# PERQemu User's Guide

For PERQemu version v0.5.0

S. Boondoggle

30-Jul-2022

**DRAFT**

Boondoggle Heavy Industries, Ltd.
1060 West Addison
Chicago, IL  60613
(800) 555-1212

Dubious modifications/enhancements to the software described by the verbiage spewed forth here contributed by S. Boondoggle, Boondoggle Heavy Industries, Ltd.

Accent and many of its subsystems and support programs were originally developed by the CMU Computer Science Department as part of its Spice Project.

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. The Reader's Comments card is distributed with this document to request users' critical evaluation to assist us in preparing future documentation.

# Table of Contents

# 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 originally ran on the PERQ 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 (PERQ-1) or 16K (PERQ-1A) of writable control store (48-bit 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 single Shugart SA851 8" 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, none of these I/O options are implemented. *Yet.*

The later PERQ-2 series, introduced in 1983, extended the original design in a few significant ways and added a number of additional IO options:

- A larger landscape display (1280 x 1024, monochrome)
- Up to two higher capacity disk drives (8" Micropolis, 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

There has been progress toward incorporating these additional models into the emulator, but much work remains. 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 will enable 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.

# What's New in *PERQemu* v0.5.0

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;
- 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 IO 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!

# The Emulator

## 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. 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.

- *Todo: All of these requirements need to be properly vetted/explained/dependencies noted and* ***installation steps*** *documented!*

The *PERQemu* master repository maintained by Josh Dersch is developed in MS Visual Studio *nn* on Windows *nn*, with Linux testing on *[TBD]*.

The "`skeezicsb/experiments`" branch is developed and tested with Xamarin Studio Community 6.3 on MacOS X 10.11 (El Capitan) and verified on Visual Studio for Mac 2017 / MacOS X 10.13 (High Sierra). Yes, really.

## 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.

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 (~5.89Mhz) and the Z80 at 407ns (2.4576Mhz for the PERQ-1) or 250ns (4Mhz for the PERQ-2).

- *Note: The emulator is still in active development and performance tuning is an ongoing concern.*

**CPU**: A "reasonably current" processor is required. A dual-core, 3.0GHz or faster x64 made in the last 5 years or so is about the minimum requirement; four cores is highly recommended. No testing has been done on Apple's ARM-based silicon (donations accepted!) or other non-x64 processors.

**Memory, disk space**: negligible (by modern standards); the emulator requires about 100MB of RAM with a typical PERQ-1/Shugart configuration loaded, and approximately 550MB of disk space (for a source tree) or less (binary distribution).

❖ *Todo: Recompute disk space requirement once a test distribution is built with compressed PERQmedia disk images, and a (possibly expanded) standard media library.*

**Display**: A typical 1920 x 1080 display is more than 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. At full speed the PERQ only refreshes its display at 60 frames/second, which should be supported by all modern displays.

❖ *Note: 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.*

**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.

## 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 currently the program does not need any special OS privileges to run.

❖ *Note:* If you prefer to build from source, a clone or download from the Github repository should contain 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.) In the examples below, the 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:

`Conf/`

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.

`Disks/`

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 `Disks/` directory by default.

`Output/`

When logging debug output to disk is enabled, those files go here by default. When screenshots and printing are implemented, that output will land here too. (Output directory will be a settable preference.)

`PROM/`

Contains dumps of PERQ ROMs 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 command-line 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 v0.4.6 ('As the sparks fly upwards.')
Copyright (c) 2006-2022, J. Dersch (derschjo@gmail.com)
Feebly assisted by S. Boondoggle (skeezicsb@gmail.com)
Type 'help' for console commands, or 'gui' to start the GUI.
>
```

❖ *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 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
              commands - Show settings commands and their descriptions
               default - Reset all program settings to defaults
        display cursor - Change the system cursor when in the display window
                  done - Exit settings mode, return to top-level
                  load - Reload saved settings
      output directory - Set directory for saving printer output and screenshots
        pause on reset - Pause the emulator after a reset
  pause when minimized - Pause the emulator when the display window is minimized
    performance option - Set performance option flags
                  save - Save current settings
                  show - Show all program settings
 unassign rs232 device - Unmap a device from a PERQ serial port
```

You could 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"[1] 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)
```

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

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

Here a string argument named "`file`" is required. Entering `g7.prqm` 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.[2]

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!

[2] *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:

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

*Todo:* History display, saving, possible "!" syntax (future), repeat buffer?

# 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 you can specify CPU type, option boards, memory and display size, peripherals to attach and more, all at runtime using the `configuration` subsystem. 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 many devices and drivers to be added to the emulator before they can be run.[3] Currently the default is a PERQ-1A with the POS F.1 image already assigned, so "out of the box" 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
```

