GNU PILOT SDK TUTORIAL

The GCC Compiler belongs to the Free Software Foundation. Check out their web site to learn more about free software.

This tutorial records what I figured out as I taught myself to program my pilot using GCC and PilRC. Maybe it can help other recreational programmers. Many thanks to the experts who built free tools for Pilot Programming. If you find errors in this tutorial or would like to contribute improvements, please let me know. For document control purposes, please do not distribute modified copies of the tutorial.

The tutorial package should include the following files:

`GNU_Pilot_SDK_Tutorial.doc`	Original MS Word 6.0/95 version of the tutorial
`GNU_Pilot_SDK_Tutorial.html, eventloop.jpg`	HTML version for on-line reading if you don't have MS Word
`GNU_Pilot_SDK_Tutorial.ps`	Postscript version for printing if you don't have MS Word. Set up for double side, letter paper.
`GNU_Pilot_SDK_Links.html`	Links to all the web sites mentioned in this tutorial, links to all my favourite development sources.
`tex2hex.rcp, tex2hex.c, hex2hex.h`	source code for example
`tex2hex.bmp, text.bmp, hex.bmp`	bitmaps for example
`makefile (or compile.bat)`	makefile for example

Table of Contents

1.0 INTRODUCTION

1.1 Aim

This tutorial will explain how to build simple Pilot applications using GCC for Pilot and PilRC. This tutorial defines a simple application as an application with one form, one or two alert dialogs, one menu, and any number of buttons, fields, bitmaps, and labels. It supplements the documentation listed in section 1.5.

Later editions of this tutorial may explain how to build more sophisticated Pilot applications. More sophisticated applications use tables, resource databases, data databases, popup lists, checkboxes, multiple forms, and other techniques.

This tutorial is written for newbies. Readers should be familiar with ANSI C.

1.2 But I Don't Know ANSI C

If you don't know ANSI C but still want to program your Pilot then you have three options:

1. Learn ANSI C. Check the web, especially
   http://www.yahoo.com/Computers_and_Internet/Programming_Languages/C_and_C__/Courses/
   Also, check used book stores. I picked up an old copy of Mastering Borland C++ version 3.1 for $5.

2. Use PILA, Darrin Massena's Pilot Assembler. Everything you need is at Darrin's web page.

3. Use JUMP, the Pilot Java Compiler.

1.3 I Know ANSI C

Be careful! Some of the standard C library functions don't work with PalmOS. For instance, PalmOS has its own string functions – StrItoA replaces stritoa and WinDrawChars must be used in place of printf. Unfortunately, I can't give you a complete list, so check the PalmOS API before you use the standard stuff.

1.4 Limitations

This manual is limited to PalmOS 1.0, because today (20 June 1997) PalmOS 2.0 documentation has not been made public. However, with only two exceptions software developed with GCC for PalmOS 1.0 will work under PalmOS 2.0. The exceptions will be identified in the following chapters. This tutorial was developed using Pilot GCC version 0.4.0 and PilRC version 1.3 on Windows 95.

1.5 Related Documentation

You should download and read all the documentation listed in this section. You will require Adobe Acrobat to read any files with a filename suffix of "pdf". Acrobat Reader can be downloaded from Adobe's website, www.adobe.com.

Developing Palm OS Applications. This document is available from Palm Computing's developer resources webpage, http://www.usr.com/palm/dresources.html. The document is in two files, guide1.pdf and guide2.pdf. Guide1 contains "Part 1: System and User Interface Management". Guide2 contains "Part 2: Memory and Communications Management." These two files describe the PalmOS application programmer interface (API) in detail. When this tutorial mentions an API call you should look it up in these guides and read the details.

Palm OS Tutorial. This tutorial describes how to build a memopad application. Many of the techniques in the Palm OS Tutorial are useful with the GNU Pilot SDK, but there are two big problems with using this with the GNU Pilot SDK. First, it was written for the Macintosh. Which means that resources aren't built the same way and some of the C code is different. Second, the beginning stages don't include a command test (you'll learn about this in Chapter 3). Which means the application will crash when you try to hotsync it to your Pilot. It's available from the developer resources web page as file tutorial.pdf.

PilRC User Manual. The Pilot Resource Compiler (PilRC) user manual comes with PilRC in HTML. You will need the PilRC user manual.

GNU documentation. Documentation for GCC, MAKE, and GDB (HTML format) can be downloaded from my website.

1.6 Copyrights and Trademarks

Some information is drawn from PilRC documentation, Palm Computing documentation, and Palm Computing header files. That information will be identified when presented and is copyright of the contributor. Palm Computing and Palm OS are registered trademarks of US Robotics. Motorola and DragonBall are registered trademarks of Motorola. Adobe and Acrobat are registered trademarks of Adobe.

1.7 Required Software

Figure 1.7 shows the minimum software required to complete this tutorial. Links to the software are provided in the accompanying file “GNU_Pilot_SDK_Link.html”.

Common	Unix	Win95
guide1.pdf	prc-tools.0.4.2.tar.gz binutils-2.7.tar.gz gcc-2.7.2.2.tar.gz	Pilot GCC Win32 0.4.0
guide2.pdf	PilRC Unix	PilRC 1.3
Adobe Acrobat	xcopilot	copilot beta 9
copilot.rom	See Annex B	See Annex A

Figure 1.7: Software Requirements

Make sure that you have the right version of PilRC. Sometimes GCC or prc tools come with an old version of PilRC. Versions before 1.3 will report errors when you try to compile the example in Chapter 6.

2.0 PILOT DATABASES

This tutorial will start by explaining Pilot databases. To new Pilot programmers, this may seem a little strange. Most C programming books start by talking about the user interface and coding, then explain how to build databases. But Pilot is a little bit different than your desktop PC. A little insight regarding how PalmOS organizes its applications and data will help you when you start building applications.

2.1 Dynamic and Static RAM

The Motorola DragonBall CPU can address up to 4GB of memory. No Pilot is currently equipped with 4GB, so most of this address space is unused. Darrin Massena figured out Figure 2.1 when he hacked the Pilot 1000.

Start Address	End Address	Comments
00000000	000003ff	68k vectors, including traps starting at $80 for instance trap #15 at $bc point to $10c03656
00000000	00007fff	Dynamic RAM. The 68k vectors are in Dynamic RAM.
00008000	0fffffff	Faults on access attempt
10000000	10007fff	Mirror of the 00000000-00007fff range above (RO w/o permission)
10008000	1001ffff	Storage RAM on 128k machine (RO w/o permission)
10020000	1003ffff	Out of bounds but doesn't cause a fault
10040000	10bfffff	Faults on access attempt
10c00000	10c7ffff	512K ROM!
fffff000	fffffb12	DragonBall registers

Figure 2.1: Pilot 1000 Memory Map

Your Pilot's RAM is divided into two areas: dynamic RAM and storage RAM. Dynamic RAM is the memory available to an application as it runs, similar to the RAM installed in your desktop computer. Dynamic RAM is used for the current application's local variables, stack space, etc. My Pilot 1000 uses 32k for dynamic RAM and my PalmPro uses 64k. Storage RAM is the RAM used to store applications, their global variables, and their data files, similar to the hard drive in your desktop computer. This analogy is handy (thank the brainy guys and gals at Palm Computing, it's their analogy from page 15 of guide2.pdf), but remember that the comparison isn't identical. For instance, when you launch a desktop application, the code is transferred from hard disk to dynamic RAM but when you launch a pilot application, Palm OS accesses the code directly from storage RAM. Palm OS uses a database style file system within storage RAM. This means that everything in storage RAM is formatted as a Pilot database, including your application. Each database has a header where header attributes identify the characteristics of the database. Some important database header fields are "type", "creatorID", and "attributes".

2.2 Type and CreatorID

PalmOS identifies different databases by their unique combinations of "type" and "creatorID". Type and creatorID are 32 bit values. In C they can be stored as unsigned long, in assembler as long. Usually they are represented by four ASCII characters, but some applications use hexadecimal or decimal representation for #DEFINE instructions. All Pilot applications are database of type 'appl' (1634758764 or 0x6170706c). If you create an application, but give it a type other than 'appl', then it won't show up when you tap the application button. At this point you should load fileman.prc into your Pilot and inspect the type and creatorID values for the databases in your Pilot. Verify for yourself that all the applications are type 'appl'.

Some other types are commonly used. The built-in applications use ‘DATA’ type for their data databases, such as your addresses and appointments. The ‘HACK’ type identifies hackmaster extensions. A good rule of thumb is to use ‘Rsrc’ type to identify resource databases and ‘Data’ type to identify data databases. Of course, type and creatorID are case sensitive because ‘A’ is ASCII 0x40 whereas ‘a’ is ASCII 0x60. All lower case creatorIDs are reserved for use by Palm Computing, so your creatorID must have at least one upper case letter.

Because every application must have a unique type/creatorID combination, and since all applications have type ‘appl’, your application’s creatorID must be different from all other creatorIDs on the Pilot. If you load an application into your Pilot with the same creatorID as an application that is already in static memory, then the previous application will be erased. If you load an application into your Pilot with the same creatorID as an application in ROM, then the ROM application will no longer be available. Usually, you will want to find a creatorID that no one is using yet. Palm Computing makes this task simple by posting a list of registered creatorIDs at their technical support webpage. Anyone can register a creatorID, providing someone else isn’t already using it. If you’re just fooling around, you don’t need to register your creatorID, but if you plan to distribute your application to other Pilot owners, you should register the creatorID with Palm Computing.

2.3 Resource vs Data Databases

Resource databases store resources: bitmaps, code, version, UI objects, strings, etc. A resource database of type “appl” is a Pilot application. Resource databases are distinguished from “data databases” by the dmHdrAttrResDB bit in the attributes field of the database header. Not all resource databases are type ‘appl’. For instance, hackmaster extensions are resource databases of type “Hack”. Palm Computing never gave a specific name to databases that aren’t resource databases - in this tutorial they will be called “data databases”. Annex C, contributed by Theodore Ts’o, shows the structure of a PRC file (which is a type of resource database).

3.0 PILOTMAIN

3.1 Introduction

PilotMain is the entry point for a Pilot application. Figure 3.1.1 shows the prototype for the PilotMain procedure. PilotMain returns zero if no errors occur, otherwise it returns the error code. There are four parts to PilotMain, in order of appearance in code:

- Command test
- Start application
- Event loop
- Stop application

Figure 3.1.2 shows a generic PilotMain procedure. The start application, event loop, and stop application code is modularized in separate procedures. This chapter will explain the command test and each procedure.

```
DWord PilotMain(Word cmd, Ptr cmdPBP, Word launchFlags)
```

Figure 3.1.1: Prototype for PilotMain.

```
DWord  PilotMain (Word cmd, Ptr cmdPBP, Word launchFlags)
{
  Word error;

  if (cmd == sysAppLaunchCmdNormalLaunch)  // Command test, could also use “if (!cmd)”
  {

    error = StartApplication();           // Application start code
    if (error) return error;

    EventLoop();                          // Event loop

    StopApplication ();                   // Application stop code
  }
  return 0;
}
```

Figure 3.1.2: PilotMain procedure

3.2 Command Test

When PalmOS calls your PilotMain, PalmOS will pass a command using the cmd argument. Figure 3.2.1 shows possible commands.

Your PilotMain procedure must check which command PalmOS issued to your application. If your application responds to the command, then PilotMain should do whatever you intend, otherwise PilotMain must return. So the first line of code in your PilotMain should be an if statement. For now, our Pilot applications will only respond to command sysAppLaunchCmdNormalLaunch.

As you can see in Figure 3.2.1, sysAppLaunchCmdNormalLaunch is defined as 0. This allows many programmers to use the test expression (!cmd) as a shortcut to test for a normal launch. I recommend you use the full expression. I’ve seen one post on pilot.programmer.gcc from someone who may have been confused by the (!cmd) notation.

