Palm OS Emulator 2.1/3.0

Engineering Specification

Keith Rollin – x6290

7/24/1998

Table of Contents

1 Introduction

1.1 Summary

The Palm OS Emulator is an application for desktop computers that emulates the operations of Palm Computing Platform hardware devices. With it, users can load ROM images, applications, and databases and use them on their desktop computers.

1.2 Goals

The Palm OS Emulator meets the needs of two target audiences: developers and end-users.

1.2.1 Developers

Developers use the Palm OS Emulator to help develop, test, and debug their Palm OS application. Because the Palm OS Emulator is completely equivalent to a Palm Computing Hardware device, it obviates the need for downloading applications to a viewer device for testing. Instead, the developer can install their application into the emulated environment and run and test it on the same desktop computer containing their development tools.

Developers can also make use of extensive debugging facilities incorporated into the emulator. With the emulator, developers can detect accesses to illegal ranges of memory (such as low-memory or the CPU's hardware registers), run Gremlins, etc. When errors are detected, developers can use debugging facilities from external source-level debuggers to pinpoint the errors.

1.2.2 End-users

End-users use the Palm OS Emulator to evaluate 3rd party software. By installing prospective software into the emulator, they can safely evaluate the software before installing it onto their hardware device, making sure that it fulfills their needs and doesn't do anything overtly bad (such as crash). They can also test the application for compatibility, making sure that their application has a good chance of working with future Palm OS versions or hardware devices.

2 Product Description

2.1 Requirements

2.1.1 Windows

- Windows 95 or better, Window NT 3.51 or better. Windows 3.x with Win32s cannot be used, as the Palm OS Emulator makes use of multi-threading facilities that Win32s does not support.
- 200MHz Pentium processor or better recommended. Lesser processors may be used, but will suffer from performance problems.
- Screen capable of greater than 640x480 resolution.
- 16 Meg RAM.
- 700K disk space, plus space for each ROM file (typically 512K, 1 Meg, 1.1 Meg, or 1.5 Meg), plus space for each document file (typically a little larger than the amount of RAM being emulated).

2.1.2 Macintosh

- Mac OS 7.5 or later. Mac OS 8.0 or later recommended.
- 200MHz PowerPC processor. Lesser processors PowerPC may be used, but will suffer from performance problems. 68K-based Macs are not supported, solely for the reason that a survey of developers didn't turn up anybody using them.
- 12" display or larger (15" display or larger to use 2x mode)
- 2.0 Meg RAM, plus 1 Meg of RAM for each Meg of emulated device RAM
- 1.4 Meg disk space, plus space for each ROM file (typically 512K, 1 Meg, 1.1 Meg, or 1.5 Meg), plus space for each document file (typically a little larger than the amount of RAM being emulated).

2.2 What Is An Emulator?

In general, an emulator is a software application that makes one computer act like another computer. In the case of the Palm OS Emulator, it is an application that allows a user to run Palm OS applications on Windows or Macintosh desktop computers. The user acquires a ROM image (typically by downloading one from their Palm OS device or from Palm's developer Web pages), launches the emulator, and tells the emulator to load that ROM image. The emulator will then "boot" that ROM image, with the result that the user sees the same General Preferences panel that they would after resetting a Palm OS device. After that, the user can install applications into the emulator, which they can then run in exactly the same way as if they were running the application on an actual device.

2.3 Technical Overview

When a Palm OS device is being emulated, a number of different aspects of emulation are going on all at the same time. The microprocessor (CPU) must be emulated, standard memory accesses must be emulated, and accesses to special memory locations called hardware registers (which control other hardware attached to the device, such as the LCD screen and serial ports) must be emulated.

2.3.1 CPU

Palm OS hardware devices currently make use of two similar CPUs: the MC68328 (Dragonball) and MC68EZ328 (DragonballEZ). These CPUs are based on the MC68000 (the same one used in the original Macintoshes), and use the same instruction set. The Palm OS Emulator can emulate both of these processors. Which one it emulates is determined by the ROM image the user loads into the emulator.

Emulation at the CPU level is fairly simple. The Palm OS Emulator loads the ROM image and initializes a virtual program counter ("PC") to a well-defined location within that ROM image. The emulator proceeds to "boot" the ROM by emulating the execution of machine language instructions (opcodes) starting at that memory location. In turn, opcodes are read from memory, decoded, and the actions that would have occurred in an actual CPU are mirrored in algorithms and data structures maintained by the emulator. For example, if the opcode being emulated is supposed to result in two numbers being added together, the emulator will know to fetch those two numbers, add them together, and store the result in the location indicated by the opcode. In other words, what a microprocessor does in hardware, the emulator does in software.

Other aspects of CPU emulation (interrupt handling, exception handling, sleep mode, etc.) are also fully emulated.

2.3.2 Memory

The CPU works in conjunction with memory accessible via its external address bus. On Palm devices, there are three types of memory to be accessed: ROM, RAM, and the memory-mapped hardware registers.

ROM memory is emulated by making use of the ROM image file that the emulator loaded. Whenever an emulated opcode needs to fetch a value that falls in the range occupied by the ROM in an actual hardware device, the emulator fetches the appropriate value from within the ROM image it loaded.

RAM memory is emulated similarly. When the emulator is started, it is told to emulate a device with a specified amount of RAM installed (128K, 256K, 512K, 1 Meg, 2 Meg, 3 Meg, 4 Meg, or 8 Meg). The emulator uses the host OS's facilities to allocate a block of memory of the requested size to be used as a "RAM buffer". Whenever an emulated opcode needs to fetch a value that falls in the range occupied by RAM in an actual hardware device, the emulator fetches the appropriate value from within the RAM buffer instead.

2.3.3 Hardware Registers

Hardware registers are special memory locations that do magic things when read from or written to. When an emulated process accesses a hardware register, the emulator detects which one, and attempts to emulate the "magic functionality" to some extent. For instance, if the emulated process writes a value to the register corresponding to the serial output port, the emulator redirects that byte to the host desktop computer's serial output port instead.

3 User Interface

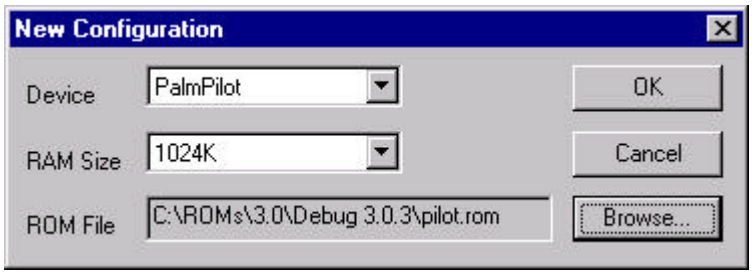
3.1 Launch Behavior

3.1.1 Entering The Application

Users start the Palm OS Emulator by double-clicking on the application icon or one of its session document icons.

If the application is started by double-clicking on its icon, it will attempt to perform some appropriate startup action.

- If the user had saved a session document the previous time the emulator had been run, the emulator will attempt to reload that document.
- If the user created a new session document during the previous session but didn't save that document, the emulator will create a document with the same settings.
- If either of those attempts fails (for instance, the previously used session document or ROM file cannot be found), or this is the first time the user has used the emulator, or the user holds down the Caps Lock key while starting the emulator, the user is presented with a "New Configuration"[1] dialog box. The user uses this dialog box to specify a new hardware/RAM/ROM configuration from which a new document is created.



If the user double-clicks on a session document icon, the emulator attempts to restore the session recorded in that file. This attempt may fail if the ROM image for the session can no longer be found. In that case, the

[1] "New Document"?

session document can no longer be used; session documents can only be used in conjunction with the exact same ROM image that created them.

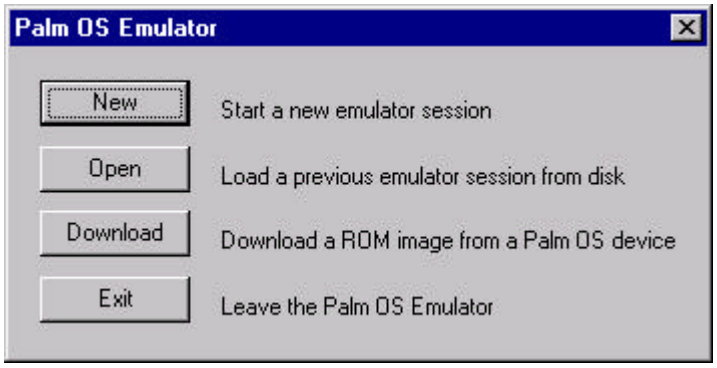
If the user double-clicks on a ROM image icon, the emulator presents the user with the "New Configuration" dialog box with the selected ROM file entered into the ROM File text slot.

