start status stop

These are the same as the top-level commands that are documented above.

All debugging commands that investigate the state of the active emulator require that the PERQ be powered on. As of this writing many of these commands are in a state of active development, or their use is fairly esoteric or directed at particular debugging scenarios. Most typical users won't need to explore these, so documentation is currently incomplete and their use may have unintended consequences. Careful with that ax, Eugene!

clear breakpoint *type* *watchpoint*

Clear a breakpoint of type *type* with the trigger *watchpoint*.

[Debug builds only]

clear interrupt *irq*

Clear a specific CPU interrupt, if asserted; *irq* is selected from the list of available interrupts.

clear logging *category*

Clear logging for certain events. *category* is selected from the list of available logging categories.

disable breakpoints

Disable breakpoint processing. Defined breakpoints are retained but ignored until enabled. Note that breakpoints are disabled by default when the emulator starts.

[Debug builds only]

disable console logging

Disable logging of debug messages to the console. Only normal command interactions are output.

disable file logging

Disable logging to files.

[Debug builds only]

disassemble microcode [address] [length]

Produce a formatted disassembly of the current contents of the microstore. If no *address* argument is provided, output starts with the current microprogram counter. If no *length* is given, the default is 16 microinstructions.

dump item

Show internal state of a number of internal devices, states, queues, or other developer-specific information. Some of these commands *may* be available in Release builds, but all of them are subject to change or removal and should be considered experimental.

[Debug builds mostly]

edit breakpoint type watchpoint

Change or reset a defined breakpoint. *type* is from the list of available types, and *watchpoint* is the value for the trigger (an address, port, etc.). This command enters a subsystem that allows modification of the breakpoint's properties; these are not yet fully documented.

[Debug builds only]

enable breakpoints

Enable breakpoint processing.

[Debug builds only]

enable console logging

Enable debug logging to the console. See also: the *Debugging Features* section for a complete breakdown of available logging categories and severities.

enable file logging

Enable saving debug logging messages to files in the `Output` directory. Currently up to 100 files of up to 10MB each are saved; the number, naming, destination directory and size of the log files will be made configurable in a future release.

[Debug builds only]

find memory address value

Find a specific *value* in the PERQ's memory starting at *address*.

[Debug builds only]

inst

Run one Q-code, then pause execution. This command executes microinstructions until the PERQ executes a `NextOp` (AMux select) or `NextInst` (jump instruction); it may execute one or more Z80 instructions as well. See also: Appendix B, *Bibliography* for links to documentation about the PERQ microengine.

jump *address*

Start or resume execution at the given *address* in the control store.

load microcode *binfile*

Load a microcode binary into the control store. You may want to manually `stop` the emulator before loading new microcode! (*PERQemu* should, but doesn't currently, prevent you from doing this.)

load qcodes *codeset*

Load Q-code definitions for opcode disassembly. Choose *codeset* from the given list of known versions. See also: the chapter *PERQ Operations* for more information about which operating systems use which Q-code versions. **Note:** the loaded Q-code set is *only* used for accurately disassembling instructions for output/debugging, and in no way affects the execution of the emulated PERQ.

raise interrupt *irq*

Assert a specific CPU interrupt *irq* from the given list of interrupt types. Note that this is highly operating system specific, and generally harmless, but the boot process will likely crash if you inject stray interrupts prior to OS establishment.

reset breakpoints *type*

Reset breakpoint counters for all breakpoints of the given *type*.

[Debug builds only]

set breakpoint *type watchpoint*

Set a breakpoint of a particular *type*. The value for *watchpoint* is specific to what you wish to trigger; a memory address, I/O port, interrupt type, etc.

[Debug builds only]

set console loglevel *severity*

Set the threshold for console logging to the *severity* level from the given list. In Release builds, only messages from the selected categories at or above the severity level are printed; in Debug builds, all Warning, Error and Heresy-level messages regardless of category are included automatically.

set execution mode (asynchronous | synchronous)

Set the execution mode for the virtual PERQ. The default is “asynchronous,” meaning the main CPU and Z80 run on separate threads; “synchronous” mode runs both processors on the same thread.

[This command may be removed in a future release.]

set file loglevel severity

Set *severity* threshold for file logging. *[Debug builds only]*

See also: the “set console loglevel” command above.

set logging category

Enable debug logging for a specific *category* of events, from the provided list. Currently just one set of categories is used for both console and file logging; this may be made specific to each in a future release.