[3] *For a sneak peek at what's in the pipeline, 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 should support* every *option on that chart someday!*

```
Configuration: PERQ2-T2
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 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 below.

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 are specified. The "`configure chassis`" command selects the type.

| Chassis Designation | 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` | Extremely rare, this "ugly duckling" was meant to be smaller and quieter than the original cabinet, but is mostly just hotter, more cramped, and harder to work on. 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.[4] |
| `PERQ2Tx` | This ICL-designed cabinet quickly replaced the early PERQ-2. The "T1" model shipped with 8" Micropolis hard drives; the "T2" model accommodates 5.25" MFM hard disks. A second hard disk is optional. 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.*

Currently only the PERQ-1 chassis is supported.

There is some confusion surrounding the early PERQ-2 model (sometimes referred to as "K0" or "Krismas", possibly "T0"), especially given its rarity (by some estimates only a dozen or so were made!?) and how few references exist in any of the surviving literature and marketing material. Selecting the `PERQ2` chassis enforces a limit of *one* 8" Micropolis hard disk and allows the 4K or 16K CPU.

[4] *I don't have enough photos/specs or personal experience with this model to fully describe it properly. Josh recently rescued one, so more information about specifics should be forthcoming.*

The `PERQ2Tx` designation represents the T1, T2 or T4 models in the ICL-designed cabinet. The Configurator enforces the subtle differences as described in the table above, or will when the PERQ-2 hardware is finally added to the emulator!

### Processor Type

The system then 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 enjoys the widest software compatibility of the available types. |
| `PERQ24` | This rare processor is an extension of the `PERQ1A` processor. It offers the same 16K of writable control store, but extends the major datapaths and physical address bus to 24 bits. This processor is *only* 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.

### Configuring Memory

A fairly wide 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 either the portrait or landscape display. Use the "`configure memory`" command to select how much RAM is available to the virtual machine. With no arguments, a helpful 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` | *PERQemu* 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 *or* an 8" Micropolis hard disk. |
| `EIO` | All PERQ-2 models use the "Ethernet I/O" board. It features a faster 4Mhz Z80, runs the v10.17 firmware, and has a real-time clock chip and a second RS-232 port. The EIO can drive one or two 8" Micropolis *or* 5.25" MFM hard disks, depending on the chassis type. |
| `NIO` | An "Ethernet I/O" board without the Ethernet, but otherwise the same as the EIO. |

*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 re-map 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.

Currently the IOB and CIO boards are available, enabling most PERQ-1 software to run. However, CIO Micropolis hard disk support is not yet implemented.

The EIO and its NIO variant are not yet implemented.

### 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*, however, 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.

*Todo:* Complete the I/O options section.
Commands: check, enable/disable, name/description, save.

# 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 "*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[5]** |
|---|---|---|---|
| Floppy drives | 1 | 1 | 2 |
| Internal hard drives | | | |
| 14″ Shugart | 1 | - | 2 |
| 8″ Micropolis | 1 | 2 | 2 |
| 5.25″ MFM | - | 2 | 4 |
| External hard drives | | | |
| SMD | - | ? | 4 |

[5] *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[5] |
|---|---|---|---|
| Tape drives | | | |
| QIC streamer | 1 | 1 | 1 |
| 9-track | - | ? | 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, the only hard drive) |
| Unit 2: | 2nd hard drive | (only in PERQ-2/T*x* models) |
| Unit 3: | Streamer tape | (optional; not yet implemented) |

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.

## 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 disk or floppy images you create or download should be dropped into `Disks/` so *PERQemu* can find them without a lot of typing!

## Supported Media Formats

As of *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 a 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.

### Loading, Saving, and Unloading Media

The new dynamic configuration of virtual machines in *PERQemu* changes the way that storage devices are managed. The original syntax provides useful shortcuts for the most common operations:

```
> load floppy <file>
> save floppy
> unload floppy

> load harddisk <file>
> save harddisk
> unload harddisk
```

Each of these commands do the expected thing in configuration mode, updating the current media assignments in the same way the new Configurator's "`assign`" command does:

```
> configure assign <file>
```

is equivalent to

```
> load floppy <file>
      or
> load harddisk <file>
```

depending on the contents of <*file*>.

Because a floppy is *removable* media, *PERQemu* allows these commands at any time.[6] The current configuration record is updated to reflect the status of the drive *and* the emulated floppy drive receives a status change signal.

❖ ***Note:*** You should take care to not load or unload a floppy 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).

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 stopped first (see "*Managing the Virtual Machine*", below).

### Creating and Formatting New Media

The new "`storage`" subsystem allows the creation of new blank media files in a wide range of predefined types. It also allows for the creation of new types, using the PRQM format. To create a new file, first check the list of available media types:

```
> storage list
Drive Class Drive Type  Description
----------- ----------  ------------------------------------------------
Floppy      Floppy1024  Shugart SA851 DS/DD (1MB) floppy disk
Floppy      Floppy256   Shugart SA851 SS/SD (256KB) floppy disk
Floppy      Floppy512   Shugart SA851 DS/SD (512KB) floppy disk
Disk5Inch   Maxtor120   Maxtor XT-1140 120MB hard disk
Disk5Inch   Microp33    Micropolis 1303 33MB hard disk
Disk8Inch   Microp35    Micropolis 1202 35MB hard disk
Disk14Inch  Shugart12   Shugart SA4004 12MB hard disk
Disk14Inch  Shugart24   Shugart SA4008 24MB hard disk
Disk5Inch   Vertex51    Priam/Vertex V150 51MB hard disk
...
```

[6] *PERQmedia accommodates other removable media types too; tape drives or even SMD hard disks that use removable "packs" will be supported in future releases.*

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.
```

This file can then be loaded using the shortcut:

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

The "format" 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.

> The complete list of known/supported 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 "for fun" included by the author.
>
> 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. The emulator does not yet support any but the basic PERQ-1 Shugart drives.

## Working with Communication Devices

*Todo:* Once full serial port support is restored, describe how RS-232 ports can be assigned and mapped to ports a/b on the emulated PERQ.

When Ethernet support is included, describe the setup and configuration of the host and emulator.

# 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`" command[7]:

> **`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. (See the section "*Errors and Boot Failures*" below on how to diagnose startup 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:

> **`start`** ← starts the CPU executing at ROM address 0x0000

[7] *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 PERQ. At any time you may click on the console window to issue commands. To pause execution of the virtual machine, 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 "*User Preferences,*" below, for information about changing your autosave settings.

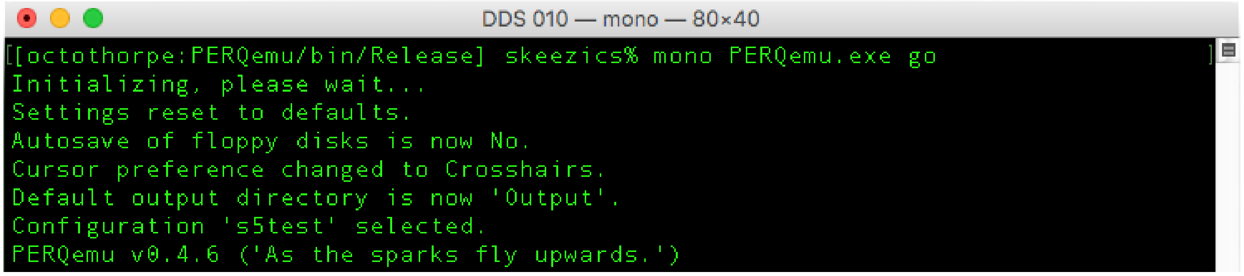
To exit *PERQemu,* as if you'd ever want to do such a thing:

> **`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 "diagnostic display" that shows the progress of the boot sequence; *PERQemu* emulates this by updating the console window's title bar to read "DDS," followed by a three-digit number:



```
[octothorpe:PERQemu/bin/Release] skeezics% mono PERQemu.exe go
 Initializing, please wait...
 Settings reset to defaults.
 Autosave of floppy disks is now No.
 Cursor preference changed to Crosshairs.
 Default output directory is now 'Output'.
 Configuration 's5test' selected.
 PERQemu v0.4.6 ('As the sparks fly upwards.')
```

*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** |
| POS: | **999** |
| Accent: | **400** (1MB of memory) or **450** (2MB) |

Because *PERQemu* doesn't emulate the 2 minute warm-up time[8] 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!

[8] *If you're a real ~~glutton for punishment~~ stickler for accuracy, you* can *enable* `AccurateStartupDelays` *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![9] 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 three[10] 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 the drive 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 "boot area" set aside (starting at cylinder 0) where the microcode for bootstrapping is written.