3.1.2 Closing the Emulator Document

Closing the emulator window ends the emulation session. Depending on the preference settings, the user may have the option to save the document, or the document may be automatically discarded or saved.

When no document is open, the user has few options available. They can create a new emulation session, open a previously saved session, download a ROM, edit preference settings (Mac only), see the About box (Mac only), or quit the emulator entirely.

On the Mac, these options are available from the menubar, which still exists even when there are no open windows. On Windows, the menu commands are available only by right-clicking on the emulator window. If there is no emulator window, there is nothing on which to right-click. Thus, if the user opts to close the emulator window but not quit the application, they are presented with a dialog box containing buttons for the commands available at that point.[23]



3.1.3 Exiting The Application

When quitting the emulator, the user can be given the option to save their session to disk. Whether or not this option is given to them is controlled by an application preference setting, where the user can specify whether they always want the session saved, never want the session saved, or want to be asked if they want the session saved.

3.1.4 Saving Session State

Session files are the "document" of the Palm OS Emulator. They store the configuration being emulated during the session along with the data generated during the session (that is, a copy of the RAM created during the session).

When a user starts up the emulator for the first time, they must define a particular configuration to emulate. The parts of the configuration that need to be specified (either explicitly or implicitly) are:

- Hardware device (that is, Pilot, PalmPilot, Palm III, etc.)

[2] We can modify this dialog box so that it allows the user to perform the same commands as on the Mac when there is no emulator window open, thus taking care of those "Mac only" comments.
[3] "Download" should probably be "Transfer" to match the menu item.

- RAM size
- ROM

Once this information is provided, the whole package can be saved to the session document. The session document contains all of the information needed for the user to quit the emulator and later restart it at exactly the same point where it was when he quit. In addition to the above information, the following is saved to the session file:

- RAM contents
- CPU state
- Dragonball register state
- Date/time adjustment (Normally, the date and time of the emulated device are tied to the host computer's date/time services. However, the user may want to change the date and/or time to some other setting for any number of reasons.)

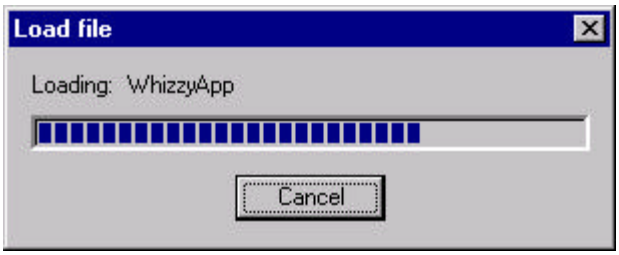
Once a session file is saved, the user can re-open it just like they would any other document in any other application.

3.2 "Autoload" Directory

If there is a directory named "Autoload" in the emulator's directory when a configuration is defined or loaded, all application and database files in that directory are automatically loaded at that time.

3.3 Loading Applications & Databases

Users can load (install) Palm OS application (.prc) and database (.pdb) files into the emulated Palm OS environment. The file is loaded just like ones installed with Palm Install tool or the "import" command from the PalmDebugger.[4]



While the file is being loaded, a progress bar is displayed to the the user know how things are going; some files (especially database files with lots of records) can take a while to load.

When multiple files are being loaded (usually because multiple .prc and/or .pdb files were dragged onto the emulator), a second progress bar appears in the progress dialog indicating how many of the files have been processed.

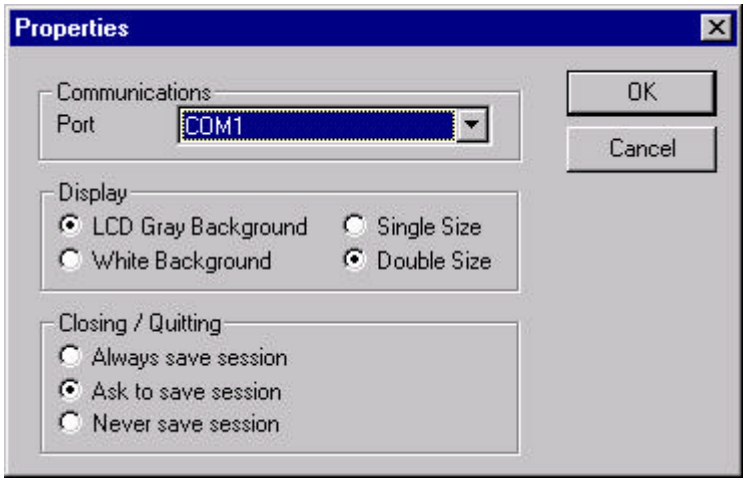
3.4 Preferences

The following options can be controlled from the Preferences / Properties dialog:

- Emulation serial port
- Screen size (1x or 2x)[5][6]

[4] Currently, there is a bug with the install process, in that the Palm OS Launcher application will crash if the application being loaded is already loaded into the emulated device, and hence is being displayed by the Launcher.

- Gray or white background
- Action to take when emulator window is closed (save the document, discard the document, or ask the user)



3.4.1 Implicit preferences

The following pieces of information are specified in locations other than the Properties/Preferences dialog, but nonetheless get saved to the registry/preferences file.

- Last specified hardware device.
- Last specified RAM size.
- Last specified ROM file.
- Last saved session document.
- Window location.
- Recent session files
- Recent .pdb/.prc files

3.5 Debug Options

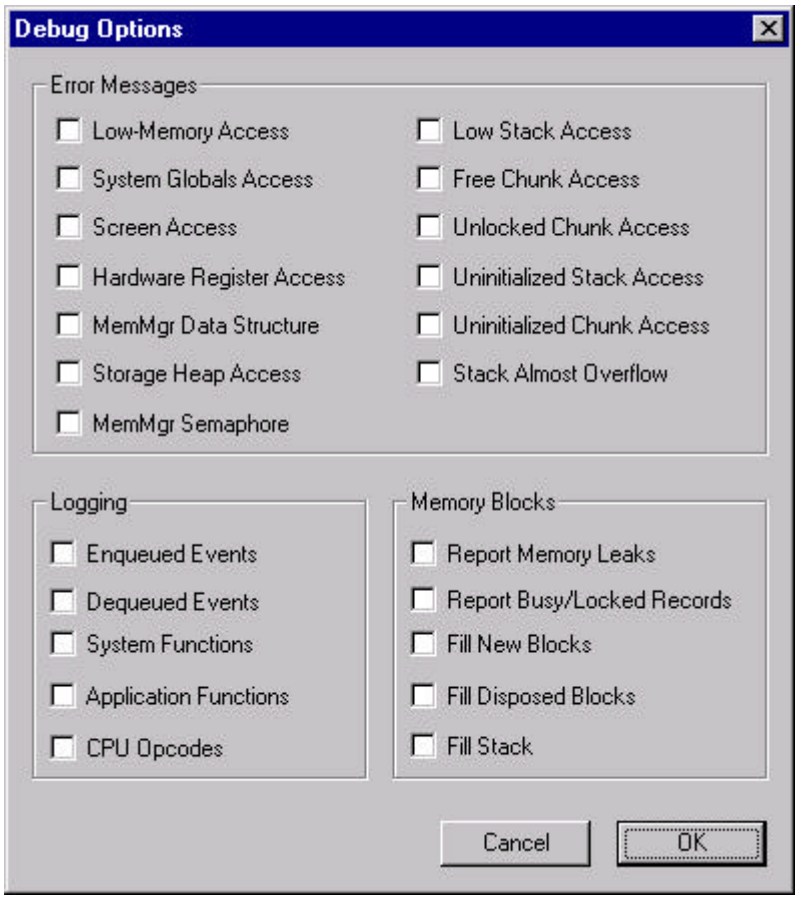
The Palm OS Emulator performs a lot of behind-the-scenes work to ensure the correct operation of the currently executing application. Not only does it emulate the CPU and other aspects of a hardware device, but it also monitors the application's actions, making sure that the application is "well-behaved". When transgressions of Palm's programming guidelines are detected, the emulator presents the user with a dialog describing what happened.

