

Programming the Palm OS™ for Embedded IR Applications

*Author: Frank Ableson
CFG Solutions Inc.
Mark Palmer
Microchip Technology Inc.*

A Palm OS application program to interface to an embedded system via IrCOMM is included in the Appendices of this application note. This source code shows the system calls that can be used for IR communication.

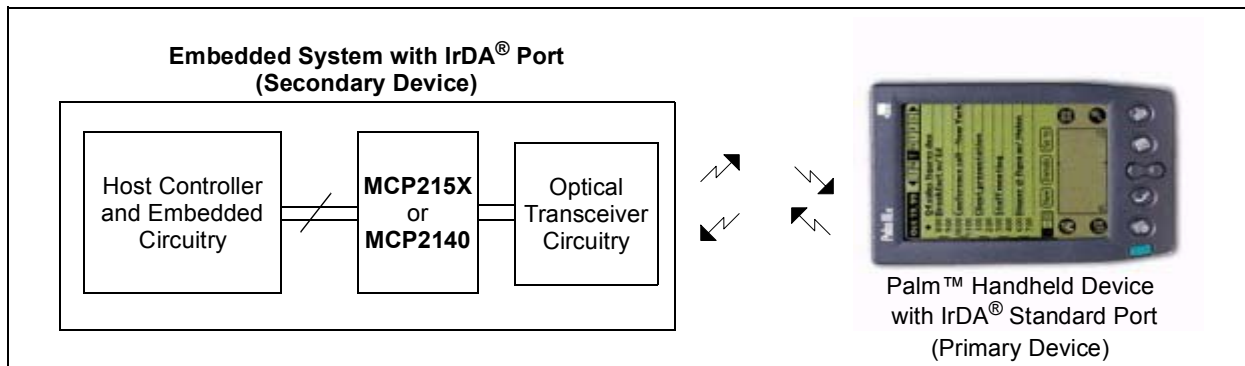
INTRODUCTION

This application note strives to impart core, fundamental programming concepts and design considerations for the development of Palm OS® application programs. Attention is given to each of the fundamental areas of Palm OS application development in the “C” programming language.

Appendix A describes the system and documents the tool used to create this Palm® application program and **Appendix B** through **Appendix G** is the Palm Application Program source code.

Figure 1 shows an IrDA® standard system, where a Palm PDA device is communicating to an embedded system. In this system, the Palm PDA operates as the Primary device and the embedded system operates as the Secondary device.

FIGURE 1: PALM™ PDA - EMBEDDED SYSTEM BLOCK DIAGRAM

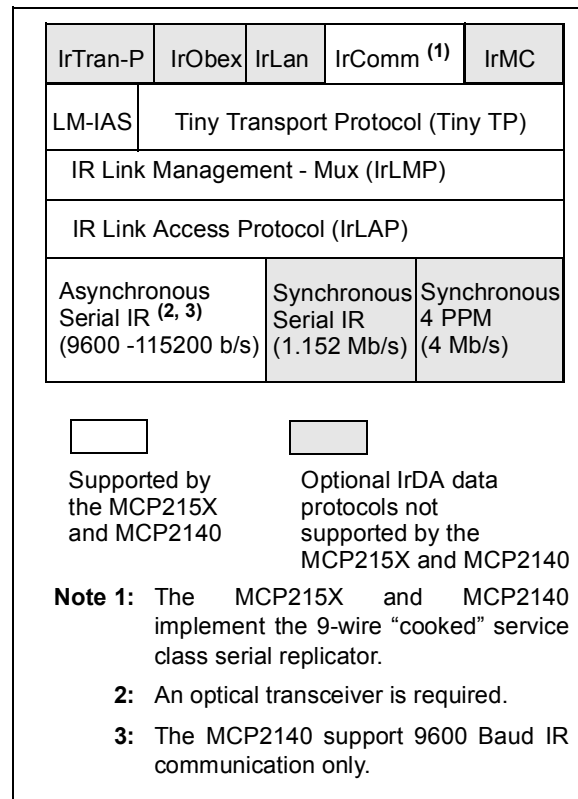


Terminology

Below is a list of useful terms and their definitions:

- **Palm OS device:** Any device running the Palm OS operating system. This includes devices from Palm Computing such as the V and m series units, the IBM® Workpad, a Sony Clie®, or a Handspring® Visor®.
- **HotSync™:** The process of synchronizing the Palm OS device to a host system.
- **Host System:** The computer with which a Palm OS device performs a HotSync. The host system is also where development takes place. Host systems are typically Windows®, Macintosh® or Linux.
- **Conduit:** Code containing the logic necessary for synchronizing a database on the Palm OS device with the host system. Each conduit is invoked during the HotSync operation. Conduits on the PC platform are packaged as dynamically linked libraries (DLLs). Conduits work differently under Linux, where each conduit is an application instead of a library.
- **POSE:** The Palm OS Emulator. A desktop tool useful for debugging applications.
- **Primary Device:** The IrDA standard device that queries for other devices.
- **Secondary Device:** The IrDA standard device that waits to detect IR communication before doing any IR communication.
- **Host Controller:** The controller in the embedded system that communicates to the MCP215X or MCP2140.
- **MCP215X:** An IrCOMM protocol handler IC that supports IR communication from 9600 baud to 115,200 baud.
- **MCP2140:** A low-cost IrCOMM protocol handler IC that supports IR communication at 9600 baud.
- **Protocol Stack:** A set of network protocol layers that work together. Figure 2 shows the IrDA standard protocol stack.
- **IrCOMM (9-wire “cooked” service class):** IrDA standard specification for the protocol to replace the serial cable (using flow control).

FIGURE 2: IrDA DATA - PROTOCOL STACKS



Environment Basics

This application note begins with a discussion of the basics of the Palm OS environment, including resource limitations, tasking model, user interface basics and deployment options. Palm OS application developers must understand these basic boundaries to successfully develop for this platform. As the application note progresses, more detailed information conveys the specifics of communications programming for the Palm OS platform.

Resource limitations

Palm OS devices face typical hand-held computer system resource limitations, such as storage capacity, available Random Access Memory (RAM), processor speed, input devices and power consumption. Fortunately, working with restricted resources is nothing new to the embedded developer. These limitations go beyond the problem of not having enough resources to run all of the applications common on the desktop. They also pose specific challenges to the developer building the applications. For example, a Palm OS application cannot allocate an arbitrarily large data structure since heap memory is limited (measured in Kilobytes instead of Megabytes). Global variables consume part of this allocation, leaving a very finite amount of space for dynamically allocated data structures.

The storage capacity of Palm OS devices currently range from 2 Mbytes to 64 Mbytes. There is no real distinction made between what is traditionally considered volatile RAM and nonvolatile, or persistent storage, as in a hard drive on a desktop. Storage of applications, database records and running applications consume this relatively scarce resource. This shared space means that the quantity of Address or DateBook records stored on a device directly impacts the space available for the applications themselves.

More importantly, this resource consumption can directly impact the application needing to allocate temporary storage for an operation at run-time. The preferred algorithm for a specific problem may fail on devices with too many Address records. For example, an application cannot slowly spool a graphical print job, due to insufficient storage space.

EMULATING A MULTI-TASK ENVIRONMENT

The operating system is multi-tasking, though the threads are not available for “user” applications; the few existing threads are for system use only. Surprisingly, Palm OS applications do an elegant job of emulating a multi-tasking environment. Applications create this effect by remembering the currently displayed record when the application ends. When the application is restarted, it jumps directly to the most recently displayed record, as if the application had been running the entire time. In reality, the application was terminated and restarted.

USER INTERFACE (UI) AND APPLICATION PROGRAMMING INTERFACE (APIs)

The Palm OS offers an economical User Interface (UI). Traditional user interface elements (such as edit boxes, lists, drop-downs, bitmaps, etc.) are available for building applications. Limited screen size and lack of a keyboard impose some restrictions on the user interface. Later-model devices add greater color depth, bringing a little more appeal to the display. The Palm OS provides a rich Application Programming Interface (API) for manipulating each type of user interface element.

The UI should be thought out in advance, based on what the customer needs to achieve with the application. Care should be taken to ensure that the UI is easy to use.

Some suggestions that can help are:

- Keep the UI “clean” by minimizing the number of buttons, pull downs, icons, etc.
- Minimize the number of steps performed to see vital information:
 - most information should be accessible in a minimal (1 or 2) number of stylus taps
- Optimize which features to include to ensure a positive user experience

APIs are available for virtually every action necessary in a Palm OS application. It is possible for a user application to go straight to the metal (the Hardware) on a Palm OS device. This practice is discouraged in most cases and is considered a programming “hack”. This may cause the user application to no longer work on a later-model device, due to different hardware characteristics. However, this technique of “hacking” can offer interesting opportunities to insert custom code into various portions of the operating system. The result is similar to the functionality of a TSR in the days of DOS, where an interrupt vector is “hooked”, allowing custom code to be invoked when certain events occur. The recommended technique for function (or trap) hooking for the Palm OS involves using a hack manager to coordinate the precedence of multiple custom hacks in use at the same time.

PALM OS DATABASE TYPES

The Palm OS file system is actually a database manager. Everything stored on a Palm OS device is a database. Applications are stored as databases, while user interface components are stored as resource databases. Of course, data, such as phone numbers and to-do lists, are stored as records in a database.

Figure 3 shows a representation of the Palm Database Header, while Example 1 shows what the programming structure looks like.

Two database types of primary concern on the Palm OS are:

- PRC
- PDB

The first database type is commonly referred to as a “PRC” because of the file extension used on host systems. The PRC file contains a “ready to run” Palm OS application, or a Palm OS Shared Library (the end result of the build process). The applications built in this application note result in files with a PRC extension, which are loaded to a Palm OS device.

The second type of database is a “PDB” file. The PDB contains data stored in what is traditionally considered a database. A data-collection application stores the captured information in a database on the Palm OS device. When this type of database is backed up to the host system, the file extension is PDB.

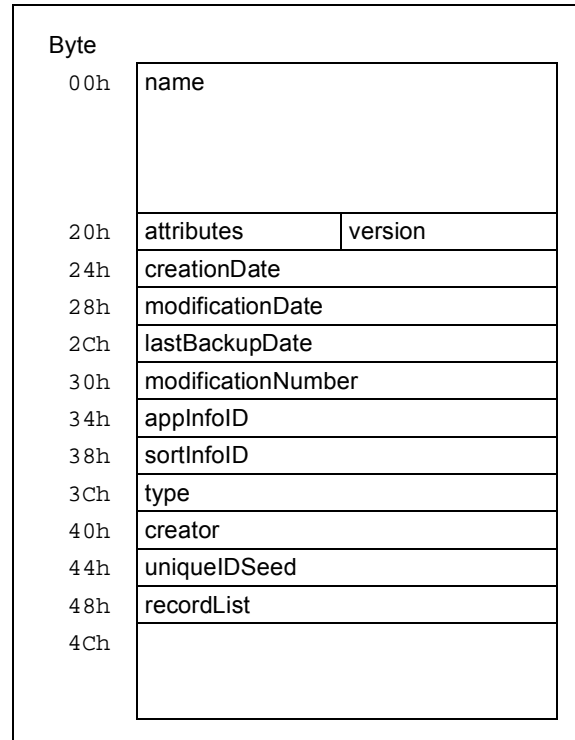
While the host system relies on file extensions, the Palm OS distinguishes databases by their “type”. The type is designated by a 32-bit value stored in the database itself.

Typical values:

- appl: Palm OS application (prc extension on host)
- libr: Palm OS Shared Library (prc extension on host)
- data: Traditional record storage (pdb extension on host)

There are other database types, but the PRC and PDB represent the majority of databases encountered when working in this environment.

FIGURE 3: PALM DATABASE HEADER



EXAMPLE 1: PALM DATABASE HEADER

```

Line # |
001 | typedef struct
002 | {
003 |     Int8 name[dmDBNameLength];
004 |     UInt16 attributes;
005 |     UInt32 creationDate;
006 |     UInt32 modificationDate;
007 |     UInt32 lastbackupDate;
008 |     UInt32 modificationNumber;
009 |     LocalID appInfoID;
010 |     LocalID sortInfoID;
011 |     UInt32 type;
012 |     UInt32 creator;
013 |     UInt32 uniqueIDSeed;
014 |     RecordListType recordlist;
015 | } DatabaseHdrType;
    
```

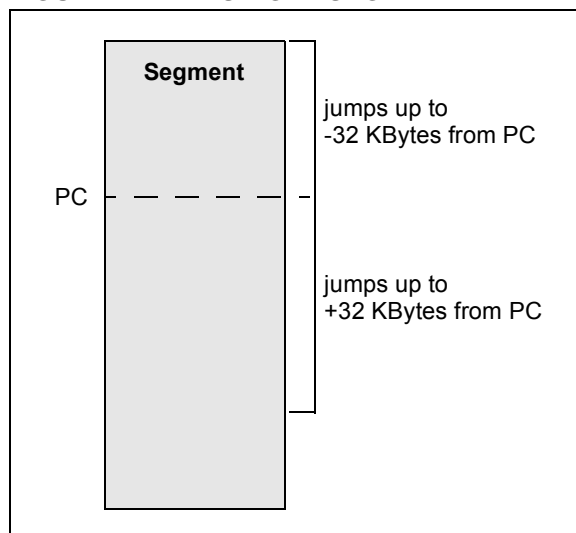
DEPLOYING PALM OS APPLICATIONS

The Palm OS database structure plays an important role in defining the available deployment models for applications. The database storage mechanism constrains individual records to approximately 64 Kbytes, minus some overhead. This constraint requires program code to fit into a single database record. Additionally, all function calls (or jumps) must be plus or minus 32 Kbytes from the current program counter location.

Palm OS applications are typically written as monolithic, single-segment programs. These applications are known as single-segment because they are contained within the 64 Kbyte single-record-size limit. As applications grow in code size, these two restrictions demand a solution. An alternative to the single-segment application is the multi-segment application. A multi-segment application has code spanning multiple database records. Development tools resolve function calls across code segments. However, some functions are now further than 32 Kbytes apart in code space. A technique that helps the linker resolve these “far” function references calls for placing commonly used code in centrally located segments, with the intent of making those functions fall within 32 Kbyte of any other locations in the application. (See [Figure 4](#)).

Another option is to “in-line” functions. This is the practice of copying the code into place as opposed to “calling” on it from another location. In-line functions resolve the linking problem at the expense of creating a larger application. When the application grows in size, the linking problems may be exacerbated. These code-placement techniques work in many instances, yet some applications require even more flexibility. The solution comes in the form of a code library.

FIGURE 4: SINGLE SEGMENT



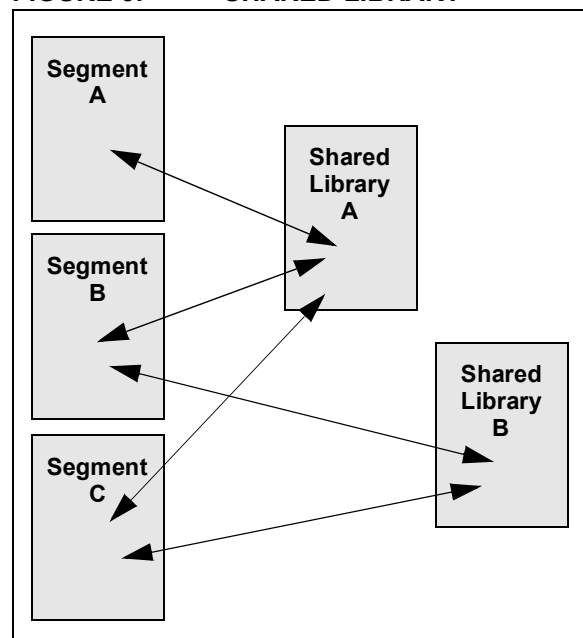
SHARED LIBRARIES

It is generally accepted as good practice to break code into reusable pieces. The most elegant way of performing this is in the form of a code library, ideally one loadable at run-time. Sharing code at run-time allows many applications to utilize the same functions without the overhead of storing multiple copies of the same code on the device. This conservation of space is crucial in a limited resource environment such as the Palm OS. These loadable, reusable units are called “Shared Libraries” on the Palm OS.

A shared library is a single-segment code resource that is post-linked into a PRC file. A shared library is installed in the same manner as traditional applications. It does not show up in the list of applications, but rather is available for use by Palm OS applications. An application loads the shared library on demand. The Palm OS ships with many shared libraries, with the most popular being used for communications functions, such as Infrared (IrLibrary) and Networking (Net.lib). Many third-party products are also packaged as shared libraries (see [Figure 5](#)).

The shared library is an attractive mechanism for deploying commonly used functionality. However, it does have two significant drawbacks. The first limitation is code size. A shared library is constrained to a single segment. A more annoying feature of shared libraries is the lack of global variables. A shared library is forbidden to utilize global variables or statically initialized variables, which become global variables at link time. There are techniques for providing global-like structures to cope with this limitation. For additional information on shared libraries, see the “[Resources](#)” section.

FIGURE 5: SHARED LIBRARY



PALM OS COMMUNICATIONS CAPABILITIES

A BRIEF INTRODUCTION

The Palm OS boasts an impressive array of communications options. The available features vary according to the manufacturer and model of the target device. For example, some devices have wireless networking capabilities, while others have a Universal Serial Bus (USB) connector instead of the traditional serial (RS-232) connector.

Infrared communications hardware is common on virtually all Palm OS devices, with later-model devices having enhanced IrDA standard software capabilities. The Palm VII model contains a radio modem providing connectivity to Palm's proprietary network, Palm.net[®]. Other Palm OS devices gain access to the Internet via a variety of commercially available modems. The area of communications is always in transition as new technologies arise. For example, Bluetooth[™] and Wi-Fi (802.11b, wireless Ethernet) technologies are finding their way into new devices (such as the Palm Tungsten[™] C).

Note 1: As with any O.S., application program compatibility issues can arise between different versions.. So a program that functions on V4.x of the O.S. may be required to be "tuned" for V5.x of the O.S.

2: Due to the number of Palm PDA processor manufacturers, hardware dependencies may be encountered. This likelihood increases as the application software "gets closer" to the hardware. This is more likely when doing "Raw IR" communication.

INFRARED COMMUNICATIONS

The application built and discussed in this application note uses a high-level, infrared protocol called IrCOMM. This protocol is designed as a wire-replacement technology. Infrared technology is very compelling for data collection for many reasons, including:

- **Availability:** Virtually every later model PDA and laptop contains an IrDA port.
- **Cost:** IrDA communications may be added to a custom design very economically, as demonstrated in this application note.
- **Convenience and Compatibility:** Working without wires means no cables, gender-changers, or any other gadgets to make two devices communicate. This is vital to the frequent traveler or technician in the field.

For more information regarding the IrCOMM protocol, visit the IrDA organization web site at:

<http://www.irda.org>

PALM OS COMMUNICATIONS LINE-UP

Serial/USB

Every Palm OS device is equipped with either a Universal Serial Bus (USB) or RS-232 UART connection. The most common example of this communication capability is during the HotSync process, which is when the Palm OS device synchronizes its contents with the host system.

APIs are available for Serial Communications, however USB interaction is limited to the HotSync process; it is not available for custom Palm applications.

Infrared Library

Programming the infrared communications port of a Palm OS device may be accomplished in a number of different ways. The SDK provides a fairly rich set of API functions for performing activities such as device discovery, packet sending, and registering callback functions (useful for notification when certain events have taken place).

Programming the IrLibrary is arguably the most complex manner by which to communicate via infrared on the Palm. This application note does not venture into the specifics of IrLibrary programming, but other resources are available on this topic. See the **“Resources”** section for more information regarding IrLibrary programming examples and applications.

IrCOMM

As mentioned earlier, IrCOMM is a wire-replacement protocol, sitting near the top of the IrDA protocol stack. The most convenient manner with which to leverage the IrCOMM protocol is through the use of the Palm OS Serial Manager APIs. Communicating with an IrCOMM device (such as the Microchip MCP215X series) requires a special identifier to the SrmOpen() function. This will be discussed in more detail later in the application note.

Object Exchange (OBEX)

The most common use of Infrared communications on the Palm OS platform is “beaming”. Users share information, such as Address records or applications, via beaming. The underlying protocol in action is known as Object Exchange, more commonly known as OBEX. OBEX is similar to HTTP in that it shares “typed” information. The Palm OS Exchange Manager APIs provide access to these functions. This application note does not discuss Exchange Manager programming. For additional information on Exchange Manager, refer to the **“Recommended Reading”** section of this application note.

BlueTooth

BlueTooth is a limited-range radio frequency technology, sharing the same frequency spectrum as 802.11b (Wi-Fi technology). BlueTooth technology is finding its way into the Palm OS arena in the form of add-ons by releasing BlueTooth cards from Palm Computing® that fit into the expansion slot found on the latest Palm devices, as well as integrated into the PDA itself (such as the Palm Tungsten T2 model). Other manufacturers provide various forms of “sleds”, or dongles, boasting BlueTooth connectivity.

BlueTooth programming may be accomplished through a rich API set, known as BtLibrary. BtLibrary may be likened to the IrLibrary, providing a rich set of functions and mechanisms for discovering devices and interacting with them. For more straight-forward, wire-replacement applications, the Palm OS Serial Manager API provides a means for communicating with another BlueTooth device when the device is operating in the BlueTooth Serial Port Profile (SPP). Wire-replacement is but one of many “profiles” defined in the BlueTooth specification. This application note does not demonstrate BlueTooth development, though more information may be found in the **“Resources”** section.

Network Programming

The Palm OS provides a rich set of network programming interfaces, roughly equivalent in function to the Berkeley Sockets API. Both TCP and UDP connections may be established over a variety of underlying transports. A network connection may be established via a traditional dial-up modem, a CDPD radio modem, an “Ethernet cradle”, a BlueTooth adapter in its LAN profile and more. Networking APIs are found in the Net.lib shared library.

<p>Note: The Palm OS does not provide a Windows networking client. More information on network programming may be found in the “Resources” section.</p>

The topic of communications programming for the Palm OS is vast and beyond the scope of this application note. For more information on this topic, please see the **“Resources”** section at the end of this application note.

PALM OS SDK & APPLICATION ARCHITECTURE BASICS

This section explores the Palm OS SDK, including API functions, data types and how they work together in a typical Palm OS application. The basic structure of a Palm OS application is also examined. An understanding of these topics is a key ingredient to developing applications for the Palm OS platform.

API Functions

It is crucial to understand the facilities provided by the SDK as they are the lifeblood of Palm OS development. Documentation for Palm OS has traditionally been supplied in the form of Acrobat® PDF files. However, some development tools, such as DeveloperStudio by Falch.net, makes the information available via context-sensitive help. Open a source file created from the new Palm OS Framework Project wizard and scan the various function calls. Highlight any function and press the F1 key for help.

Syntax, style and data types require some attention. SDK functions are organized into categories based on their role. All functions of the same category have the same two or three-letter prefix. Table 1 shows some examples.

TABLE 1: SDK FUNCTIONS

Prefix	SDK Category
Frm	Form manipulation functions
Fld	Field manipulation functions
Net	Networking functions
Exg	Exchange Manager functions, used for Infrared Beaming
Srm	Serial Manager functions
Ir	Infrared Library functions

EXAMPLE 2: DATA TYPE SYNTAX

Line #	
001	typedef struct
002	{
003	UInt8 attributes; // record attributes;
004	UInt8 uniqueID[3]; // unique ID of record
005	} SortRecordInfoType;

EXAMPLE 3: POINTER DATA TYPE AND VARIABLE SYNTAX

Line #	
001	typedef SortRecordInfoType *SortRecordInfoPtr;
002	
003	SortRecordInfoPtr sortinfoP;

Data Types

Data types fall into two categories:

1. Scalar.
2. Composite.

Scalar data types are variables that are not broken down into sub fields. Table 2 shows some Palm OS scalar data types and their equivalent data type in other environments.

TABLE 2: DATA TYPES

Palm OS® Scalar Data Type	Equivalent Data Type	Format
UInt8	byte	unsigned 8 bits
Char	char	signed 8 bits
UInt16	word	unsigned 16 bits
Int16	short	signed 16 bits
UInt32	dword	unsigned 32 bits
Int32	long	signed 32 bits
UInt8*	unsigned char *, or byte *	

The other category is the composite, or user-defined, data type. The SortRecordInfoType (shown in [Example 2](#)) is an example of this kind of data type. It uses other data types to construct the composite representation required. This example was taken from the DataMgr.h header file.

A pointer data type is designated with a 'Ptr' at the end, as shown in line #1 of [Example 3](#).

Pointer variables are denoted with a capital 'P' at the end. In line #3 of [Example 3](#), the syntax for defining a pointer to a SortRecordInfoType is shown.

“C” Runtime Functions

On most platforms, it is common to utilize standard C runtime functions, such as string manipulation routines like `sprintf()` or `strcpy()`. Palm OS provides its own version of these and other “runtime” functions. String manipulation functions all start with 'Str' and can be found in the documentation.

- Note 1:** Do not attempt to use the standard C “runtime” routines.
- 2:** The `StrPrintf` function is unforgiving with regard to improper data type sizes in the format string.

Event-Driven System

The Palm OS is an event-based operating system. Typical events are pen taps, button presses, menu selections and so forth. The operating system and application work in tandem to service the event queue. The operating system enqueues events as they occur. The primary responsibility of any Palm OS application is to service the event queue. Both the operating system and application may service the queue. Both may add events, and both may remove events.

The operating system supports a single user-interface application. Multitasking is not available for custom applications. There are limited 'background tasks' in the operating system, though these are minimal and deal with low-level communications drivers. These drivers enable modem or network access for the device. A network connection takes place at a lower level than the user application, though the user application often initiates and terminates the underlying connection. It is perfectly feasible (and common) for one application to establish a connection and then terminate itself. This allows another application, the active user interface, to make use of the network connection, if so desired. This is the manner in which the Network Preferences panel operates. The operating system is responsible for managing the low-level communications driver and the single user-interface task. Custom applications, such as those built in this application note, have complete control of the device. The custom application, though, must be mindful of the underlying OS's need for processor cycles to handle and respond to interrupts sufficiently.

Starting the Application

There are multiple ways to start an application. It may be initiated from the main application launcher interface as well as from another application. Launch codes are arguments passed to the application at start-up. These launch codes permit the application to make appropriate decisions on how to behave and operate. For example, with a normal launch, the application displays its full user interface. A 'Find' launch code causes the program to search its database, but not necessarily display the normal user interface.

There are three topics fundamental to a Palm OS application. These are:

1. Launch Codes
2. Main EventLoop
3. Event Handlers

The next few sections walk through code examples of “boiler plate” Palm OS applications.

PilotMain

PilotMain is the entry point for a Palm OS application. PilotMain performs the same role as the main() function in other C language environments. The arguments to PilotMain determine the role the application is to perform. [Example 5](#) shows the PilotMain code.

The cmd parameter represents the launch code (see Line #1 of code). This code may request a normal launch, as this example demonstrates, or it may be one of a finite list of values. These values include launch codes to inform applications that the HotSync operation is complete or that the date or time has been changed. The SDK documentation enumerates all available launch codes.

The parameter cmdPBP (see Line #1 of code) may point to a structure required for a particular launch code. For example, the launch code sysAppLaunchCmdGoto is used to instruct an application to initiate its user interface and display a particular record. The structure accessible via the cmdPBP parameter contains the necessary information to perform the action of displaying a specific database record. [Example 4](#) shows what the structure looks like.

The launchFlags parameter (see Line #1 of code) allows further refinement of how the application operates. One interesting flag is sysAppLaunchFlagSubCall. This value allows the application to call its own PilotMain function again and again.

Under a normal launch, the application performs any initialization required and then enters its main EventLoop (see Line #14 of code).

EXAMPLE 4: STRUCTURE

Line #	
001	typedef struct
002	{
003	Int16 searchStrLen;
004	UInt16 dbCardNo;
005	LocalID dbID;
006	UInt16 recordNum;
007	UInt16 matchPos;
008	UInt16 matchFieldNum;
009	UInt32 matchCustom;
010	} GoToParamsType;

EXAMPLE 5: PILOTMMAIN

Line #	
001	UInt32 PilotMain(UInt16 cmd, void *cmdPBP, UInt16 launchFlags)
002	{
003	Err error;
004	
005	switch (cmd)
006	{
007	case sysAppLaunchCmdNormalLaunch:
008	// Application start code
009	error = StartApplication();
010	if (error)
011	return error;
012	
013	// Maintain event loop
014	EventLoop();
015	
016	// Stop application
017	StopApplication();
018	break;
019	default:
020	break;
021	
022	}
023	
024	return 0;
025	}

EventLoop & Event Handlers

A Palm OS application's primary activity is to service events received from the operating system. This process is accomplished via a function named `EventLoop` (see Line #1 of code). A typical `EventLoop` function is presented in [Example 6](#).

The `EventLoop` continually looks for new events by calling the `EvtGetEvent` function (see Line #10 of code). Once an event has been received, the `EventLoop` passes the event to a series of event handlers (see Lines #12 - #20 of code). There is a priority to the event handlers, demonstrated in the calling sequence as events "trickle down" through various

handlers. The `EventLoop` function continues until an `appStopEvent` is received (see Line #22 of code), which signals the Event Loop to terminate.

The operating system itself is given the first opportunity to process an event. The operating system services events, such as a hard key press. If the operating system is not interested in the event, the event is passed on to the `MenuHandleEvent` handler (see Line #15 of code). The `MenuHandleEvent` function is responsible for tracking menu interaction. If both the Sys and Menu handlers decline to process an event, the event is passed to application-defined functions.

EXAMPLE 6: EVENTLOOP

Line #	
001	<code>static void EventLoop(void)</code>
002	<code>{</code>
003	<code> Err error;</code>
004	<code> EventType event;</code>
005	
006	<code> // Main event loop</code>
007	<code> do</code>
008	<code> {</code>
009	<code> // Get next event</code>
010	<code> EvtGetEvent(&event, evtWaitForever);</code>
011	
012	<code> // Handle event</code>
013	<code> if (!SysHandleEvent(&event))</code>
014	<code> {</code>
015	<code> if (!MenuHandleEvent(0, &event, &error))</code>
016	<code> {</code>
017	<code> if (!ApplicationHandleEvent(&event))</code>
018	<code> FrmDispatchEvent(&event);</code>
019	<code> }</code>
020	<code> }</code>
021	<code> }</code>
022	<code> while (event.eType != appStopEvent);</code>
023	<code> }</code>

AN888

Example 7 shows the ApplicationHandleEvent function (see Line #1 of code). This function generally deals with only one event, the FrmLoadEvent (see Line #10 of code).

EXAMPLE 7: APPLICATION HANDLE EVENT

Line #	
001	static Boolean ApplicationHandleEvent(EventPtr event)
002	{
003	UInt16 formID;
004	FormPtr form;
005	Boolean handled = false;
006	
007	// Application event loop
008	switch (event->eType)
009	{
010	case frmLoadEvent:
011	// Handle form load events
012	formID = event->data.frmLoad.formID;
013	form = FrmInitForm(formID);
014	FrmSetActiveForm(form);
015	
016	switch (formID)
017	{
018	case frmMain:
019	// Set event handler for frmMain
020	FrmSetEventHandler(form,
021	(FormEventHandlerPtr) FrmMain_HandleEvent);
022	break;
023	default:
024	break;
025	}
026	handled = true;
027	break;
028	default:
029	break;
030	}
031	
032	return handled;
033	}

Form Event Handlers - An Introduction To User Interface Events

As an event trickles down through the series of event handlers without being serviced, it will eventually fall to the current form event handler. This function is application-defined to service relevant events, such as menu events, button selects, pop-up list selections and so on. In short, the form event handler deals with user-interface events, the point at which most substantive Palm OS development starts. The other functions are more or less boiler-plate code that is copied from project to project.

Each event carries with it specific information pertinent to that event. For example, a button selection event contains the identifier for the button, while a pen tap contains the screen coordinates of the pen at the time the tap was recorded. The following code snippet demonstrates the handling of a button press. This application's user interface contains a single button labeled 'Hit Me'.

[Example 8](#) shows how the `frmMain_HandleEvent` initializes the form and assigns the form's event handler. 'Cases' are added for each additional form in order to assign the appropriate event handler function.

EXAMPLE 8: FORM EVENT HANDLER

Line #	
001	Boolean frmMain_HandleEvent(EventPtr event)
002	{
003	FormPtr form;
004	Boolean handled = false;
005	
006	switch (event->eType)
007	{
008	case ctlSelectEvent:
009	switch (event->data.ctlSelect.controlID)
010	{
011	// HitMe receives an event
012	case HitMe:
013	handled = frmMain_HitMe_OnSelect(event);
014	break;
015	}
016	break;
017	case frmOpenEvent:
018	// Repaint form on open
019	form = FrmGetActiveForm();
020	FrmDrawForm(form);
021	handled = true;
022	break;
023	default:
024	break;
025	}
026	
027	return handled;
028	}

AN888

Code Organization

When an event occurs in the UI, such as a button being tapped on the device, a `ctlSelectEvent` makes its way to the form event handler. Referring to the code sample above, the event handler examines the `controlID` of the event and dispatches it accordingly to another function named `frmMain_HitMe_OnSelect`, listed in [Example 9](#).

This function simply displays a message dialog with the words "Hi there".

It is not necessary to break out each and every control or action to a separate function. This is a matter of organization, ultimately driven by programming style and readability.

EXAMPLE 9: DATA TYPE

Line #	
001	<code>static Boolean frmMain_HitMe_OnSelect(EventPtr event)</code>
002	<code>{</code>
003	<code> // Insert code for HitMe</code>
004	<code> FrmCustomAlert(Info, "Hi there", NULL, NULL);</code>
005	
006	<code> return true;</code>
007	<code>}</code>

PALM OS SERIAL MANAGER PROGRAMMING

This section will discuss the Serial Manager API functions and the parameter that allows the function to communicate using the IrDA standard.

Serial Manager - the Universal Communications API

The Palm OS exposes a versatile communications programming interface known as the Serial Manager. The Serial Manager permits a developer to interact with an arbitrary stream-based communications resource via a common API library, regardless of the characteristics of the underlying resource. For example, opening a physical serial port, a “raw” infrared port, an IrComm resource or Bluetooth resource may be accomplished with the same Serial Manager function call, as seen in [Example 10](#).

The <port id> is a 32-bit value identifying the specific port (see Line #5 of code).

SrmOpen Details

The first argument of the SrmOpen function (see Line #1 of code) indicates the desired communications resource. The value may be either a logical port number or the name of a particular port. Common values are:

- 0x8000 - logical value for the RS-232 serial port
- 0x8001 - logical value for the IR Port
- 'ircm' - IrCOMM virtual port name
- 'rfcm' - Bluetooth serial port profile

The second argument of the SrmOpen function indicates the baud rate desired. In [Example 9](#), this value is set to 9600. In standard serial communications, it is common to change the speed and other port settings after a successful SrmOpen call. When IrCOMM is selected as the port, the baud rate will automatically start at 9600 baud for the negotiation process. After negotiation, the interface will operate at the negotiated baud rate (which will be limited to a maximum by the baud parameter of the SrmOpen function).

The Serial Manager function SrmControl is used for controlling port characteristics, such as baud rate and flow control settings. The functions SrmGetStatus and SrmGetDeviceInfo allow querying of current port settings.

The third, and final, argument of the SrmOpen function is the address of a variable of type UInt16. Upon a successful open, this variable is populated with a value identifying the open communications resource, or simply, the port identifier. All subsequent invocations of Serial Manager functions require this value as the first parameter.

The return value from the SrmOpen function is the standard Palm OS Err data type. A return value of zero indicates success. Any other value represents an error condition. The Palm OS SDK documentation enumerates the possible error conditions for the SrmOpen function.

EXAMPLE 10: SrmOpen

Line #	
001	Err err = 0;
002	UInt16 port = 0;
003	Char buf[100];
004	
005	err = SrmOpen(<port id>,9600,&port);
006	if (err)
007	{
008	StrPrintf(buf,"Unable to open port, error is [%d]",err);
009	FrmCustomAlert(Info,buf,NULL,NULL);
010	}
011	// successful open

Sending/Transmitting Data

Once the communications resource is open and the communications characteristics are set to their desired state, the transfer of information may commence. The function below demonstrates data transmission via the Serial Manager function `SrmSend`.

[Example 11](#) shows how the function attempts to send the entire string represented by the single argument to the function `dataP`. If an error occurs during the attempt, an Alert notifies the user. Notice that this function expects a null-terminated string. However, the `SrmSend` function can accept binary data as well.

The `SrmSend` function (see Line #8 of code) requires a valid port identifier as the first argument. The additional arguments to the function are the base address of a memory buffer containing the data to be sent, the number of bytes to send and the address of a variable to record any errors that may occur during the send attempt. The return code of the function indicates the number of bytes actually sent to the port.

EXAMPLE 11: SENDING DATA

Line #	
001	<code>void SendData(Char * dataP)</code>
002	<code>{</code>
003	<code> Err err;</code>
004	<code> UInt32 bytestosend = StrLen(dataP);</code>
005	<code> UInt32 bytessent = 0;</code>
006	<code> Char buf[100];</code>
007	
008	<code> bytessent = SrmSend(port,(const void *)dataP,bytestosend,&err);</code>
009	<code> if (bytessent != bytestosend err != 0)</code>
010	<code> {</code>
011	<code> StrPrintf(buf,"Error during send [%ld] out of [%ld] err = [%d]",</code>
012	<code> bytessent,bytestosend,err);</code>
013	<code> FrmCustomAlert(Info,buf,NULL,NULL);</code>
014	<code> }</code>
015	<code>}</code>

Options When Sending

There are many functions available in the Serial Manager concerning the sending of data. The most commonly used function is `SrmSend`, as demonstrated above. When the `SrmSend` function is invoked, the data passes through a FIFO (first in, first out) buffer. All functions have a special purpose in manipulating this FIFO buffer.

The `SrmSendCheck` function obtains the number of bytes remaining to be sent in the FIFO transmit buffer. The first argument is, as usual, the port identifier. The second argument is the address of a `UInt32` variable. This second value is populated with the current depth of the FIFO transmit buffer. A zero return value indicates success, with a non-zero value indicating an error.

The `SrmSendFlush` function flushes the transmit FIFO buffer without sending the contents. This function is commonly used when initializing a communications resource. The port is first opened with `SrmOpen` and then the transmit buffer is flushed to ensure that no stray data finds its way to a target device with a call to `SrmSendFlush`.

The `SrmSendWait` function attempts to send any data remaining in the transmit buffer. It returns when either the buffer has been completely sent, or the elapsed time exceeds the port time-out value, known as `ctsTimeout`. The `ctsTimeout` is set by the `SrmControl` function. For more information regarding these functions, please refer to the Palm OS SDK help, found through the Falch.net DeveloperStudio Help menu or from the Palm OS SDK documents directly.

Reading/Receiving Data

The code listed below is responsible for moving data from a Serial Manager-managed receive FIFO buffer into the application's own buffer. It attempts to read `<bytesavailable>` characters from the resource.

Example 12 shows how this code uses the `SrmReceive` function to read data from the receive FIFO. The arguments to the `SrmReceive` function include the usual port identifier, the starting address of a buffer to store the read data, the number of bytes to read, a time-out value and a variable to record any errors during the read activity. The function returns the number of bytes actually read.

`SrmReceive`'s signature is similar to the `SrmSend` function, except that it includes an additional parameter, namely the time-out. A call to `SrmReceive` returns after reading the requested number of bytes, or the time-out period has elapsed. The time-out is expressed in Palm OS system ticks. The SDK function `SysTicksPerSecond` is used for determining the timing values on a specific device. Using this function to calculate a time-out value, rather than a fixed constant, allows an application to run on any Palm OS device, even if the underlying time granularity changes.

EXAMPLE 12: READING DATA

Line #	
001	<code>void ProcessPort(UInt32 bytesavailable)</code>
002	<code>{</code>
003	<code>UInt32 bytestoread = 0;</code>
004	<code>UInt32 bytesread = 0;</code>
005	<code>Err err;</code>
006	<code>Char buf[100];</code>
007	<code>Char buffer[<sufficient size>];</code>
008	
009	<code>// read data into buffer</code>
010	<code>bytesread = SrmReceive(port,buffer,bytestoread,SysTicksPerSecond(),&err);</code>
011	<code>if (bytesread != bytestoread err != 0)</code>
012	<code>{</code>
013	<code>StrPrintf(buf,"Error during read [%d] out of [%d] err = [%d]",</code>
014	<code>bytesread,bytestoread,err);</code>
015	<code>FrmCustomAlert(Info,buf,NULL,NULL);</code>
016	<code>}</code>
017	<code>}</code>

Options When Receiving

Similar to the Send functions, the Serial Manager also provides functions for manipulating the receive FIFO buffer.

The SrmReceiveCheck function checks for the number of available bytes in the receive FIFO buffer. This function is often used in an application's EventLoop when looking to process data received in an asynchronous fashion. This application note's sample application demonstrates this technique. [Example 13](#) shows the relevant code snippet.

SrmReceiveFlush is a useful function typically used immediately after opening a communications resource. This function removes all data from the receive buffer, minimizing the chances that an application reads stale, unwanted data from the buffer. In addition to the port identifier, this function takes an argument specifying a time-out period. When invoked, the receive buffer is flushed. The function then waits the requested number of system ticks, expecting more data to arrive. If additional data arrives, the newly-arrived data is removed from the receive buffer and the function again waits for the full time-out period. The function will only return when an entire time-out period has elapsed without the receipt of additional data.

The SrmReceiveWait function takes three arguments after the required port identifier, byte count and time-out value arguments are used to direct this function. The function does not return until either the receive FIFO depth reaches the requested byte count or the time-out period expires.

Though it is beyond the scope of this application note, there are additional mechanisms for an application to be notified when data has arrived in a "callback" fashion. For more information regarding this technique, refer to the ["Recommended Reading"](#) section.

EXAMPLE 13: CHECKING FOR DATA

Line #	
001	if (commsactive == true && rawreadmode == true)
002	{
003	SrmReceiveCheck(portid, &bytestoread);
004	if (bytestoread > 0)
005	{
006	ReadIntoBuffer(bytestoread);
007	}
008	}

Closing the Port

When the communications resource is no longer required, it may be closed with a call to the `SrmClose` function. The lone parameter of this function is the port identifier. It is important to close a communications resource when the Palm application terminates. If not closed, the application shall remain open and, effectively, unavailable to the application when it next attempts to open the port without additional complexity in the application. A common technique for avoiding this troublesome situation is to place a call to `SrmClose` at the conclusion of the application's event loop.

Serial Programming and the MCP21XX Family

This application note discusses and demonstrates the steps necessary for communicating with the MCP215X devices. These devices implement the IrCOMM protocol in the Application Layer. This is in the upper echelon of the IrDA stack (see [Figure 2](#)). When interacting with the IrComm protocol, there are two options on the Palm OS platform:

1. To use the Serial Manager functions as described above and demonstrated in the application note's demonstration application. This provides basic connectivity and is appropriate for most applications.
2. To use the IrLibrary directly. This is a much more complex approach, particularly for developers new to the Palm OS platform. However, there is additional control available to the developer at this level. For example, the IrDA standard discovery process is available to the application when it is not available to an application choosing to employ the Serial Manager.

For more information regarding the IrLibrary, please see the [“Resources”](#) section at the end of this application note.

When communicating with the MCP2120 (an UART to IrDA standard encoder/decoder device), the only option available to the Palm OS developer is to use the Serial Manager API. The port identifier for the “raw ir” port is 0x8001. In addition, the `SrmControl` function must be utilized to enable and disable the IR transceiver. [Example 14](#) shows an example of these calls.

It is important to enable and disable the receive capability of the IR transceiver, as it is common for some devices to “receive” that which has been sent. It does not occur on all devices, but an application developer is well advised to perform the enable/disable when appropriate. The Rx capability should be disabled when sending data. Be sure that the application sends all data via a call to `SrmSendWait()` and then re-enables the Rx capability.

EXAMPLE 14: IrDA CONTROL

Line #	
001	<code>UInt16 paramlength;</code>
002	<code>UInt32 flags;</code>
003	
004	<code> // enable ir</code>
005	<code> flags = 0;</code>
006	<code> paramlength = sizeof(flags);</code>
007	<code> err = SrmControl(portid, srmCtlIrDAEnable,</code>
008	<code> (void *) &flags,&paramlength);</code>
009	
010	<code> // enable rx</code>
011	<code> flags = 0;</code>
012	<code> paramlength = sizeof(flags);</code>
013	<code> err = SrmControl(portid, srmCtlRxEnable,</code>
014	<code> (void *) &flags,&paramlength);</code>
015	
016	<code> // disable rx</code>
017	<code> flags = 0;</code>
018	<code> paramlength = sizeof(flags);</code>
019	<code> err = SrmControl(portid, srmCtlRxDisable,</code>
020	<code> (void *) &flags,&paramlength);</code>

Communications Routines

This section attempts to break down the pertinent code snippets found in the comms.c module. This module contains the majority of the logic required for the demonstration application. In addition, the EventLoop function, found in main.c, is examined to learn the modifications necessary for receiving data in an asynchronous fashion. The remaining modules simply provide a framework and boiler plate code for a Palm OS application.

The application makes use of a handful of global variables listed in [Example 15](#).

The commsactive flag guides the behavior of the EventLoop code (see [Example 19](#)).

The portid represents the communications resource opened with the SrmOpen function. All Serial Manager functions use this variable to identify the open port.

The rawreadbuffer accepts data as it is received by the SrmReceive function.

rawreadmode and rawreadsize are helper variables to assist in properly reading data via the IR port.

To establish the connection to the Secondary device, the code in [Example 16](#) is invoked .

This code attempts to open the port with a call to SrmOpen (line # 007) using the 'ircm' parameter, instructing the Palm's Serial Manager to open the communications port and implement the IrComm protocol. The requested speed is 9600 baud.

Line # 016 is a call using the SrmSend function. This call has a payload size of zero (third parameter). Without this line, the link is not actually established (the connection negotiation does not take place). This call to send data with a payload size of zero bytes tricks the stack into negotiating and sending an empty data frame, thereby establishing the link without actually sending application data. Without this approach, the link would not be established until an application sent a packet to the device.

Note 1: The behavior of this "trick" varies from device to device and OS versions. It is advised to thoroughly test this on your target platform/device.

2: This trick appears not to be well behaved in Palm OS V5.2.1.

EXAMPLE 15: GLOBAL VARIABLES

Line #	
001	/* communications global variables */
002	Boolean commsactive = false;
003	UInt16 portid = 0;
004	Boolean rawreadmode = false;
005	UInt8 rawreadbuffer[1024];
006	UInt16 rawreadsize;

EXAMPLE 16: OPENING THE PORT

Line #	
001	Boolean OpenPort()
002	{
003	Err e;
004	char buf[100];
005	
006	// attempt to connect
007	e = SrmOpen('ircm',9600,&portid);
008	if (e)
009	{
010	StrPrintf(buf,"Failed to open port [%d]",e);
011	FrmCustomAlert(ErrorAlert,buf,NULL,NULL);
012	return false;
013	}
014	// now set any port parameters desired
015	// push out a zero sized packet to bring up the interface
016	SrmSend(portid,buf,0,&e);
017	commsactive = true;
018	
019	return true;
020	}

Transmitting a Byte of Data

The code shown in [Example 17](#) will transmit a byte of data on the IR port (once the port has been successfully opened).

This code sends a single byte of value 5 (or hex 0x35) to the IR Demo board (line # 014 and line # 015). Notice the use of the `SrmSendFlush` (line # 009), `SrmReceiveFlush` (line # 011) and `SrmSendWait` (line # 020) routines. These routines take measures to clean out buffers prior to initiating this “transaction”. Once the data has been sent, the `QueryVendingMachineResponse()` function (line # 021) is invoked to process the results. Here is a description of the code, as it is too lengthy to present in full here.

The `QueryVendingMachineResponse` checks for data availability in the receive buffer while in a main loop. This loop will run for no more than 5 seconds, or until a complete message has been received from the Secondary device. As data is received, it is parsed, with the relevant information being extracted from the application message received from the Secondary device. Any data received is then displayed on the screen of the Palm device.

EXAMPLE 17: QUERY IR DEMO BOARD

Line #	
001	<code>void QueryVendingMachine()</code>
002	<code>{</code>
003	<code>Err e;</code>
004	<code>UInt8 b;</code>
005	
006	
007	<code>// clear out send and receive buffers</code>
008	
009	<code>SrmSendFlush(portid);</code>
010	
011	<code>SrmReceiveFlush(portid, SysTicksPerSecond()*1.5);</code>
012	
013	<code>// send query to the device ... ascii 5 or hex 0x35</code>
014	<code>b = '5';</code>
015	<code>if (SrmSend(portid, (void *) &b, 1, &e) != 1)</code>
016	<code>{</code>
017	<code> FrmCustomAlert(ErrorAlert, "Failed To Query Demo Board", NULL, NULL);</code>
018	<code> return;</code>
019	<code>}</code>
020	<code>SrmSendWait(portid);</code>
021	<code>QueryVendingMachineResponse();</code>
022	<code>}</code>

Data Receive Window

When the Palm application is expected to receive a large amount of data from the Secondary Device, the `ReadIntoBuffer()` function (shown in [Example 18](#)) can be used. This routine is invoked from the `EventLoop` when data is available in the receive FIFO.

[Example 18](#) shows the relevant code snippet from the `EventLoop`, demonstrating a technique for reading data as it is received.

EXAMPLE 18: READ DATA

Line #	
001	/*
002	read data and then update the trace byte count UI
003	*/
004	void ReadIntoBuffer(UInt32 bytestoread)
005	{
006	UInt32 bytes;
007	FormPtr f = FrmGetActiveForm();
008	ControlPtr cptr =
009	(ControlPtr) FrmGetObjectPtr(f, FrmGetObjectIndex(f, TRACECOUNT));
010	Err e;
011	
012	bytes = SrmReceive(portid, &rawreadbuffer[rawreadsize], min(bytestoread,
013	sizeof(rawreadbuffer) - rawreadsize), SysTicksPerSecond(), &e);
014	rawreadsize += bytes;
015	StrPrintf(tracebuffermsg, "Trace : %d Bytes", rawreadsize);
016	CtlSetLabel(cptr, tracebuffermsg);
017	CtlDrawControl(cptr);
018	FrmDrawForm(f);
019	
020	}

EXAMPLE 19: CHECKING FOR DATA

Line #	
001	Err error;
002	EventType event;
003	UInt32 bytestoread;
004	// Main event loop
005	do
006	{
007	// Get next event
008	EvtGetEvent(&event, 10);
009	
010	if (event.eType == nilEvent)
011	{
012	// check to see if we should service the communications port
013	if (commsactive == true && rawreadmode == true)
014	{
015	SrmReceiveCheck(portid, &bytestoread);
016	if (bytestoread > 0)
017	{
018	ReadIntoBuffer(bytestoread);
019	}
020	}
021	}
022	...
023	}

Overview of Conduits

This application note has focused on the specifics of writing a Palm OS application and, in particular, interacting with the MCP215X demo board. While IrDA standard communications are extremely useful for communicating with a device in the field, a common question arises when determining the best manner to move data to and from the host environment, which is typically a Windows-based computer. The prescribed manner for this activity is a Palm OS desktop conduit. As defined in the introduction of this application note, the conduit acts as a loadable module to the HotSync Manager. The conduit is responsible for intelligently exchanging data between a Palm resident database and the desktop.

There are two typical usage profiles for Palm/MCP215X applications, of which conduits play a role in both:

1. The first, and most common, is the classical data collection scenario where the Palm OS based device is employed to collect data in the field. The application communicates with the device via the infrared port and stores collected information into a Palm resident database. Upon return to the office or lab, the data is moved to the desktop via a Conduit for subsequent recording and/or analysis.
2. The second scenario involves the use of a Palm application for updating firmware and/or configuration information of a device in the field. In this case, the Conduit loads data (which may be a Hex file) onto the Palm device and the Palm-based application is responsible to transmit that information to the device in the field.

Custom conduits must be written to perform these functions. Palm provides a conduit development kit that provides the scaffolding and hooks to create conduits with Microsoft® Visual C++®, Microsoft VB/.Net and Java™. In addition, it is possible to write conduits for the Linux® platform as well.

Example IrDA Standard System

An example system was developed for testing of the Palm application software. This system uses a hardware board to implement the embedded IrDA standard system, a Palm PDA and the Palm Application Program. [Appendix A](#) discusses the system, including the operational aspects that determine the requirements for the Palm Application Program.

[Appendix B](#) through [Appendix G](#) show the actual source code for the Palm Application Program. This program was developed with Falch.net's DeveloperStudio. The Falch.net's DeveloperStudio IDE automatically generates much of the application code (User Interface aspects), leaving only the "meat" of the key tap functions to be defined.

Falch.net's DeveloperStudio was chosen for its clean packaging, compatibility, ease-of-use and comparative low-cost. An attractive feature of DeveloperStudio is its integration with, and use of, the open source PRC-Tools suite. This compatibility permits the use of various GNU tools and utilities, such as *make*. Projects created with DeveloperStudio may be compiled from the command line with *make*. Additionally, source-code management of a PRC/DeveloperStudio project fits nicely with existing tools in many programming shops. These features are helpful to developers who port their products to the Palm OS. Falch.net's context-sensitive help system, which also provides "tool tips" function prototypes, is a nice bonus, particularly for the beginner.

Note 1: Falch.net's DeveloperStudio has a demo version for evaluation purposes.

2: The Palm Application Program in [Appendix B](#) - [Appendix G](#) exceeds the allowances of the Falch.net's DeveloperStudio Demo Version capabilities.

While the Palm Application Program is developed using Falch.net DeveloperStudio, the programming concepts and API calls are identical to CodeWarrior®. The Falch.net DeveloperStudio source code should be easily modified for the CodeWarrior development platform.

Table 3 shows the versions of the different products that were used in the development and validation of the Palm Application Program.

TABLE 3: DEVELOPMENT TOOL VERSIONS

Tool	Version	Comment
Falch.net IDE	V2.7.2.0	
Palm Desktop	V4.1	
Palm m105	V3.5.1	PDA model O.S. Version
Sony PEG-SJ22	V4.1	PDA model O.S. Version
Sony PEG-S360	V4.0	PDA model O.S. Version
Palm® Zire™ 21	V5.2.1	PDA model O.S. Version

Palm Application Code Descriptions

The Palm Application Program called MCP215XDemo is shown in [Appendix B](#) through [Appendix G](#). This program is created with two forms. These forms are:

- The primary user interface form
- An “About” form

There is no requirement that each form have its own C module, though it is common practice to do so.

Table 4 briefly describes the role of each source file and has a link to the Appendix which contains that source file.

The following section describes the operation of some of the code from these source files.

For more information about the operation of the system (Embedded System and Palm Application Program), please refer to [Appendix A](#).

TABLE 4: MCP215XDEMO SOURCE FILES

File Name	Description	Appendix
Main.c	Entry Point of the application, fundamental, boiler-plate code. Contains PilotMain and the EventLoop, among other functions. With the exception of a few variable references and slight modification to the EventLoop function, this source file has not changed from its original state as created by the Falch.net Framework wizard.	Appendix B
frmMain.c	Handles interactions with the primary user interface. There is quite a bit of customizing done in this module. Each user interface element (sans the Menus) is backed by code in this module.	Appendix C
Menu.c	Handles menu selections. This module is very minimal and represents a typical menu-handling module useful for dispatching menu selections. Note the use of a switch statement for the specific menu command invoked.	Appendix D
Comms.c	This module contains communication variables and function usage examples. In addition, this module contains routines for formatting and parsing commands to and from the IR Demo Board. This particular module is explored in more detail in the next section.	Appendix E
FrmAbout.c	Code for the “About” form	Appendix F
MCP2150Demo.h, Comms.h, MCP2150Demo_Res.h	Include Files	Appendix G

Resources

For information on HackMaster, a popular tool for managing multiple “hacks”, or Trap handlers, used to modify normal Palm OS behavior, visit:

<http://www.daggerware.com/hackmstr.htm>

For additional books and tutorials on Palm OS development, visit:

<http://www.palmos.com/dev/support/docs/other.html>

Additional Palm OS tutorials may be found at:

<http://www.palm-communications.com>

For information on Shared Library development, visit:

<http://oasis.palm.com/dev/kb/papers/1670.cfm>

For support with Falch.net's DeveloperStudio, visit:

<http://www.falch.net/Support/>

Recommended Reading

Table 5 gives a list of additional documentation on Palm OS development and Table 6 shows the documentation available from PalmSource™.

TABLE 5: ADDITIONAL PALM OS READING

Title	Author	ISBN
Palm OS Programmer's API Reference	Palm Corporation	0595737110
Palm OS Programmer's Companion, vol. I	Palm Corporation	3004-006-HW *
Palm OS Programmer's Companion, vol. II	Palm Corporation	3005-003-HW *
Palm File Format Specification	Palm Corporation	1400527384
Using Palm OS Emulator	Palm Corporation	0595737153
Palm OS Programming Bible, 2nd Edition	Lonnon R. Foster	0764549618
Palm OS Programming for Dummies	Liz O'Hara and John Schettino	0764505637
Palm OS Programming: The Developer's Guide, 2nd Edition	Neil Rhodes and Julie McKeehan	1565928563
Palm OS Programming from the Ground Up	Robert Mykland	0072222891
Palm Programming (Sam's Professional Series)	Glenn Bachmann	0672314932
Teach Yourself Palm Programming in 24 Hours	Gavin Maxwell	0672316110
Programming Visual Basic for the Palm OS (O'Reilly Palm)	Matthew Holmes, Patrick Burton, and Roger Knoell	0596002009
NS Basic Programming for Palm OS	Michael J. Verive	0969584466
Palm Database Programming	Eric Gigure	0471354015
Official Pendragon Forms for Palm OS Starter Kit	Debra Sancho and Ivan Phillips	0764546511
Advanced Palm Programming: An Expert's Guide...	Steve Mann and Ray Rischpater	0471390879
Palm OS Network Programming	Greg Winton	0596000057

* Available for download at: www.palmos.com/dev/support/docs/

TABLE 6: PALM OS DOCUMENTATION (AVAILABLE AT WWW.PALMSOURCE.COM)

Title	Date	Description
Palm OS Companion and Reference	12/12/2002	Provides extensive conceptual and "how-to" development information in the Companion, and official reference information of Palm OS functions and data structures in the Reference. Includes: <ul style="list-style-type: none"> • Palm OS Programmer's API Reference • Palm OS Programmer's Companion, vol. I • Palm OS Programmer's Companion, vol. II
Palm File Format Specification	5/1/2001	Provides the data layout specifications of installable files (PRC), databases (PDB), and web clipping applications (PQA).
Using Palm OS Emulator		
Creating Content for Web Browser	8/30/2002	Describes how to create small screen content for Palm OS 5 Web Browser.
Constructor for Palm OS	11/14/2002	Describes how to use Constructor for Palm OS to build graphical user interfaces for Palm OS applications.
Front-End Processor Developer's Guide	12/06/2002	Describes how to create a language front-end processor for Palm Powered handhelds and documents how applications can interface with front-end processors.
Integrating Palm OS Applications with Web Browser	8/30/2002	Describes how Palm OS applications can interact with Palm OS 5 Web Browser.
Palm Desktop Extensibility Framework	11/11/2002	Describes how to write software for the Palm Desktop Extensibility Framework. Includes step-by-step descriptions of how to develop add-ins with Visual Basic® and C++, and how to develop extensions with C++.
Palm OS Development Tools Guide	11/14/2002	Describes Palm tools that can be used to develop, test, and debug Palm OS applications: Palm Simulator, Palm Debugger, Palm Reporter, console window and resource overlay tools.
Palm OS 5 ARM Programming	5/31/2002	Describes how to use ARM subroutines to improve the performance of Palm OS 5 applications.
Palm OS User Interface Guidelines	12/11/2002	Describes how to design applications for Palm Powered handhelds so that they conform to the Palm OS user interface guidelines.
Testing with Palm OS Simulator	11/14/2002	Describes how to use Palm OS Simulator to test your Palm OS applications.
Using Palm OS Emulator	12/16/2002	Describes how to use Palm OS Emulator to test your Palm OS applications.
Web Clipping Developer's Guide	5/1/2001	Describes how to develop wireless applications using "lightweight" HTML.
Zen of Palm	12/06/2002	Describes design philosophies and practices for developing Palm OS applications.
Conduit Documentation - Windows	7/1/2002	Provides extensive conceptual and reference information on developing conduits to exchange and synchronize data between a Windows application and a Palm Powered handheld.
Palm OS Programming Recipes	—	Palm OS Programming recipes are technical articles with step-by-step explanations describing how to perform specific tasks related to Palm OS programming. They are different from existing sample code because they have step-by-step explanations about specific subjects.

SUMMARY

This application note has shown some of the fundamental “C” programming concepts and design considerations for the development of Palm OS application programs. Attention was given to the Serial Manager API calls for IrCOMM communications.

Using the source code from the example Palm Application Program should allow you to get your custom application to connect to an embedded IrDA Standard system using either the MCP215X or MCP2140 devices.

Biography

Frank Ableson is a consultant specializing in the development of IrDA application programs for Palm OS, PocketPC OS, Symbian® OS and Windows OS systems. For inquiries into consulting services, please contact Frank via e-mail at mchp@cfgsolutions.com.

APPENDIX A: EXAMPLE IrDA STANDARD SYSTEM DESCRIPTION

A description of the example IrDA Standard system is provided to allow better understanding of the Palm Application Program functions. This Palm OS Application Program communicates with an embedded system to transfer data and control operation/status. The embedded system acts as an IrDA Standard Secondary device. [Figure A-1](#) shows this example IrDA Standard system with a Primary device (Palm PDA) and a Secondary device (embedded system). [Figure A-2](#) shows a detailed block diagram of the embedded system (Secondary device). For additional information on the implementation of an Embedded System, please refer to AN858, "Interfacing the MCP215X to a Host Controller", DS00858.

The embedded system (IR Demo Board 1) uses a 40-pin PICmicro[®] microcontroller and an MCP215X device.

Note: The "IR Demo Board 1", which is used as the embedded system, is not currently available for purchase. This system can be created using a PICDEM[™] 2 Plus demo board and an MCP2150 developer's board (in the MCP2120/MCP2150 Developer's Kit).

FIGURE A-1: PALM[™] PDA - EMBEDDED SYSTEM BLOCK DIAGRAM

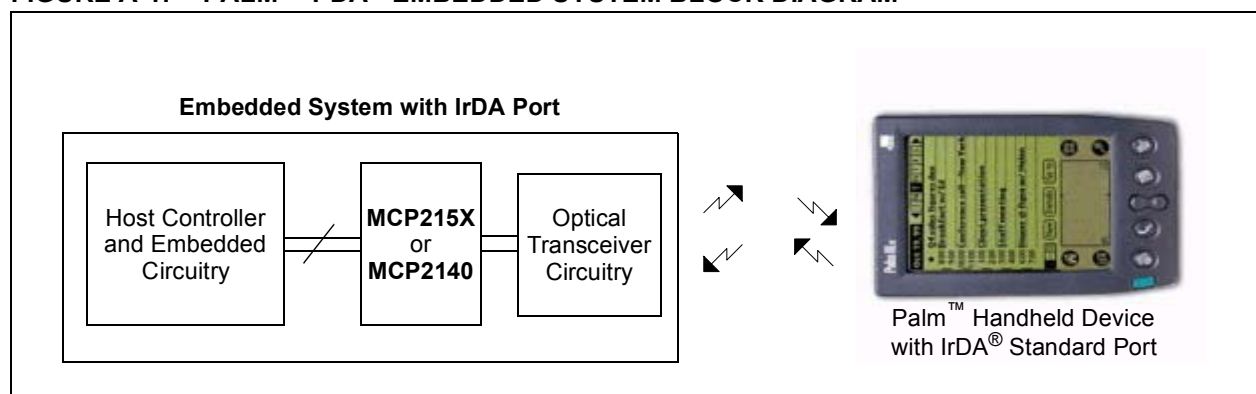
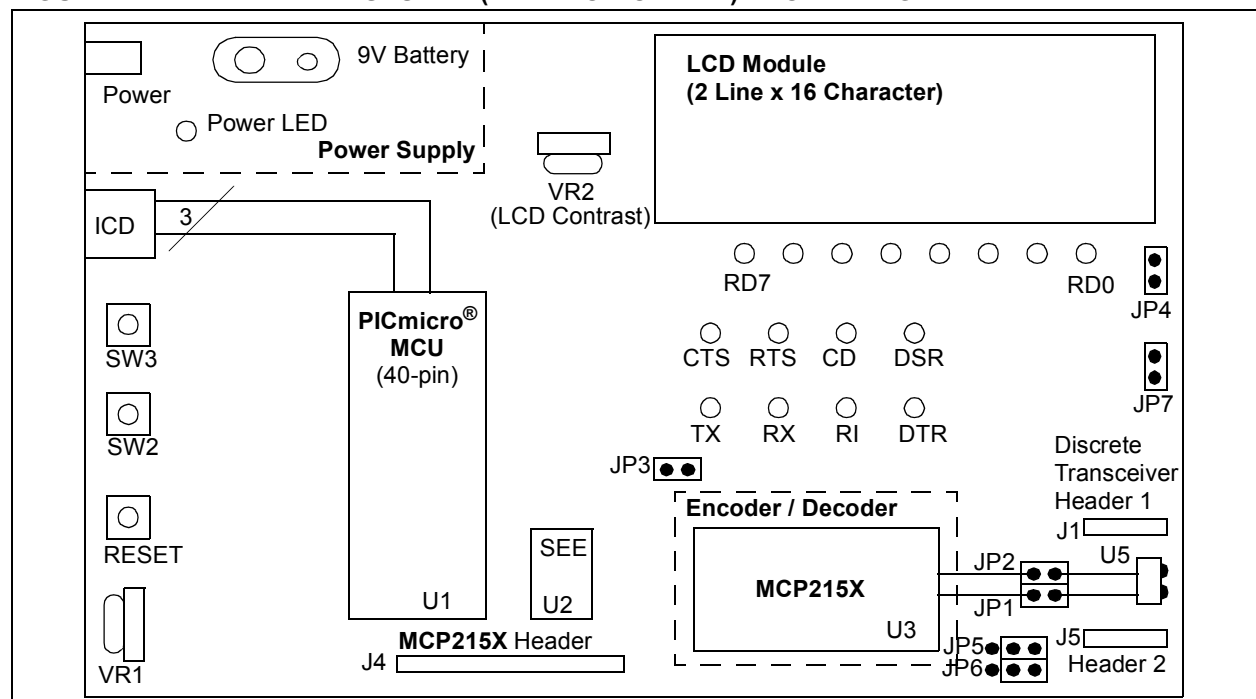


FIGURE A-2: EMBEDDED SYSTEM (IR DEMO BOARD 1) BLOCK DIAGRAM



Embedded System Firmware Operation

The embedded system has two programs that can be selected to run. The first is a “Vending Machine” and the second is a “240 Byte Data Transfer”.

VENDING MACHINE

This demo emulates a “Vending Machine” by counting the number of each item (“Soda” and “Candy”) dispensed.

Each time the SW2 button is depressed, the counter for the number of “Sodas” is incremented. Each time the SW3 button is depressed, the counter for the number of “Candies” is incremented. Each Counter is an 8-bit value and can display a value from 0 to 255 (decimal).

The program monitors for “data” being received from the IR port (received on the Host UART) and will then respond with the appropriate data. Table A-1 shows the two commands of the “Vending Machine” program.

TABLE A-1: “VENDING MACHINE” COMMANDS

Command Value (ASCII)	Hex Value	Demo Program
5	0x35	Transfer the current “Soda” and “Candy” counter values to the Primary device.
6	0x36	Clear the current “Soda” and “Candy” counters.

Note: All other values are ignored

240 BYTE DATA TRANSFER

Depressing SW2 and SW3 will cause the program in the PICmicro microcontroller to execute the “Xfer 240 Bytes” routine. In this demo, the PIC16F877 receives a single byte from the IrDA standard Primary device. This received byte is moved to PORTD (displayed on the LEDs) and then a 240 byte table is transmitted back to the Primary device.

Note: The byte sent by the Primary device is expected, since most PDA’s will not establish a link until data is sent. This application program forces the link open when the “connect” button is depressed by transmitting a null data packet (a packet with 0 data bytes).

Palm Application Program User Interface

In this case, the main User Interface (UI) form (Figure A-3) either displays all the information required, has a button to do the requested action or has a button to display the information (Trace Buffer).

The Connect button causes the application to attempt to connect to the Secondary device. Once this command is completed, the N changes to a Y.

Note: After tapping on the Connect button, the other buttons of the application can be tapped for their desired operation. The CD (DSR) signal will not “turn on” until one of these other buttons is tapped. This modification was done to address Palm OS V5.2.1 operation.

FIGURE A-3: IrDA™ DEMO MAIN FORM



VENDING MACHINE

To interface to the embedded system running the Vending Machine program, the main UI form displays all the information the user needs (Figure A-3).

The Read Data button can then be tapped, causing the “read data command” to be sent to the embedded system. The embedded system will respond with strings that include the following information:

- # of Sodas Sold
- # of Candies Sold
- Change Box (% Full)

If the Clear Data button is tapped, the Clear Data Confirm dialog box will be presented to verify the request. Tapping the Yes button will then send the “clear data command”. A “read data command” is then sent to verify that the embedded system received the “clear data command”.

240 BYTE DATA TRANSFER

To interface to the embedded system running the Vending Machine program, the main UI form displays some of the information the user needs (Figure A-3).

Once the Palm has connected to the embedded system (Secondary device), tap on one of the keyboard buttons (“123” or “ABC”). This will bring up either a numeric or alpha keyboard. Tap on a single character and then tap on the Done button. That character will be displayed on the line titled “TX Data (ASCII)”. To transmit that byte, tap on the Send button.

Notice that the Trace line indicates the number of bytes received (initially it was ‘0’). To view the Trace buffer, tap on the Show button. To clear the trace buffer, tap on the Reset button.

Description of Graphical User Interface (GUI)

The GUI consists of a number of user interface elements, including command buttons, text labels and a text entry field.

- The “Connect” button attempts to establish a connection to the IR demo board. The Palm device is acting as the Primary device and the demo board as the Secondary device. To the Palm Developer, it is a call to SrmOpen with a parameter of 'ircm'. The label to the left of the button provides an indication of connection.
- The “Read Data” button causes a query to be sent to the demo board, requesting the number of sodas, candies and change box information. Data received from the demo board are parsed and displayed in text labels.
- The “Clear Data” button sends a command to the demo board instructing it to reset the application level counters. The command to “Read Data” is then sent to ensure that the registers were cleared.
- The “ASCII/HEX” button toggles the application between ASCII and HEX modes. This value is used when preparing and transmitting data to the IR Demo board. This is useful when there is a need to send a “non-printable” value, such as low-order ASCII. For example, to send the value 0x03, use the keyboard or Graffiti to enter “03” (without the quotes). This will be converted to 0x03 and transmitted.
- The “Send” button initiates the aforementioned sending process. If the “HEX” mode is selected and the string is not properly formatted as two characters per “byte value”, an error message will display. The error checking performed in this application is minimal so be sure to enter the appropriate values.
- The “Send File” button
- The “Show” button causes a message box or Alert in Palm parlance to be displayed. This data may be larger than the viewable area of the Alert. To see all of the data, drag the stylus down the length of the Alert’s window, which will cause it to scroll the additional data into view.
- The “Reset” button clears the Palm application’s receive data buffer. Note that this buffer is different than the operating system buffer used by the SrmReceive functions.

Development Tools

There are a number of options available for developing applications for the Palm OS. Some environments focus on form-based applications, insulating the developer from some of the underlying details of the operating system. "C" language is the most common, and best-supported, language for the platform and is used in this application note.

Two tool options for C language development are generally available:

1. The commercial option is CodeWarrior for Palm OS, produced by Metrowerks®.
2. The open source option is a toolset called "PRC-Tools". PRC Tools includes the open source GNU C compiler (gcc) popular on many other platforms.

In conjunction with free SDK bundles from Palm Computing, both packages provide everything to build applications for Palm OS. The various books available on Palm OS programming cover both of these tools.

A third toolset option available is a hybrid option which is a commercial product available from Falch.net. The Falch.net DeveloperStudio provides a modern GUI interface (for Windows only), yet leverages the PRC tools (open source compiler) under the hood to actually compile and link the application. The graphical form layout and debug environment are very intuitive. Falch.net provides an evaluation license suitable for creating the applications demonstrated in this application note.

The best source for information on Palm OS development tools is the Palm Computing web site:

<http://www.palmos.com/dev>.

Links to all the major development tools, as well as in-depth reviews and comparisons, are available.

BUILD STAGES

Regardless of the C-based toolset developers have chosen, there are three stages of the build process for a Palm OS application or library.

1. The first stage is compilation, where source code is transformed into object code.
2. The next stage is linking, where all function references are resolved.
3. The final stage is known as post-linking. Post-linking combines the executable code and user interface elements into a database suitable for storage on the Palm OS device. Once the post-linking step is complete, the database (application or library) is ready for installation to the device via the HotSync operation.

Development for the Palm OS is cross-platform because the build process takes place on the host system, not on the run-time platform of the Palm OS device itself. The iterative steps of edit, compile, load and debug can be very tedious. The Palm OS Emulator (or POSE) is an invaluable tool for the application developer. POSE runs as a graphical application on the host environment and is capable of running Palm OS applications and libraries. POSE catches errors that may take weeks to find on a real device. It is also capable of testing communications applications by redirecting communications through the host environment's resources, such as a communications port or a network interface.

Other useful tools for this application note include a terminal emulation program, such as the Embedded Companion Suite for Palm OS or HyperTerminal® for Windows® based systems.

Note: This application note assumes that the reader has successfully established the required build environment including either Falch.net DeveloperStudio or CodeWarrior and has also installed a current version of the Palm Desktop Software.

TABLE A-2: PDA DEVELOPMENT TOOL LINKS

Tool	Manufacturer	Web Link	Comment
CodeWarrior	Metrowerks	http://www.metrowerks.com	'C' language
PRC-Tools	OSDN (Open Source Development Network, Inc.), or SourceForge, or VA Software Corporation	http://prc-tools.sourceforge.net/ http://sourceforge.net/projects/prc-tools/	Open source toolset
DeveloperStudio for Palm OS	Falch.net	http://www.Falch.net	Modern GUI interface for PRC-Tools
Palm OS and ROM files	PalmSource	http://www.palmsource.com/	Good source for information

Code Module Description

The MCP215X demo application is created with two forms, the primary user interface and an “About” form. There is no requirement that each form have its own C module, though it is common practice to do so. In this

application, the About form is simply a dialog and therefore has no need of its own C module. In addition to the form handling C module, there are three additional C files in the project. Table A-3 briefly describes the role of each module.

TABLE A-3: PALM APPLICATION PROGRAM FUNCTIONS

File Name	Description	Appendix
Main.c	Entry Point of the application, fundamental, boiler-plate code. Contains PilotMain and the EventLoop, among other functions. With the exception of a few variable references and slight modification to the EventLoop function, this source file has not changed from its original state as created by the Falch.net Framework wizard.	Appendix B
frmMain.c	Handles interactions with the primary user interface. There is quite a bit of customizing done in this module. Each user interface element (sans the Menus) is backed by code in this module.	Appendix C
Menu.c	Handles menu selections. This module is very minimal and represents a typical menu handling module useful for dispatching menu selections. Note the use of a switch statement for the specific menu command invoked.	Appendix D
Comms.c	This module contains communications variables and function usage examples. In addition, this module contains routines for formatting and parsing commands to and from the IR Demo Board. This particular module is broken out in more detail in the next section.	Appendix E
FrmAbout.c	Code for the “About” form	Appendix F
MCP2150Demo.h, Comms.h, MCP2150Demo_Res.h	Include Files	Appendix G

APPENDIX B: PALM SOURCE CODE - MAIN.C

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") is intended and supplied to you, the Company's customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

FIGURE B-1: MAIN.C - PAGE 1

```
/*
 *
 * Created with Falch.net DeveloperStudio
 * http://www.falch.net/
 *
 * File : main.c
 *
 * Description :
 *
 * History:
 * Name          Date          Description
 * ----          -
 * Frank Ableson March 2003    Created
 *
 */

#include <PalmOS.h>

#include "MCP215XDemo.h"
#include "MCP215XDemo_res.h"

extern Boolean commsactive;
extern UInt16 portid;
extern Boolean rawreadmode;

/*
 *
 * FUNCTION:      StartApplication
 *
 * DESCRIPTION:   This routine will launch the application's main form
 *
 * PARAMETERS:   none
 *
 * RETURNED:     returns nothing
 *
 * REVISION HISTORY:
 * Name          Date          Description
 * ----          -
 * Administrator 11/21/2002 11:55:59 PM Created
 *
 */
```

FIGURE B-2: MAIN.C - PAGE 2

```

static int StartApplication(void)
{
    FrmGotoForm(frmMain);
    return 0;
}

/*****
 *
 * FUNCTION:      ApplicationHandleEvent
 *
 * DESCRIPTION:   This routine is called from the event loop, and its
 *                main responsibility is to load forms and set form
 *                eventhandlers.
 *
 * PARAMETERS:   EventPtr  event    Pointer to the event
 *
 * RETURNED:    handled/not handled
 *
 * REVISION HISTORY:
 * Name          Date              Description
 * ----          -
 * Administrator 11/21/2002 11:55:59 PM  Created
 *
 *****/

static Boolean ApplicationHandleEvent(EventPtr event)
{
    UInt16 formID;
    FormPtr form;
    Boolean handled = false;

    // Application event loop
    switch (event->eType)
    {
        case menuEvent:
            // Set event handler for application
            handled = ApplicationHandleMenu(event->data.menu.itemID);
            break;
        case frmLoadEvent:
            // Handle form load events
            formID = event->data.frmLoad.formID;
            form = FrmInitForm(formID);
            FrmSetActiveForm(form);

            switch (formID)
            {
                case frmMain:
                    // Set event handler for frmMain
                    FrmSetEventHandler(form,
                                       (FormEventHandlerPtr) frmMain_HandleEvent);
                    break;
                default:
                    break;
            }
            handled = true;
            break;
        default:
            break;
    }
    return handled;
}

```

FIGURE B-3: MAIN.C - PAGE 3

```

/*****
 *
 * FUNCTION:      EventLoop
 *
 * DESCRIPTION:   The eventloop polls the event que for new events and
 *               dispatches them to the different event handlers.
 *
 * PARAMETERS:   Nothing
 *
 * RETURNED:     Nothing
 *
 * REVISION HISTORY:
 *   Name          Date          Description
 *   ----          -
 *   Administrator 11/21/2002 11:55:59 PM  Created
 *
 *****/

static void EventLoop(void)
{
    Err error;
    EventType event;
    UInt32 bytestoread;

    // Main event loop
    do
    {
        // Get next event
        EvtGetEvent(&event, 10);
        if (event.eType == nilEvent)
        {
            // check to see if we should service the communications port ....
            if (commsactive == true && rawreadmode == true)
            {
                SrmReceiveCheck(portid,&bytestoread);
                if (bytestoread > 0)
                {
                    ReadIntoBuffer(bytestoread);
                }
            }
        }
        // Handle event
        if (!SysHandleEvent(&event))
        {
            if (!MenuHandleEvent(0, &event, &error))
            {
                if (!ApplicationHandleEvent(&event))
                    FrmDispatchEvent(&event);
            }
        }
    }
    while (event.eType != appStopEvent);
    if (commsactive) ClosePort();
}

```

FIGURE B-4: MAIN.C - PAGE 4

```
/*
 *
 * FUNCTION:      StopApplication
 *
 * DESCRIPTION:   this routine closes all open forms by calling
 *               FrmCloseAllForms()
 *
 * PARAMETERS:   Nothing
 *
 * RETURNED:     Nothing
 *
 * REVISION HISTORY:
 *   Name          Date              Description
 *   ----          -
 *   Administrator 11/21/2002 11:55:59 PM  Created
 *
 */
static void StopApplication(void)
{
    // Insert stop code here
    FrmCloseAllForms();
}
```

FIGURE B-5: MAIN.C - PAGE 5

```
/*
 *
 * FUNCTION:      PilotMain
 *
 * DESCRIPTION:   Main entrypoint for the application
 *
 * PARAMETERS:   cmd           Application Launchcode
 *               cmbBBP       Pointer to a structure that is
 *                             associated with the launch code.
 *               launchFlags   value providing extra information about
 *                             the launch
 *
 * RETURNED:     Result of the launch
 *
 * REVISION HISTORY:
 *   Name          Date          Description
 *   ----          -
 *   Administrator 11/21/2002 11:55:59 PM Created
 *
 */
*****/

UInt32 PilotMain(UInt16 cmd, void *cmdBBP, UInt16 launchFlags)
{
    Err error;
    switch (cmd)
    {
        case sysAppLaunchCmdNormalLaunch:
            // Application start code
            error = StartApplication();
            if (error)
                return error;

            // Maintain event loop
            EventLoop();

            // Stop application
            StopApplication();
            break;
        default:
            break;
    }
    return 0;
}
```

APPENDIX C: PALM SOURCE CODE - FRMMAIN.C

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") is intended and supplied to you, the Company's customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

FIGURE C-1: FRMMAIN.C - PAGE 1

```

/*****
 *
 * Created with Falch.net DeveloperStudio
 * http://www.falch.net/
 *
 * File : frmMain.c
 *
 * Description :
 *
 * History:
 *   Name           Date           Description
 *   ----           -
 *   Frank Ableson  March 2003      Created
 *
 *****/

#include <PalmOS.h>
#include "MCP215XDemo.h"
#include "MCP215XDemo_res.h"

#include "comms.h"

extern Boolean commsactive;
extern UInt8 rawreadbuffer[1024];
extern UInt16 rawreadsize;
extern Boolean rawreadmode;
char tracebuffermsg[100];

static UInt8 AsciiHexMode = 0; // 0 is ascii, 1 is hex
static Char * AsciiHexModeStrings[] = {"ASCII", "Hex  "};

static Char * YorN[] = {"Y  ", "N  "};
static Char * ConnectDisconnect[] = {"Disconnect", "Connect"};

```

FIGURE C-2: FRMMAIN.C - PAGE 2

```
UInt8 hexdecode(UInt8 c)
{
    switch (c)
    {
        case 'a':
        case 'b':
        case 'c':
        case 'd':
        case 'e':
        case 'f':
            return (UInt8) (c - 'a' + 10);
        case 'A':
        case 'B':
        case 'C':
        case 'D':
        case 'E':
        case 'F':
            return (UInt8) (c - 'A' + 10);
        case '0':
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':
        case '6':
        case '7':
        case '8':
        case '9':
            return (UInt8)(c - 0x30);
    }
    return 0;
}
```


FIGURE C-3: FRMMAIN.C - PAGE 3

```
static Boolean frmMain_CONNECT_OnSelect(EventPtr event)
{
  Err e = 0;
  char buf[100];
  FrmPtr f = FrmGetActiveForm();
  ControlPtr cptr = (ControlPtr)FrmGetObjectPtr(f, FrmGetObjectIndex(f, AREWECONNECTED));
  ControlPtr cptr2 = (ControlPtr)FrmGetObjectPtr(f, FrmGetObjectIndex(f, CONNECT));

  // Insert code for CONNECT/DISCONNECT
  if (commsactive == false)
  {
    if (OpenPort())
    {
      //FrmCustomAlert(InfoAlert, CtlGetLabel(cptr), NULL, NULL);
      // tell the user that we think we have a connection ...
      CtlSetLabel(cptr, YorN[0]);
      CtlSetLabel(cptr2, ConnectDisconnect[0]);
      //CtlDrawControl(cptr);
      FrmDrawForm(f);
      // toggle the button display to say "Disconnect"
    }
  }
  else
  {
    // FrmCustomAlert(InfoAlert, "Disc", NULL, NULL);
    // FrmCustomAlert(InfoAlert, CtlGetLabel(cptr), NULL, NULL);
    // disconnect ....
    ClosePort();
    CtlSetLabel(cptr, YorN[1]);
    //CtlDrawControl(cptr);
    CtlSetLabel(cptr2, ConnectDisconnect[1]);
    FrmDrawForm(f);
  }
  return true;
}
```

FIGURE C-4: FRMMAIN.C - PAGE 4

```
static Boolean frmMain_SEND_OnSelect(EventPtr event)
{
FormPtr f = FrmGetActiveForm();
FieldPtr pfield = FrmGetObjectPtr(f,FrmGetObjectIndex(f,TXDATA));
MemHandle h;
char *p;
Int16 i,j,k;
char c;
    // Insert code for SEND
    if (!commsactive) return true;
    // get the text in the field and send it in either ascii or interpret it as "hex"
    p = NULL;
    h = FldGetTextHandle(pfield);
    if (h != 0)
    {
        p = MemHandleLock(h);
        if (!p)
        {
            FrmCustomAlert(ErrorAlert,"Unable To Send",NULL,NULL);
            return;
        }
        // we now have data ... let's process it!
        if (AsciiHexMode == 0)
        {
            // send ascii ....
            Send(p,StrLen(p));
        }
        else
        {
            // check that this string is an even length
            j = (UInt8) StrLen((Char *)p);
            if ((j % 2) != 0)
            {
                FrmCustomAlert(ErrorAlert,"This is an invalid hex string!",NULL,NULL);
            }
            else
            {
                // walk thru this, converting every 2 characters into a byte to send
                for (i=0;i<j;i+=2)
                {
                    k = 16 * hexdecode(p[i]);
                    k += hexdecode(p[i+1]);
                    c = (Char) k;
                    Send((UInt8*) &c,1);
                }
            }
        }
        StrCopy (p,"");
        MemSet((Char*)p,(Int16) MemHandleSize(h),0x00);
        FldSetTextHandle(pfield,h);
        FldDrawField(pfield);
        MemHandleUnlock(h);
    }
}
```

FIGURE C-5: FRMMAIN.C - PAGE 5

```
else
{
    FrmCustomAlert(ErrorAlert,"Unable To Send",NULL,NULL);
    return;
}
rawreadmode = true;
return true;
}

static Boolean frmMain_CLEARDATA_OnSelect(EventPtr event)
{
    // Insert code for CLEARDATA
    if (!commsactive) return true;
    if (FrmCustomAlert(ConfirmAlert,"Clear Counters, Are You Sure",NULL,NULL) == 0)
    {
        ClearVendingMachine();
        QueryVendingMachine();
    }
    return true;
}

static Boolean frmMain_READDATA_OnSelect(EventPtr event)
{
    //UInt8 payload = 0x35;
    //UInt8 response[100];

    // Insert code for READDATA
    if (!commsactive) return true;
    QueryVendingMachine();
    return true;
}

static Boolean frmMain_ASCIIHEX_OnSelect(EventPtr event)
{
    FormPtr f = FrmGetActiveForm();
    ControlPtr cptr =
        (ControlPtr)FrmGetObjectPtr(f,FrmGetObjectIndex(f,ASCIIHEXINDICATOR));

    // Insert code for ASCIIHEX
    if (AsciiHexMode == 0)
    {
        AsciiHexMode = 1;
    }
    else
    {
        AsciiHexMode = 0;
    }
    CtlSetLabel(cptr,AsciiHexModeStrings[AsciiHexMode]);
    FrmDrawForm(f);
    return true;
}
```

FIGURE C-6: FRMMAIN.C - PAGE 6

```
static Boolean frmMain_SHOWTRACE_OnSelect(EventPtr event)
{
    // Insert code for SHOWTRACE
    FrmCustomAlert(RawDataAlert,(char *) rawreadbuffer,NULL,NULL);
    return true;
}

static Boolean frmMain_RESETRACE_OnSelect(EventPtr event)
{
    FormPtr f = FrmGetActiveForm();
    ControlPtr cptr = (ControlPtr) FrmGetObjectPtr(f,FrmGetObjectIndex(f,TRACECOUNT));

    // Insert code for RESETRACE
    if (FrmCustomAlert(ConfirmAlert,"Reset Trace Buffer, Are You Sure",NULL,NULL) == 0)
    {
        MemSet(rawreadbuffer,sizeof(rawreadbuffer),0x00);
        rawreadsize = 0;
        StrPrintf(tracebuffermsg,"Trace : %d Bytes",rawreadsize);
        CtlSetLabel(cptr,tracebuffermsg);
        //CtlDrawControl(cptr);
        FrmDrawForm(f);
    }
    return true;
}

static Boolean frmMain_cmdABC_OnSelect(EventPtr event)
{
    FormPtr f = FrmGetActiveForm();
    ControlPtr cptr = (ControlPtr)FrmGetObjectPtr(f,FrmGetObjectIndex(f,cmdABC));

    // set focus to TXData field
    FrmSetFocus(f,FrmGetObjectIndex(f,TXDATA));
    // show keyboard
    SysKeyboardDialog(kbdAlpha);
    // deselect button
    CtlSetValue(cptr,0);
    return true;
}

static Boolean frmMain_cmdOneTwoThree_OnSelect(EventPtr event)
{
    FormPtr f = FrmGetActiveForm();
    ControlPtr cptr = (ControlPtr)FrmGetObjectPtr(f,FrmGetObjectIndex(f,cmdOneTwoThree));
    // set focus to TXData field
    FrmSetFocus(f,FrmGetObjectIndex(f,TXDATA));
    // show keyboard
    SysKeyboardDialog(kbdNumbersAndPunc);
    // deselect button
    CtlSetValue(cptr,0);
    return true;
}

static Boolean frmMain_cmdOneTwoThree_OnSelect(EventPtr event)
{
    FormPtr f = FrmGetActiveForm();
    ControlPtr cptr = (ControlPtr)FrmGetObjectPtr(f,FrmGetObjectIndex(f,cmdOneTwoThree));
    // set focus to TXData field
    FrmSetFocus(f,FrmGetObjectIndex(f,TXDATA));
    // show keyboard
    SysKeyboardDialog(kbdNumbersAndPunc);
    // deselect button
    CtlSetValue(cptr,0);
    return true;
}
}
```

FIGURE C-7: FRMMAIN.C - PAGE 7

```

/*****
 *
 * FUNCTION:          frmMain_HandleEvent
 *
 * DESCRIPTION:      Handles a Form event
 *
 * PARAMETERS:       event    pointer to an event structure
 *
 * RETURNED:         returns handled/not handled
 *
 * REVISION HISTORY:
 *   Name            Date                Description
 *   ----            -
 *   Administrator   11/21/2002 11:55:59 PM  Created
 *
 *****/

Boolean frmMain_HandleEvent(EventPtr event)
{
    FormPtr form;
    Boolean handled = false;

    switch (event->eType)
    {
        case ctlSelectEvent:
            switch (event->data.ctlSelect.controlID)
            {
                // SENDFILE receive an event
                case SENDFILE:
                    handled = frmMail_SENDFILE_OnSelect (event);
                    break;
                // RESETTRACE receives an event
                case RESETTRACE:
                    handled = frmMain_RESETTRACE_OnSelect(event);
                    break;
                // SEND receives an event
                case SEND:
                    handled = frmMain_SEND_OnSelect(event);
                    break;
                // CLEARDATA receives an event
                case CLEARDATA:
                    handled = frmMain_CLEARDATA_OnSelect(event);
                    break;
            }
    }
}

```

FIGURE C-8: FRMMAIN.C - PAGE 8

```
    // READDATA receives an event
    case READDATA:
        handled = frmMain_READDATA_OnSelect(event);
        break;
    // CONNECT receives an event
    case CONNECT:
        handled = frmMain_CONNECT_OnSelect(event);
        break;
    // ASCIIHEX receives an event
    case ASCIIHEX:
        handled = frmMain_ASCIIHEX_OnSelect(event);
        break;
    // SHOWTRACE receives an event
    case SHOWTRACE:
        handled = frmMain_SHOWTRACE_OnSelect(event);
        break;
    // cmdABC receives an event
    case cmdABC:
        handled = frmMain_cmdABC_OnSelect(event);
        break;
    // cmdOneTwoThree receives an event
    case cmdOneTwoThree:
        handled = frmMain_cmdOneTwoThree_OnSelect(event);
        break;
    }
    break;
case frmOpenEvent:
    // Repaint form on open
    form = FrmGetActiveForm();
    FrmDrawForm(form);
    WinDrawLine(0,42,160,42);
    handled = true;
    break;
default:
    break;
}

return handled;
}
```

APPENDIX D: PALM SOURCE CODE - MENU.C

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") is intended and supplied to you, the Company's customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

FIGURE D-1: MENU.C - PAGE 1

```

/*****
 *
 * Created with Falch.net DeveloperStudio
 * http://www.falch.net/
 *
 * File : Menu.c
 *
 * Description :
 *
 * History:
 *   Name           Date           Description
 *   ----           ----           -
 *   Frank Ableson  March 2003      Created
 *
 *****/

#include <PalmOS.h>
#include "MCP215XDemo.h"
#include "MCP215XDemo_res.h"

```

FIGURE D-2: MENU.C - PAGE 2

```
/*
 *
 * FUNCTION:      ApplicationHandleMenu
 *
 * DESCRIPTION:   This routine handles menu events
 *
 * PARAMETERS:   id of the menu recieveing a click
 *
 * RETURNED:     handled/not handled
 *
 * REVISION HISTORY:
 *   Name          Date          Description
 *   ----          -
 *   Administrator 11/21/2002 11:55:59 PM Created
 *
 */
***** /

Boolean ApplicationHandleMenu(UInt16 menuID)
{
    Boolean handled = false;
    EventType evt;

    switch (menuID)
    {
        case mnuHelp:
            FrmCustomAlert(InfoAlert,"help is on the way !",NULL,NULL);
            handled = true;
            break;
        case mnuAbout:
            FrmDoDialog(FrmInitForm(frmAbout));
            handled = true;
            break;
        case mnuClose:
            evt.eType = appStopEvent;
            EvtAddEventToQueue(&evt);
            handled = true;
            break;
        default:
            break;
    }

    return handled;
}
```


APPENDIX E: PALM SOURCE CODE - COMMS.C

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") is intended and supplied to you, the Company's customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

FIGURE E-1: COMMS.C - PAGE 1

```
// comms.c
// this source file contains the bulk of communications interaction
// for the demo application.
// Frank Ableson
// frank@cfgsolutions.com
// 973 448 1844

#include <PalmOS.h>
#include "MCP215XDemo_res.h"
#include "comms.h"

#define min(a,b) (a < b ? a : b )

/*
communications global variables ....
*/
Boolean commsactive = false;
UInt16 portid = 0;
Boolean rawreadmode = false;
UInt8 rawreadbuffer[1024];
UInt16 rawreadsize;

/* Descriptive Text Labels */
char sodalabel[100];
char candieslabel[100];
char tracebuffermsg[100];

Boolean OpenPort()
{
Err e;
char buf[100];

// attempt to connect ....
e = SrmOpen('ircm',9600,&portid);
if (e)
{
StrPrintf(buf,"Failed to open port [%d]",e);
FrmCustomAlert(ErrorAlert,buf,NULL,NULL);
return false;
}
// now set any port parameters desired ...
// push out a zero sized packet to bring up the interface ....
SrmSendFlush(portid);
SrmReceiveFlush (portid, SysTicksPerSecond ()/2);
// buf[0] = 0xff;
// SrmSend (portid,buf,1,&e);
commsactive = true;
return true;
}
```

FIGURE E-2: COMMS.C - PAGE 2

```
void ClosePort()
{
    SrmClose(portid);
    commsactive = false;
}

static void DoEvents()
{
    EventType evt;

    do
    {
        EvtGetEvent(&evt,10);
        SysHandleEvent(&evt);
    } while (evt.eType != nilEvent);
}

void Send(UInt8 * data,UInt32 size)
{
    Err e;

    // clear out send and receive buffers
    SrmSendFlush(portid);
    SrmReceiveFlush(portid,SysTicksPerSecond()*1.5);

    if (SrmSend(portid,(void *) data,size,&e) != size)
    {
        FrmCustomAlert(ErrorAlert,"Failed To Send Data",NULL,NULL);
        return;
    }
    SrmSendWait(portid);
}

void QueryVendingMachine()
{
    Err e;
    UInt8 b;

    // clear out send and receive buffers
    SrmSendFlush(portid);
    SrmReceiveFlush(portid,SysTicksPerSecond()*1.5);

    // send query to the device ... ascii 5 or hex 0x35 ....
    b = '5';
    if (SrmSend(portid,(void *) &b,1,&e) != 1)
    {
        FrmCustomAlert(ErrorAlert,"Failed To Query Demo Board",NULL,NULL);
        return;
    }
    SrmSendWait(portid);
    QueryVendingMachineResponse();
}
```

FIGURE E-3: COMMS.C - PAGE 3

```
void QueryVendingMachineResponse()
{
  UInt32 bytestoread,bytesread;
  Boolean done = false;
  char msg[100];
  char buf[100];
  char search[] = {0x0d,0x0a,0x0a,0x00};
  char c;
  char * p1;
  char * p2;
  char * p3;
  char * p4;
  Boolean foundstart,foundend;
  UInt32 offset;
  Err e;
  UInt32 starttime,now;
  UInt32 b;
  EventType evt;
  ControlPtr cptr;

  // go get the response...
  foundstart = foundend = false;
  offset = 0;
  MemSet(buf,sizeof(buf),0x00);
  MemSet(msg,sizeof(msg),0x00);
  starttime = TimGetSeconds();
  DoEvents();
}
```

FIGURE E-4: COMMS.C - PAGE 4

```

while (!done)
{
    e = 0;
    bytestoread = 0;
    SrmReceiveCheck(portid,&bytestoread);
    if (bytestoread > 0)
    {
        bytesread = SrmReceive(portid,&buf[offset],
                               bytestoread, SysTicksPerSecond(), &e);
        if (e == 0)
        {
            // process whatever data has been read ....
            //FrmCustomAlert(InfoAlert,buf,NULL,NULL);
            p1 = StrStr(buf,search);
            if (p1)
            {
                p1+=3; // skip past start of message signature
                p2 = StrStr(p1,search);
                if (p2)
                {
                    // found both start and end of message ... parse it out ...
                    // 0d0a0atext = number0d0atext = number0d0a0a
                    //      p1          p3          p2
                    p3 = StrChr(p1,0x0d);
                    if (p3)
                    {
                        // read to the left until the character is a space ....
                        p4 = p3;
                        while (*p4 != ' ')
                        {
                            p4--;
                            if (p4 == p1)
                            {
                                FrmCustomAlert(ErrorAlert,
                                                "Failed to Parse Response.",NULL,NULL);
                                goto leave;
                            }
                        }
                        // ok, let's see what we're left with!
                        *p3 = 0x00;
                        //FrmCustomAlert(InfoAlert,p4,NULL,NULL);
                        StrPrintf(sodalabel,"# of Sodas Sold = %s ",p4);
                        cptr = (ControlPtr)FrmGetObjectPtr(FrmGetActiveForm(),
                                                         FrmGetObjectIndex(FrmGetActiveForm(),SODAS));
                        CtlSetLabel(cptr,sodalabel);
                        // now do the same for the second value ....
                        p4 = p2;
                        while (*p4 != ' ')
                        {
                            p4--;
                            if (p4 == p3)
                            {
                                FrmCustomAlert(ErrorAlert,
                                                "Failed to Parse Response.",NULL,NULL);
                                goto leave;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

FIGURE E-5: COMMS.C - PAGE 5

```
        // ok, let's see what we're left with!
        *p2 = 0x00;
        //FrmCustomAlert(InfoAlert,p4,NULL,NULL);
        StrPrintf(candieslabel,"# of Candies Sold = %s  ",p4);
        cptr = (ControlPtr)FrmGetObjectPtr(FrmGetActiveForm(),
            FrmGetObjectIndex(FrmGetActiveForm(),CANDIES));
        CtlSetLabel(cptr,candieslabel);
        done = true;
        FrmDrawForm(FrmGetActiveForm());
    }
    else
    {
        FrmCustomAlert(ErrorAlert,
            "Couldn't Parse Response Message!",NULL,NULL);
    }
}
}
goto leave;
}
else
{
    // error occurred reading ...
    StrPrintf(msg,"Error Reading Response From Demo Board - %d",e);
    FrmCustomAlert(ErrorAlert,msg,NULL,NULL);
}
}
// check to see how long we've been waiting for a complete response ....
if (TimGetSeconds() - starttime > 5)
{
    // there's a problem, this should've come back right away...
    FrmCustomAlert(ErrorAlert,
        "Error Getting Response From Demo Board- Timeout.",NULL,NULL);
    goto leave;
}
}
}
leave:
}
```

FIGURE E-6: COMMS.C - PAGE 6

```
void ClearVendingMachine()
{
Err e;
UInt8 b;
    // clear out send and receive buffers
    SrmSendFlush(portid);
    SrmReceiveFlush(portid, SysTicksPerSecond()*1.5);

    // send command to the device ... ascii 6 or hex 0x36 ....
    b = '6';
    if (SrmSend(portid, (void *) &b, 1, &e) != 1)
    {
        FrmCustomAlert(ErrorAlert, "Failed To Query Demo Board", NULL, NULL);
        return;
    }
    SrmSendWait(portid);
}
/*
    read data and then update the trace byte count UI ....
*/
void ReadIntoBuffer(UInt32 bytestoread)
{
    UInt32 bytes;
    FormPtr f = FrmGetActiveForm();
    ControlPtr cptr = (ControlPtr) FrmGetObjectPtr(f, FrmGetObjectIndex(f, TRACECOUNT));
    Err e;

    bytes = SrmReceive(portid, &rawreadbuffer[rawreadsize],
        min(bytestoread, sizeof(rawreadbuffer) - rawreadsize),
        SysTicksPerSecond(), &e);
    rawreadsize += bytes;
    StrPrintf(tracebuffermsg, "Trace : %d Bytes", rawreadsize);
    CtlSetLabel(cptr, tracebuffermsg);
    //CtlDrawControl(cptr);
    FrmDrawForm(f);
}

    send file (large amount of data) to secondary device
*/
void SendFile()
{
    Int16 packets, i;
    Char c;
    Char buf[16+1];
    Err e;

    c = 48;
    for (packets=0; packets<15; packets++)
    {
        i=0;
        for (i=0; i<16; i++)
        {
            buf[i] = c++;
            if (c > 122) c = 48;
        }
        SrmSend(portid, buf, 16, &e);
        SrmSendFlush(portid);
        //    buf[16] = 0x00;
        //    FrmCustomAlert(InfoAlert, buf, NULL, NULL);
    }
}
```

APPENDIX F: PALM SOURCE CODE - FRMABOUT.C

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") is intended and supplied to you, the Company's customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

FIGURE F-1: FRMABOUT.C - PAGE 1

```

/*****
 *
 * Created with Falch.net DeveloperStudio
 * http://www.falch.net/
 *
 * File : frmAbout.c
 *
 * Description :
 *
 * History:
 *   Name           Date           Description
 *   ----           -
 *   Frank Ableson  March 2003      Created
 *
 *****/

#include <PalmOS.h>
#include "MCP215XDemo.h"
#include "MCP215XDemo_res.h"

/*****
 *
 * FUNCTION:        frmAbout_HandleEvent
 *
 * DESCRIPTION:     Handles a Form event
 *
 * PARAMETERS:      event  pointer to an event structure
 *
 * RETURNED:        returns handled/not handled
 *
 * REVISION HISTORY:
 *   Name           Date           Description
 *   ----           -
 *   Administrator  11/21/2002 11:55:59 PM  Created
 *
 *****/

```

FIGURE F-2: FRMABOUT.C - PAGE 2

```
Boolean frmAbout_HandleEvent(EventPtr event)
{
    FormPtr form;
    Boolean handled = false;
    switch (event->eType)
    {
        case frmOpenEvent:
            // Repaint form on open
            form = FrmGetActiveForm();
            FrmDrawForm(form);
            handled = true;
            break;
        default:
            break;
    }
    return handled;
}
```


APPENDIX G: PALM SOURCE CODE - INCLUDE FILES

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") is intended and supplied to you, the Company's customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

FIGURE G-1: MCP2150DEMO.H

```

/*****
 *
 * Created with Falch.net DeveloperStudio
 * http://www.falch.net/
 *
 * File : MCP215XDemo.h
 *
 * Description :
 *
 * History:
 *   Name           Date           Description
 *   ----           -
 *   Administrator  11/21/2002 11:55:59 PM  Created
 *
 *****/

Boolean ApplicationHandleMenu(UInt16 menuID);
Boolean frmMain_HandleEvent(EventPtr event);
Boolean frmAbout_HandleEvent(EventPtr event);

```

FIGURE G-2: COMMS.H

```

Boolean OpenPort();
void ClosePort();
void QueryVendingMachine();
void QueryVendingMachineResponse();
void ClearVendingMachine();
void ReadIntoBuffer(UInt32 bytestoread);
void Send(UInt8 * data,UInt32 size);
void SendFile ();

```

FIGURE G-3: MCP2150DEMO_RES.H

```
#define frmMain 1000
#define cmdOneTwoThree 1000
#define cmdABC 1001
#define CONNECT 1002
#define READDATA 1003
#define CLEARDATA 1004
#define SODAS 1005
#define CANDIES 1006
#define ASCIIHEX 1007
#define TXDATA 1008
#define SHOWTRACE 1009
#define RESETTRACE 1010
#define ASCIIHEXINDICATOR 1011
#define TRACECOUNT 1012
#define AREWECONNECTED 1013
#define SEND 1014
#define SENDFILE 1015
#define frmAbout 1001
#define cmdClose 1000
#define InfoAlert 1000
#define ErrorAlert 1001
#define ConfirmAlert 1002
#define RawDataAlert 1003
#define mnufrmMain 1000
#define mnuClose 1000
#define mnuConfigCommunications 1001
#define mnuConfigTerminal 1002
#define mnuHelp 1003
#define mnuAbout 1004
```

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, MPLAB, PIC, PICmicro, PICSTART, PRO MATE and PowerSmart are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


AmpLab, FilterLab, microID, MXDEV, MXLAB, PICMASTER, SEEVAL, SmartShunt and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Application Maestro, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, PICKit, PICDEM, PICDEM.net, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, rLAB, rPIC, Select Mode, SmartSensor, SmartTel and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2004, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

**QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949:2002 ==**

Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003. The Company's quality system processes and procedures are for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, non-volatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.



WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: <http://www.microchip.com>

Atlanta

3780 Mansell Road, Suite 130
Alpharetta, GA 30022
Tel: 770-640-0034
Fax: 770-640-0307

Boston

2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848
Fax: 978-692-3821

Chicago

333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071
Fax: 630-285-0075

Dallas

4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423
Fax: 972-818-2924

Detroit

Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo

2767 S. Albright Road
Kokomo, IN 46902
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles

18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888
Fax: 949-263-1338

Phoenix

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966
Fax: 480-792-4338

San Jose

1300 Terra Bella Avenue
Mountain View, CA 94043
Tel: 650-215-1444

Toronto

6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Australia

Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing

Unit 706B
Wan Tai Bei Hai Bldg.
No. 6 Chaoyangmen Bei Str.
Beijing, 100027, China
Tel: 86-10-85282100
Fax: 86-10-85282104

China - Chengdu

Rm. 2401-2402, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-86766200
Fax: 86-28-86766599

China - Fuzhou

Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506
Fax: 86-591-7503521

China - Hong Kong SAR

Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Shanghai

Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700
Fax: 86-21-6275-5060

China - Shenzhen

Rm. 1812, 18/F, Building A, United Plaza
No. 5022 Binhe Road, Futian District
Shenzhen 518033, China
Tel: 86-755-82901380
Fax: 86-755-8295-1393

China - Shunde

Room 401, Hongjian Building
No. 2 Fengxiangnan Road, Ronggui Town
Shunde City, Guangdong 528303, China
Tel: 86-765-8395507 Fax: 86-765-8395571

China - Qingdao

Rm. B505A, Fullhope Plaza,
No. 12 Hong Kong Central Rd.
Qingdao 266071, China
Tel: 86-532-5027355 Fax: 86-532-5027205

India

Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaughnessy Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan

Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471-6166 Fax: 81-45-471-6122

Korea

168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5932 or
82-2-558-5934

Singapore

200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

Taiwan

Kaohsiung Branch
30F - 1 No. 8
Min Chuan 2nd Road
Kaohsiung 806, Taiwan
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan

Taiwan Branch
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Austria

Durisolstrasse 2
A-4600 Wels
Austria
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark

Regus Business Centre
Lautrup høj 1-3
Ballerup DK-2750 Denmark
Tel: 45-4420-9895 Fax: 45-4420-9910

France

Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - 1er Etage
91300 Massy, France
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany

Steinheilstrasse 10
D-85737 Ismaning, Germany
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy

Via Quasimodo, 12
20025 Legnano (MI)
Milan, Italy
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands

P. A. De Biesbosch 14
NL-5152 SC Drunen, Netherlands
Tel: 31-416-690399
Fax: 31-416-690340

United Kingdom

505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44-118-921-5869
Fax: 44-118-921-5820

11/24/03