Only declarations and the final return statement should be outside of the command test. GNU Pilot SDK users have reported strange behaviour, such as Pilot hanging during hotsync, when even innocent looking code is added before the command test.

```
//  Copyright(c) 1994, Palm Computing Inc., All Rights Reserved
//  FileName: SystemMgr.h
//  Created by Ron Marianetti

// SysAppLaunch Commands

#define sysAppLaunchCmdNormalLaunch          0   //  Normal Launch
#define sysAppLaunchCmdFind                  1   //  Find string
```

```
#define sysAppLaunchCmdGoTo                 2   //  Launch and go to a particular record
#define sysAppLaunchCmdSyncNotify           3   //  Sent to apps whose databases changed
                                                //  during HotSync after the sync has been
                                                //  completed
#define sysAppLaunchCmdTimeChange           4   //  The system time has changed
#define sysAppLaunchCmdSystemReset          5   //  Sent after System hard resets
#define sysAppLaunchCmdAlarmTriggered       6   //  Schedule next alarm
#define sysAppLaunchCmdDisplayAlarm         7   //  Display given alarm dialog
#define sysAppLaunchCmdCountryChange        8   //  The country has changed
#define sysAppLaunchCmdSyncRequest          9   //  The "HotSync" button was pressed
#define sysAppLaunchCmdSaveData            10   //  Sent to running app before
                                                //  sysAppLaunchCmdFind
                                                //  or other action codes that will cause
                                                //  data searches or manipulation.
#define sysAppLaunchCmdInitDatabase        11   //  Initialize a database; sent by
                                                //  DesktopLink server to the app whose
                                                //  creator ID matches that of the database
                                                //  created in response to the "create db"
                                                //  request.
#define sysAppLaunchCmdSyncCallApplication 12   //  Used by DesktopLink Server command
                                                //  "call application"
```

Figure 3.2.1: System command codes

3.3 Start Application

The application start code typically has three functions: open the application's data database (if any), load system preferences (if any), and set the first form.

```
static int StartApplication(void)
{
  int error;

  error = OpenDatabase();
  if (error) return error;

  FrmGotoForm(formID_StockMarket);
}
```

Figure 3.3.1: StartApplication procedure

If the application uses a data database, then the application should open it before doing anything else. I use a boolean procedure called OpenDatabase to open my database and create the database if no database exists. Figure 3.3.2 shows my OpenDatabase procedure.

```
static Boolean OpenDatabase(void)
{
  UInt          index = 0;
  VoidHand      RecHandle;
  Ptr           RecPointer;

  StockMarketDB = DmOpenDatabaseByTypeCreator(StockMarketDBType, StockMarketAppID,
                                              dmModeReadWrite);

  //  if StockMarketDB doesn't exist, create it
  //  and create a zero record with new game state information

  if (!StockMarketDB) {
    if (DmCreateDatabase(0, StockMarketDBName, StockMarketAppID, StockMarketDBType, false))
      return 1;
    StockMarketDB = DmOpenDatabaseByTypeCreator(StockMarketDBType, StockMarketAppID,
                                                dmModeReadWrite);
    RecHandle = DmNewRecord(StockMarketDB, &index, 42);
    RecPointer = MemHandleLock(RecHandle);
    DmWrite(RecPointer, 0, &SM_Values, 42);
    MemPtrUnlock(RecPointer);
    DmReleaseRecord(StockMarketDB, index, true);
  }

  //  load game state information

  RecHandle = DmQueryRecord(StockMarketDB, index);
  RecPointer = MemHandleLock(RecHandle);
  MemMove(&SM_Values, RecPointer, 42);
  MemPtrUnlock(RecPointer);
```

```
  return 0;
}
```
Figure 3.3.2: OpenDatabase procedure from “Stock Market”

Another thing you may want to do in your OpenDatabase procedure, which isn’t shown in Figure 3.3.2, is check the version of the database. When you hotsync a new version of an application to your Pilot, the previous version of the application is erased. But although hotsync erases the previous “appl” type, the previous “Data” type data database is still there. In many cases using the old version of the data database with the new version of the application causes errors. This is why new versions of Pilot software sometimes come with a warning to delete previous version before installing. When you delete an application, every database with that creatorID is deleted, even if the database has a different type. More sophisticated applications will check the version of their data database when they run. If the data database is an old version then the a sophisticated application will either delete the old one and create a new one, or upgrade the old data database to the new format.

System preferences are typically used for restoring the state of the application to whatever it was when the user last closed the application. For instance, in a chess game the board configuration might be stored in system preferences. In the memopad application system preferences records which category was last open and which memo was open, if a memo was open. There’s no defined limit on the sized of your system preferences record - in one of my applications I stored the entire LCD display memory, about 3k, in my syspref record. However, I don’t use the system preferences API calls anymore. When Palm Computing released Palm OS 2.0, they changed something in the system preferences API. After the change, code written using the GNU Pilot SDK stopped working, even though it conformed to the published API. So if you plan on using GCC, don’t use system preferences. Instead of system preferences, create a data database. It requires a bit more code, but the database is much more versatile. I now use record zero of my data database in the same way that I used to use the system preferences API. I usually load the state information in my OpenDatabase procedure, as can be seen in Figure 3.3.2.

After opening and checking the database, the next thing to do is set the initial form. Usually programmers use FrmGotoForm to define the initial form, but there are some possible optimizations for applications that use only one form.

3.4 The Event Loop

Pilot applications are event driven. This means that the user, the operating system, and even the application itself generate operating system events which are placed in the operating system’s event queue. Figure 3.4.1 lists Palm OS events. One at a time, the application takes events from the event queue and handles them, in first-in-first-out order. The application programmer decides how the application handles these events. So the purpose of the code in a Pilot application is to get events from the queue, pick out the events that you are interested in, handle the events that you are interested in and pass the others to the operation system’s default event handlers.

```
//  Copyright (c) Palm Computing 1994
//  All Rights Reserved
//  FILE:     Event.h
//  AUTHOR:   Art Lamb: September 26, 1994

// Types of events

enum events {
        nilEvent = 0,
        penDownEvent,
        penUpEvent,
        penMoveEvent,
        keyDownEvent,
        winEnterEvent,
        winExitEvent,
        ctlEnterEvent,
        ctlExitEvent,
        ctlSelectEvent,
        ctlRepeatEvent,
        lstEnterEvent,
        lstSelectEvent,
        lstExitEvent,
        popSelectEvent,
        fldEnterEvent,
        fldHeightChangedEvent,
        fldChangedEvent,
        tblEnterEvent,
        tblSelectEvent,
        daySelectEvent,
        menuEvent,
```

```
        appStopEvent,
        frmLoadEvent,
        frmOpenEvent,
        frmGotoEvent,
        frmUpdateEvent,
        frmSaveEvent,
        frmCloseEvent,
        tblExitEvent,
        firstUserEvent = 32767
}
```

Figure 3.4.1: Palm OS Events

Figure 3.4.2 shows program flow for the event loop. As you can see, it's very simple. The application gets an event from the event queues using EvtGetEvent and handles the event. If the event is an AppStopEvent then execution exits the event loop, otherwise execution loops back to EvtGetEvent.

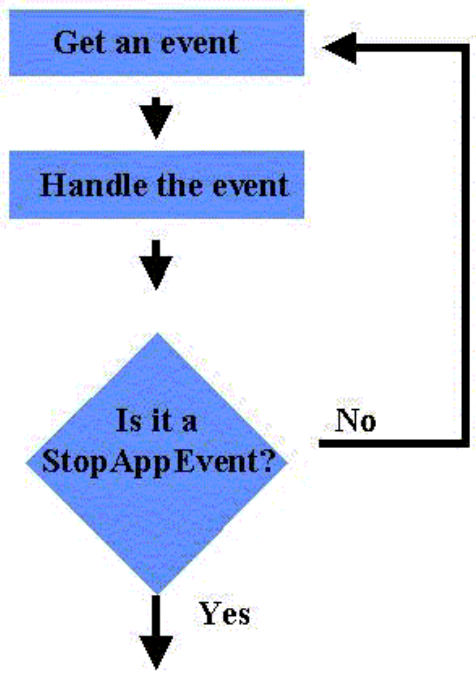


Figure 3.4.2: The Event Loop

The programmer has some choices when getting an event. You could choose to wait forever for an event. Or the programmer can define a time limit, in units of .01 seconds. This is handy if you want something to happen at specific intervals. For instance,

EvtGetEvent(&event, 200); // wait 2 seconds for an event. After 2 seconds, NilEvent is returned

EvtGetEvent(&event, -1); // wait forever for an event

EvtGetEvent(&event, WaitForever); // wait forever for an event. WaitForever is defined in Event.h

In the first example, if no events happen within 2 seconds then EvtGetEvent will return with a nilEvent in the event structure. Setting the time-out value can be used as a simple method of timing for some applications. However this technique doesn't work for games/applications that use a lot of user input, because the timer gets reset every time an event occurs. For instance, it wouldn't work in Space Invaders, because every time the player pressed a button to move or fire, EvtGetEvent would exit with a keyDownEvent and the 2 second limit might never be reached. Setting a time-out on EvtGetEvent is good for timing simple games, but better timing methods must exist. You can read more about the event queues on pages 152 to 154 of guide1.pdf.

The event handler part of the loop is a little more complicated. Before your application gets a chance to handle the event, various parts of the operating system should be offered the event. If the operating system handles the event, then you shouldn't need to. For instance, if a penDownEvent occurs within the bounds of a button, you probably don't want to interfere. SysHandleEvent will handle it. For example, a penUpEvent will occur after the penDownEvent. If the penUpEvent is also within the button, then SysHandleEvent generates a ctlEnterEvent, which indicates that the button was pushed. But you typically don't want to know when a button was pushed – you want to know when it was released. So SysHandleEvent waits for a penUpEvent. If the penUpEvent occurs within the button, and the preceding software event was a ctlEnterEvent for that button, then SysHandleEvent generates a ctlSelectEvent. So ctlSelectEvent is the event we want to handle for buttons. To understand that example you really should know the difference between software and hardware events, which means you have to read pages 152 to 154 of guide1.pdf. But my point is, if you don't call SysHandleEvent or if you carelessly interfere with the processing of hardware events you can screw up buttons and many other UI objects.

My EventLoop doesn't determine exactly what the event is, it just passes the event to other handlers in a certain order. Of course, there is an exception - frmOpenEvent I handle inside EventLoop. Here's why. Most of my applications have multiple forms. Switching from one form to another (and setting the initial form) generates a FormOpenEvent. Each form

has an event handler. When the form changes, you must change the form event handler. This is done using the API call FrmSetEventHandler. So in my EventLoop I have a FormOpenEvent handler which detects the FormOpenEvent and sets the correct form event handler.

Some Pilot programmers use a ApplicationEventHandler procedure to handle frmOpenEvent and appStopEvent. My thinking is that frmOpenEvent isn't an app event, so I wouldn't handle it with an app event handler. And I don't need a handler for appStopEvent - when appStopEvent occurs the event loop breaks and the StopApplication code gets executed. I don't use an ApplicationEventHandler, but many other programmers do and there are examples at my web page.