This monitoring is not strictly essential to the successful execution of the application, and can in fact slow down the applications performance. Additionally, a user may know full well that the application they are running is not strictly conforming to Palm's programming guidelines, but may wish to run the application anyway without any annoying dialogs.

For both of these reasons, the user can control what types of monitoring the emulator performs. The options presented in the following dialog are emulator-wide (as opposed to being different for each session document).

---

[5] Also need a way to minimize the sucker, perhaps via the taskbar button.
[6] One user has requested a "Maximize" option for presentation purposes. This option may not be a good idea, as it would require non-integral scaling of the LCD bits. Profiling shows that LCD updating can take a significant amount of time at times.



This dialog presents the user with three classes of "Debug Options".

The first box contains options that control whether or not the emulator monitors the application's access to memory. For instance, the emulator can report when the application reads from or writes to low-memory or causes a "stack overflow". These options are *not* saved in the preferences file, and will always be turned back on when you restart the emulator.

The second box controls what kinds of helpful debugging information the emulator writes to a "Log File" on disk. This Log File contains descriptions of what the application was doing up to the point where the logging was stopped. In addition to the five categories of information logged by the emulator itself, application developers can write their applications to add their own logging information to the Log File.

The third box controls monitoring and manipulation of heap blocks (and one stack-related option thrown in for no really great reason).

3.5.1 Memory Access

3.5.1.1 Low-memory Access

Enables / disables checking of low-memory access by applications / processes running in RAM. A low-memory access is defined as a read from or write to a memory location in the range 0x0000 – 0x00FF.

3.5.1.2 System Globals Access

Enables / disables checking of system global variable access by applications / processes running in RAM. A system global variable access is defined as a read from or write to a memory location in the range 0x0100 to the end of the trap dispatch table.

3.5.1.3 Screen Access

Enables / disables checking of LCD screen buffer access by applications / processes running in RAM. An LCD screen buffer access is defined as a read from or write to the memory range indicated by the hardware registers lcdStartAddr, lcdScreenWidth, lcdScreenHeight, and lcdPanelControl (for screen depth).

3.5.1.4 Hardware Register Access

Enables / disables checking of hardware register access by applications / processes running in RAM. A hardware register access is defined as a read from or write to the memory range 0xFFFFF000 – 0xFFFFFFFF.

3.5.1.5 MemMgr Data Structure

Enables / disables checking of read/write access to Memory Manager data structures. Memory Manager data structures are defined as heap headers, master pointer tables, memory chunk headers and memory chunk trailers (if they exist). Access to these areas of memory are restricted to the Memory Manager; not even other parts of the Palm OS are allowed to touch them.

3.5.1.6 Storage Heap Access

Enables / disables checking of naked access to the storage heap by applications. In general, access to the storage heap should be performed by calling DmRead and DmWrite.

3.5.1.7 MemMgr Semaphore

Enables / disables checking of how long the Memory Manager semaphore has been acquired for write access via MemSemaphoreReserve and MemSemaphoreRelease. Applications should, in general, not be making these calls, but if they do, they should not hold the semaphore for longer than 10 milliseconds.

3.5.1.8 Low Stack Access

Enables / disables checking of low stack access, that is, access to the range of memory below the stack pointer.

3.5.1.9 Free Chunk Access

Enables / disables checking of free chunk access. Nothing, except possibly the Memory Manager, should access the contents of a chunk deallocated with MemChunkFree, MemPtrFree, or MemHandleFree.

3.5.1.10 Unlocked Chunk Access

Enables / disables checking of unlocked relocatable chunk access. Nothing, except possibly the Memory Manager, should access the contents of an unlocked chunk.

3.5.1.11 Uninitialized Stack Access

Enables / disables checking of read access from uninitialized portions of the stack. This option is good for detecting read access to uninitialized local variables.

3.5.1.12 Uninitialized Chunk Access

Enables / disables checking of read access from uninitialized portions of memory chunks allocated by MemChunkNew, MemPtrNew, or MemHandleNew. This option is good for detecting read access to uninitialized portions of dynamically allocated memory blocks. Since an application's global variables are

stored in a memory chunk allocated by these calls, this option is also good for detecting read access to uninitialized globals.

3.5.1.13 Stack Almost Overflow

Enables / disables a “stack sniffer” that ensures that the stack pointer has not dipped below the space allocated for it by the kernel. When enabled, this option warns the user when the application stack is getting close to full. If the stack actually overflows, the user will be told regardless of the setting of this option.

3.5.2 Logging

3.5.2.1 Enqueued Events

Enables / disables the logging of events posted to the event queue.

3.5.2.2 Dequeued Events

Enables / disables the logging of events removed from the event queue. Note that this sequence of events is not necessarily the same as the sequence of events posted to the event queue, as posted events are sometimes morphed into one or more other events before they are pulled off. Additionally, some events are synthesized as needed (such as nil events), and are never really posted onto any queue.

3.5.2.3 System Functions

Enables / disables the logging of system functions that are called.

3.5.2.4 Application Functions

Enables / disables the logging of application functions that are called.

3.5.2.5 CPU Opcodes

Enables / disables the logging of CPU opcodes that are executed. The corresponding CPU state is also included. This information is useful for post-mortems, where an application has crashed due to some reason in the near past.

3.5.3 Memory Blocks

3.5.3.1 Report Memory Leaks

Enables / disables the checking of memory leaks when an emulated application quits. If any are found, a report is created on the disk, and a dialog reporting that there was a problem is presented. The report file contains information that can be used to identify the source of the problem.

3.5.3.2 Report Busy / Locked Records

Enables / disables the checking for locked and/or busy database records when an emulated application quits. If any are found, a report is created on the disk, and a dialog reporting that there was a problem is presented. The report file contains information that can be used to identify the source of the problem.

3.5.3.3 Fill New Blocks

Enables / disables the filling of newly created memory blocks to some standard value. The reason for doing this is to help trip up applications that don’t completely initialize memory that they’ve allocated.

3.5.3.4 Fill Disposed Blocks

Enables / disables the filling of newly disposed memory blocks to some standard value. The reason for doing this is to help trip up applications that attempt to use memory blocks that they’ve disposed.

Overwriting the memory block destroys any data that the application may otherwise be able to use for some time after the block has been disposed.

3.5.3.5 Clean Stack

Enables / disables the filling of the sections of the stack below the stack pointer. The reason for doing this is to help trip up applications that create and user pointers to stack variables from activation frames that no longer exist.

3.6 User Interaction

Users interact with the emulated Palm OS environment using the mouse and the keyboard.

3.6.1 Mouse Used As Pen

In lieu of a pen, the user uses the mouse to interact with the representation of the LCD screen in the emulator window.

The hardware buttons (on/off, the four application buttons, up, and down) are simulated on the emulator display window. Clicking on the button images activate them. Holding down the mouse button is the same as holding down the corresponding Palm OS device button.

3.6.2 Function Keys

The buttons are also simulated on the desktop keyboard as follows:

Palm Computing Hardware Platform device button	**Desktop Key**[7]
On/Off	Esc
Date Book	F1
Address	F2
To Do List	F3
Memo Pad	F4
Up	Page Up
Down	Page Down

3.6.3 Text Entry

Users can type on their desktop keyboards to enter text into the emulated device. The characters typed are automatically inserted into the Palm OS event queue. This means that users can enter data with their keyboard instead of trying to do Graffiti strokes with their mouse.

3.7 Drag & Drop

Users can use Drag & Drop in the following ways:

- Drag a session document file onto an existing emulator window to close that window and open the dragged session document.
- Drag a .prc or .pdb file (or multiple such files) onto an existing emulator window to install those files into the current emulator session.

[7] One reviewer suggested that the F1-F4 keys be changed to F9-F12 to match the key assignments in the Palm OS Simulator. However, these keys have been defined for other functions under Mac OS 8.1, and apparently can't be defined by the application.