[9] *The author wryly acknowledges the irony of this statement. See Appendix A.*

[10] *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 VFY and SYSB 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 ("have you tried turning it off and back on again?") and polls the keyboard several times to see if a boot character is pressed. Based on the character received (defaulting to "`a`" boot if none pressed) it attempts to load the operating system from either the hard disk or floppy. Two OS-specific microcode files (with the extensions ".boot" and ".mboot") contain the kernel and I/O microcode as well as the language interpreter which defines the instruction set used by the system. (For POS and Accent these are "Q-codes," 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.

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 OS takes over all the 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, with only a hint of sarcasm…

# 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.[11] 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, preference 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.

[11] *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* virtue *after all!*

## User Preferences

*TBD*.
Things that work now: autosave settings for disks, pause on reset, cursor type, serial port device assignment, pause-on-minimize?

Output options, default debugging radix, performance options not yet implemented. Last successfully loaded configuration is not reliably updated/resaved (minor bug).

## Debugging Features

*TBD*.
Highlights: logging/tracing, breakpoints, :variable syntax, etc. Most of these are implemented and working, though some are incomplete but aren't needed in normal day-to-day use. Y'know, for the legion of fellow PERQ microcoders out there, right guys? Hello? <tap tap> Is this thing on?

# 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 | PERQ-2 | Host |
|---|---|---|
| HELP | HELP | F1 or HELP |
| OOPS | OOPS | F2 or CLEAR or Ctrl-u |
| INS | INS/ESC | ESC or INSERT |
| DEL | REJ/DEL | DELETE |
| LINEFEED | LINEFEED | F3 or PAGE DOWN or Ctrl-j |
|  | SETUP | PAGE UP |
|  | NOSCRL | SCROLL LOCK |
|  | BREAK | SYS REQ |
|  | PF1..4 | F1..F4 |

*Table 5: Special key mapping.*

*Todo:* The PERQ-2 keyboard layout is defined but can't be tested yet.

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 damn function keys. Sigh.

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.[12]

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" mode, 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 "on" 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 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

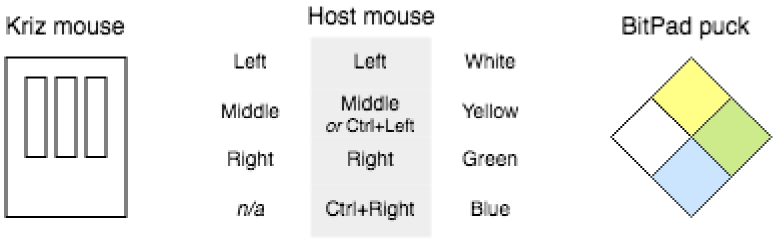
[12] *PERQ documentation also mentions a one button stylus option for the BitPad; this is not currently emulated.*

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 stylus[13] 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.



| Kriz mouse | Host mouse | BitPad puck |
|---|---|---|
| Left | Left | White |
| Middle | Middle<br>*or* Ctrl+Left | Yellow |
| Right | Right | Green |
| *n/a* | Ctrl+Right | Blue |

*Figure 2: Host to PERQ mouse button mapping.*

## PERQ 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.

[13] *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 but we have no way to really test that...*

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. An external, standalone utility (`floppy.pl`) is also available upon request. It allows reading and writing PERQ floppies on the host, but currently only supports the "raw" image format; a port to C# is planned to take advantage of the *PERQmedia* library.

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 eee-vill; 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 "tar" formatted floppies used for data interchange.

### PERQ 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. One PERQ hard disk can contain several partitions shared among them; software limitations are discussed in the chapter below. PNX, however, uses an incompatible on-disk format that is based on the Unix V7/System-III filesystem(?). It is not clear if PNX takes advantage of the logical header.

Two external programs exist to access older "PHD" style hard disk images: `perqdisk.pl` is a reasonably full-featured Perl utility, while *PERQdisk* is a fairly bare-bones C# implementation that allows a limited subset of commands. Both can extract files from POS filesystems to the host, but neither can write files directly into an image. Work is underway to merge these two into a new utility that will leverage the *PERQmedia* library and allow access to all hard disk image formats.

There are not currently any host tools to access the contents of PNX disk images, as details of the formats used are unknown at this time.

# 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 ICL).

*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);
- PNX version 2.[14]

The following are planned but will not be available until PERQ-2 support is complete:

- Accent S7 (was to be Release III; later available from Accent Systems?);
- PNX versions 3 through 5 (public releases from ICL);
- FLEX (depending on availability).

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.