After all these other handlers get their chance, then your application has an opportunity to handle the event in your customized way. The event gets passed to FrmHandleEvent. FrmHandleEvent will do some stuff with the event (I don't know exactly what) then send the event to the form event handler which you defined using FrmSetEventHandler. Figure 3.4.3 shows a GCC implementation of an event loop for one of my applications.

```
static void EventLoop(void)
{
  do
  {

    EvtGetEvent(&event, 200);

    if (SysHandleEvent(&event)) continue;
    if (MenuHandleEvent((void *)0, &event, &err)) continue;

    if (event.eType == frmLoadEvent)
    {
      formID = event.data.frmLoad.formID;
      form=FrmInitForm(formID);
      FrmSetActiveForm(form);
      switch (formID)
      {
      case formID_Buy:
        FrmSetEventHandler(form, BuyEventHandler);
        break;

      case formID_Sell:
        FrmSetEventHandler(form, SellEventHandler);
        break;

      case formID_StockMarket:
        FrmSetEventHandler(form, StockMarketEventHandler);
        break;

      case formID_About:
        FrmSetEventHandler(form, AboutEventHandler);
        break;

      case formID_Help:
        FrmSetEventHandler(form, HelpEventHandler);
        break;
      }

      FrmDispatchEvent(&event);
    } while(event.eType != appStopEvent)
  }
}
```

Figure 3.4.3: PilotMain Event Loop

3.5 Application Stop Code

Application stop code should save the application's state and close any databases that your application may have opened. Both of these tasks are optional. Your application may not need to save any state information. The M+ key in the built-in calculator is a very simple example of state information. If someone stores a number in the calculator's memory register, it should still be there next time they launch the application. So when the application ends, Calculator has to store it's memory register. It probably uses SysPrefs to do it, but as I mentioned earlier your applications should use a data database. The second function of application stop code is to shut down databases. If your application shuts down and leaves databases open, PalmOS will close them automatically. However I prefer to explicitly close any databases I may have opened. I usually have use a separate CloseDatabase procedure. You may choose to reduce the final PRC size by omitting this call.

3.6 Optimizations and Customizations

Before closing this chapter, you should understand that there are many variations on PilotMain. As I mentioned, some programmers write an ApplicationEventHandler. Some programmers write a MenuEventHandler (MenuEventHandler is always recommended for large apps) and call it directly from their event loop procedure. You should write some code and collect samples from different authors to find out what works best for you.

4.0 FORMS

4.1 Introduction

Humans interact with the Pilot's graphical user interface via user interface objects. PalmOS 1.0 supports the following UI objects: bitmap, button, checkbox, field, gadget, graffiti shift state indicator, help string (tips), label, list, menu bar, popup trigger, push button, repeating button, selector trigger, table, title. PalmOS organizes UI objects using "forms". A form is a set of UI objects grouped together to achieve some common purpose. For instance, if you press the MemoPad hardware button a form appears on the screen. The form contains a title object "Memo List" in the upper left corner, a popup trigger in the upper right corner (categories), a button in the lower left corner "New", and a two column, eleven row table in the middle of the screen. These UI objects achieve the common objective of listing memos of the category selected with the popup trigger. UI objects are not resources, but they might be associated with a resource, which provides the UI object with data (see FORMBITMAP).

"Windows" are related to forms, but a little different. A window is just a section of the screen. For instance, all forms include a window where the UI objects are displayed. But not all windows are forms. For instance, the screen space occupied by a pull down menu is a window. The screen space occupied by an alert box is a window. PalmOS also permits off-screen windows. Just imagine an invisible section of screen on the back of your Pilot that follows all the same rules as normal windows. If you want to draw on the LCD screen inside your window, use the WinDraw API calls, such as WinDrawChars, WinDrawBitmap, etc. However stuff you draw using WinDraw calls isn't part of the form, so it will be erased by a FrmDrawForm call.

A menu might be associated with the form using the MENUID identifier. The form can be specified as modal or non-modal. "Modal forms ignore pen events outside their boundaries" (guide1.pdf, page 74). Helpstrings are only usable with modal forms. A modal form displays the information icon on the right side of the title bar. If a helpstring is defined, then the helpstring text is displayed in an information alert box when the user taps the title bar information icon. You can define a default button for the form. If the form quits, then the system will add a ctlSelectEvent to the Event queue with buttonID equal to the button ID set by DEFAULTBTNID. For instance, suppose your form has two buttons, "do it" and "cancel". This form is displayed when the application stops due to some other user input, for instance another application was launched from the Applications screen. You may want the "cancel" button to execute automatically to release locked handles, reset global variables, etc. This is a situation where you could use DEFAULTBTNID.

To build forms and other resources, Wes Cherry has provided the Pilot Resource Compiler, "PilRC". Defining a form requires multiple PilRC instructions because every option and UI object must be specified. PilRC uses the FORM identifier to identify the start of a form definition, followed by the form options, then the UI objects are contained between BEGIN and END identifiers. I indent the UI object definitions by four spaces, but this is not required. Order of the UI objects is not important. This chapter will discuss the most commonly used UI objects:

- title
- label
- button
- field
- formbitmap
- graffitistateindicator

PilRC prototypes start with an identifier, followed by a number of fields. Optional fields are enclosed by [ ], required fields are enclosed by < >. The field suffix identifies the data type. Figure 4.1 is taken from PilRC's user guide.

Suffix	Data Type	Description
.s	string	May contain the following character escapes: \t (tab) \n (cr) \nnn (where nnn is an octal character code)
.ss	multi-line string	PilRC will concatenate strings on separate lines enclosed with quotes and terminated by the \ character. Example: "Now is the time for all good "\

		“men to come to the aid of their ”\ “country”
.n	number	Defined constant or simple arithmetic expression. Valid operators are + - * /. Precedence is left to right. Math is integer.
.p	position coordinate	Which may be either a number, expression or one of the following keywords: AUTO Automatic width or height. The width/height of the item is computed based on the text in the item. Valid only for widths or heights of items. CENTER Centers the item either horizontally or vertically. Only valid for left or top coordinate of an item. PREVLEFT Previous item’s left coordinate PREVRIGHT Previous item’s right coordinate PREVWIDTH Previous item’s width PREVTOP Previous item’s top coordinate PREVBOTTOM Previous item’s bottom coordinate PREVHEIGHT : Previous item’s height

Figure 4.1: PilRC field data types

4.2 Title

```
TITLE  <Title.s>
```
Figure 4.2.1: Form Title Object Prototype

Title gives your form a title. The title appears at the top of the form as white characters on black background. The characters are always font 1. (The fonts are listed in figure 4.2.2. Download the ASCIIChart tool to see the different fonts on your Pilot.) For modal forms the title is centered. In a non-modal form, the title bar uses the top fifteen pixel rows of the form. In a non-modal form the title bar uses only fourteen pixel rows.

FONT ID	Description	Height	Width	Comments
0	Standard	11	variable	extended ASCII character set
1	Bold	11	variable	extended ASCII character set
2	Large	14	variable	extended ASCII character set
3	Symbol	10	variable	alarm clock, up and down arrows, memo, bullet, etc.
4	Symbol11	11	variable	AKA checkbox font
5	Symbol7	8	11	up and down triangles
6	LED	19	variable	calculator numbers

Figure 4.2.2: Pilot fonts

4.3 Label

```
LABEL  <Label.s> ID <Id.n> AT (<Left.p> <Top.p>) [USABLE] [NONUSABLE] [FONT <FontId.n>]
```
Figure 4.3: Form Label Object Prototype

The label object allows you to fix text strings on the form. The font and location is user selectable. Usable and nonusable supposedly allows the user to determine whether or not the Label is drawn. I suggest you set all labels USABLE. If you want text to appear and disappear use the Field UI object or the WinDrawChars API call.

4.4 Button

```
BUTTON  <Label.s> ID <Id.n> AT (<Left.p> <Top.p> <Width.p> <Height.p>) [USABLE] [NONUSABLE]
           [DISABLED] [LEFTANCHOR] [RIGHTANCHOR] [FRAME] [NOFRAME] [BOLDFRAME] [FONT <FontId.n>]
```
Figure 4.4: Form Button Object Prototype

A button is exactly what it sounds like: an area on screen, usually framed, which activates some process when touched. You have the option of labeling the button, you decide where the button is located, you decide whether it’s usable or non usable. For instance, if your application is a list of words, and you flip from word to word with “Previous” and “Next” buttons, then

you may want to make the “Previous” button unusable on the first word and the “Next” button unusable on the last word. Initially you can set the object usable using it’s PilRC definition. During runtime, you can set it usable and nonusable using the FrmSetObjectUsable and FrmSetObjectNonusable calls. Setting a button non-usable removes it from the screen. Buttons cannot be “grayed out” as done in some other operating systems. However, some programmers cheat by making a bitmap that looks like a “grayed out” button, and then calling WinDrawBitmap after FrmSetObjectNonusable.

When you tap a button SysHandleEvent puts a ctlSelectEvent onto the queue. The event.data.ctlEnter.controlID field contains the resource ID of the button that was tapped.

4.5 Field

```
FIELD  ID <Id.n> AT (<Left.p> <Top.p> <Width.p> <Height.p>)[USABLE] [NONUSABLE] [DISABLED]
        [LEFTALIGN] [RIGHTALIGN] [FONT <FontId.n>] [EDITABLE] [NONEDITABLE] [UNDERLINED]
        [SINGLELINE] [MULTIPLELINES] [MAXCHARS <MaxChars.n>]
```

Figure 4.5.1: Form Field Object Prototype

Fields are probably the most powerful tool in the PalmOS UI. You can use a field to display a string. Unlike LABEL, however, you can change the string at runtime using FldSetText, FldSetTextHandle, or FldSetTextPointer. FldSetText can only be used with NONEDITABLE fields. That leads into the important thing about fields - the user can be allowed to edit the field. Using a field the user can write memos, record names and addresses, add to do list items. Lots of applications use fields. The great thing is that PalmOS handles all the messy text processing stuff like backspace, changing focus, where’s the cursor, selecting text, etc. Your application just deals with the final result.

It’s best to use FldSetTextHandle when setting the text for editable fields. When you use handles the memory manager can resize the allocated memory. As far as I can tell, if you use FldSetTextPointer then the allocated memory cannot change. When you work with handles, if the user only enters one character, memory manager only allocates one byte. If the user enters ten thousand characters, then memory manager allocates ten thousand bytes. Figure 4.5.2 shows my WordPower app using handles with fields and a data database. Each database record stores a text string. To edit the database records, all you have to do is get the handle of the record and assign it to a field. PalmOS will use the database record as it’s field text and will resize the database record as required. This is cool. And it saves Pilot programmers time and RAM. But remember to set the field handle to NULL before shutting down the form, or else you’ll get an error.

```
static void EditGetRecord(void)
{
  Handle    recHandle;

//  Set handle for Word field to Word database

  recHandle = (Handle) DmGetRecord(WordDB, CurrentRecord);
  FldSetTextHandle(fieldptr_Word, recHandle);

//  Set handle for Definition field to Definition database

  recHandle = (Handle) DmGetRecord(DefinitionDB, CurrentRecord);
  FldSetTextHandle(fieldptr_Definition, recHandle);
}
```

Figure 4.5.2: FldSetTextHandle and a database

You can make a field UNDERLINED or NONUNDERLINED. For debugging, I find it best to make all fields underlined, that way I can see where they are. You can limit the field to one line (SINGLELINE) or you can permit multiple lines (MULTIPLELINES). For multiple line fields remember to use the FldRecalculateField call after setting the text to recalculate the word wrapping parameters. Your application can change the field options during run-time using the FldSetAttributes call.

4.6 Formbitmap

```
FORMBITMAP  AT (<Left.p> <Top.p>) BITMAP <BitmapId.n> [NONUSABLE]
```

Figure 4.6: Form Formbitmap Object Prototype

A bitmap is actually a separate resource, defined outside a form. But to mount a bitmap inside a form, you must use the FORMBITMAP object. For instance

FORMBITMAP AT (20 20) BITMAP 1000

puts bitmap resource 1000 on the form with the top left corner at (20, 20). Only use the FORMBITMAP object when you want the BITMAP to be fixed on the screen at the same location visible all the time. For instance, if you want a fancy frame around a certain form, you could add the frame to your application as a bitmap resource and lock it into your form using

FORMBITMAP AT (0 0) BITMAP bitmapID_FancyFrame

If, on the other hand, you want the bitmap to appear and disappear, to be changed with other bitmaps, to move around the screen don't use FORMBITMAP. Instead use the technique described in section 5.4.