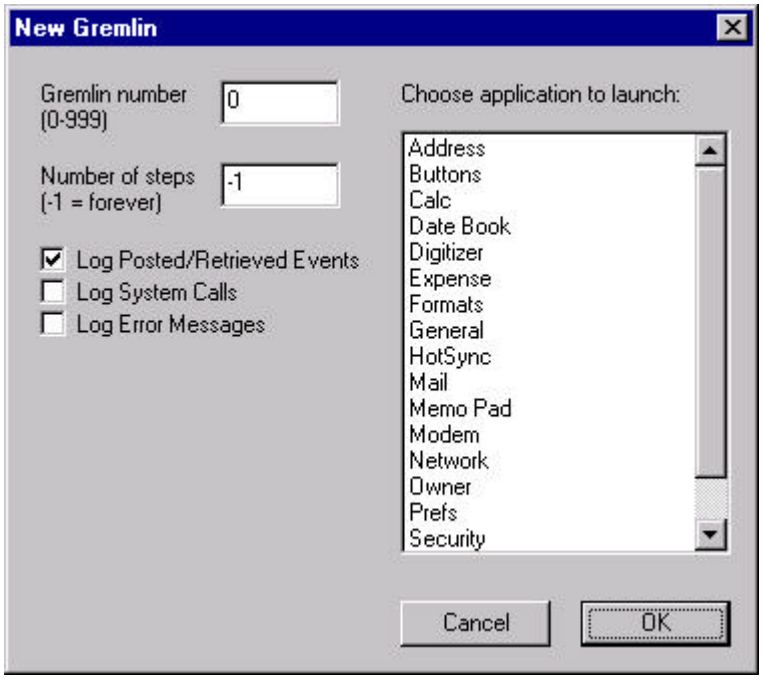
- Drag a ROM image file onto an existing emulator window to open a "New Configuration" dialog based on that ROM.

3.8 Gremlins

Gremlins is a facility to generate random pen and key events. Each Gremlin is a unique sequence of random taps, strokes, and so on. The user can use Gremlins to reveal program problems.

Gremlins range from 0 to 999 and each Gremlin is repeatable.

Selecting the "Gremlin / New" menu item to start a Gremlin session brings up the following dialog:



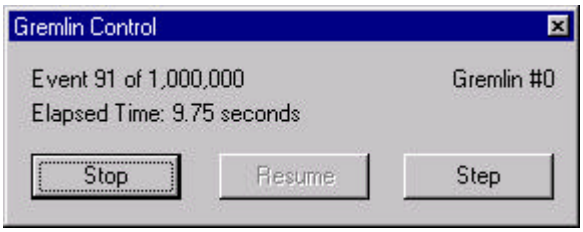
The "Gremlin number" is the seed value used for the random number generator used to create the random events posted to the operating system.

"Number of steps" is the number of events the user wants posted to the application. Entering "-1" means to continue posting events until the user selects the "Gremlin / Stop" menu item.

The list of loaded applications (including other application-like executables, such as control panels) lets the user specify which application or applications to run Gremlins on. Initially, the last loaded or selected application(s) is selected by default.

The checkboxes enable the logging of information that may be useful to the developer when debugging his/her application. The third checkbox causes the text of any error dialogs that come up during the execution of the Gremlin to be written to the log file, and for the dialog's "Continue" button to be automatically pressed.

When a Gremlin is running, the following dialog box is displayed to monitor and control the process:



When a Gremlin is running, the only source of events for the emulated environment is the Gremlin subsystem. That is, users can't interact with the emulated environment with the mouse or keyboard.

Selecting "Gremlin / Stop" stops the current Gremlin. After that, the Gremlin can be resumed with "Gremlin / Resume" or single-stepped with "Gremlin / Step".

3.9 Profiling

"Profiling" is the process of running an application through one or more scenarios and monitoring that application to determine where it is spending most of its time. Profiling is used to determine how applications can be made to run more quickly.

A profiling session is started with the "Profiling / Start" menu item. Choosing "Profiling / Initialize" can optionally be done beforehand to set some profiling-related values.

While a profiling session is active, the emulator monitors which application and system functions are being executed and how long those functions take to execute. That timing information is collected until the user selects "Profiling / Stop". Selecting "Profiling / Dump" menu item writes the collected information to disk in a format compatible with the "MW Profiler" application that comes with CodeWarrior Pro for Macintosh.[8]

Unlike the profiling facilities found in development environments like VC++, CodeWarrior, and MPW, the profiled application does not have to be specially prepared before it can be profiled. In those environments, such preparation is necessary for the profiling framework to find out when functions are entered and exited. An emulator, on the other hand, can figure that stuff out on its own (it knows when JSR and RTS instructions are being executed). In an emulator, the profiling calculations are handled "between cycles". It is, however, a good idea to set the compiler switch so that the debug symbols get embedded in your code, otherwise you won't be able to identify your functions in the output.

3.10 Error Handling

This section discusses reporting and handling errors in the Palm OS Emulator. It describes what errors/conditions are detected, what happens when they occur, what messages are displayed, and what options are available to the user.

When something untoward occurs, the Palm OS Emulator generates some error text. The Palm OS Emulator then tries to send this text to an external debugger (such as PalmDebugger or MWDebug) and "enters" the debugger (that it, it halts opcode emulation until the debugger says it's OK to go).[9]

[8] There currently is no corresponding solution for Windows users.

[9] The Palm OS Emulator may not be able to easily send this text to the debugger. First, there are some issues right now with trying to send the text via the DbgMessage facility. Second, we'd need to bring the external debugger forward to highlight the fact that we've just entered the debugger; this may not be possible.

If it can't send the text (for example, there's no external debugger running), the Palm OS Emulator presents the error text to the user. The text is presented in two parts: an initial user-friendly message, and a second, optional, developer-friendly message. In this document, these are referred to as the "first-line" and "second-line"[10] messages.

The Palm OS Emulator first presents the first-line message in a dialog box. This dialog box has three buttons in it: Continue, Reset, and More Info. Clicking on Continue (which may not always be enabled, depending on the severity of the error) attempts to continue opcode processing. Clicking on Reset emulates a soft reset. Clicking on More Info brings up a second dialog containing the second-line message.

When the second-line message is displayed, the dialog box also has three buttons: Continue, Reset, and Debug. Continue and Reset do the same things as the same buttons in the first dialog box. Clicking on Debug attempts to again enter the debugger. By the time the user clicks on this button, the developer has presumably launched a suitable external debugger.[11]

The Palm OS Emulator detects and reports three types of conditions: exceptions, memory access, and application error messages. Exceptions are processor exceptions that involve the CPU pushing the current PC and Processor State on the stack and then branching through a low-memory vector. Memory access conditions are those in which some executing code accesses (reads or writes are differentiated from each other) some memory location it shouldn't. Application error messages are messages displayed when something invokes the ErrDisplayFileLineMsg or SysFatalAlert system functions.

When the Palm OS Emulator checks memory accesses, it generally applies four levels of accessibility: application, system, memory manager, and none. Applications have the least access to memory, the system has more, and the memory manager has most. There are also sections of memory that *nothing* should access. Currently, an "application" is anything running in RAM, the "system" is anything running in ROM, and the memory manager is any function operating within the context of a memory manager call (that is, a memory manager function has been called but has not yet exited).[12][13]

Error messages are displayed via "error message templates". Error message templates contain "message parameters" interspersed with the main error text. In the error message templates below, percent (%) signs indicate the message parameters. When the final message is prepared, these parameters are replaced with the appropriate text. In general:

- %Application and %application are replaced with the name of the currently running application (even if the error occurred in a system call). If the Palm OS Emulator can't determine the current application's name, %Application is replaced with "The current application" and %application is replaced with "the current application".
- %version is replaced with the current application's version information (as stored in the 'tver'(1) and 'tver'(1000) resources). If the Palm OS Emulator can't determine this information, %version is replaced with "(unknown version)".
- %operation is replaced with some verb. In many cases, the verb is "read directly from" or "written directly to".

Other parameter replacements should be apparent from their context and names.

[10] It's possible that "second-line" messages won't be implemented for 2.1.
[11] If not, the Palm OS Emulator can just keep presenting the dialog box.
[12] A facility needs to be added that detects patches or system updates running in RAM.
[13] The Palm OS Emulator tracks what system functions are called by tracking TRAP $F invocations. However, a mechanism is afoot (SYSTRAP_FASTER) that would allow the ROM to call other ROM functions without using the TRAP $F mechanism. If/when we move over to that mechanism, a new method would be needed to track system function calls.

3.10.1 Hardware Register Access

The Palm OS Emulator detects and reports application access to the Dragonball and Dragonball EZ hardware registers.

Users can prevent this message from appearing by selecting a preference option.