[14] *Currently PNX 2 boots from floppy but fails to install on the hard disk. I hope to remedy this soon.*

# 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.

Like most OS environments on the PERQ, the POS taxonomy is split into the "old Z80" and "new Z80" eras. The version naming/numbering scheme is simple, with a letter representing the major release, and a number the minor release. In terms of hardware/emulator support, this means:

- POS D[15] and F releases ran the "old Z80" ROMs, only on the PERQ-1 with the IOB, one Shugart hard disk, and the portrait display;
- POS D.6 and F.1 releases (included in the bundled disk images) can only use the BitPad tablet; they don't support the Kriz. POS F.15 can support both tablet types.

---

- POS version G requires the "new Z80" ROMs; this means a PERQ-1 with the CIO I/O board, or any PERQ-2 configuration;
- POS G supports all hard disk types; on the PERQ-2/T*x* a second 8" or 5.25" hard drive can be provisioned;
- POS G also supports both tablets *and* both display types!
- POS version G does *not* run on the 24-bit processor.[16]

Once POS has booted, enter the date and login with user "`guest`" and a blank password (or alternately just hit `RETURN` at the login prompt).

Full POS documentation can be found on Bitsavers:

[http://www.bitsavers.org/pdf/perq/PERQ_SysSWRefManual_Feb1982.pdf](http://www.bitsavers.org/pdf/perq/PERQ_SysSWRefManual_Feb1982.pdf)

Text versions are also in `sys:user>doc>` in the F.1 hard disk image, or `:boot>docs>` in F.15.

[15] *No versions of POS prior to D.6 are known to have survived.*

[16] *POS G.7 and G.85 may have been recovered; they might run on the 24-bit CPU. [To be confirmed.]*

Interaction with POS is through the 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:

`?` 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 `.RUN` and directories are suffixed with `.DR`

`PATH` change directory. Often aliased to `CD`, more or less like your Unix or DOS equivalent. Note that the directory delimiter is '>' (this may freak out Unix heads!)

```
CD foo>bar>baz>       - change to directory foo>bar>baz
PATH ..               - change to parent directory
PATH sys:user>        - change to root of user 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 `.SEG` file which must be linked

`LINK` run the linker. Produces a `.RUN` file which may be executed

`COPY` copy a file from one location to another. `RSX:` is a special file which can be used to transfer files to/from the emulator host machine. *See section n for more details.*

`EDITOR` a fairly simple text editor (see also `PEPPER`, a more Emacs-like editor, available in most of the POS images)

To run a program, just type its name (you can leave off the `.RUN`). Switches are prefaced by '/' as in many OSes of the day. 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?

There are games under `sys:user>games>`, demos under `sys:user>demo>`, various utilities and OS source are under `sys:boot>`. User directories are generally kept under `sys:>user>` but (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.

You can log out of POS by typing "`bye`" to the Shell. The PERQ-1 hardware actually could do "soft" power! To log out *and* power down the machine, type "`bye off`" and POS will tidy up and shut down the virtual machine. In *PERQemu* that means returning you to the CLI, the same as pressing the close button on the Display window. Depending on your user preferences, you may be prompted to save any unsaved changes to your loaded disk(s) before the emulator closes up shop.

*Todo:* Using floppies
Using the RSX: device for copying files to/from the host
Using an actual serial device with Chatter, Jet or Kermit

## 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 Shug24 and the floppies!

Most interactions with MPOS are the same as POS D/F, so only the differences are highlighted here.

*Todo:* Device interaction?
Window manager!
See if the recently found Ethernet code works, once Ethernet is implemented.

# Accent

Accent is a message-based, multi-tasking OS developed in the early 1980s by Carnegie-Mellon University as part of the "SPICE Project." Successor to RIG and forerunner 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* is now able to run several versions of this interesting and groundbreaking operating system. Please see Appendix B, *Bibliography*, for links to a wealth of documentation.

## Availability

In the bundled POS D.6 disk image (`Disks/d6.phd`), an early release of the Accent S4 from CMU is available. 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 by the time *PERQemu* v0.5.0 is released.

A fresh installation of Accent S6, paired with POS G.7, is now available and included in the standard distribution as `Disks/g7.prqm`. Additional applications are being prepared to augment the base OS. Hopefully a working Lisp M3 environment will be included once all the right pieces can be found and tested. Accent S6 is '`z`' boot on this image.

Accent S7, the final (unreleased) version known to exist, is *tantalizingly* close. Watch this space.

## 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 practical minimum of 1MB (though it will boot with less) and a maximum of 2MB. On a PERQ-1 with a 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 was 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.

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

*Figure 3: Sample Accent S4 configuration dialog.*

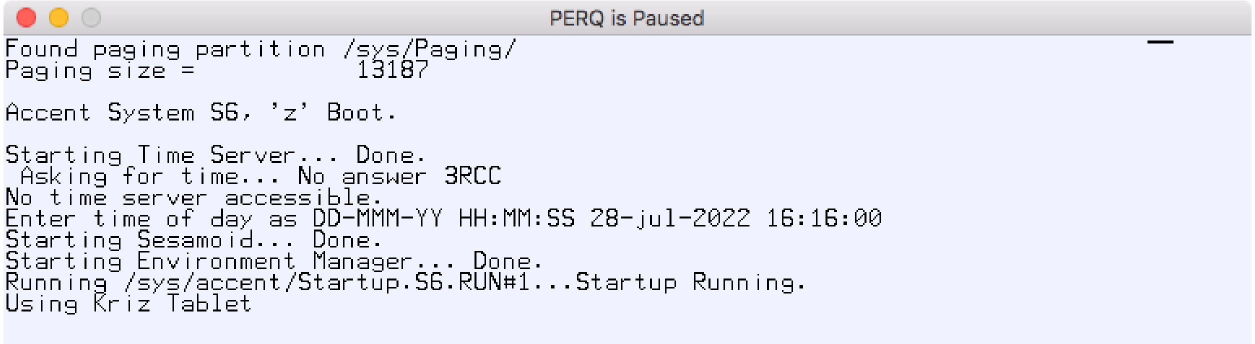
Accent S6 runs on the PERQ-1A/CIO/Shugart configuration. It supports both tablet types, and both display types! As with all versions of Accent, performance is best with more memory configured. Selecting the Kriz tablet gives a slightly more responsive interaction with the window manager, as its binary protocol is more efficient than the ASCII data stream sent by the BitPad.

```
> configure load perq1a
> configure assign g7
> configure memory 2
> configure tablet kriz
> debug load qcodes accentv5
> bootchar z
```

*Figure 4: Sample Accent S6 configuration dialog.*

### Login and User Interface

Accent's startup sequence 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:



```
Found paging partition /sys/Paging/
Paging size =          13187

Accent System S6, 'z' Boot.

Starting Time Server... Done.
 Asking for time... No answer 3RCC
No time server accessible.
Enter time of day as DD-MMM-YY HH:MM:SS 28-jul-2022 16:16:00
Starting Sesamoid... Done.
Starting Environment Manager... Done.
Running /sys/accent/Startup.S6.RUN#1...Startup Running.
Using Kriz Tablet
```

*Figure 5: 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 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.[17]

The interface to Accent is provided by SAPPHIRE, a window manager that unifies the "native" (POS-like) shell environment, a Common Lisp environment, and a Unix-compatibility layer.[18] 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, scrollback, multiple shells and multi-tasking, and "real" window management.

Window manager commands may be invoked by mouse click or the keyboard. Keyboard commands are prefixed by `Ctrl-Del`, 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 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*.

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.) As with POS, control characters are case sensitive!
- You can scroll back to see text that scrolled off the screen (`^V/^v` for backward/forward by roughly a page full of text);
- Interrupting or aborting the current program goes through the window manager, so `^c` or `^C` doesn't work as in POS.

[17] *PERQ-2s have a battery-backed real-time clock chip, and most Accent machines were connected to Ethernet networks where a time server would provide the time, eliminating this manual step. PERQemu will support those options in the future as well!*

[18] *Work is ongoing to make these additional Accent environments available.*

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 -size -part`

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.

## 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

Designed to be a distributed, networked OS, all versions of Accent will be greatly enhanced by the availability of Ethernet once that's added to *PERQemu*. Currently both distributed Accent images have their network servers disabled, as something in the emulation environment causes too many problems at startup.

One side effect of this is that S6 seems to be unable to launch the FloppyServer process, making it impossible to load data from floppies directly. However, as a workaround the POS G.7 floppy program ('`a`' boot) can be used to copy data to and from the Accent partitions.

Accent S4 seems to get 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.

*Todo:* Get S5 and Lisp booting! See if MatchMaker will run too
Load the S6 demos and test them
See if the C environment is available, and Qnix/Spoonix
Ethernet, Ethernet, Ethernet!

# PNX

A complete PNX 1.3 hard disk image and a full floppy set is available!

PNX, another unfortunately named PERQ operating system, was an early Unix V7/System III mashup that included 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 used 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 documentation.

Note that this early 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;
- Has one further restriction: version 1.3 supports a maximum of 1MB of memory. If you configure 2MB PNX will fail to initialize and will drop into its debugger, displaying an "@" prompt and generally only responding to a few cryptic (and undocumented) commands.

There is currently no extra software for this version of PNX besides the very basic install image. With upcoming enhancements to the emulator, there are several newer versions of PNX to try out *and* some exciting news about a possible software archive unearthed in the UK; watch this space!

PNX 2.0 will now boot on the "CIO" PERQ-1, but a full installation on the hard disk has not yet been completed successfully. This version promises a number of improvements over the 1.x release. Versions 3 and 5 will *likely* require the EIO/PERQ-2, but more research is needed to understand their hardware requirements. 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.

*Todo:* Get PNX 2 to properly install & boot; will anything later run on the PERQ-1?
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.

PERQ FLEX is an exceedingly rare 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 the first real 8″ Micropolis hard disk image to test with once PERQ-2 emulation features are complete.

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

## More to come…

(each OS in turn: startup, login, basic commands, getting help, logging out)
UI features (windows, pop-ups, command keys, etc)
How the emulator maps keys and clicks (need to expand this)
Demos and games!!
Applications
Dev tools and languages
Documentation and links to other help
Logging out and shutting down

# Command Reference

Below is an itemized description of the command syntax for *PERQemu* v0.5.0. Command arguments are illustrated using the following typographic conventions:

| | |
|---|---|
| *argument* | a required argument supplied by the user |
| (`this` \| `that`) | a required argument from a list of specific choices |