4.7 Graffiti State Indicator

```
GRAFFITISTATEINDICATOR AT (<Left.p> <Top.p>)
```

Figure 4.7: Graffiti State Indicator Object Prototype

When you edit a memo, the graffiti state indicator is the symbol in the bottom right corner which shows special symbols for shift, shift lock, and punctuation. The graffiti state indicator isn't automatically there - you have to put it on the screen - but once it's on the screen you can forget about it. You don't have to keep track of shift penstrokes, PalmOS will do that and make sure the graffiti state indicator shows the appropriate symbol.

4.8 Internal Representation

In Palm OS, the form and each UI object exist as data structures. Each data structure is linked with the object on the screen, so if you change the value of the "pos.x" element of a FormLabelType structure, then the label will be in a different horizontal position the next time the form is drawn. The structures for resources and UI objects are contained in the Pilot header files. Figure 4.8 shows some of the structures defined in form.h. As you can see, the structures defining UI objects are defined in terms of other structures, some of which are found in other header files.

```
*       Copyright (c) Palm Computing 1994 -- All Rights Reserved

typedef struct {
        WindowType     window;
        Word    formId;
        FormAttrType   attr;
        WinHandle      bitsBehindForm;
        FormEventHandlerPtr    handler;
        Word    focus;
        Word    defaultButton;
        Word    helpRscId;
        Word    menuRscId;
        Word    numObjects;
        FormObjListType *      objects;
} FormType;


typedef struct {
        Word    id;
        PointType      pos;
        FormObjAttrType        attr;
        FontID  fontID;
        Char *  text;
} FormLabelType;


typedef struct {
        RectangleType  rect;
        Char *  text;
} FormTitleType;


typedef struct {
        FormObjAttrType        attr;
        PointType      pos;
        Word    rscID;
} FormBitmapType;
```

Figure 4.8: UI Object Structures

4.9 FormEventHandler

The form declaration in PilRC will put objects on the screen, but if you want anything to happen you have to write a form event handler in GCC or Pila. Section 3.4 mentioned that the event loop calls FrmEventHandler and FrmEventHandler calls your customized form event handler. Section 3.4 also demonstrated how you inform PalmOS of your custom form event handler using the FrmSetEventHandler call.

My convention is to give the form event handler the same name as the form, however many programmers add the words "EventHandler" to the end of all their event handler procedure names, which is probably a good idea. You could even put "FormEventHandler" at the end of your procedure name, if you want to make it really obvious. The form event handler deals with events relevant to your form. If your form has a button, then your form event handler has to detect ctlEnterEvent. If your form has a table, your form event handler will have to detect and process a tblSelectEvent. If your form has many objects, then the form event handler can be quite long. To reduce the procedure's size and improve it's readability, typically the form event handler will call other procedures which performs the activities associated with the detected events. Figure 4.9 shows my form event handler for the main form of my Stock Market game.

```
static Boolean StockMarketEventHandler(EventPtr event)
{
  int handled;

  switch (event->eType)
    {
  case    frmOpenEvent:
    StockMarketDrawForm();
    break;

  case ctlSelectEvent:  // A control button was pressed and released.
    if (event->data.ctlEnter.controlID== buttonID_Next)
    {
      RollDice();
      SM_Values.turn++;
      StockMarketDrawForm();
      handled = true;
    }
    if (event->data.ctlEnter.controlID== buttonID_Buy)
    {
      FrmGotoForm(formID_Buy);
      handled = true;
    }
    if (event->data.ctlEnter.controlID== buttonID_Sell)
    {
      FrmGotoForm(formID_Sell);
      handled = true;
    }
    break;

  case nilEvent:
    RollDice();
    SM_Values.turn++;
    StockMarketDrawForm();
    handled = true;
    break;

  default:
    return 0;
  }
  return handled;
}
```

Figure 4.9: StockMarket form event handler

5.0 OTHER RESOURCES

Chapter four discussed the form resource, but your application will require other resources in order to work. This chapter will explain the following resources:

- version resource
- string resource
- alert resource
- bitmap resource
- menu resource
- code resource
- data resource
- icon resource
- applicationname
- application

5.1 Version (tVER)

```
VERSION ID <VersionResourceId.n> <Version.s>
```
Figure 5.1: Version Resource Prototype

Optional. You can use this to indicate the version of the application or resource database. The resource is a string, so you can include characters as well as numbers.

5.2 String (tSTR)

```
STRING ID <StringResourceId.n> <String.ss>
```
Figure 5.2: String Resource Prototype

You can define strings in the global section of your code, or you can define them as tSTR resources. Modal forms can be associated with a string resource for their help or “tips” button. If you code constant global strings as string resources, then they can be accessed by other applications. Instead of using a data database to save memos, you could use a resource database and save each memo as a string resource. I haven’t tried this yet, so no sample code.

5.3 Alert (tALT)

```
ALERT ID <AlertResrouceId.n>
[HELPID <HelpId.n>]
[INFORMATION] [CONFIRMATION] [WARNING] [ERROR]
BEGIN
    TITLE <Title.s>
    MESSAGE <Message.ss>
    BUTTONS <Button.s> <BUTTON.s>...
END
```
Figure 5.3.1: Alert Resource Prototype

There are four different types of Alert resources: information, confirmation, warning and error. The only difference between the four is the bitmap displayed when it pops up. Alert resources are handy for displaying a message and halting the application until the user acknowledges the message. The alert resource is modal, meaning that it stays on the screen until the button is pushed. For this reason, your Alert IDs should almost always have at least one button.

You can use an alert for user input. You don’t need to write an event handler for the alert’s buttons. Instead, the FrmAlert and FrmCustomAlert calls return the number of the button that the user pushed. The buttons are numbered in the order that they are declared in the PilRC file. When you call FrmAlert the alert box is displayed. Your application halts until the user presses one of the buttons, then the number of the button is returned to your application.

Most alert IDs are constant, meaning that you define what they look like in the PilRC file and they always look the same. But you can also make dynamic alert boxes. Use the FrmCustomAlert call. When you call FrmCustomAlert you pass up to three strings. Palm OS looks for the symbols “^1” “^2” and “^3” in the MESSAGE part of the requested alert box, and replaces these symbols with the strings defined in the FrmCustomAlert call. This is a fairly powerful capability. Figure 5.3.2 demonstrates.

```
// Pilrc file
ALERT ID 1000
INFORMATION
BEGIN
    TITLE “Custom Alert Demo”
    MESSAGE “First choice is ^1 \n” \
            “Second choice is ^2 \n” \
            “Third choice is ^3”
    BUTTONS “First” “Second” “Third”
END

// GCC file
switch FrmCustomAlert(1000, “Apples”, “Oranges”, “Bananas”);
{
case 0:
  WinDrawChars(“Apples”, 6, 30, 30);
  break;

case 1:
  WinDrawChars(“Oranges”, 7, 30, 30);
  break;

case 2:
  WinDrawChars(“Bananas”, 7, 30, 30);
  break;
}
```

Figure 5.3.2: FrmCustomAlert demonstration

5.4 Bitmap (Tbmp)

```
BITMAP ID <BitmapResourceId.n> <BitmapFileName.s>
```

Figure 5.5: Bitmap Resource Prototype

PilRC will convert a windows monochrome bitmap to Pilot format. The <BitmapFileName.s> field identifies the bitmap that you want to include. Bitmaps are stored uncompressed, so they can gobble up a lot of memory. For instance download tricorder.prc from a Pilot file site. Cool app but uses too much RAM on my Pilot 1000. Some programmers are using their own compression procedures to save memory. Roger Critchlow mentions some public graphics routines at his web site. I wrote a zero suppression routine, but the code took up more RAM than it saved - when the image is source is only 160x160x2, I wonder if there’s enough source data to make compression routines economical.

Use WinDrawBitmap to draw a bitmap. To erase a bitmap use WinEraseRectangle or redraw the form using FrmDrawForm. To “permanently” embed a bitmap in a form, define it as a UI object in the form using the FORMBITMAP identifier in the PilRC file.

5.5 Menu (MBAR)

```
MENU ID <MenuResourceId.n>
BEGIN
    <PULLDOWNS>
END

<PULLDOWNS>: one or more of:

PULLDOWN <PulldownTitle.s>
BEGIN
    <MENUITEMS>
END

<MENUITEMS>: one or more of:

MENUITEM <MenuItem.s> <MenuItemId.n> [AccelChar.c]
```

Figure 5.6: Menu Resource Prototype

Menus are maybe the most distinctive feature of a graphical user interface. When you define a form in PilRC you can associate the form to an MBAR resource. If you want, you should be able to have multiple menus for the same form. You can only associate one in the PilRC definition, but supposedly your app can switch menus using the MenuSetActiveMenu. I’ve never done it, so I can’t provide sample code.

PalmOS 1.0 doesn’t allow grayed out menu items, and doesn’t support check marks beside menu items, as some other GUIs do. It does allow “keyboard shortcuts” (Palm Computing’s name) which are optional and declared with the [AccelChar.c]

field of the MENUITEM identifier. The keyboard shortcut allows the user to choose the menu item using the “Command” graffiti stroke followed by [AccelChar.c].

Don’t be confused by guide1.pdf. Guide1.pdf defines two menu UI resources: menu bar (MBAR) and menu (MENU). Guide1.pdf describes the pulldowns as separate MENU resources. As far as I can tell, MENU resources only exist in the “official” SDK, not the GNU Pilot SDK or the Pilot. I’ve inspected PRC files and can’t find any MENU resources. The menus are included within the MBAR resource.

5.6 Code (code)

Code is stored as a resource in the database. PalmOS doesn’t provide any documentation on the code resource, but you can thank Theodore Ts’o for hacking the PRC. Theodore has allowed his web page to be reprinted as Annex C. A link to his web page is included in the links sheet so you can read about his most recent discoveries.

5.7 Data (data)

Theodore Ts’o has also figured out how the data resource works. The explanation is included in Annex C.

5.8 Application (APPL)

```
APPLICATION ID <ApplResourceId.n> <APPL.s>
     <APPL.s> must be 4 characters long
```

Figure 5.8: Application Resource Prototype

I never use this resource. Although it’s defined in the PilRC documentation, I couldn’t find any references in Palm Computing’s documentation. I’ve inspected many PRC files but have never seen an APPL resource.

5.9 Application Icon Bitmap (tAIB)

```
ICON <IconFileName.s>
```

Figure 5.4: Icon Resource Prototype

The PalmOS application launcher uses this resource. If an application has a tAIB resource then the launcher will use it as the application’s icon using standard bitmap commands. If the application doesn’t have a tAIB resource, then the launcher just leaves an empty space above the application icon name. Resource ID 1000 is automatically assigned.

5.10 Application Icon Name (tAIN)

```
APPLICATIONICONNAME ID <AINResourceId.n> <ApplicationName.s>
```

Figure 5.9: Applicationname Resource Prototype

The Palm OS application launcher uses this resource. If an application has a tAIN resource, then it will be displayed in the launcher as the application name. Otherwise, the application launcher uses the name field from the application’s resource database header. I don’t use this resource - the name field in the database works fine.

6.0 SIMPLE EXAMPLE

Now that you know how to create resources and write a PilotMain procedure, it's time to put these together and build a simple Pilot application. A simple application is an application with one form, a few alert dialogs, one menu, buttons, fields, bitmaps, and labels. We will build the application in eight steps:

1. State the aim of the application
2. Design the form
3. Design other resources
4. Write the header file
5. Write the PilotMain
6. Write the form handler
7. Write the makefile
8. Testing and debugging

Before starting this chapter, you should have installed GCC (see Annex A and B). The tex2hex source files should be unzipped in a convenient subdirectory. You should have your editor up and aimed at the tex2hex source and have your MS-DOS window open for making the file. You should also open Paint and have a look at the two bitmap files, "text.bmp" and "hex.bmp".