3.10.1.1 First-line message

%Application %version has just %operation the hardware registers.

This technique is not permitted for Palm OS applications and may not work in future Palm products which have hardware changes. %Application will likely crash this or future Palm devices. When run on a real Palm device, a soft reset with a paper clip will usually restore the Palm device without data loss although a hard reset and full data loss are known to happen.

Users should upgrade this product immediately to safeguard their data.

3.10.1.2 Second-line message

Memory location = 0xFFFFFxxx
Register name = fooRegister
PC = 0x00000000
Function = FooFunction + 0x1234 (Perhaps several of these for a stack crawl)

3.10.2 Low-memory Access

The Palm OS Emulator detects and reports *all* access to low-memory (the first 256 bytes, containing the exception vectors). Some allowances are made during boot-up (when the vectors are initialized), HwrSleep (when some vectors are replaced), and PrvSystemTimerProc, where the Debug ROMs perform the low-memory checksum calculation.[14][15][16]

Users can prevent this message from appearing by selecting a preference option.

3.10.2.1 First-line message

%Application %version has just %operation low memory.[17]

This technique is not permitted for Palm OS applications and may not work in future Palm products which have hardware changes. %Application will likely crash this or future Palm devices. When run on a real Palm device, a soft reset with a paper clip will usually restore the Palm device without data loss although a hard reset and full data loss are known to happen.

Users should upgrade this product immediately to safeguard their data.

3.10.2.2 Second-line message

Memory location = 0x000000xx
PC = 0x00000000

[14] An exception is made for GrfProcessStroke in 3.0 and earlier ROMs. A bug in that function causes it to read 2 bytes from memory location 2.
[15] An exception is made for BackspaceChar. A bug in that function causes it to read 2 bytes from NULL.
[16] An exception is made for NetPrvTaskMain in 3.0 and earlier ROMs. A bug in that function causes it to read from low-memory.
[17] If the memory location happens to be NULL, the message is: "%Application %version has just %operation NULL (memory location zero)".

Function = FooFunction + 0x1234

3.10.3 System Variable Access

The Palm OS Emulator detects and reports application access to the system variables. System variable space is defined as the range of memory starting just after low-memory and extending to the end of the system function dispatch table. Presumably, this takes us up to the heap header for the dynamic heap.

Users can prevent this message from appearing by selecting a preference option.

3.10.3.1 First-line message

%Application %version has just %operation Palm OS global variables.

This technique may not work in future Palm OS versions. Palm OS provides supported calls that will generally achieve the same result and work in future Palm OS versions. Because %application bypasses this support, it will likely crash future Palm OS versions. When run on a real Palm device, a soft reset with a paper clip will usually restore the Palm device without data loss although a hard reset and full data loss is possible.

Users should expect to upgrade this product for future Palm OS versions.

3.10.3.2 Second-line message

Memory location = 0x00001234
Global name = GSysFooVariable
PC = 0x00000000
Function = FooFunction + 0x1234

3.10.4 Screen Buffer Access

The Palm OS Emulator detects and reports application access to the LCD screen buffer, as defined by the Dragonball registers lcdPanelControl, lcdScreenWidth, lcdScreenHeight, and lcdStartAddr.

Users can prevent this message from appearing by selecting a preference option.

3.10.4.1 First-line message

%Application %version has just %operation screen memory.

This technique may not work in future Palm products that have hardware changes. Palm OS provides supported calls that will both achieve the same result and work in future hardware. Because %application bypasses this support, it will likely crash future Palm devices. When run on a real Palm device, a soft reset with a paper clip will restore the device and no data loss should occur.

Users should expect to upgrade this product for future Palm devices.

3.10.4.2 Second-line message

Memory location = 0x00000000
Screen start/end = 0x0000xxxx / 0x0000xxxx
PC = 0x00000000
Function = FooFunction + 0x1234

3.10.5 Memory Manager Data Structure Access

The Palm OS Emulator detects and reports application and system access to memory manager data structures. Memory manager data structures include heap headers, master pointer tables[18], chunk headers, and chunk trailers.[19][20][21][22][23][24][25][26]

Users can prevent this message from appearing by selecting a preference option.

3.10.5.1 First-line message

%Application %version has just %operation memory manager data structures.

This problem indicates an error in the application. Users should upgrade this product immediately to safeguard their data.

3.10.5.2 Second-line message

Memory location = 0x00000000
One of the following:
That location is in the heap header.
That location is in a master pointer block.
That location is between two allocated chunks. (Possibly list those chunks, showing their locations, sizes, and possibly their contents.)
PC = 0x00000000
Function = FooFunction + 0x1234

3.10.6 Unlocked Chunk Access

The Palm OS Emulator detects and reports application and system access to unlocked memory chunks. Unlocked memory chunks are chunks with a lockcount of zero.[27]

Users can prevent this message from appearing by selecting a preference option.

3.10.6.1 First-line message

%Application %version has just %operation an unlocked memory chunk.

This problem indicates an error in the application. Users should upgrade this product immediately to safeguard their data.

---

[18] Actually, the Palm OS Emulator allows system access to master pointer tables in order to support the MemDeref macro.
[19] One nice check to make would be to detect when a base+offset access is made where base is in one chunk but base+offset is outside that chunk (and perhaps "safely" inside of another chunk).
[20] Exceptions are made for PrvMisAlignedForwardInnerBitBlt and PrvMisAlignedBackwardInnerBitBlt, which sometimes get a running start and come to a skidding halt outside of a memory chunk's bounds.
[21] An exception is made for MenuHandleEvent in 3.0 and earlier ROMs. A bug in that function causes it to possibly access invalid memory. This bug has been fixed in post-3.0 ROMs.
[22] An exception is made for NetPrvSettingSet. A bug in that function causes it to possibly access invalid memory.
[23] DmWrite and PrvFindMemoryLeaks are also allowed to access memory manager data structures.
[24] An exception is made for callback functions called by pre-3.0 versions of SysBinarySearch. A bug in that function can pass a pointer to a non-existent n+1st element.
[25] An exception is made for FindSaveFindStr, which can read off the end of the source handle.
[26] An exception is made for FntDefineFont, which can read off the end of a small buffer it's resizing.
[27] An exception is made for FindShowResults in 2.0 and earlier ROMs. A bug in that function causes it to use a stale pointer to access an unlocked handle.

3.10.6.2 Second-line message

Memory location = 0x00000000
Chunk location/size: 0x00000000 / 0x00000000
First 16 or 32 bytes of chunk
PC = 0x00000000
Function = FooFunction + 0x1234

3.10.7 Low-stack Access

The Palm OS Emulator detects and reports *all* access to areas of the stack below the stack pointer. The stack is defined by values in the result of SysGetAppInfo when SysAppStartup is called. If the Palm OS Emulator can't determine the stack range, low-stack access checking is not performed.

Users can prevent this message from appearing by selecting a preference option.

3.10.7.1 First-line message

%Application %version has just %operation an invalid section of memory known as the "stack".

This problem indicates an error in the application. Users should upgrade this product immediately to safeguard their data.

3.10.7.2 Second-line message

Memory location = 0x00000000
Stack bottom (A7) = 0x00000000
PC = 0x00000000
Function = FooFunction + 0x1234

3.10.8 Uninitialized Stack Access[28]

The Palm OS Emulator detects and reports *all* read access to uninitialized memory[29][30]. No executing code should read from a memory location without it first being written to.[31]

Users can prevent this message from appearing by selecting a preference option.

3.10.8.1 First-line message

%Application %version has just %operation an uninitialized section of memory known as the "stack".

This problem indicates an error in the application. Users should upgrade this product immediately to safeguard their data.

---

[28] This one may get merged with "Uninitialized Chunk Access" into a general "%Application has just %operation uninitialized memory" message. This message would further define what section of memory was accessed (memory chunk, stack, etc.).

[29] Exceptions would be made for some parts of the system, such as that which detects if the memory card has been formatted during bootup.

[30] This feature will likely not make it into the Palm OS Emulator 2.1.

[31] As we found out in dress rehearsal, this isn't always true. There are some perfectly valid cases where reads of uninitialized memory can be performed before a write. For instance, if an application applies a struct to a range of memory, and that struct has bitfields in it, and the application attempts to initialize the bitfields, the generated code will first read in the (byte, word, long) containing the bitfield and partially update that value before writing it back out. Tracking this type of operation is the reason why uninitialized memory access will probably be deferred.

