

CIRCUIT CELLAR^{INK}

THE COMPUTER APPLICATIONS JOURNAL

Richard Ames

PalmPilot Application

Using Open Source Tools for Development

Did you know the PalmPilot is a programmable device? Probably not. Richard shows us how the same feature that lets us link to a desktop PC and download our daily schedule can be used to download new application programs.

Scientists are seemingly rare these days, but certain characteristics make identifying scientists easy, should one walk near your workbench. When travelling between the study habitat and the lab habitat, a scientist often uses a pacing gait, which isn't very efficient but does make it easier to think about scientific problems.

Also, the left shirt pocket is often puffed out by a small wire-bound notepad containing notes on what the scientist is thinking about and what will be considered next. When sharing a habitat with a scientist, the ordinary engineer's attention is often fixed on the small notebook, which surely contains a wonderful archive of great ideas.

That's what I thought, but I realized that I couldn't just run out to the drugstore, buy a notepad, and start filling it with insightful notes. After all, wire-bound shirt-pocket notepads are not programmable. Being an engineer, I need to be able to write code.

Well, it's the end of the twentieth century and little programmable notepads are available at the local office supply store. Of the

pocket-sized programmable devices, the 3Com PalmPilot is one that has a healthy developer community that shares tips, tools, and source code to ease the path to application development for the platform.

It may not be immediately apparent that the PalmPilot is a programmable device, perhaps because it is pitched primarily as an organizer and extension to your desktop computer. However, the same desktop PC-link that enables you to transfer the data in an address book or schedule can also be used to download new applica-

tion programs into the PalmPilot. From the PalmPilot's perspective, an application looks like another internal database.

You can also download an interpreter into the PalmPilot and develop applications entirely on the device—a native hosted environment. There are interpreters for BASIC, C, Forth, and Lisp.

With some of these systems, you can enter your program using the onboard memo writing application and execute the program with the interpreter application. This may bring back the thrill of hanging out in the back of the classroom covertly trying to set up your programmable calculator to display real-time graphics with an eight-digit seven-segment LED display.

But, the PalmPilot has a lot more than an LED display and there are more accommodating application-development environments than the native environment of the PalmPilot itself. The first PalmPilot applications were developed on Macintosh computers using a tool chain targeted to generate code for the PalmPilot.

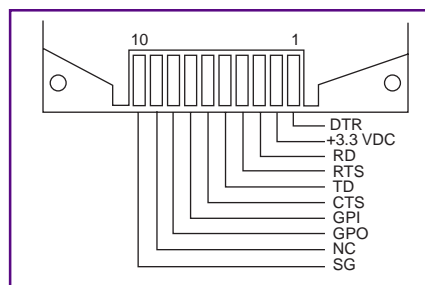


Figure 1—The connector on the back of the PalmPilot provides access to serial communication signals and general-purpose I/O.

The original cross-development environment (from Metrowerks) has been ported for use in Windows environments and has been bolstered by other cross-development solutions. One of them has an interesting price tag—it's free.

The GNU C compiler has been a long-standing focal point for free software development tools. Now there's a package for the GNU compiler that enables you to develop PalmPilot applications.

You don't need to stop at the compiler. It's possible to assemble a development environment for the PalmPilot or other targets entirely from free open-source software. This software goes from the OS that boots the system to the text editor, the target simulator, and the download utility.

I'll show you how to put your own code into the PalmPilot. And just for fun, let's do it all using free software.

BRINGING UP GNU/LINUX

Linux is a Unix-like OS developed by Linus Torvalds while he was a student at the University of Helsinki in Finland. The OS has grown from being a very ambitious personal project into something of a cult. When the Linux kernel is mated with the Unix-like tools produced by the GNU project and joined by additional application software, it creates an environment that rivals your average desktop box loaded with a commercial OS and applications. Some highlights of its directory structure are listed in Table 1.

The process of assembling, installing, and maintaining the components composing a GNU/Linux system can take a lot of effort. Luckily, the task was taken on by a number of organizations that produce Linux distributions. The first steps on the road to your own Linux system are to find a suitable distribution and select some hardware to run the software.