In this chapter code fragments will be interspersed with the discussion. I've removed the "Figure 6.x.x" labels because they were annoying. Code is in fixed font (Courier New 8), discussion is in proportional font (Times New Roman 10).

6.1 State the Aim of the Application

It's important to state the aim of the application at the beginning, especially if you are writing the application for someone else. If you're just writing for fun, it's still important to know what you want before you start coding. For this example, we want an application that converts text strings to their hexadecimal equivalents. I want to enter a text string, push a button, and see the hexadecimal representation of the string. In the hexadecimal representation, I want each two character byte translation separated by hyphens for easy viewing. If someone tries to convert a null string, an error message should be displayed.

I'm an egoist, so I want an "about" screen which will show my name. To get to the about screen there must be a menubar with an "about" selection on it. The menu should also have a "Convert" command, which does the same thing as the on-screen button.

Just for practice, I want a few bitmaps. At the bottom of the screen there should be a bitmap of some ASCII characters. Every two seconds it should change to hexadecimal representation, and vice versa.

I want to be able to copy the text string to the clipboard and paste text strings from the clipboard to the application. The copy and paste commands should appear in the menu.

If someone enters something in the field and leaves the application, it should be there when they relaunch Tex2Hex.

6.2 Design the Form

The application needs one form. It doesn't matter whether or not the form is modal or if it has a frame. I choose a non modal form with no frame, because I like the look. The form must have a menu bar. It's non-modal so it can't have a helpstring. It doesn't need a default button. I'm going to call the form "Tex 2 Hex". I want the form to cover the whole screen.

The form must have a field for entering the text string. I want the text field to be centred near the top of the screen, about 100 pixels wide. This gives it a top left corner around (40 40). I will fine tune the layout later. The field has to be editable, should be leftaligned and use the standard font. Underlined is good, so the user knows how much room is available. Maximum number of characters will be 20 (100 pixels, average width for standard font is about five pixels). I can change that later, if I want.

The form must have a field for displaying the hexadecimal conversion. The hex field should be underneath the text field, centered, 100 pixels wide. Left align the field, use the standard font. Twenty text characters means sixty hex characters. At

five pixels per character that won't fit on a single line, so the field must be multiline. It doesn't have to be underlined. It can't be editable - the user doesn't change the hex representation.

The form needs a command button to initiate the conversion. The button should go between the two fields and have the caption "Convert to hex".

The alternating bitmaps will go at the bottom of the screen. Because they alternate, we won't use FORMBITMAP, but we should make sure that there's an empty space where we want the bitmaps to be. The bitmaps are 120 pixels wide and 40 pixels high.

The user may want to enter shifted characters, so we should put a graffitistateindicator in the bottom right corner.

```
FORM ID formID_tex2hex AT ( 0 0 160 160 )
NOFRAME
USABLE
MENUID menuID_tex2hex
BEGIN
    TITLE "Tex 2 Hex"
    FIELD ID fieldID_text AT (30 25 100 AUTO ) USABLE LEFTALIGN FONT 0 EDITABLE
                          UNDERLINED SINGLELINE MAXCHARS 20
    BUTTON "Convert to Hex" ID buttonID_convert AT ( 40 48 AUTO AUTO) USABLE FRAME FONT 0
    FIELD ID fieldID_hex AT (30 70 100 50 ) USABLE LEFTALIGN FONT 0 NONEDITABLE
                          MULTIPLELINES MAXCHARS 90
    GRAFFITISTATEINDICATOR AT (145 150)
END
```

6.3 Design other Resources

This is Tex2Hex version 1.0, so I'll use a tVER resource. I used Paint to make a customized icon. It's in file "tex2hex.bmp". I used Paint to make two bitmaps - "text.bmp" and "hex.bmp" - that will alternate at the bottom of the screen.

To make the user interface consistent with other applications, I'll use two pull down menus, "Commands" and "Edit". Commands will include the Convert and About instructions. Edit will include Copy and Paste. Each command will have a graffiti shortcut character.

We need two alert resources. We can use an information alert for the About screen. We can use an error alert if the user tries to convert a null string.

```
VERSION 1 "1.0"
ICON "tex2hex.bmp"
BITMAP ID bitmapID_text "text.bmp"
BITMAP ID bitmapID_hex  "hex.bmp"

MENU ID menuID_tex2hex
BEGIN
  PULLDOWN "Commands"
  BEGIN
    MENUITEM "Convert" menuitemID_convert
    MENUITEM "About"   menuitemID_about   "A"
  END
  PULLDOWN "Edit"
  BEGIN
    MENUITEM "Copy"    menuitemID_copy    "C"
    MENUITEM "Paste"   menuitemID_paste   "P"
  END
END

ALERT ID alertID_about
INFORMATION
BEGIN
  TITLE "About Tex2Hex"
  MESSAGE "Demo program for the GNU Pilot SDK tutorial"
  BUTTONS "Done"
END

ALERT ID alertID_errornullstring
ERROR
BEGIN
  TITLE "Error"
  MESSAGE "Cannot convert a null string"
  BUTTONS "Oops"
END
```

6.4 Write the header file

You have probably noticed that I'm not using literal numbers for resource IDs. I put all my resource IDs in a header file and include the header file in both the gcc source file and the PilRC source file. Figure 6.4 shows the header file so far.

```
#define formID_tex2hex           1000

#define menuID_tex2hex           1000

#define menuitemID_convert       1000
#define menuitemID_about         1001
#define menuitemID_copy          1002
#define menuitemID_paste         1003

#define bitmapID_text            1000
#define bitmapID_hex             1001

#define fieldID_text             1000
#define fieldID_hex              1001

#define alertID_about            1000
#define alertID_errornullstring  1001

#define buttonID_convert         1000
```

6.5 Write PilotMain, StartApplication, StopApplication, and EventLoop

The PilotMain procedure is standard.

```
DWord  PilotMain (Word cmd, Ptr cmdPBP, Word launchFlags)
{
  int error;

  if (cmd == sysAppLaunchCmdNormalLaunch)
 {

    error = StartApplication();     // Application start code
    if (error) return error;

    EventLoop();                    // Event loop

    StopApplication ();             // Application stop code
  }
  return 0;
}
```

StartApplication is standard. OpenDatabase creates the database if it doesn't exist and creates record zero containing a null string. In section 3.3 I said that the OpenDatabase procedure usually loads saved values from record zero, but in this case we can't. We're going to use the technique of setting the text field handle to record zero's database handle. Then whatever is in record zero automatically appears in the field. But we can't set the field handle until the form has been initialized. So we'll have to set the handle in the form event handler.

```
static int StartApplication(void)
{
  int error;

  error = OpenDatabase();
  if (error) return error;

  FrmGotoForm(formID_tex2hex);
}

static Boolean OpenDatabase(void)
{
  UInt          index = 0;
  VoidHand      RecHandle;
  Ptr           RecPointer;
  char          nullstring = 0;

  Tex2HexDB = DmOpenDatabaseByTypeCreator(Tex2HexDBType, Tex2HexAppID, dmModeReadWrite);
  if (!Tex2HexDB) {
    if (DmCreateDatabase(0, Tex2HexDBName, Tex2HexAppID, Tex2HexDBType, false)) return 1;
    Tex2HexDB = DmOpenDatabaseByTypeCreator(Tex2HexDBType, Tex2HexAppID, dmModeReadWrite);
    RecHandle = DmNewRecord(Tex2HexDB, &index, 1);
    RecPointer = MemHandleLock(RecHandle);
```

```
    DmWrite(RecPointer, 0, &nullstring, 1);
    MemPtrUnlock(RecPointer);
    DmReleaseRecord(Tex2HexDB, index, true);
  }
  return 0;
}
```

Because there's only one form, the EventLoop doesn't need to use a switch statement to process the frmLoadEvent. After all, there's nothing to switch - event.data.frmLoad.formID can only have one value, formID_tex2hex, because that's the only form in the application. In fact, for single form applications you can move the FrmInitForm, FrmSetActiveForm, and FrmSetEventHandler form calls into the StartApplication procedure. Then you don't need to switch on frmLoadEvent at all and you can delete the FrmGotoForm call from StartApplication. This optimization will save you some RAM. I've included the switch anyway, because eventually you will want to write multi-form applications, and then you'll need it. If you want, after completing this chapter you can optimize the example by yourself and see how much RAM you can save.

```
static void EventLoop(void)
{
  short      err;
  int        formID;
  FormPtr    form;
  EventType  event;

  do
  {

    EvtGetEvent(&event, 200);

    if (SysHandleEvent(&event)) continue;
    if (MenuHandleEvent((void *)0, &event, &err)) continue;

    if (event.eType == frmLoadEvent)
    {
      formID = event.data.frmLoad.formID;
      form = FrmInitForm(formID);
      FrmSetActiveForm(form);
      switch (formID)
      {
      case formID_tex2hex:
        FrmSetEventHandler(form, tex2hexHandler);
        break;
      }
    } while(event.eType != appStopEvent)
  FrmDispatchEvent(&event);
  }
}
```

Note the FldSetTextHandle call in StopApplication. Since we're using the technique of setting the field handle to record zero's database handle, we have to reset the field handle to null before the form is closed. Otherwise PalmOS would try to deallocate a handle which is still being used by the database. Strangely, however, in this application you can leave out the FldSetTextHandle(fieldptr_text, NULL) call and it still works. Maybe it's because the database closes before the field gets deallocated. Nevertheless, I recommend leaving the FldSetTextHandle call in, otherwise you might forget in some other app where it might cause an error.

```
static void StopApplication(void)
{
  FldSetTextHandle(fieldptr_text, NULL);
  DmCloseDatabase(Tex2HexDB);
}
```

You may have noticed some global variables and a few new definitions in this code. The global declarations and definitions will go at the beginning of the tex2hex.c file. The definitions have to go in tex2hex.c because we have included the header in our tex2hex.rcp file, and PilRC 1.3 doesn't accept characters (eg 'TxHx') as definition data. Tex2HexAppID and Tex2HexDBType are used in the DmCreateDatabase call. By specifying Tex2HexAppID the same as the application's creator ID, PalmOS knows that this data database belongs to our Text to Hex application.

```
FieldPtr  fieldptr_text;   // FieldPtr and DmOpenRef are defined in the PalmOS header files.
FieldPtr  fieldptr_hex;    // See if you can find where they are defined.
DmOpenRef Tex2HexDB;
char      Tex2HexDBName[]="Tex2HexDB";

#define   Tex2HexAppID        'TxHx'
```

```
#define   Tex2HexDBType       'Data'
```

6.6 Write the Form Handler

```
static void tex2hexHandler(void)
{
  int       handled = 0;
  FormPtr   form;

  switch (event->eType)
  {
  case    frmOpenEvent:
    form = FrmGetActiveForm();
    fieldptr_text = FrmGetObjectPtr(form, FrmGetObjectIndex(form, fieldID_text));
    FldSetTextHandle (fieldptr_text, (Handle) DmGetRecord(Tex2HexDB, 0));
    fieldptr_hex = FrmGetObjectPtr(form, FrmGetObjectIndex(form, fieldID_hex));
    FrmDrawForm(form);
    handled = 1;
    break;

  case ctlSelectEvent:  // A control button was pressed and released.
    if (event->data.ctlEnter.controlID== buttonID_Convert)
    {
      Convert();
      handled = 1;
    }
    break;

case menuEvent:
    switch (e->data.menu.itemID)
    {
    case menuitemID_convert:
      Convert();
      break;
    case menuitemID_about:
      FrmAlert(alertID_about);
      break;
    case menuitemID_copy:
      FldCopy(fieldptr_text);   // user has to select (highllight) the text to copy
      break;
    case menuitemID_paste:
      FldPaste(fieldptr_text);
      break;
    }
    handled = 1;
    break;

case nilEvent:
    ChangeBitmap();
    handled = 1;
    break;
  }
  return handled;
}
```