3.10.8.2 Second-line message

Memory location = 0x00000000
Stack bottom (A7) = 0x00000000
Stack frame (A6) = 0x00000000
PC = 0x00000000
Function = FooFunction + 0x1234

3.10.9 Free Chunk Access

The Palm OS Emulator detects and reports application and system access to unallocated memory chunks.[32]

Users can prevent this message from appearing by selecting a preference option.

3.10.9.1 First-line message

%Application %version has just %operation an unallocated chunk of memory.

This problem indicates an error in the application. Users should upgrade this product immediately to safeguard their data.

3.10.9.2 Second-line message

Memory location = 0x00000000
Chunk location/size: 0x00000000 / 0x00000000
First 16 or 32 bytes of chunk
PC = 0x00000000
Function = FooFunction + 0x1234

3.10.10 Uninitialized Chunk Access

The Palm OS Emulator detects and reports *all* read access to uninitialized memory. No executing code should read from a memory location without it first being written to.

Users can prevent this message from appearing by selecting a preference option.

3.10.10.1 First-line message

%Application %version has just %operation an uninitialized chunk of memory known.

This problem indicates an error in the application. Users should upgrade this product immediately to safeguard their data.

3.10.10.2 Second-line message

Memory location = 0x00000000
Chunk location/size: 0x00000000 / 0x00000000
First 16 or 32 bytes of chunk
PC = 0x00000000
Function = FooFunction + 0x1234

[32] An exception is made for ScrInit. The LCD buffer is set up before the Memory Manager is initialized. Later, when the Memory Manager is initialized, the LCD buffer happens to fall within an unallocated chunk. When ScrInit eventually allocates a *real* chunk to use as an LCD buffer, it needs to copy the contents of the old buffer into the new one.

3.10.11 Storage Heap Access

The Palm OS Emulator detects and reports application access to the storage heap.

Users can prevent this message from appearing by selecting a preference option.

3.10.11.1 First-line message

%Application %version just tried to write to the storage heap and that's just plain not allowed! Try using DmWrite.[33]

3.10.11.2 Second-line message

Memory location = 0x00000000
Chunk location/size: 0x00000000 / 0x00000000
First 16 or 32 bytes of chunk
PC = 0x00000000
Function = FooFunction + 0x1234

3.10.12 Stack Overflow

The Palm OS Emulator detects and reports stack overflows. Stack overflows are were an application pushed more information onto the stack than was allocated for the stack.

3.10.12.1 First-line message

%Application %version has just overflowed its stack.

This problem indicates an error in the application. Users should upgrade this product immediately to safeguard their data.

3.10.12.2 Second-line message

TBD

3.10.13 Stack Almost Overflow

The Palm OS Emulator detects and reports conditions where the stack is close to overflowing. "Close to overflowing" means that the stack pointer is within a small number of bytes (currently 100) from the end of the stack.

Users can prevent this message from appearing by selecting a preference option.

3.10.13.1 First-line message

%Application %version is getting close to overflowing the stack.

While not an error, this indicates that the application is running close to the edge. For future robustness, the application should be modified to either use a larger stack or use less space in the current stack.

3.10.13.2 Second-line message

TBD

[33] Error text courtesy of Steve Lemke.

3.10.14 Memory Manager Semaphore Acquisition time

The Palm OS Emulator keep track of how long the memory manager semaphore has been acquired for write access and reports if it has been acquired for more than 10 milliseconds[34].

Users can prevent this message from appearing by selecting a preference option.

3.10.14.1 First-line message

> %Application %version has held the "Memory Manager semaphore" for approximately %elapsed milliseconds. It is recommended that applications not hold the semaphore longer than 10 milliseconds

3.10.14.2 Second-line message

> TBD

3.10.15 Invalid Heap

The Palm OS Emulator frequently needs to traverse the contents of the dynamic heap. While doing so, it makes certain sanity checks to make sure the heap data structures have not been corrupted. The Palm OS Emulator will report any corruption found.

3.10.15.1 First-line message

> During a regular checkup, Poser determined that the dynamic heap got corrupted. %corruption_type.

%corruption_type is one of the following message fragments:

- The chunk was not within the heap it was supposed to be
- The size of the chunk (%chunk_size) was larger than the currently accepted maximum (%chunk_max)
- Some unused flags were set to "1"
- The "hOffset" field of the chunk header did not reference a memory location within a master pointer block
- The master pointer referenced by the "hOffset" field in the chunk header didn't point back to the chunk
- The lock count for a fixed block was not 15

3.10.15.2 Second-line message

> TBD

3.10.16 Invalid Program Counter

The Palm OS Emulator detects and reports when the Program Counter (PC) has been set to an invalid memory location. An invalid memory location for a PC is one outside of a 'CODE' resource.

3.10.16.1 First-line message

> %Application %version has just set the Program Counter (PC) to an invalid memory location.
>
> This problem indicates an error in the application. Users should upgrade this product immediately to safeguard their data.

[34] Times are based on the same timing mechanisms used for profiling. That is, instruction cycles and memory access cycles are determined and added up.

3.10.16.2 Second-line message

TBD

3.10.17 Unimplemented Trap

The Palm OS Emulator detects and reports attempts to invoke unimplemented system functions. Unimplemented system functions are those with trap numbers outside the range of the system function dispatch table, or whose entry in the table matches that of the SysUnimplemented function.[35]

3.10.17.1 First-line message

%Application %version tried to call Palm OS routine #%trapNum (%trapName). This routine does not exist in this version of the Palm OS.

This problem indicates an error in the application. If this is the latest version of %application, please report this to the application author.

3.10.17.2 Second-line message

PC = 0x00000000
Function = FooFunction + 0x1234

3.10.18 SysFatalAlert

The Palm OS Emulator patches the SysFatalAlert function and presents the message in its own dialog. It intercepts the trap so that the user can have the option of continuing, entering the debugger or resetting the CPU.

3.10.18.1 First-line message

%Application %version has failed, reporting "%message". If this is the latest version of %application, please report this to the application author.

3.10.18.2 Second-line message

PC = 0x00000000
Function = FooFunction + 0x1234

3.10.19 Unhandled Exception

The Palm OS Emulator catches all exceptions and attempts to handle them in some way. For instance, TRAP $F exceptions result in a system function being called, TRAP $0 and TRAP $8 result in the debugger being entered, etc. If the Palm OS Emulator can't find a part of itself to handle the exception, it presents the user with a dialog box saying what happened.

3.10.19.1 First-line message

%Application %version has just performed an illegal operation. It performed a "%exception". If this is the latest version of %application, please report this to the application author.

3.10.19.2 Second-line message

PC = 0x00000000
Function = FooFunction + 0x1234

[35] Regardless of the ROM, sysTrapHostControl is *always* implemented when running under the Palm OS Emulator.

3.11 Scripting[36][37]

Gremlins is a fabulous tool for testing, but it can't be directed. That is, it can't be used to test a specific part of an application. Nor, when using it, can developers be 100% sure that the entire application gets covered.

For developers who want or need more control over how events are sent to the application, a built-in scripting language will be added. Such a scripting language would allow developers to create scripts that say "Click here", or "Tap on that button", or " Enter this sequences of characters". By building up such scripts, developers can ensure that the entire application gets coverage, and has a powerful tool for targeted regression testing.

<<< Need startup scripts. >>>

There are four candidates for the scripting language: Perl, Tcl, GUILE, and LUA. All scripting systems are free and come with source code.

Perl – http://language.perl.com/index.html

> (From the Web page) Perl is an interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language for many system management tasks. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal). It combines (in the author's opinion, anyway) some of the best features of C, sed, awk, and sh, so people familiar with those languages should have little difficulty with it. (Language historians will also note some vestiges of csh, Pascal, and even BASIC-PLUS.) Expression syntax corresponds quite closely to C expression syntax. Unlike most Unix utilities, Perl does not arbitrarily limit the size of your data--if you've got the memory, Perl can slurp in your whole file as a single string. Recursion is of unlimited depth. And the hash tables used by associative arrays grow as necessary to prevent degraded performance. Perl uses sophisticated pattern matching techniques to scan large amounts of data very quickly. Although optimized for scanning text, Perl can also deal with binary data, and can make dbm files look like associative arrays (where dbm is available). Setuid Perl scripts are safer than C programs through a dataflow tracing mechanism which prevents many stupid security holes. If you have a problem that would ordinarily use sed or awk or sh, but it exceeds their capabilities or must run a little faster, and you don't want to write the silly thing in C, then Perl may be for you. There are also translators to turn your sed and awk scripts into Perl scripts.
>
> Embeddable and Extensible
> Perl may now be embedded easily in your C or C++ application, and can either call or be called by your routines through a documented interface. The XS preprocessor is provided to make it easy to glue your C or C++ routines into Perl. Dynamic loading of modules is supported.

Tcl - http://www.scriptics.com/

> Tcl's advantage is that it is more popular than Guile and therefore possibly more approachable by developers, especially by those with previous Tcl experience. It also has Tk for developers who would like to have a simple UI for their scripts. (Actually, it looks like Perl supports Tk, too).

GUILE - http://www.gnu.org/software/guile/guile.html

> (From the Web page) GUILE uses Scheme, a powerful yet simple dialect of Lisp. One advantage of GUILE over TCL is that Scheme is a more powerful language. Scheme was designed as a

[36] This section is incomplete.
[37] Scripting will not likely be added to 2.1. If developers say they want it, we could add it to a subsequent version and call it 3.0.

"programming language", not as a "scripting language". Scheme is also simpler and cleaner than other extension languages such as Perl and Python. But the big advantage of GUILE is that it allows support for multiple languages. This is because Scheme is powerful enough that other languages can conveniently be translated into it. (In fact, GUILE already comes with CTAX, a C-like language.)

LUA - http://www.tecgraf.puc-rio.br/~lhf/lua/

Python – http://www.python.org

3.11.1 User API

Following is a suggested set of user-callable functions from Steve Haneman.

Note: char* controlStr can be in this form: "text", "#id", "@index". The Palm OS Emulator then detects whether it is a string or needs to convert it to an ID. For example: there is a button with the text = "MyButton" and ID = 100. ButtonFind("MyButton", 5) would try to find a button with text = "MyButton". ButtonFind("#100", 5) would try to find a button with ID = 100. ListItemSelect("#100", "@1") would select the first item from a List with ID = 100.

Note: All APIs should return a zero on success and a non-zero on failure.

3.11.1.1 Priority 1 APIs

```
long StartEmulator(char *machine type, long ramSize, char *romPath);
```

Start up the Palm OS Emulator with specified machine type, RAM size and ROM file.

```
long ShutDownEmulator();
```

Shut down the Palm OS Emulator.

```
long LoadApp(char *fileName);
long LoadAndRunApp(char *fileName);
```

Load and/or run a PRC file. Pass in a PRC file name.

```
long RunApp(char *appName);
```

Run an app that is already in memory. Pass in the apps name.

3.11.1.2 Priority 2 APIs

Form:

```
long FormFind(char* controlStr, long timeout);
long FormGetTitle(char* controlStr);
```

Menu:

```
long MenuFind(char* controlStr, long timeout);
long MenuItemFind(char* menuControlStr, char* itemControlStr, long timeout);
long MenuItemSelect(char* menuControlStr, char* itemControlStr);
```

Button:

```
long ButtonFind(char* controlStr, long timeout);
long ButtonPress(char* controlStr);
```

CheckBox:

```
long CheckBoxFind(char* controlStr, long timeout);
long CheckBoxCheck(char* controlStr);
long CheckBoxUncheck(char* controlStr);
```

Field:
```
long FieldFind(char* controlStr, long timeout);
```

List:
```
long ListFind(char* controlStr, long timeout);
long ListItemFind(char* listControlStr, char* itemControlStr, long timeout);
long ListItemSelect(char* listControlStr, char* itemControlStr);
```

Popup Trigger:
```
long PopupTriggerFind(char*  controlStr, long timeout);
long PopupTriggerItemSelect(char* ptControlStr, char* itemControlStr);
```

Push Button:
```
long PushButtonFind(char* controlStr, long timeout);
long PushButtonItemSelect(char* controlStr, char* itemControlStr);
```

Scrollbar:
```
long ScrollbarFind(char* controlStr, long timeout);
long ScrollbarUp(char* controlStr, long lineNum);
long ScrollbarDown(char* controlStr, long lineNum);
```

Selector:
```
long SelectorFind(char* controlStr, long timeout);
long SelectorItemFind(char* selectorControlStr, char* itemControlStr, long timeout);
long SelectorTap(char* controlStr);
long SelectorItemSelect(char* selectorControlStr, char* itemControlStr);
```

Table:
```
long TableFind(char* controlStr, long timeout);
long TableItemTap(char* tableControlStr, int row, int column);
long TableItemEdit(char* tableControlStr, int row, int column);
```

Others:
```
long Pause(long mseconds);
```
Pause the script.

```
long GetObjectInfo(long objectID, ObjectInfo objectInfo);
```

ObjectInfo is a structure that contains some information about the object such as: Left Origin, Top Origin, Width, Height, Label, ID.

```
long WriteGraffiti(char* string);
```

Simulates writing into the Graffiti area.

```
long PenDrag(Point start, Point end);
```

Simulates a Pen dragging from one point to another.

```
long ScreenCapture(char* filename);
```

Captures a screen and saves it to a file.

```
long ScreenCompareScreen(char* filename);
```

Compares a saved file to the screen.

```
long ScreenCompareFiles(char* filename1, char* filename);
```

Compares two saved files.

```
long HilightData(long objectID, long startOffset, long endOffset);
```

Hilight data in specified object. Pass in start and end.

```
long TapArea(long x, long y);
```

Tap area indicated by x and y coordinates.

3.12 Serial Port Emulation

Serial port operations performed by the emulated application or Palm OS are supported. When the CPU's serial port hardware registers are accessed, the appropriate operations are mirrored with the desktop computer's serial port[38]. Thus, when outgoing data is written to the emulated UART's "tx" register, that data is redirected to the desktop computer's serial port. When the emulated process attempts to read data from the emulated UART's "rx" register, data is read from the computer's serial port and placed in that register.

3.13 Sound Emulation

Calls to the emulated Palm OS's sound functions result in the emulator calling the corresponding sound functions on the host OS. Note that emulation is thus performed at a higher level than, say, serial port emulation. Rather than monitoring accesses to the appropriate registers at the hardware level, monitoring is performed at the system function level.

3.14 Auto-off

Auto-off is turned off by default. The user can use the General Preference panel to turn auto-off back on. "Resetting" the device will turn auto-off back off.

3.15 Menu Commands

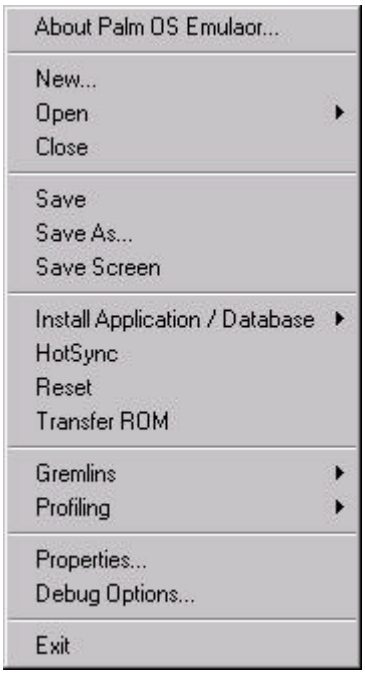
On the Macintosh, the menubar is laid out as described below. On Windows, the emulator doesn't have a menubar. Instead, right-clicking anywhere within the emulator window activates the menu. Alternatively, the user can press Shift-F10 to activate the menu.

The Windows and Macintosh menu arrangements are identical except for the following differences:

- The Macintosh menu bar is turned into a popup menu on Windows.
- The Windows menu doesn't have Undo, Cut, Copy, or Clear menu items.
- The Windows "options" menu item is called Properties; on the Mac it is called Preferences.

The Windows popup menu is displayed below.

[38] This approach has turned out to be pretty flaky. We may instead merely patch out all the SerFoo routines and implement them with native code.



3.15.1 About Palm OS Emulator…

Displays a dialog box containing application name and version number. Also gives credit to the past emulator contributors, along with contact information (in the form of mailto and http URL hyperlinks).

3.15.2 New…

Opens a dialog allowing the user to specify a ROM file, hardware type, and RAM size to be emulated. A new emulator window is opened based on this information.

3.15.3 Open

A hierarchical menu item letting the user open a session document. The submenu contains an "Other…" menu item allowing the user to select a session document on disk, and subsequent items that target recently-used session documents.

3.15.4 Close

Closes the emulation session, allowing the user to save the current state.

3.15.5 Save

Saves the current RAM image and associated data (CPU state, associated ROM file, etc.) to an existing file.

3.15.6 Save As…

Saves the current RAM image and associated data (CPU state, associated ROM file, etc.) to a new file.

3.15.7 Save Screen

Allows the user to take a snapshot of the current LCD image. The snapshot will be saved to a .BMP file on Windows and a PICT file on the Mac[39]. The snapshot will appear exactly as it does on the screen. That is, if the user has the LCD in "2x" mode, the snapshot will be 2x.

3.15.8 Install Application / Database

Allows the user to specify one or more .prc or .pdb files for loading into the current emulator. The submenu contains an "Other…" menu item allowing the user to select a .prc file on disk, and subsequent items that target recently-installed .prc files.

3.15.9 HotSync

Simulates the pressing of the HotSync button on the cradle, causing the ROM to attempt to HotSync over the serial port.

3.15.10 Reset

Simulates a device reset.[40]

3.15.11 Transfer ROM

Starts a ROM download session.

3.15.12 Gremlins

3.15.12.1 New

Starts a new Gremlin session. Brings up a dialog allowing users to specify the following:

- Which application or applications to be run while Gremlins is running
- Which Gremlin number to run (0-999)
- How many Gremlin events to simulate
- What kind of logging to perform

3.15.12.2 Step

After a Gremlin has been started with "New" and stopped with "Stop", this menu item can be used to emulate a single event.

---

[39] We may want to simply put the snapshot onto the clipboard, so that the user doesn't have to

- Futz around with finding the file,
- Opening it,
- Copying the contents,
- Closing the file,
- Deleting the file,
- Going to the document where they probably wanted the snapshot in the first place,
- And pasting it there.

[40] If the Ctrl key is held down during a Reset, a cold/hard reset is performed. Note that this applies to *any* reset, including a reset performed as part of loading a session document, or clicking on Reset in an error dialog.

3.15.12.3 Resume

After a Gremlin has been started with "New" and stopped with "Stop", this menu item can be used to restart continuous event emulation.

3.15.12.4 Stop

Stops the current Gremlin from inserting emulated events. The elapsed time since the Gremlin was started and the number of events emulated are displayed in a dialog box.

3.15.13 Profiling

3.15.13.1 Initialize

Establishes parameters used while profiling, such as buffer sizes.

3.15.13.2 Start

Starts a profiling session. Subsequent to selecting this item, all executed functions will be timed so that performance problems can be detected and analyzed.

3.15.13.3 Stop

Stops a profiling session. Function execution time will no longer be recorded.

3.15.13.4 Dump

Dumps the results of the last profiling session to a file.

3.15.14 Preferences… / Properties…

This menu item is "Preferences" on the Mac and "Properties" on Windows.

3.15.15 Debug Options…

Displays the Debug Options dialog box.

3.15.16 Quit / Exit

This menu item is "Quit" on the Mac and "Exit" on Windows.

3.15.17 Edit Menu (Mac only)

3.15.17.1 Undo

Standard menu item. May or may not actually do anything.

3.15.17.2 Cut

Standard menu item. May or may not actually do anything.

3.15.17.3 Copy

Standard menu item. May be used for clipboard data exchange (that is, users may be able to Copy text from a text field in an emulated application into their desktop clipboard).

3.15.17.4 Paste

Standard menu item. May be used for clipboard data exchange (that is, users may be able to paste text from their desktop clipboard into a text field in an emulated application).

3.15.17.5 Clear

Standard menu item. May be used to clear the selected text in a text field in an emulated application.

4 Programmer Interface

4.1 "Host" API

The Palm OS Emulator has a mechanism by which Palm OS applications running inside of it can call emulator-defined functions. For instance, it is possible to start and stop profiling from under application control.

A special header file (HostControl.h) defines the functions that emulated applications can call. These functions are invoked by executing a TRAP $F/selector combination defined for use by the emulator (as well as other foreign host environments, such as the Palm OS Simulator library). The Palm OS Emulator catches calls to the selector defined for it and then branches to the correct internally-implemented function.

Following is a description of the services available to emulated applications:

4.1.1 Types

```
// File operations create and use an opaque data structure
// defined by this type.

typedef struct HostFILE
{
         long  _field;
} HostFILE;

// Various other functions take and/or return values of these
// types. All parameters are actually 32-bits wide, but are
// given special typedef names so that it's clearer as to what
// the parameters are used for.

typedef  long  HostBool;
typedef  long  HostErr;
typedef  long  HostID;
typedef  long  HostPlatform;
typedef  long  HostBG;
```

4.1.2 Constants

```
enum    // HostErr values
{
        hostErrNone = 0,

        hostErrBase = hostErrorClass,

        hostErrUnknownGestaltSelector,

        hostErrDiskError,

        hostErrOutOfMemory,
        hostErrMemReadOutOfRange,
        hostErrMemWriteOutOfRange,
        hostErrMemInvalidPtr,

        hostErrInvalidParameter
};


enum    // HostID values
{
```

```
        hostIDPalmOS,
        hostIDPalmOSEmulator,  // The Copilot thingy
        hostIDPalmOSSimulator  // The Mac libraries you link with thingy
};


enum    // HostPlatform values
{
        hostPlatformPalmOS,
        hostPlatformWindows,
        hostPlatformMacintosh
};


enum    // HostBG values
{
        hostBGGray,
        hostBGWhite
};
```

4.1.3 Functions

```
long         HostGetHostVersion             (void);
HostID       HostGetHostID                  (void);
HostPlatform HostGetHostPlatform            (void);
HostBool     HostIsSelectorImplemented      (long);
HostErr      HostGestalt                    (long gestSel, long* response);

HostErr      HostProfileInit                (long maxCalls, long maxDepth);
HostErr      HostProfileStart               (void);
HostErr      HostProfileStop                (void);
HostErr      HostProfileDump                (const char* filename);
HostErr      HostProfileCleanup             (void);

long         HostErrNo                      (void);

long         HostFClose                     (HostFILE*);
long         HostFEOF                       (HostFILE*);
long         HostFError                     (HostFILE*);
long         HostFFlush                     (HostFILE*);
long         HostFGetC                      (HostFILE*);
long         HostFGetPos                    (HostFILE*, long*);
char*        HostFGetS                      (char*, long n, HostFILE*);
HostFILE*    HostFOpen                      (const char*, const char*);
long         HostFPrintF                    (HostFILE*, const char*, ...);
long         HostFPutC                      (long c, HostFILE*);
long         HostFPutS                      (const char*, HostFILE*);
long         HostFRead                      (void*, long, long, HostFILE*);
long         HostFSeek                      (HostFILE*, long offset, long origin);
long         HostFSetPos                    (HostFILE*, long);
long         HostFTell                      (HostFILE*);
long         HostFWrite                     (const void*, long, long, HostFILE*);
HostFILE*    HostTmpFile                    (void);

void*        HostMalloc                     (long size);
void*        HostRealloc                    (void*, long size);
void         HostFree                       (void*);

HostBool     HostGremlinIsRunning           (void);

HostErr      HostImportFile                 (const char* fileName, long cardNum);
HostErr      HostExportFile                 (const char* fileName, long cardNum,
                                             const char* dbName);

HostBool     HostSetDebugOption             (HostDebugOption, HostBool newSetting);
long         HostSetWindowScale             (long newScale);
HostBG       HostSetWindowBackground        (HostBG newBackground);
```

4.2 Debugger API

The Palm OS Emulator supports an API for external debuggers. This API (described in a separate document) allows external debuggers to set breakpoints, handle exceptions (such as address errors), single step, examine memory, etc.

5 Differences Between 2.0 and 2.1

5.1 Consistency Between Platforms

5.2 Basis on Session Files

5.3 "Autoload" Directory

5.4 Drag & Drop

5.5 DragonballEZ Emulation

5.6 More/Better Error Dialogs

5.7 Debug Options dialog

5.8 Scripting

5.9 "HostXXX" API

5.10 Networking, IR, and Serial emulation