See also: the *Debugging Features* section for a great deal more information about logging.

set memory address value

Write a specific 16-bit *value* into the PERQ's memory at *address*. *[Debug builds only]*

set xy register reg value

Change the value of XY register *reg* to *value*. *[Debug builds only]*

show

Print a summary of current debugger settings.

show breakpoints [type]

Print the status of defined breakpoints of the given *type*. *[Debug builds only]*

With no arguments, *type* defaults to “all”.

show cpu registers

Display the values of the basic CPU registers.

show cstack

Display the contents of the Am2910 call stack. For the 16K CPU, the extended stack is also shown.

show estack

Display the contents of the 16-level hardware expression stack.

show execution mode

Show the execution mode for the virtual PERQ. See also: “`set execution mode`” above.

show memory *address* [*length*]

Dump the PERQ's memory starting at *address*. If not given, *length* defaults to 64 words (128 bytes).

show memstate

Dump information about the memory controller's state.

[Debug builds only]

show opfile

Display the contents of the opcode cache and current Byte Program Counter (BPC).

show variables

Show debugger variables and their descriptions. The contents of the variables can be printed using the “*:var*” syntax as described in the section *Debugging Features*.

show xy register [*reg*]

Display the value of the XY register *reg*. If no argument is given, dump all 256 registers.

step

Run the next microinstruction, then pause execution. This single steps the main processor by one microcycle; the Z80 may or may not execute one opcode.

z80 [...]

Enter the Z80 debugging subsystem.

Z80 Debugging Commands

The “`debug z80`” prefix enters the Z80 debugging subsystem. Currently the most useful feature is a source-level disassembly of the instruction stream (when single stepping); the CLI doesn't offer much more, yet. The available commands are:

clear breakpoint *type watch*

Clears a Z80 breakpoint previously set with `set breakpoint` (see below). *[Debug builds only]*

dump *item*

Show internal state of a number of Z80 internal devices, states, queues, or other developer-specific information. Some of these commands *may* be available in Release builds, but all of them are subject to change or removal and should be considered experimental. *[Debug builds mostly]*

edit breakpoint *type watch*

Edits a Z80 breakpoint to set or change flags. Operates like the PERQ debugger's breakpoint editor (with slight differences); not all *types* are implemented. *[Debug builds only]*

inst

Run one Z80 opcode, then pause execution. The PERQ will execute an equivalent number of microinstructions so that the two processors remain in sync (given the disparity of execution rates).

reset breakpoints *type*

Reset Z80 breakpoint counters for all breakpoints of the given *type*. *[Debug builds only]*

set breakpoint *type watch*

Set a breakpoint in the Z80's execution. This operates much like the PERQ's debugger, but the categories are mapped to the Z80 context. This feature is incomplete and not well documented; support for all breakpoint types is added as needed to debug specific instances during emulator development. *[Debug builds only]*

show breakpoints [*type*]

Print the status of Z80 breakpoints of the given *type*. *[Debug builds only]*
With no arguments, *type* defaults to “all”.

show memory *address* [*length*]

Display contents of Z80 memory starting at *address*. If not given, *length* defaults to 64 bytes.

show registers

Display contents of the Z80 registers.

top

Return directly to the top level menu. (Use “done” to return one level to the debug menu.)

Settings Commands

The “settings” prefix allows you to tailor *PERQemu* to your system and to your preferences. At first release, a subset of the configurable settings are supported.

assign audio device *hostDevice*

Map a host audio interface to the PERQ's emulated “speech” output device. *[Not yet implemented]*

assign ethernet device *hostDevice*

Map a host Ethernet interface to the PERQ's emulated network device. For Ethernet configuration options refer to chapter two, *Working with Communication Devices*.

assign rs232 device *port hostDevice [baud] [charBits] [parity] [stopBits]*

Map a host serial interface *hostDevice* or the pseudo-device RSX: to a PERQ RS-232 *port* (‘a’ or ‘b’). You can optionally specify the baud rate, bits per character, parity and stop bits for the host device; defaults are 9600, 8, None and One. See also: the “configure enable rs232” command above; chapter two, *Working with Communication Devices*.

autosave (floppy | harddisk | tape) (yes | no | maybe)

Set the preferred action to take when modified media is unloaded. See also: chapter two, *Working with Storage Devices*.

canon output format (raw | png | tiff | jpeg)

Set the preferred format for output from the Canon laser printer. These files are written to the default output directory at the printer's resolution (see below) and centered on the chosen paper size. PNG files use a lossless compression and are much smaller, but contain only one page; TIFF files are not compressed and are approximately 1.1MB in size (per page). These conform to the TIFF 6.0 specification and should be easy to convert to PDF through an external utility.