The point of this tutorial is to explain pilot programming, not how to convert ascii to hex, so I'm not going to talk about the Convert procedure very much. But there are a few things worth mentioning. FldGetTextPointer will give a pointer to the field's text. Then an index (textpos) can be used to move through the field's characters. FldSetTextPtr should only be used on read only fields. Also, FldSetTextPtr doesn't copy the text to some field data storage area, it just sets a pointer to whatever area you've already set up. For this reason the string hex[] has to be a global variable. If the variable were local, it would cease to exist at the end of the procedure and the hex field would point to nothing.

```
char   hex[60];

static void Convert(void)
{
  char     textchar;
  int      textpos;
  int      lownybble, hinybble;
  char     hexchar;
  int      textlength;
  CharPtr  textpointer;

  textpointer = FldGetTextPtr(fieldptr_text);
  textlength = FldGetTextLength(fieldptr_text);
  if (textlength)
```

```
  {
    for (textpos = 0; textpos <textlength; textpos++)
    {
      textchar = textpointer[textpos];
      lownybble = textchar % 16;
      hinybble = textchar / 16;
      if (hinybble < 10) hexchar = 48 + hinybble;
        else hexchar = 55 + hinybble;
      hex[textpos * 3] = hexchar;
      if (lownybble < 10) hexchar = 48 + lownybble;
        else hexchar = 55 + lownybble;
      hex[textpos*3 + 1] = hexchar;
      hex[textpos * 3 + 2] = 45;
    }
    hex[textpos * 3 - 1] = 0;
    FldSetTextPtr(fieldptr_hex, hex);
    FldRecalculateField(fieldptr_hex, true);
    FldDrawField(fieldptr_hex);
  }
  else
  {
    FrmAlert(alertID_errornullstring);
  }
}

static void ChangeBitmap(void)
{
  VoidHand   bitmaphandle;
  BitmapPtr  bitmap;

  WhichBitmap = (WhichBitmap + 1) % 2;
  if (WhichBitmap == 0) bitmaphandle = DmGet1Resource('Tbmp', bitmapID_text);
  else bitmaphandle = DmGet1Resource('Tbmp', bitmapID_hex);
  bitmap = MemHandleLock(bitmaphandle);
  WinDrawBitmap(bitmap, 20, 120);
  MemHandleUnlock(bitmaphandle);
}
```

6.7 Write the Makefile

The GNU Pilot SDK uses the unix utility make to make the application. Annex A describes a few things about make. You can get GNU MAKE documentation in HTML format from my website.

Here's the makefile I use. "all" is a phony target. When a make file produces more than one output file, then "all" is important. For a single prc you don't need to use it. The list (ls) command isn't necessary. It's there because I like to see the filesize of the built prc. In the last rule, I've used gcc's -c option to separate the compiling and linking. You don't have to separate the compiling and linking. You could use the single command m68k-palmos-coff-gcc -O1 tex2hex.c -o tex2hex.

```
all : tex2hex.prc

tex2hex.prc : code0001.tex2hex.grc tfrm03e8.bin
          build-prc tex2hex.prc "Text to Hex" TxHx *.grc *.bin
          ls -l *.prc

tfrm03e8.bin: tex2hex.rcp tex2hex.h tex2hex.bmp
          pilrc tex2hex.rcp

code0001.tex2hex.grc: tex2hex.c tex2hex.h
          m68k-palmos-coff-gcc -O1 -c tex2hex.c -o tex2hex.o
          m68k-palmos-coff-gcc -O1 tex2hex.o -o tex2hex
          m68k-palmos-coff-obj-res tex2hex
```

Everything's ready to go. Now you should go to your MAKE shell and compile the prc. The next section will be more valuable if you read it while working on the file.

6.8 Testing and Debugging

You read through the entire tutorial, compiled the code in the example, and it didn't work! Of course not. I'm not a great programmer. I don't think I've ever had an app work right on the first make. In my opinion testing and debugging is an essential part of building an application. I left a couple bugs in the code to illustrate the testing and debugging function. There were other mistakes on the first compile, but going through all of them would be too boring. As we identify the bugs and make corrections, you should correct the source code using your editor.

Before we start, I'll tell you something now that you should remember whenever app you're debugging uses a data database. When you debug you may load the same app into your pilot or copilot again and again. Reread section 2.2. When an application is hotsync'd to your Pilot (or loaded into copilot) the previous application is erased. But it's data database is not erased. If you have a problem with your database calls, then something like this might happen: load app first time, find a user interface error, fix the user interface error, load app second time, app crashes when its launched. The first time you launched the app it created a bad database. The second time you launched the app, it tried to load the bad database and crashed. I suggest that if your application uses a data database, use the memory application to delete the application before you load in a new copy. The memory application will delete all data databases having the same Creator ID as your application.

So try make again. GCC barfed lots of stuff. If you're new to C don't be dismayed. This probably means you've forgotten to terminate a line with a semicolon. Make it again, this time use the pause key to stop the display before the first lines scroll off the screen. I see this:

```
m68k-palmos-coff-gcc -O1 -c tex2hex.c -o tex2hex.o
tex2hex.c: In function 'ChangeBitmap':
tex2hex.c:34: parameter 'Tex2HexDBName' is initialized
tex2hex.c:39: parse error before '{'
tex2hex.c:39: declaration for parameter 'PilotMain' but no such parameter
tex2hex.c:36: declaration for parameter 'WhichBitmap' but no such parameter
tex2hex.c:35: declaration for parameter 'hex' but no such parameter
tex2hex.c:34: declaration for parameter 'Tex2HexDBName' but no such parameter
tex2hex.c:33: declaration for parameter 'Tex2HexDB' but no such parameter
tex2hex.c:32: declaration for parameter 'fieldptr_hex' but no such parameter
tex2hex.c:31: declaration for parameter 'fieldptr_text' but no such parameter
tex2hex.c:39: number of arguments doesn't match prototype
cc1.exe: prototype declaration
```

Look for the first line in the file that has an error. Sometimes the errors aren't reported in order. The first reported error is in line 31, so look there first. Line 31 looks fine. Work your way back. The previous line is missing a semicolon. Add the semi-colon and make again.

```
m68k-palmos-coff-gcc -O1 -c tex2hex.c -o tex2hex.o
tex2hex.c: In function 'EventLoop':
tex2hex.c:114: parse error before '}'
MAKE.EXE: *** [code0001.tex2hex.grc] Error 1
```

Check line 114. It's a "}". There must be something wrong with the {} nesting. This took a minute or two to figure out. It's the while(event.eType != appStopEvent) statement. It's at the end of the "if" statement, it should be at the end of the "do" statement. Here's a correction.

```
      case formID_tex2hex:
        FrmSetEventHandler(form, tex2hexHandler);
        break;
      }
    }
  FrmDispatchEvent(&event);
  } while(event.eType != appStopEvent);
}
```

Make it agai

n. Warning from line 219 "tex2hex.c:219: warning: passing arg 1 of `WinDrawBitmap' from incompatible pointer type", but the code compiles, links, and builds properly. Lets check the pointer type. Guide1.pdf says that the argument for WinDrawBitmap should be of type "BitmapPtr". Guide2.pdf says that DmGet1Resource is of type "VoidHand". But does DmGet1Resource return a handle or a pointer? Later, guide2.pdf claims"Result: Returns a pointer to resource data, or nil if unsuccessful." For now we can ignore the warning and see what happens.

So load, launch, and try converting some numbers. The graphics don't show up. So the warning was probably important. Use MemHandleLock to convert the handle to a pointer. The corrected code is below.

```
static void ChangeBitmap(void)
{
  VoidHand   bitmaphandle;
  BitmapPtr  bitmap;

  WhichBitmap = (WhichBitmap + 1) % 2;
  if (WhichBitmap == 0) bitmaphandle = DmGet1Resource('Tbmp', bitmapID_text);
  else bitmaphandle = DmGetResource('Tbmp', bitmapID_hex);
```

```
  bitmap = MemHandleLock(bitmaphandle);
  WinDrawBitmap(bitmap, 20, 120);
  MemHandleUnlock(bitmaphandle);
}
```

Make gives no errors or warnings. Load and launch the application on copilot. Try to convert a number. The bitmaps work, but they flash too fast when the field has the focus. The graphics seem to be synchronized with the cursor. The cursor turning on and off must be accompanied by a nil event. Cool, I didn't know that. If you don't like the graphic flashing that fast you can find your own way to deal with it, for instance a global counter. Next, test all the menu options. Test the grafitti shortcut commands for the menu options.

Time to test the database. Put something in the text field. Change to another application. Fine. Change back to Tex to Hex. Data manager error getting record. We must have neglected to do something when we closed the database. Looking up DmCloseDatabase in guide2.pdf "This routine doesn't unlock any records in the database which have been left locked, so the application should be careful not to leave records locked." Is the record locked? Under DmGetRecord guide2.pdf says "Return a handle to a record by index and mark the record busy." On page 30, guide2.pdf says "The busy bit is set when an application currently has a record locked for reading or writing." So the first time Tex2Hex runs it gets record zero and locks it. The second time it runs, the record is still locked. The normal way to unlock a record is to release it before leaving the application.

```
static void StopApplication(void)
{
  FldSetTextHandle(fieldptr_text, NULL);
  DmReleaseRecord(Tex2HexDB, 0, false);
  DmCloseDatabase(Tex2HexDB);
}
```

Test it again. Everything works. Wow, that was an easy debug. But next time you'll have to do it by yourself. Don't panic, though. If you can't debug your code you can post your problem at news.massena.com, pilot.programmer.gcc. But try to debug the problem yourself first, because every time you post a stupid question your credibility goes down. When your credibility hits zero, other programmers will put your name on their kill lists. Also, notice that news.massena.com keeps all the messages since pilot.programmer.gcc was first started. So you can download all the headers and do a quick search to see if your question has been asked before.

When everything works, and if you want to distribute your application, then it's time to connect to www.usrobotics.com and register the CreatorID "TxHx". If someone has already claimed it, then you have to choose an unused creator ID and make the appropriate changes to tex2hex.h and makefile. Then you can distribute your application. If you want, as an exercise you could go to USRobotic's site and find out if the creator ID TxHx is registered.

6.9 Debugging with the Error API

When I figure out how to use the GNU debugger with Pilot, then I'll let you know. For now, I'll offer my techniques for in-line debugging code. I usually keep these four lines commented out at the end of every application:

```
//        char  debugstring1[20], debugstring2[20];
//        StrIToA(debugstring1, **111**);
//        StrIToA(debugstring2, **222**);
//        ErrDisplayFileLineMsg (debugstring1, **333**, debugstring2);
```

They can be used to enter breakpoints in your code. The first line has to be copied to the top of your source code where you declare global variables. The next three lines go where you want the breakpoint. You can display three integers. Replace **nnn** with the variable names of the integers you want displayed. Another useful cal is ErrFatalDisplayIf. The prototype is

```
void ErrFatalDisplayIf (Boolean condition, char* message)
```

The boolean test is useful for entering conditional breakpoints. Suppose you don't know exactly why an application is misbehaving, but you do know that bad things will happen if the integer "MyNumber" exceeds 250. Then you can use this code snippet to display the value of "MyNumber" if it exceeds 250:

```
          StrIToA(debugstring1, MyNumber)
          ErrFatalDisplayIf( MyNumber>250, debugstring1);
```

That's the end of this tutorial. If you find out something new, remember to post it on Darrin's news service.

If you find this tutorial useful please send me a thank you note and tell me what country you're from. I collect emails from all over the world.

Andrew Howlett
Ottawa, Canada
June 1997

ANNEX A: INSTALLING AND USING GNU PILOT SDK ON WIN95

Installing the Tools

First, download GCC-0.40 for Pilot and PilRC. I use the links at Adam's Pilot software archive. Getting GCC for Pilot might be harder than it sounds. Some of the links are broken. J.J. Lehett recommends Ray's Pilot Software Archive. You should also get a copy of copilot.