Which Linux distribution to use is yet another one of those areas of the computer world that inspires spirited discussion. Choosing a distribution affects the ease of installation, maintenance, and the ability to upgrade your system. There's no perfect answer. The references

Listing 1—This resource script provides the information for the main form for the Meter-Reader application shown in Photo 1.

```
FORM idReadForm 0 0 160 160 USABLE NO FRAME BEGIN
TITLE "MeterReader"
FIELD ID idLocField AT 20 20 AUTO AUTO NONEDITABLE
FIELD ID idReadField AT 20 50 AUTO AUTO EDITABLE
BUTTON "<prev" ID idPrevButton 40 120 AUTO AUTO
BUTTON "<next" ID idNextButton 60 120 AUTO AUTO
END
```

point you to more discussion on this topic, so I'll lead on through example by discussing the the Debian GNU/Linux distribution.

DEVELOPMENT HARDWARE

Your development system needs at least a '386-family processor, 200 MB of hard disk space, a VGA display, and 4 MB of memory. If you have an older machine, you may want to put it to work running Linux. The latest hardware always seems more satisfying though, and Linux has support for multiprocessor motherboards.

The system can be set up to boot multiple OSs or you can dedicate the system to running Linux. My system here runs only Linux. In a world of change, you can expect something in this installation process to change as well (check www.debian.org/debian/install.html).

BOOT DISKS

To start installing your Linux system, you need a boot disk, also called the rescue disk. The image for that disk is available from the Debian web site or one of its mirrors. The mirrors usually provide faster download times and ease the burden on the main server.

You also need five disks' worth of compressed base-system software and a driver disk. You should have a spare disk handy for writing an emergency boot floppy once the system is installed on the hard drive.

The disk images can be retrieved using a web browser or ftp program. A typical disk-image file has a path such as /debian/disks-i386/base14-1.bin. Not all of the installation diskettes are in MS-DOS format, so a special utility (rawrite2) is used to write the disk image files to floppy disk. This utility is available at the distribution site.

Here's an overview of the installation process with the scenario I installed on an '486-based system with a 212-MB hard disk and 16 MB of RAM. It can receive information from the outside world through a 1.44-MB floppy drive and a modem.

Once you've prepared your seven special Debian boot disks, put the rescue disk into the drive and reboot the machine. The initial screen should inform you that you are about to transform your machine into a Linux box and it gives you your first chance to turn back.

Assuming that you don't have any attachment to what is on the hard drive, pressing Enter boots a minimal Linux kernel, spits out more than a screen's worth of detailed information about your system hardware, and starts in on the process of configuring your system.

The configuration dialogs use a simple text screen-oriented user interface that lets you use the tab key to move between fields and Enter to move on to the next screen. After setting up your basic video and keyboard type, it's time to divvy up your hard drive for the new OS. This step can be done with the cfdisk program,

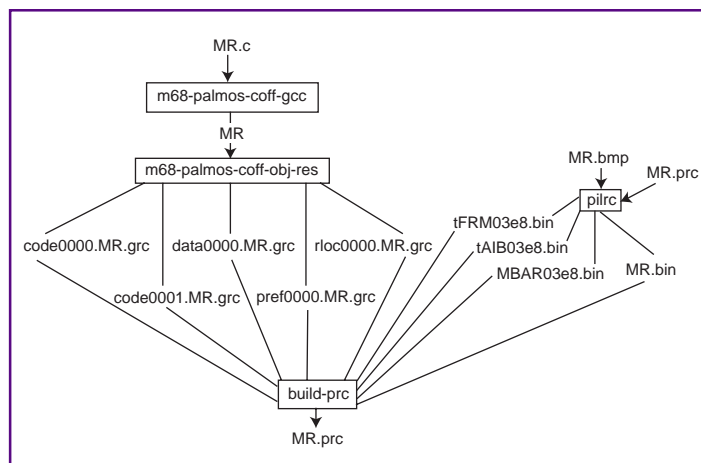


Figure 2—The PalmOS development tools are used to create the resource components that make up the application and then gather them into a prc file that can be loaded into the PalmPilot.