JPEG output is not currently implemented. “Raw” format is literally just a dump of the bitmap for the page as written by the microcode, useful for debugging. *[Other formats may be added in a future release; raw format may be withdrawn.]*

canon paper size (nocassette | a4 | b5 | uslegal | usletter)

Set the default size of the paper cassette for the Canon printer. Choosing `NoCassette` effectively takes the printer offline; this mechanism may be used to enable simulating “manual feed” in a future enhancement. While these are the four sizes of input tray listed and supported by the Canon hardware, the PERQ software only currently supports `USLetter` (8.5” x 11” / 216mm x 279mm).

canon resolution (240 | 300)

Set the output resolution for the Canon laser printer in dots per inch. The LBP-10 is 240 dpi; the LBP-CX is 300dpi. No other resolutions are allowed.

default

Reset all program settings to defaults.

display cursor (crosshairs | defaultarrow | none)

Change the system cursor when in the PERQ Display window.

load

Reload saved settings, discarding any unsaved changes.

output directory *dir*

Set directory for saving debugging logs, printer output and screenshots. Currently incomplete; does not yet require that *dir* exists nor will it create it for you. *[Work in progress]*

pause on reset (true | false)

If true, the emulator is paused after a `reset` command is issued.

pause when minimized (true | false)

If running when the PERQ Display window is minimized, the emulation is paused and restarted when the window is restored.

rate limit *option*

Set performance *option* flags. If set to “none” *PERQemu* does **not** regulate the speed of the emulated CPU, peripherals or video and runs as fast as your computer can manage. If set to “default” the emulator strives to exactly match the 5.88Mhz rate of the processor and the 60fps display rate, with the

Z80 running at the speed defined by the selected I/O board. The *option* flags are independent, so you may want to accurately emulate the CPU/video but speed up some peripheral accesses, for example.

save

Save current settings.

show

Show all current settings.

show ethernet devices

Show available host Ethernet adapters.

unassign audio device

Unmap a host device.

[Not yet implemented]

unassign ethernet device

Unmap a host Ethernet adapter (equivalent to choosing the “null” device).

unassign rs232 device *port*

Unmap the host device from PERQ serial port *port*.

Appendix A: A Brief History of PERQ

First publicly demonstrated in 1979, the PERQ's design was heavily influenced by the legendary Xerox Alto and the family of "D-machines" from Xerox PARC in the 1970s. Sometimes referred to as the "Pascalto," the machine's most distinguishing feature is its high-resolution, non-interlaced bitmapped display, supported by high memory bandwidth and a custom microcoded processor. Although early production problems delayed volume shipments until 1981, PERQ was arguably the first *commercially available* workstation-class computer to meet the ["3 M" criteria](#). Its hardware specifications were broadly in line with the requirements of Carnegie-Mellon University's SPICE Project proposal, where it was used as the development platform for the Accent kernel.

One of the strengths of the PERQ is its extremely flexible microarchitecture. A microprogrammer can program the "bare metal," even tailoring the instruction set to whatever environment is desired. The downside to this flexibility, of course, is the difficulty it posed to OEMs who had to write nearly *everything* from scratch, as POS provides little more than a thin MS-DOS like single user shell. While a number of factors contributed to the PERQ losing its lead in the workstation race of the early 1980s, the lack of a standard software base – specifically Unix during its meteoric rise in popularity – is generally cited as the primary failing. By the time PNX and Accent (with a System V compatibility environment called Qnix) came along, other Unix vendors like Sun Microsystems had surpassed the PERQ solely on the strength of their software support. While a PERQ T2 could hold its own against the competition in raw graphics performance, contemporary Motorola 68k-based workstations were quickly pulling away in overall speed and capacity.

By the time Three Rivers Computer (renamed in 1983 to PERQ Systems Corporation) went bankrupt in 1986, it is estimated that approximately 4,000 PERQs had been built, mostly sold to universities or a few OEMs building turnkey pre-press, page layout or document management systems. Only a handful of systems are known to exist today in museums and private collections.

PERQemu exists to preserve access to some innovative and interesting software from the heyday of modern computing, when technology was advancing rapidly and the industry as a whole was growing by leaps and bounds. Representing one of the first steps onto dry land from the primordial soup of Xerox PARC, the PERQ is a fascinating "missing link" from the world of text-based timesharing to the fully graphics-driven interactive world we take for granted today.