GCC maybe be zipped or it may be tar'ed and gzipped into an archive. Some Win95 users might ask "Why tar it?" or "What is tar?" TAR is the tape archive and retrieve facility used on unix. It packs a directory structure with all it's files into one big uncompressed file. After tarring, unix operators compress. The procedure is historical - the original unix compress didn't support directory trees. But it's also helpful - tar and gzip produce a smaller archive than just zipping the directory structure, especially when lots of small files are tar'ed. For example, my archive of GNU documentation in HTML format is about 6M when tarred and zipped and about 18M when just zipped. Of course the hard drive space isn't a big deal, but download time matters to many people.

Winzip will untar and decompress the archive. It will take longer than you expect to open the archive and you should have at least 20M hard drive space, because winzip has to decompress the file before it can read the directory structure from the uncompressed tar file. If you abort the operation you may want to clean up your temp directory. I installed all the files to c:\cygpilot. In the following instructions, substitute whatever directory you used instead of c:\cygpilot.

Create a batch file in your c:\cygpilot directory called "setvar.bat". Here's my batch file:

```
echo running setvar.bat
PATH=%path%; c:\cygpilot\H-i386-cygwin32\bin;
set GCC_EXEC_PREFIX=//c/cygpilot/H-i386-cygwin32/lib/gcc-lib/
doskey
```

Copy PilRC version 1.3 (or better) to c:\cygpilot\H-i386-cygwin32\bin. Create a folder to hold the icons for your Pilot development tools. Add shortcuts for your favourite editor, copilot, Adobe Acrobat (set the working directory to wherever you keep your Pilot programming documentation), Calculator, Pilot Install (set the working directory to c:\cygpilot), Hotsync, Paint (set the working directory to c:\cygpilot), and your web browser. Create a new MS-DOS shortcut in your development folder. Rename it "MAKE". Use this shortcut to open the command shell where you run the MAKE utility. Use these program parameters for the MS-DOS shortcut:

```
Name:         MAKE
Command line: C:\WIN95\COMMAND.COM /e:1024
Working:      c:\cygpilot
Batch file:   setvar.bat
```

Now you're ready to code.

What the Tools Do

The GNU Pilot SDK includes four essential tools which process your source code: m68k-palmos-coff-gcc, m68k-palmos-coff-obj-res, pilrc, and build-prc.

m68k-palmos-coff-gcc. This is the GCC cross compiler for the Motorola 68000 type CPU with some custom startup code for PalmOS. It produces a single file containing all the M68K binary. There are lots of compiler options. Figure A.1 lists the options most relevant to Pilot. For a full list, download the GCC docs. The suffix on the input file determines what GCC does. Figure A.2 shows suffixes and actions

Option	Description	Comments
-Ox	optimize	x can be 1, 2, or 3. 1 is least optimization, 3 is most.
-o	output file	Defines the output file name.

-c	no linking	Don't run the linker. Output is object code.
-S	stop before compiling	Preprocesses to assembly, but doesn't compile. Output is assembly code.

Figure A.1: Some GCC Command Options

Suffix	Action
file.c	C source code which must be preprocessed.
file.i	C source code which should not be preprocessed.
file.ii	C++ source code which should not be preprocessed.
file.m	Objective-C source code. Note that you must link with the library `libobjc.a' to make an Objective-C program work.
file.h	C header file (not to be compiled or linked).
file.cc, file.cxx file.cpp, file.C	C++ source code which must be preprocessed. Note that in `.cxx', the last two letters must both be literally `x'. Likewise, `.C' refers to a literal capital C.
file.s	Assembler code.
file.S	Assembler code which must be preprocessed.
other	An object file to be fed straight into linking. Any file name with no recognized suffix is treated this way.

Figure A.2: GCC file extensions

m68k-palmos-coff-obj-res. This processor breaks the M68K binary file into separate code resources suitable for inclusion in a PalmOS resource database (ie a PRC file). This processor produces a number of files of the pattern "codeXXXX.yourfilename.grc" where XXXX is the code resource ID. For instance, my WordPower PilotMain comes out as "code0001.wordpower.grc".

PilRC produces all the resources except your code resource. It processes your *.rcp file into separate *.bin files for each resource. The format of the output files is "tFRM03e8.bin" where tFRM is the resource type and 03e8 is the resource ID in hexadecimal.

build-prc combines the resources produces by the previous three tools into a resource database. Format for the command is.

```
build-prc <destination.prc>  <Database name> <CreatorID> <resource 1>  \
  [resource 2] [resource 3] [resource 4] ...
```

Make

You don't have to use the MAKE utility. A batch file will do almost the same thing. I've included a batch file called "compile.bat" which will compile the example. This tutorial will assume that you've chosen to use MAKE. I won't get into MAKE too much - you can get the manual from my website. There are a few things I'll say here.

```
wordpower.prc : code0001.wordpower.grc tfrm03e8.bin
        build-prc wordpower.prc "Word Power" WPow code0001.wordpower.grc  \
                code0000.wordpower.grc data0000.wordpower.grc *.bin pref0000.wordpower.grc
```

In your makefile there are rules and commands. The first line of the above example is a rule. The next line is a command. **In make, all commands must start with a tab**. Be careful that your editor doesn't replace tabs with spaces. Also, notice that the build-prc command is spread over two lines. In make the "\" character concatenates lines. This is handy because usually you will have lots of resources to add to your PRC. The order of the resources doesn't matter. Notice the "*.bin" in the last line. build-prc supports pattern matching in the resource list. *.bin will grab all the resources produces by PilRC. Be careful - if you don't clean up your directory between makes, then there could be old resources that your app doesn't use any more still hanging around. These unused resources will be built into your prc and waste Pilot RAM. So before you build a release version of your app, always clean up the resources in your directory.

Another thing to know about make is that each command runs in a separate shell. So if you change directories in a command, then in the next command the new shell will restore the original working directory. You can combine multiple commands in the same shell by separating them with semi-colons. Then you can change directories, semi-colon, execute command in new directory. MAKE uses the "bourne again shell" whose filename is "bash.exe" in your c:\cygpilot\H-i386-cygwin32\bin directory. Actually, MAKE always looks for "sh.exe" so you will find "sh.exe" in your c:\cygpilot\H-i386-cygwin32\bin directory - sh.exe is just a copy of bash.exe. Now the weird part. **You have to have a c:\bin directory and sh.exe has to be in it.** I don't know why. Even though c:\cygpilot\H-i386-cygwin32\bin is in your path, and sometimes MAKE will find sh.exe there, sometimes MAKE needs c:\bin\sh.exe. I don't know why.

ANNEX B: INSTALLING AND USING GNU PILOT SDK ON LINUX

This annex has been contributed by Roger Critchlow and Blake Winton. The latest version is available at Roger's website.

Developing Pilot Applications,

by Roger E. Critchlow Jr.

Sources for stuff

My platform for developing Pilot applications is a PC running Linux, which I bought from VAResearch back when they were still called Fintronic. The tools which I use to build doodle are:

pilot-link

Routines and programs for talking to the Pilot and manipulating Pilot databases. Available from Jeff Dionne's archives.

prc-tools

Patches to gcc and programs for assembling Pilot applications. Available from Jeff Dionne's archives.

gcc

The GNU C compiler. Available from prep.ai.mit.edu.

binutils

The GNU binary object utilities. Available from prep.ai.mit.edu.

libgr

A collation of the free graphics image processing tools available for Unix. Available from ftp.ctd.comsat.com.

Palmos Documents

The documentation for PalmOS is available in Acrobat format at www.usr.com.

Jeff Dionne's archives

The primary resources for developing pilot applications under Linux and other Unix systems are collected on Jeff Dionne's PalmOS archive, which is now mirrored in two additional locations:

- Dionne's archive
- Daveltd mirror
- Uiuc mirror

Compiling with gcc under Linux

Here is Blake Winton's summary of how to build the Linux based development tools for the Pilot:

1. Get:
   - prc-tools.0.4.2.tar.gz
   - binutils-2.7.tar.gz

- gcc-2.7.2.2.tar.gz

2. tar xvzf prc-tools.0.4.2.tar.gz
   - cd prc-tools-0.4.2
   - vi gcc-2.7.2.2.palmos.diff
   - :%s/Lenght/Length/g
   - :wq
   - cd ..
3. tar xvzf binutils-2.7.tar.gz
   - cd binutils-2.7
   - ./configure --prefix=/usr/local --target=m68k-palmos-coff
   - su
   - make all; make install
   - exit
   - cd ..
4. tar xvzf gcc-2.7.2.2.tar.gz
   - patch -p < prc-tools-0.4.2/gcc-2.7.2.2.palmos.diff
   - cd gcc-2.7.2.2
   - ./configure --prefix=/usr/local --target=m68k-palmos-coff
   - su
   - make LANGUAGES=c install
   - cd /usr/local/lib
   - chmod -R a+rX .
   - exit
5. cd prc-tools-0.4.2
   - vi ’find . -name "Makefile" -print’
   - remove all gdb hooks.
   - su
   - cd /usr/local/m68k-palmos-coff/lib
   - cp /pilot/prc-tools-0.4.2/lib* .
   - chmod a+r *
   - cd /pilot/prc-tools-0.4.2
   - make install
   - exit
6. You’re done!!! :)

The last three steps of number 5 should, in theory, be enough, but I’ve found that for one reason or another, they didn’t work for me, and I needed to copy over the lib files myself.

Created April 17, 1997, updated May 7, 1997 by rec@elf.org.

The PRC Format

as guessed/compiled from various sources by
Theodore Ts'o
Last updated May 30, 1997

Recent Changes to This Document

Thanks to Ian Goldberg, the format of the RLE compression in the Data resource is now documented.

Introduction

Up until now, there really hasn't been a public documentation of the PRC format. This was was frustrating for me since I wanted to write a BFD backend for PRC executables. I tried looking at the sources for some of the alternative software Pilot development programs, such as Pila and prc-tools. Unfortunately, these programs would often treat header fields as "magic", and often different programs would do completely different things with the same fields.

This document is my attempt to rectify this situation. It is the product of both research into existing implementations, as well as experimentation to clarify some minor points of how the Pilot tools work. Some of my sources include:

- The Pila compiler for Unix
- The obj-res and emit-prc programs from the prc-tools package
- Inside Macintosh

Any mistakes in this document should be considered my responsibility, however, and not the responsibility of these sources. If you find any mistakes, or have anything to contribute, please contact me via email at tytso@mit.edu.

High-level format of a PRC file

An application for the pilot is simply a Pilot resource database with a number of mandatory resources (CODE 0, CODE 1, DATA 0, PREF 0, etc.) The PRC file, then, is simply the flat file representation of a Pilot resource database. When the PRC file is loaded into the Pilot, it is converted into a resource database using the PalmOS routine dmCreateDatabaseFromImage().

The PRC format consists of the following major pieces:

1. *PRC Header*
2. *PRC Resource Headers*
3. *PRC Resources*

PRC header

The PRC Header is located at the very beginning of the file, and contains the following information:

name	type	size	notes
Name	char		
32	[1]		
Flags	int		
2	[2]		
Version	int		
2	[3]		
create_time	pilot_time_t	4	[4]
mod_time	pilot_time_t	4	[4]
backup_time	pilot_time_t	4	[4]

mod_num	int	4	[5]
app_info	int	4	[5]
sort_info	int	4	[5]
Type	int		
4	[6]		
Id	int		
4	[7]		
unique_id_seed	int	4	[5]
next_record_list	int	4	[5]
num_records	int	4	[8]

[1] The name field is zero terminated and is usually zero padded. The pila assembler sneaks ’Pila’ into the last 4 bytes of this field

[2] The ’flags’ field is 0x01 for PRC executables

[3] The ’version’ field is 0x01 for PRC executables

[4] Pilot time is defined to be the number of seconds since January 1, 1904 (i.e, Macintosh time).