Listing 2—The boilerplate main loop for a PalmOS application doesn't leave much room for creativity. After the initial form is set up, execution continues in the event processing loop.

```
DWord PilotMain(Word cmd, Ptr cmdPBP, Word launchFlags)
{
  Int err;
  if (cmd == sysAppLaunchCmdNormalLaunch) {
    err = StartApplication();
    if (err)
      return (err);
    FrmGotoForm(MainForm);
    EventLoop();
    StopApplication();
  }
  return 0;
}
```

which should automatically start when you get to this point in the installation.

There's plenty of advice available on the best way to set up these partitions, depending on the size of the hard disk and system memory. In my example system, a typical configuration is an 80-MB root partition, a 100-MB user partition, and a 32-MB swap partition.

The root and user partitions should be set to type Linux ext, and the swap partition should be type Linux Swap. Select the Write option to write the new settings to the disk before leaving `cfdisk`.

The next step should be to initialize the swap, root, and user partitions. The root file system is then mounted and the OS kernel will be copied there. Once you have the OS on the hard disk, you are prompted to provide the driver's disk so that additional device driver support can be loaded.

If you have a minimal system, you don't need anything special from this disk. If you want to expedite the process by installing from a CD-ROM, you may need to load a CD-ROM driver.

The next step is configuring the network, which is only a matter of thinking up an appropriate name for your machine. The details of setting up your link to your ISP come later in the process.

Now it's time to feed the remaining disks into the machine, configure the clock, set up to boot into Linux, and make a backup-bootable disk. Once all of this is done, you can reboot the machine and load the kernel directly from the hard disk.

You'll be prompted to set up accounts for the superuser and yourself. Then, you can set up the PPP connection. Here you need IP addresses for name servers, user name and password, and the number to dial.

Now you have a base system on your hard drive, but it's not very useful. You're bound to miss the familiar environment of editors, compilers, GUIs, and the copy of Tetris. Don't worry, all of these will be coming because they've been set up in neat little packages.

These packages are special-purpose files that integrate all the information needed by an installation utility so the contents can be installed and configured and ready for immediate use. At this point, you have the opportunity to load whole sets of packages, minimal system requirements, or the full-fledged Swiss army knife setup.

If you want to pull your software in over the PPP connection, it makes sense to skip the options and individually select the packages you want. This option is offered next from within a utility called `dselect`.

With `dselect` you can specify PPP as your access method and select specific packages. The packages can also be loaded from a Debian CD-ROM. Some software packages depend on other packages, and `dselect` advises you in selecting a set of compatible packages.

You need the following packages for PalmPilot development: `xserver`, `man-db`, `pillrc`, `prc-tools`, and `xcopilot`. While browsing through the list of packages, you're bound to see others that you want to check out, so add them to the shopping cart, too.

Now it's time to go through the check-out line behind the Install Packages menu item. If all goes well, the PPP connection transfers the files and after several hours you receive a report that the software installation was successful. While you're waiting, track down that Debian CD-ROM. After `dselect` finishes, you can start exploring your new system.

One of the first things you may want to do is edit a file. In the Unix world there are two common text editors—`vi` and `emacs`.

`vi` is popular in part because it is the most likely editor to be found on a Unix system. If you find yourself wanting to edit a file on an unfamiliar Unix system, chances are, you can find `vi` and get the job done. `vi` also happens to be rather capable.

`emacs` is also common and quite programmable. Say you're in the middle of an edit session and want to review your e-mail without switching to a new window. Extensions enable `emacs` to display your e-mail from within the editor and do a number of other useful tasks. You may never even need to leave the text editor.

There are other PC-like text editors available such as `joe` and `xjed`. `joe` will be comfortable if you're happy with the default DOS screen editor. `xjed` is more of a programmer's editor that takes advantage of the X Windows environment.

For more information about migrating from DOS to a Unix-based system, see the sidebar on page 42.

INSIDE THE PALMPILOT

The PalmPilot is built around a Motorola 68328 DragonBall processor running at 16 MHz. There's also a 32-bit address space that provides up to 4 GB of memory.