Appendix B: Bibliography

A growing collection of PERQ documentation is available online to help you learn more about the PERQ and the software you can run with *PERQemu*. Links are provided below to get you started; as the Bitsavers collection does not guarantee that their links won't move or change, copies (along with some additional material not yet published on Bitsavers) are provided on Github as well.

POS Documentation

The *POS Software Reference Manual* is **the** primary reference for older versions D and F:

Bitsavers: [PERQ System Software Reference Manual, Feb. 1982](#)
Github: [POS D Collection](#)

A more extensive collection of POS G documents breaks down the original SRM into more manageable chunks:

Bitsavers: [POS G Collection](#)
Github: [POS G Collection](#)

Accent Documentation

Two full sets of Accent documents exist. CMU provided full size manuscripts with whimsical artwork and supplemental information about code that was not included as part of the general releases from Three Rivers/PERQ Systems Corporation. The “official” documentation was done in the small format popular with the WoD²⁴ approach popular in the ‘80s and ‘90s. The content is largely the same, with a few changes between S5 and S6:

Bitsavers: [Accent S5 Programmer's Manuals Collection \(CMU\)](#)
[Accent S5 User's Manuals Collection \(CMU\)](#)
[Accent S5 Users Manual \(Supplements, 3RCC\)](#)
Github: [Accent Collection](#)

²⁴ Popularized by the VAX VMS manual set, the WALL OF DOCUMENTATION approach comprised sprawling collections of annoying little half-sized binders that needlessly increased page counts, but they looked pretty all lined up on the shelf! Back then we killed trees to print paper documentation that was outdated and ignored, rather than killing salmon or burning fossil fuels to power datacenters full of broken web links and unindexed, un-OCRed scans of PDFs that nobody reads. Documentation, like QA, is a long dead tradition from a bygone era.

PNX Documentation

There is very little PNX documentation available here in the US, as most PNX installations were supported by ICL in the UK. Some scans of the *ICL Guide to PNX* manual have recently become available, although the PNX version 3.0 floppies are incomplete and the OS itself cannot yet be installed. The documents are linked here in the hope that they will soon be relevant to *PERQemu*.

Bitsavers: *Not yet available*

Github: [PNX Collection](#)

Additional Documentation

Several “miscellaneous” or OS-independent documents are gathered to provide additional background material. The *Fault Dictionary* (available in several versions or included in a number of other docs) is especially useful for tracking down problems at boot time.

Bitsavers: [POS G Fault Dictionary](#)

Github: [POS G Fault Dictionary](#)

Applications

In November, 2024, a massive collection of nearly 1,200 floppy images was contributed by yours truly to the Bitsavers software archive. These were archived by several PERQ enthusiasts and collectors (including the *PERQemu* authors) and are now available in IMD format for use with the emulator and real PERQ hardware. These have been sorted into broad categories and their contents loosely catalogued. There is a *lot* of really interesting stuff to discover here.

Efforts are ongoing to curate this raw collection to produce tested, complete installation sets for specific operating systems and applications. This additional software will in time be added to the Github [PERQmedia](#) repository, along with QIC tape dumps and many more pre-built hard disk images with demos, games, and applications to explore. Watch this space!

Appendix C: Additional Resources

Additional applications are being developed to supplement the emulator, and some of these are now being made available.

PERQdisk: Filesystem Munger Tool

PERQdisk is another C# application that lets you open and explore PERQ media archives without having to run the emulator. Perhaps its most useful feature is to extract files from the virtual disk or floppy image directly to the host, from individual files to entire partitions at once.

The software requirements for *PERQdisk* are the same as *PERQemu*: any Windows, Mac or Linux environment that supports the .NET Framework v4.8 should run it. The computational and storage requirements are much, *much* lower though.

Github: [PERQdisk Repository](#)

Currently in its “almost 1.0” early release, *PERQdisk* can read POS filesystems from hard disk images or floppies, and it can both read and write floppy images using the RT-11 volume format. It cannot yet read PNX disks due to a lack of documentation about PNX filesystem formats. *PERQdisk* can also view and edit text labels in the archive formats that support them, as well as save disks to different formats. The interface is similar to *PERQemu*'s CLI; documentation is included with the package.

Stut: The Sreamer Utility

Stut is a utility that reads PERQ tape images. It is currently being updated to use the *PERQmedia* library so that it can load 20MB QIC tape images saved in “.tap” or “.prqm” format. A tool to convert raw dumps of extracted tapes into these archive formats should be integrated, along with label editing and format transcribing capabilities. *Stut* is not yet available on Github.

Todo: Links to “tumble” for turning Canon TIFFs → PDF

Appendix D: Release History

Summaries of changes in prior releases on the Boondoggle branch, gathered here for posterity.

What's New in *PERQemu* v0.6.5

The v0.6.5 interim release is a fairly large step forward toward completing the hardware support for the full lineup of PERQ-2 configurations. Completing the work begun with v0.5.8, this release offers:

- PERQ-2 emulation:
 - Full support for the MFM 5.25" hard drive controller for PERQ-2/T2 configurations (one or two drives);
 - Low-level formatting support for both EIO DIB types (8" and 5.25");
- User Guide updates and numerous small bug fixes.

As of this writing it appears that .NET Framework 4.8.1 will remain supported by Microsoft for quite some time, so for now the target environment remains unchanged. With the v0.7.x release the baseline for MacOS X compatibility will move up to 10.13 (High Sierra) and Visual Studio 2019, as Mac OS X 10.11 has become untenable as a development platform. For now at least cursory testing on Ubuntu Linux (LTS 20.4) continues using the Mono 6.x runtime.

What's New in *PERQemu* v0.5.8

This interim release provides the first working PERQ-2 emulation. While incomplete and still in active development, snapshots posted to the "experiments" branch offer several new features:

- PERQ-2 emulation:
 - Ethernet I/O board (EIO) emulation now allows the basic PERQ-2 model to be configured and run; so far POS G, Accent S6 and PNX versions 2 and 3 are all confirmed to boot!
 - Micropolis 8" hard drive controller provides PERQ-2 and PERQ-2/T1 configurations with a larger 35MB disk vs. the 24MB Shugart in the PERQ-1.

What's New in *PERQemu* v0.5.5

This interim release completes the peripheral support for the PERQ-1 models and refactoring to integrate PERQ-2 support. It incorporates numerous bug fixes and refinements of the user interface. Significant new features include:

- Ethernet support:
 - For PERQ-1 configurations, the I/O Option board (OIO) 10Mbit Ethernet interface is fully emulated but still under active development;
 - Currently requires host `root` / Administrator privileges;
- Canon laser printer support:
 - The LBP-10 and CX models can be configured as an I/O Option, with software available in POS and Accent to drive them;
 - Output may be in PNG (single page) or TIFF (multiple pages) format;
 - Hardware allows multiple paper sizes (but software limited to US Letter);
- Screen captures from the CLI are available:
 - Portrait or landscape display saved as 100 dpi, 1bpp, PNG format.

There have been other bug fixes, refinements, and UI changes since v0.5.0 that will be enumerated at the next major release when I have time to write them all up.

What's New in *PERQemu* v0.5.0

Release v0.5.0 marks the “halfway” point in the emulator’s development. This release does not offer many new emulation options compared to prior *PERQemu* releases; it still emulates the original PERQ-1 model and standard peripherals. However, a *huge* amount of work has been done “under the hood” to enable the addition of all of the PERQ-2 series options going forward:

- True Z80 emulation, running from actual PERQ Z80 ROM images:
 - The original IOB with v8.7 ROMs (“old Z80”) supports the PERQ-1 options that prior versions of *PERQemu* can run;
 - The enhanced CIO board with v10.17 ROMs (“new Z80”) now enables newer versions of all of the PERQ-1 operating systems to run;
- Uses the SDL2 library for the display and keyboard/mouse interface:
 - Runs on 64-bit Mono/MacOS and eliminates 32-bit WinForms limitations;

- A new, unified *PERQmedia* storage architecture and file format:
 - Provides a flexible framework for managing *all* existing PERQ floppy, hard disk and tape image formats;
 - Adds a new common “PRQM” format with support for text and graphical labels, data compression, and checksums;
- Dynamic runtime configuration of all PERQ models and features:
 - Quickly load and run from a library of predefined machine configurations, or easily customize your own;
 - New peripheral support: emulated streaming QIC tape interface added;
- Enhanced command line interface:
 - More prompts, in-line help, and interactivity;
 - Basic Unix/Emacs control keys added for editing;
- Expanded debugging facilities:
 - Extensive logging options, to the console and/or log files;
 - Breakpoint support in Debug builds for watching I/O ports, memory locations, micro addresses, and more; breakpoints can trigger the execution of user-supplied scripts to perform more complex actions;
- Persistent user preference settings, and much, much more!

Please refer to the Readme files or Github Releases page for additional pre-v0.5.0 information.