| [*argument*] | 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. The "`done`" command returns to the previous level in the hierarchy.

## Top-level Commands

### about

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

### bootchar [*char*]

With no argument, shows the current boot character (if set). With a character argument, sets the desired boot character (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*]

Shows 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

Shows the last 99 commands entered.

### load floppy *filename*
### load harddisk *filename*

Loads the supplied *filename* as the current floppy or hard disk. These shortcuts are provided for compatibility with prior versions of *PERQemu*. See also: Configuration "`assign`" command.

### power (on | off)

The "`power on`" command validates the current Configuration, loads the assigned media and starts the emulator if it isn't already running; "`power off`" shuts down the emulator, optionally giving the opportunity to save modified hard disk or floppy media.

### quit
### quit [without save]

Exits *PERQemu*. If the emulation is running, performs a "`power off`" first. 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, shuts down and exits immediately and unsaved changes are lost.

### reset

If the emulator is running, "`reset`" simulates the press of the PERQ's "boot button" triggering 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 | harddisk)
### save (floppy | harddisk) (as) [*filename*]

Commits changes to a modified disk 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. Currently the new copy is always created in the new *PERQmedia* format and given a "`.prqm`" extension. *[Check this]*

### save history

### save screenshot

Not yet implemented.

### settings [...]

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

### start

Used to start or restart emulation when paused. The machine must already be powered on.

### status

Prints 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

Used to 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). Updates the current configuration.

### unload harddisk [unit]

Removes a hard disk from the current configuration. If *unit* is not specified, unloads the default unit #1. (Dual-drive PERQ-2 systems are not yet supported.) Fixed hard disks cannot be unloaded when the emulator is running; hard disks with removable "packs" are not yet supported. See also: Configuration "`unassign`" command.

# Configuration Commands

From the top level menu, "`configure`" enters the Configuration subsystem. See *The Configurator* in the previous chapter 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, although currently only PERQ-1 configurations with one floppy (unit 0) and one hard disk (unit 1) are supported.

### chassis *type*

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

### check

Validates the current configuration to make sure the machine is a valid PERQ that can be emulated. *PERQemu* will print warnings about (but will allow) rare or experimental selections that may not have adequate software support and could fail to boot.

### cpu *type*

Selects a processor to emulate from the list of supported *types*.

### default

Resets the current configuration to the built-in default.

### description *string*

Saves a one-line textual description of the configuration. *string* must be surrounded in quotes ("") if it contains spaces.

### disable rs232 [*port*]

Disables the use of a host RS-232 device. If *port* is not specified, '`a`' is assumed.

### display (portrait | landscape)

Selects the format of the PERQ's display. See also: the chapter *PERQ Operations* for restrictions on operating support for the landscape display.

### enable rs232 [*port*]

Enables 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.

### io board *type*

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

### list

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

### load *config*

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

### memory *capacity*

Configures the PERQ's memory capacity. The *capacity* given may be specified in kilobytes (256, 512, 1024, etc.) or in megabytes (1, 2, 4, 8) and must be valid for the given CPU type. *PERQemu* will attempt to round the given argument to the nearest power of two.

### name *string*

Provides a filename for the given configuration. The *string* specified should be a short, unique designation suitable for use as a filename. See also: the "`description`" command to provide more information about the configuration; the section "*The Configurator*" for details of how *PERQemu* saves and loads configuration files.

### option board *type*

Selects 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 only a very limited subset of options.

### save

Saves the current configuration if it has been modified. It must have an assigned name; see the "`name`" command, above.

### show [*config*]

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

### tablet (bitpad | kriz | both | none)

Selects 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*

Removes a media file from the configuration. The specified *filename* or the specific *unit* is unassigned. If the emulator is running… *[Check this]*

## Debug 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, while Debug builds of the program provide a much wider range of tools.

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* [Debug builds only]

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

### 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.

### clear rasterop debug [Debug builds only]

Disable extended RasterOp debugging. [Deprecated; to be removed in a future release.]

### disable breakpoints [Debug builds only]

Disable breakpoint processing. Defined breakpoints are retained but ignored until enabled.

### disable console logging

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

### disable file logging [Debug builds only]

Disable logging to files.

### disassemble microcode [*address*] [*length*]

Produces 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.

### edit breakpoint *type watchpoint* [Debug builds only]

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.

### enable breakpoints [Debug builds only]

Enables breakpoint processing.

### 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 [Debug builds only]

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.

### find memory *address value* [Debug builds only]

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

### inst

Run one Q-code. 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. Please see *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 should 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* [Debug builds only]

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

### set breakpoint *type watchpoint* [Debug builds only]

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.

### 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* [Debug builds only]

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

Write a specific *value* into the PERQ's memory at *address*.

### set rasterop debug [Debug builds only]

Enable extended RasterOp debugging. [Deprecated; to be removed in a future release.]

### set xy register *reg value* [Debug builds only]

Change the value of XY register *reg* to *value*.

### show [...]

With no arguments, "`show`" prints current debugger settings.

### show breakpoints [*type*] [Debug builds only]

With no arguments, *type* defaults to "`all`". Prints the status of defined breakpoints of the given *type*.

### 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, 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* [*words*]

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

### show memstate **[Debug builds only]**

Dump information about the memory controller's state.

### show opfile

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

### show rasterop (fifos | registers | state) **[Debug builds only]**

Print internal RasterOp unit state. Highly esoteric. [Deprecated; to be removed in a future release.]

### 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. 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

*Todo:* add these. There aren't too many useful ones at the moment.

## Settings Commands

*Todo:* describe the settings.

## Storage Commands

*Todo:* storage subsystem commands.

# Appendix A: A Little PERQ history

First publicly demonstrated in 1979 but not released in production quantities until 1981, the PERQ's hardware 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. As arguably the first *commercially-available* workstation-class computer to meet the "3 M's" criteria,[19] the PERQ's specifications were in line with the requirements for Carnegie-Mellon University's SPICE Project, where it was used as the initial 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 early lead in the workstation race of the early 1980s, the lack of 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 the "Qnix" environment) came along, other Unix vendors like Sun Microsystems had surpassed the PERQ solely on the strength of their software base. (A PERQ T2 can more than hold its own against an early Sun 3 in raw graphics performance, but the contemporary Motorola 68k-based workstations were pulling away in overall speed and capacity.)

By the time Three Rivers Computer (renamed in 1983 to PERQ Systems Corporation) went bankrupt in early 1986, it is estimated that ~ 4,000 PERQs had been built, mostly sold to universities or a couple of 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 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 full GUI-driven interactive world we take for granted today. Woo, hyperbole baby! Yeah! *[Rewrite…]*

[19] *The graphics capabilities differentiated the then-emerging "engineering workstation" from low-end personal computers or traditional text-based minicomputers of the era. The "3 Ms" criteria reflected the challenges of late 1970s technology at the cusp of breakthroughs in VLSI, storage capacity, networking and graphics.*

*Todo*: Links to other PERQ media

Github, bitsavers, future web home, etc.

Acknowledgements and thanks

# Appendix B: Bibliography

*Todo:* Add links to user documentation:

- Fault dictionary, POS manuals, Accent manuals, any additional PNX docs?
- Programming documentation, for the legion of new PERQ hackers waiting to explode onto the scene, man.
- More background info, links, hardware info?
- Reader's comments card to be printed and mailed in. Ha ha ha, are you kidding? With today's broken USPS? Sigh. But what a lovely throwback to a kinder, gentler era when you had to handwrite, lick a stamp and snail-mail in your flames without the instant gratification of just spamming them online.