The PalmOS and a set of built-in

Command	Description	Command	Description
<code>/bin</code>	Essential command binaries	<code>/tmp</code>	Temporary files
<code>/boot</code>	Files for the boot loader	<code>/usr/X11R6</code>	X Window system files
<code>/dev</code>	Device files	<code>/bin</code>	User commands
<code>/etc</code>	System configuration files	<code>/doc</code>	Details on installed packages
<code>/home/username</code>	Your home directory	<code>/include</code>	Standard C headers
<code>/lib</code>	Essential shared libraries	<code>/lib</code>	Libraries for programming
<code>/mnt</code>	Temporary file system mounting point	<code>/man</code>	Home of the man pages
<code>/sbin</code>	System binaries	<code>/var/lock</code>	Lock files for resources
		<code>/log</code>	System event log files

Table 1—Here are some highlights of the directory structure in a GNU/Linux system.

**Table 2—
Here is the
PalmPilot
family, including
discontinued mod-
els. Memory capacity
follows the usual gener-
ous curve.**

Product	PalmOS	ROM	RAM	Processor
Pilot 1000	1.0	512 KB	128 KB	68328
Pilot 5000	1.0	512 KB	512 KB	68328
PalmPilot Personal	2.0	1 MB	512 KB	68328
PalmPilot Professional	2.0	1 MB	1 MB	68328
Palm III	3.x	2 MB	2 MB	68328
Palm IIIx	3.1	2 MB	4 MB	68328EZ
Palm V	3.1	2 MB	2 MB	68328EZ

applications live on a 512-KB or 1-MB ROM on a plug-in memory card, which contains between 128 KB and 2 MB of pseudo-static RAM, depending on the model. The external data bus is 16 bits wide to reduce cost.

The system communicates to the outside world through its serial port, which has OS support for interrupt-driven output at 56 kbps. CTS and RTS signals are brought out to the contacts at the base of the device as shown in Figure 1, and there are eight hardware buttons on the system.

The system is designed to run for 40 h on two AAA alkaline cells. Because the device is likely to spend most of the time in sleep mode, the batteries are good for a few months of use. But, for constant active access to the PalmPilot or for a possible fixed application, there are suggested ways to patch a constant power source through the external connector.

Remember, the PalmPilot's purpose is to provide an electronic notepad. Thus the

backlit 160 × 160 pixel LCD is overlaid with a digitizer and capable of four levels of gray. PalmOS provides up to 50 points/s from the digitizer with 0.35-mm accuracy.

PalmOS APPLICATIONS

PalmOS applications have special features that make them different from the usual embedded system software. The software in a simple embedded device can initialize the system hardware and memory and then go into an endless loop, checking for conditions and calculating responses.

Even though the underlying OS kernel does preemptive multitasking, PalmOS does applications are single-threaded event-driven programs. The applications need to listen for special messages from the OS and play nicely with other applications.

The applications do have some characteristics that should be familiar to embedded-system software developers. The application must be frugal with memory because the application code and data

come directly out of the system memory (which can be as little as 128 KB in the Pilot 1000). This memory allocation is less of a concern with more recent members of the PalmOS family, thanks to 2 MB of system memory (see Table 2).

There are no disk drives in the system, so instead of using a file system for application programs and data storage, a database is used to store applications, their data, and their state information. So, applications open and close databases that are maintained in nonvolatile memory rather than disk-based files. The OS is specially designed to provide efficient access to the database records, even though they may be scattered through the system memory.

PalmOS also makes good on its privileged position between the hardware and your application by interpreting pen strokes on the writing surface so your application can be fed characters instead of tracking x-y coordinates. There's also a rudimentary beep from the PWM on the 68328, a timer with 10-ms resolution, a real-time clock that's always running (set to wake up an application at an arbitrary time), and a TCP/IP stack.

SAMPLE APPLICATION

Let's consider an application where the PalmPilot is used to help a meter reader collect and deliver readings from utility meters. The program presents a series of addresses to visit and the readings at each address are entered into the PalmPilot. At the end of the day, the information is downloaded into a database without having to be transcribed from a log sheet.

When envisioning this application, you're likely to think in terms of how the screen of the PalmPilot will look and how the user will interact with that screen. In implementing the application, the visual layout of the screen (or form, in Palm-speak) is defined by a resource script. This script specifies objects that appear in the form, such as buttons, fields, checkboxes, and other elements.

For the meter-reader application, you can imagine that

Migrating from DOS to Unix

DOS	Unix	Action
dir	ls	Show working directory contents
time	date	Show current time
ver	uname -a	Show OS information
type myfile.txt more	cat myfile.txt more	Display myfile.txt, pausing between pages
mkdir doc	mkdir doc	Create doc dir below working directory
cd doc	cd doc	Move to doc directory
del hello.c	rm hello.c	Delete the file hello.c
copy \home\sam\hello.c	cp /home/sam/hello.c	Copy a file to the working directory
dir /s myfile.txt	find -name hello.c -print	Search for hello.c in this dir and all subdirs
unzip -d hello.zip	tar -zxvf hello.tar.gz	Unpack a heirarchical archived set of files
help dir	man ls	Show online manual info

- Long filenames are available, and case is significant.
- DOS text files indicate the end of a line with the characters CR and LF. Unix text files use just LF.
- The default shell has a number of features. Old commands can be accessed with the up and down arrow keys. Information that has scrolled off the screen can be reviewed with Shift-Pg Up. File names can be completed by pressing Tab.
- To shutdown gracefully, change to superuser with the su command and enter shutdown -h now. Wait for the message "system halted" before switching off the power.

Listing 3—The event loop in a PalmOS application processes events as they are served up by the system, giving first dibs to default event handlers.

```
void EventLoop(void)
{
    Word error;
    EventType event;
    do {
        EvtGetEvent(&event, evtWaitForever);
        if (!SysHandleEvent(&event))
            if (!MenuHandleEvent(NULL, &event, &error))
                if (!ApplicationHandleEvent(&event))
                    FrmDispatchEvent(&event);
    }
    while (event.eType != appStopEvent);
}
```

the address to visit is prominently displayed and there's an area to write the meter reading at this address. The next address can be called up by pressing a forward arrow and previously visited addresses can be reviewed by pressing a back arrow. Listing 1 shows what this might look like as a resource script, and Photo 1 shows the result.

Executable code is linked with the objects in the form. For example, when the user presses the forward arrow button, the next address to visit is displayed.

STARTUP ROUTINE

In a traditional embedded system implemented in C, code execution starts at the location pointed to by the reset vector, runs some startup code, and then jumps to the `main()` function. For a PalmOS application, you can consider the system software to always be running, and the execution of the application starts in the `PilotMain()` function. Listing 2 shows the boilerplate body of a PalmOS application.

The first step of the application is to check the launch conditions. For simple applications, the only condition handled is a normal launch (when the user selects the application icon from the Application picker screen). The `cmd` parameter that is passed in the `PilotMain()` call is checked to see if the application was called under this condition. If it was, the application starts. Otherwise `PilotMain()` returns.

Some applications must retrieve state information when they are first started so they can present a screen that looks the same as it did when the application was last running. This is the purpose of `StartApplication()`, which is specific to the application. A simple application can return false from this function, indicating

that there were no problems in starting the application.

Execution continues by calling `EventLoop()`, which is the main event loop for the application. This loop processes a stream of events that are fed to the application by the OS.

Many events can be handled directly by the OS without any special processing by the application. But, the application is given dibs on these events as well, in case special processing is needed. The structure of the event loop is shown in Listing 3.

The first step is to retrieve the next event in the event queue with a call to `EvtGetEvent()`. Many events can be handled directly by PalmOS functions so `SysHandleEvent()` and `MenuHandleEvent()` are first given an opportunity to act on the event.

These functions return true if they were

able to completely handle the event. If the application sees these calls returned true, it doesn't do any more event processing. Opening menus and pressing buttons on the PalmPilot are examples of events that can be handled completely by these OS functions.

If execution continues to fall through beyond these functions that the OS handles, the current event may be one that should be handled by the application.

STOP ROUTINE

The stop routine shuts down the application gracefully and saves the information that will be needed the next time the application is activated.

The heavy lifting is in `ApplicationHandleEvent()`, which looks for events such as button presses and updates the screen in response to these events.

MEET THE TOOLS

Once the application is coded, it must be converted into a format that can be loaded into the PalmPilot. The compiler (`m68k-palmos-coff-gcc`) used in developing PalmOS applications is a patched version of the GNU C compiler. It is used to compile and link the application C source code (see Figure 2).

PalmOS expects the application source code to be divided into a number of resources, including ones dedicated to initializing application global memory and specifying application preferences. The `obj-res` utility in the code converter (`m68k-palmos-coff-obj-res`) converts the object code into the expected resources.

A PalmPilot application is implemented as a collection of resources, of which the application program code is just one. The other resources are prepared using the `pilrc` resource compiler. This utility takes a text description of the menus and other resources and converts them into binary resource files, which can be combined into a PalmPilot file.

Ultimately, the PalmPilot expects to receive a new application in a format known as `prc`. `build-prc` takes all the binary resources and builds them into a `prc` file.

`pilot-xfer` speaks the appropriate serial protocol to synchronize with the PalmPilot to update an application in the system memory. This utility is also used to backup and restore application data.



Photo 1—The CoPilot is a virtual PalmPilot on your computer screen that can be used to test applications without downloading to a physical device.

x-copilot is a virtual PalmPilot sitting on the screen of your development system. This simulator enables you to quickly load your latest code and exercise it without the having to download to the physical device.

This sample session shows the process by which these tools are used together to build an application. `M68k-palms-coff-gcc -O1 MeterReader.c -o MeterReader` compiles the main module and stores the output in the file `MeterReader`. The `-O1` specifies level 1 optimization.

`M68k-palms-coff-obj-res MeterReader` takes the file produced in the previous step and splits it into the set of resource components that are expected by PalmOS. `pilrc MeterReader.rcp` runs the resource script through the resource compiler to generate resources for the forms and menus.

`Build-prc MeterReader.prc "MeterReader" Mete *.grc *.bin` is the final step and combines the resources into a properly formatted `prc` file that can be loaded directly into the PalmPilot. The command line specifies an output file (`MeterReader.prc`), an application name ("`MeterReader`"), and a creator ID (`Mete`). All the files with suffixes of `grc` (code resources) and `bin` (forms and menus) are combined to create the `prc` file.

Now, the `prc` file can be loaded into the simulator or the PalmPilot, but there will be no list of addresses to visit. The list is stored in a separate database file—`MeterReader.pdb`. This file can be built from an ASCII text file using `makepdb`, a utility specific to this application.

Once the database is loaded, verify that you can scroll through the address list and fill in meter readings. After the readings are filled in, a sync operation with the desktop system can move the database to the desktop, and the readings can be extracted.

Normally, this process is integrated into the PalmPilot desktop application using special software known as conduits. But, this support doesn't exist for Linux yet.

NEXT STOP

If you program with free tools, consider sharing your discoveries with the user community that put together the framework. We all benefit from well-considered

postings, improved tools, beefed-up documentation, and shared source code. [EPC](#)

Richard Ames fulfills his need to write code at Oresis Communications, preferably in the lab habitat. You may reach him at richard_c_ames@yahoo.com.

SOFTWARE

The complete meter-reader application is available via the Circuit Cellar web site.

REFERENCES

3Com PalmOS reference, tutorials, and articles, www.palm.com/devzone/info.html, Software Developer Documentation
 Debian GNU/Linux homepage and distribution, www.debian.org
 DragonBall databooks, www.mot.com/SPS/WIRELESS/products/m68328.html
 GNU project, www.gnu.org
 Linux distributions summary, www.linuxresources.com/apps/ftp.html
 Linux documentation project, www.metalab.unc.edu/LDP
 PalmOS software and source code examples, www.palmcentral.com
 PalmPilot resource compiler, www.scumby.com/scumbysoft/pilot/pilrc
 PDA software development site, www.roadcoders.com
 Pilot application development tutorial, www.iosphere.net/~howlett/pilot/GNU_pilot_SDK_Tutorial.zip
 Pilot-to-GNU communications tutorial, www.iosphere.net/~howlett/pilot/GNU_Pilot_SDK_Tutorial.zip
 Pilot programming newsgroups, news://news.massena.com/pilot.programmer.gcc, and www.acm.rpi.edu/~albert/pilot
 X-Copilot, xcopilot.cuspy.com

SOURCE

PalmPilot
 3Com
 (800) 638-3266
 (408) 326-5000
 Fax: (408) 326-5001
www.palm.com