[5] This field must be zero for PRC executables

[6] The ’type’ field must be ’appl’ for PRC executables

[7] The ’id’ field is a four character "creator code", ala the Macintosh

[8] The ’num_records’ field contains the number of resources in the PRC file.

PRC Resource Headers

The Resource headers follow immediately after the PRC Header field. The num_records field in the PRC Header indicates the number of resources contained in the PRC file, and there is a 10 byte resource header for each resource.

name	type	size	notes
Name	char	4	Name of the resource
Id	int	2	ID number of the resource
Offset	int	4	Pointer to the resource data

PRC Resources

The actual data for the resources follow after the resource headers. The resource data records are stored in order as they appeared in the resource headers. (Since the resource header does not have a size field, the size is determined by examining the where the offset pointer for the next resource.)

The Mysterious Code #0 Resource

The contents of this resource have been (up until now) somewhat mysterious, with different packages --- Metroworks, Pila, and the obj-res program from the prc-tools package --- generating in some cases very different values.

Pila creates an 8 byte resource, with the first four bytes described as the initialized data size and the next four bytes described as the unitialized data size. Pila stores the size of the data segment in the first field, and the second field is always filled with zeros.

The obj-res program from the prc-tools package does something quite different. It creates a 24 byte resource, which is filled in as follows:

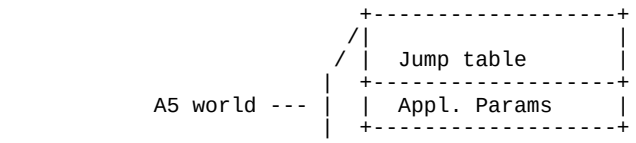
offset	value
0	0x00000028
4	[bss+data segments rounded up to 4 bytes]
8	0x00000008
12	0x00000020
16	0x0000
18	0x3F3C
20	0x0001
22	0xA9F0

The obj-res program was quite obviously treating most of the fields in the 24 byte resource as magic values, obtained from looking at the contents of the code 0 resource from PRC files generated by the Metroworks compiler.

I believe that the code 0 resource is identical to that which is used by the 68k Macintosh; this theory explains why pila can use such a different code 0 resource, and yet still produce working appications. To explain this, though, we need to take a detour and look at the Macintosh memory management model.

The Code 0 Resource on the Macintosh

On the Macintosh, when an application is lauched, MacOS allocates allocates a block of memory known as an *application partition*, which has the following format:



The *A5 world* is very important to a Macintosh application. In general, 68k Macintosh executables don't seem to bother with relocations; instead, the code segment of an application uses only position-idependent code, and it references its jump table, application parameters, and application global variables as fixed offsets (positive and negative) from the A5 register. The A5 register is always pointing at a fixed location inside the application's A5 world.

The code 0 resource is generated by the linker (for example, the Metrowerks linker) and contains the necessary information so that MacOS can setup an application's A5 world. It has the following structure:

offset	size	description
0	4	size above A5 (jump table+parameters)
4	4	size of application globals
8	4	size of jump table
12	4	A5 offset of jump table
16	8	Jump table entry #0
24	8	Jump table entry #1
...		
16+8n	8	Jump table entry #n

The jump table is used to transfer control between different code segments, which may not yet be loaded into the system. Jump table entry #0 points to the start address of the application. Jump table entries have two forms, depending on whether the destination code segment to which the entry points is loaded or unloaded. Initially, all code segments are unloaded, and so all jump table entries have the following form:

offset	size	description
0	2	Offset of this routine from the beginning of the segment
2	2	m68k push instruction
4	2	segment number (arg for push instruction)
6	2	_LoadSeg trap

When an application transfers control through a jump table to an unloaded segment, it casues a call to the _LoadSeg trap, which loads the segment and then modifies all of the jump table entries for the application that point to the now-loaded segment with the following jump table entry:

offset	size	description
0	2	Offset of this routine from the beginning of the segment
2	6	m68k long jump instruction to the routine in another segment

Hence the calling macintosh application can always transfer to another code segment by jumping to offset 2 for a particular jump table entry. This scheme has the effect of a "poor man's virtual memory", since it allowed code segments to be demand-loaded as necessary, without requiring an MMU (which early Macinoshes didn't have!).

What Does All This Have to do with the Pilot?

An examination of Pilot applications which were created using the Metrowerks compilers makes it clear that the code 0 resource of the Pilot follows the format of the code 0 resource of a 68k Macintosh. In particular, the format of the jump table entry is unmistakeable. Consider the code 0 resource for the PalmPilot's Expense application:

offset	size	value	description
0	4	0x00000030	size above A5 (jump table+parameters)
4	4	0x00000060	size of application globals
8	4	0x00000008	size of jump table
12	4	0x00000020	A5 offset of jump table
16	2	0x0000	Jump table entry --- offset
18	2	0x3f3c	Jump table entry --- push instruction
20	2	0x0001	Jump table entry --- segment
22	2	0xA9F0	Jump table entry --- SegLoad trap

All of the fields from this pilot application match up correctly with a 68k macintosh code 0 resource. The size of the jump table is correct (8 bytes), as in the start address of the application (code segment 1, offset 0) in the first (and only) jump table entry. Hex 0x3f3c is a push instruction which places segment 1 on the stack.

Now that we have confirmed this hypothesis, what does this have to tell us about the A5 world of a Pilot application? First of all, like the Macintosh memory model, the application globals are located **below** the A5 register. Hence, accessing application globals requires making negative offsets to the A5 register. The expense application reserves 48 bytes of space above the A5 register for the jump table and "application parameters". What gets stored in this space? More on that a little later.

However, apparently not all of the code 0 resource is used by the Pilot, at least not in PalmOS 1.0 or 2.0. For example, the Pila assembler only creates a code 0 resource which is 8 bytes long. Selective corruption of the jump table does not harm a pilot application, which seems to indicate that the Pilot does not use the jump table to determine the application start address.

More interesting, perhaps, is the Pila's alternative memory model. In the code 0 resource generated by the Pila, the "Application Global" size is 0, and the size above A5 is set to the size of the Pila program's data segment. In other words, Pila programs have their data segment **above** A5, instead of below it.

Does the fact that the data segment for Pila-compiled programs is located where the "jump table" and "application parameter" section cause any problems? Yes, although Pila has a workaround that apparently works for PalmOS 1.0 and 2.0. Currently the PalmOS loader stores a pointer to the application's SysAppInfo at the four bytes starting at the A5 register, and some of the PalmOS ROM routines depend on this. To avoid overwriting this pointer, Pila's startup routine reserves four bytes of data segment, and when Pila constructs the compressed Data segment, it avoids overwriting the first four bytes above the A5 register.

One useful data point which we can infer from Pila's non-standard memory module is that only the first four bytes of memory above the A5 register currently appear to be in use. Otherwise, Pila compiled programs would likely cause some kind of crash or Pilot malfunction. Apparently the rest of the 32 bytes reserved by the Metrowerks compiler for "Application Parameters" is reserved for future expansion, but is currently not used. This is another hint that indicates that the jump table currently not used.

This causes me to raise a cautionary note that while Pila-compiled programs work now, they may fail in the future if later versions of PalmOS use additional memory above the A5 register beyond the first four bytes. The PalmPilot Developer Technical Brief explicitly warns that "If your application was not developed with the Metrowerks CodeWarrior for Pilot, it may run into problems." Developers would do well to heed this warning, especially in the case of Pila where it is using a radically different memory model where the data segment of the assembly language program is overloading memory space reserved for application preferences and for the jump table.

The Code #1 resource

This resource contains the actual code for the application. For some reason, PRC executables generated by the Metrowerks compiler have the a four-byte word 0x00000001 (ori.b #1, %d0) at the beginning of the code resource. The obj-res program duplicates this behaviour, although Pila does not, and it doesn't seem to make a difference.

It is not clear whether the four byte word is meant to a flag or bitfield, or whether it is some other kind of signal. When PalmOS starts executing the application, it obviously starts at beginning of code segment #1. To test to see if the initial four byte word was intended to be interpreted as a instruction, I tried replacing it with a rts instruction. This test made it clear that the ori.b #1, %d0 instruction is actually executed. However, this instruction doesn't appear to do anything useful. It merely sets the low bit in data register 0; however, data register 0 is never used until it is later re-initialized.

OPEN QUESTION: Why does the Metrowerks plance an initial 4 byte prefix (0x00000001, or ori.b #1, %d0) in the CODE 1 segment, and why does it matter?

The Data Resource

The data resource is perhaps the most mysterious resource, because it is neither documented by the USR-provided Pilot Tutorial and Cookbook books, nor in Inside Macintosh, since the Data resource is unique to the Pilot. (MPW uses a similar, although different, mechanism which is used to initialize global variables, involving the use of the A5init sigment.) Most of the information in this section has been taken from comments in the Pila assembler. Apparently Darrin Massena, the author of Pila, had some contacts inside the Pilot development group which gave him some of the necessary technical information.

The major purpose of the data resource is to initialize global variables. The data resource can also contain relocation tables to handle arrays containing pointers to static data (for either constant data stored in the code 1 segment, or writeable data which is stored in the data resource). The high-level format of the data resource is:

offset	size	description
0	4	offset of CODE 1 xrefs (4+n+m)
4	n	compressed global initializers
4+n	m	compressed DATA 0 xrefs
4+n+m	p	compressed CODE 1 xrefs

The compressed global initializer section

The compressed global initializer section contains the following substructure repeated three times:

- A5 offset (4 bytes), which specifies the destination where the uncompressed data should be written.
- Compressed stream, in blocks
- ASCII 0 (separator).

This flexible structure allows up to three contiguous regions of the application's A5 world be initialized with compressed data. If the flexibility is not needed, one or more of these initializer substructures may be replaced by 5 nulls (4 nulls for the A5 offset and one ASCII null to indicate the lack of compressed blocks).

The Pilot uses an enhanced RLE scheme for its compressed stream. The compressed stream contains a series of RLE blocks, followed by a zero byte to terminate the compressed stream. Courtesy of Ian Goldberg, the following RLE blocks are recognized by the Palm Pilot Pro:

byte stream	description
(0x80 + n) b_0 ... b_n	n+1 bytes of literal data (n <= 127)
(0x40 + n)	n+1 repetitions of 0x00 (n <= 63)
(0x20 + n) b	n+2 repetitions of b (n <= 31)
(0x10 + n)	n+1 repetitions of 0xFF (n <= 15)
0x01 b_0 b_1	0x00 0x00 0x00 0x00 0xFF 0xFF b_0 b_1
0x02 b_0 b_1 b_2	0x00 0x00 0x00 0x00 0xFF b_0 b_1 b_2
0x03 b_0 b_1 b_2	0xA9 0xF0 0x00 0x00 b_0 b_1 0x00 b_2
0x04 b_0 b_1 b_2 b_3	0xA9 0xF0 0x00 b_0 b_1 b_2 0x00 b_3
0x00	end compressed data

OPEN QUESTION: Are all of these compression blocks supported on the old PalmOS 1.0 machines? I am particularly paranoid about the RLE blocks beginning with 0x01 -- 0x04. Also, why did PalmOS define special cases for 0xA9 0xF0?

Code 1 XREF and Data 0 XREF sections

Unfortunately, the format of the XREF sections is totally unknown. The Pila and obj-res programs currently emit 6 longwords of zeros.

The obj-res program supports its relocation of initialized data by manually including a relocation table which get processed by a custom startup routine. It would be cleaner to allow the PalmOS loader to do this work for the application automatically. (Although relying on this would probably limit that application to PalmOS 2.0 devices.)

OPEN QUESTION: What is the format of xrefs?

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- **a)** You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

- **b)** You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

- **c)** If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- **a)** Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

- **b)** Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

- **c)** Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major

components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the

Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS