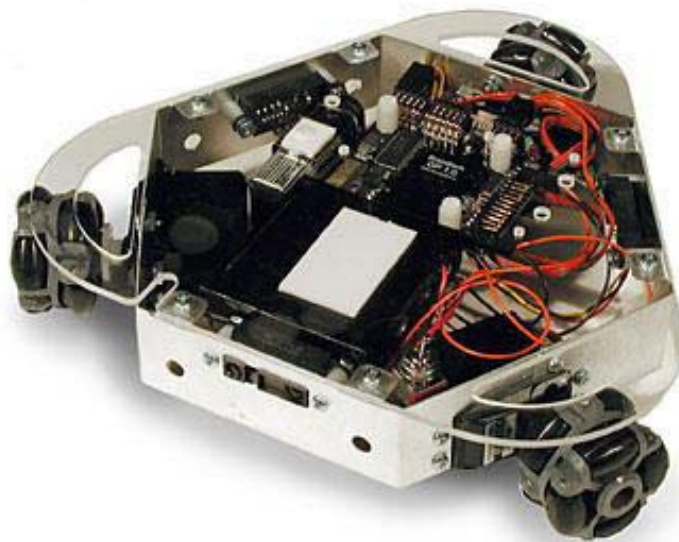


A Beginners Guide To Programming For The PPRK



Compiled By:
Sam Utley
December 2003

Table of Contents

Introduction.....	3
Assemble the PPRK.....	3
Setup the Palm Pilot.....	3
Setup Codewarrior.....	3
Test Your PPRK.....	4
Get Your Own Project Started.....	4
Renaming Your Project.....	4
Setting Up the Palm Emulator.....	5
Compiling Your Project for the First Time.....	6
Changing the User Interface.....	6
Connecting the User Interface with the Code.....	7
Forms.....	7
Buttons.....	8
Menus.....	9
Code for Controlling PPRK.....	9
Communicating With the SV203.....	10
Controlling the Servos.....	10
Reading the IR Rangers.....	11
Math Library.....	11
Additional Functions.....	12
Conclusions.....	13

Introduction

The Palm Pilot Robot Kit (PPRK) designed by the Carnegie Mellon Robotics Institute¹ is an easily assembled robotics kit that allows almost anyone to begin building and programming a robot at relatively little cost. Well...almost anyone. Anyone that knows how to create a program for PalmOS² for communicating through the Palm's serial port and can understand some inline assembly commands is well suited to take on the challenge. That sounds daunting, but it's not impossible. The hardest part is getting started off on the right foot. A program that contains good examples and a few simple instructions on getting together a program that compiles can allow a user to have a PPRK moving in no time. The following guide was written after using Metrowerks' Codewarrior³ for PalmOS v8 to write a program for controlling a PPRK purchased from Acroname, Inc⁴.

Assemble the PPRK

To begin with you'll want to assemble your PPRK, so go ahead and excitedly open the seal on your kit box and follow the instructions found on Acroname's website at http://www.acroname.com/robotics/info/PPRK/assembly/assm1_1.html. You'll have to do some soldering, but not much. Remember to have your shrink wrap already on your wires before soldering them in place. If you've never soldered before you may want to check out some soldering guides on the web (e.g. <http://www.circuittechctr.com/guides/7-1-1.shtml> or <http://www.mindhertz.com/Solder.php>).

Setup the Palm Pilot

After your robot is together, you'll need to get together all of your Palm equipment. You can get by with a minimal install of the Palm Desktop software. All you need is the HotSync Manager and Install Tool. The Palm cradle connects to your computer's serial port, but if you have a newer laptop you may not have a traditional serial port available. If you only have a USB port available you can obtain a USB to Serial PDA adapter that will allow you to connect the Palm to the laptop. While writing this guide, a Keyspan USB PDA adapter was used (<http://www.keyspan.com/products/usb/pdaadapter/>) on a Sony Viao laptop running Windows XP Home. If Hotsyncing your Palm doesn't work immediately, you'll want to change the Com port settings for the Hotsync Manager.

Setup Codewarrior

Finally you'll need to install a compiler for creating and making programs for the Palm. This guide is targeted towards using Metrowerks' Codewarrior IDE and Constructor. Just follow the instructions on the install. You're now ready to begin programming for the Palm and running your PPRK.

¹ <http://www-2.cs.cmu.edu/~pprk/>

² <http://www.palmsource.com/>

³ <http://www.metrowerks.com/>

⁴ <http://www.acroname.com/index.html>

Test Your PPRK

To test your robot, download the MathLib.prc, ServoTest.prc, and Robot1.prc files from <http://www-2.cs.cmu.edu/~pprk/>. The *.prc extension is to the Palm what the *.exe is to the PC. It is a premade application ready to be executed. After downloading the files you can double click on them to add them to the Install Manager and sync your Palm. With your Palm in the cradle, first install MathLib.prc and then ServoTest.prc and Robot1.prc. After installing these applications you can connect your Palm to your PPRK and try them out. Warning: these applications may not appear fully functional but they give a good sense of what type of user interface is friendly and what the PPRK can do.

Get Your Own Project Started

To make your own application, you'll first want to start with a project that actually compiles. If you're not a programming wiz and you just try to download the great examples from Carnegie Mellon (<http://www-2.cs.cmu.edu/~pprk/>) and compile them you'll spend a lot of time trying to figure out what's wrong. It may be easiest to start with a blank project and copy and paste code from the examples on Carnegie Mellon's site.

To begin a new project:

1. Launch Codewarrior IDE
2. Create a new project and launch the Palm OS Application Stationary Wizard.
 File >> New >> Project >> Palm OS Application Stationary
3. Point the location field to a convenient place to save your project.
4. Give a project name that reflects its purpose (e.g. RobotMapper).
5. Select OK
6. Select Palm OS C++ App as the stationary type.
7. Select OK

Codewarrior generates a set of files for the new project. Files can be added or deleted from the project tree structure, but deleting files from the tree structure does not delete them from the computer. Deletion simply alters the *.mcp (project) file which keeps track of project settings and folder structures. The Starter.cpp file contains all the code that controls the actions of the Palm (i.e. form handling, opening/closing serial port, etc.). The Starter.rsrc file is the project file for the Palm user interface that allows the user to interact with the functions contained in Starter.cpp. The *.rsrc file can be opened and modified using the Constructor program included in the Codewarrior package.

Renaming Your Project

The name "Starter" isn't really specific, and some modifications can be made to get rid of it. If you are happy with this name then just skip to the next section.

To get rid of the name "Starter":

1. In Codewarrior IDE, double click on Starter.cpp to open it.
2. At the top you'll see: `#include "StarterRsc.h"`. Replace `StarterRsc` with `<your new name>.h` (e.g. `#include "RobotMapper.h"`).

3. Select
 - File >> Save As >> <your new name>
 and close the newly saved file. Your project will update by replacing the Starter.cpp file with <your new name>.cpp
4. Double click on Starter.rsrc to open it with Constructor.
5. Under Project Settings at the bottom of the Constructor window, change Header File Name from StarterRsc.h to <your new name>.h and the Application Icon Name to <your new name>
6. Resave it as <your new name>.rsrc in your project's Rsc folder and close Constructor
7. Select: Starter.rsrc and press delete
8. Right click on the Resources folder and select Add Files
9. Add <your new name>.rsrc and <your new name>.h
10. Choose: Edit >> Starter-Debug Settings
11. Change: Target >>Target Settings >> Target Name to <your new name>-Debug
12. Change: Target >> 68K Target >> File Name to <your new name>-d.tmp
13. Change: Linker >> PalmRez Post Linker >> Mac Resource Files to <your new name>-d.tmp
14. Change: Linker >> PalmRez Post Linker >> Output File to <your new name>.prc
15. Change: Linker >> PalmRez Post Linker >> Database Name to <your new name>
16. Select Apply
17. At the top of your main project window, use the drop box to select Starter-Release
18. Choose: Edit >> Starter-Release Settings
19. Change: Target >>Target Settings >> Target Name to <your new name>-Release
20. Change: Target >> 68K Target >> File Name to <your new name>.tmp
21. Change: Linker >> PalmRez Post Linker >> Mac Resource Files to <your new name>.tmp
22. Change: Linker >> PalmRez Post Linker >> Output File to <your new name>.prc
23. Change: Linker >> PalmRez Post Linker >> Database Name to <your new name>
24. Select Apply

Your project should now be completely customized name-wise to your liking. For convenience, the application will continue to be referred to as "Starter" instead of <your new name>.

Setting Up the Palm Emulator

Before beginning to program, you want to get a Palm emulator up and running so that user interfaces and simple functions can be tested quickly without having to download the program to an actual Palm. In order to get the emulator working, a ROM

image of the Palm being programmed for is needed. This can be uploaded from a FRESH Palm (nothing has been installed on it yet) or downloaded from “<http://www.palmos.com/dev/tools/emulator/>”. Once the correct ROM image has been obtained it should be extracted to a known place on your computer where it will permanently reside.

To get the emulator working:

1. Select: Edit >> Preferences >> Debugger >> PalmDebugger Settings
2. Change: Target to *Palm OS Emulator*.
3. Change: Location to <Path to Palm Emulator executable>. This is probably located at <Path to CodeWarrior>\Palm Tools\Palm OS Emulator\Emulator.exe.
4. Select OK
5. Select: Palm >> Launch Emulator. This will launch a new emulator session.
6. Select the ROM file to use (e.g. palmos33-en-v.rom). You’ll probably have to choose “other” and find the ROM file on the computer the first time it is launched.
7. Select the correct device corresponding to the ROM file chosen (e.g. Vx).
8. Select OK

You now have a working emulator. The Palm emulator can be controlled by using the mouse pointer just as you would the stylus. The skin can be changed to make it look like the Palm being programmed for, but that isn’t necessary. Please note that when programming for the PPRK you may begin to get errors on the emulator that you wouldn’t get on the actual Palm.

Compiling Your Project for the First Time

Using Codewarrior IDE, you can now make and run your project. Pressing the F7 key will make the project. Ctrl+F5 will run the newly made *.prc file in the emulator. There isn’t much to the new program. There is just an *About Start App* menu on the *Main* form and an *About Starter Form*.

Changing the User Interface

The user interface can be easily and quickly changed by modifying the Starter.rsrc file. For example:

1. Double click on Starter.rsrc to launch it in Constructor
2. Double click on *Main* under *Forms*
3. Select Window >> Catalog
4. Click on *Button* in the Catalog window and drag it onto the *Main* form shown
5. Save the changed Starter.rsrc file: File >> Save
6. Regenerate the header file: File >> Generate Header File

Remake the project, launch the emulator, and run the program. Notice a button now appears on the main form.

To add a form (analogous to a new window in Windows):

1. Open the *.rsrc file with Constructor
2. Single click on the *Forms* section title
3. Press Ctrl+R or Edit >> New Form Resource
4. Give the new form an appropriate name

To edit the form, just double click on it and use the objects available in the catalog.

Extra menus can also be added. A menu bar was auto generated with the project. To add another menu to that menu bar:

1. Double click on *Main Form* under *Menu Bars*
2. Press Ctrl+M or Edit >> New Menu
3. Add menu items with Ctrl+R or Edit >> New Menu Item

Connecting the User Interface with the Code

Unlike some development environments, Codwarrior does not automatically add code snippets for each object added to the user interface to the project. If you want to use the button, forms, menus, etc. you'll have to add the code yourself. By opening the header (*.h) file generated by Constructor, you can see how objects are named. For forms Constructor uses the format <Form Name>Form. For buttons the format is <Form Name><Button Name>Button. Menus are named <Menu Name>Menu, and menu items are named <Menu Name><Menu Item Name>. If you're ever unsure of what an object would be called, just open up the *.h file generated by Constructor and look at the name it was given.

Forms

You'll probably want to have a from handling event function for each form you create. The wizard created one in Starter.cpp called `MainFormHandleEvent` to handle events that occur on the form named *Main*. For example, if you add another form named *Functions* that does something more than just return to the form *Main*, you'll want to copy and paste the `MainFormHandleEvent` function and rename it `FunctionsFormHandleEvent`. If the main purpose of your form is to just display a message (e.g. the About Form), then having a separate event handler for it is unnecessary.

```
static Boolean FunctionsFormHandleEvent(EventPtr eventP)
{
    Boolean handled = false;
    FormPtr frmP;
    switch (eventP->eType)
    {
        case menuEvent:
            return MainFormDoCommand(eventP->data.menu.itemID);
        case frmOpenEvent:
            frmP = FrmGetActiveForm();
            MainFormInit( frmP);
            FrmDrawForm ( frmP);
    }
}
```

```

        handled = true;
        break;
    case frmUpdateEvent:
        break;
    default:
        break;
    }
    return handled;
}

```

Make sure you add this form to your AppHandleEvent so that it can be accessed.

```

static Boolean AppHandleEvent(EventPtr eventP)
{
    UInt16 formId;
    FormPtr frmP;
    if (eventP->eType == frmLoadEvent)
    {
        formId = eventP->data.frmLoad.formID;
        frmP = FrmInitForm(formId);
        FrmSetActiveForm(frmP);
        switch (formId)
        {
            case MainForm:
                FrmSetEventHandler(frmP,
                    MainFormHandleEvent);
                break;
            case FunctionsForm:
                FrmSetEventHandler(frmP,
                    FunctionsFormHandleEvent);
                break;
            default:
                break;
        }
        return true;
    }
    return false;
}

```

To call upon a form handling event use:

```
FrmGotoForm(FunctionsForm);
```

To draw a form on top of another (e.g. the About Form) use:

```

frmP = FrmInitForm (FunctionsForm);
FrmDoDialog (frmP); // Display the Form named Functions
FrmDeleteForm (frmP); // Erase the Form when it returns

```

Buttons

One way to allow a user to launch forms or other events and functions is to use buttons. You'll need to add a `ctlSelectEvent` to your form handling event. For instance, if a "Go" button is added to the Main Form, then this snippet of code could be added to the `MainFormHandleEvent`.

```

static Boolean MainFormHandleEvent(EventPtr eventP)
{
    ....existing code generated by wizard....
    switch (eventP->eType) {
        case menuEvent:
            return MainFormDoCommand(eventP->data.menu.itemID);
        case ctlSelectEvent:
            switch (eventP->data.ctlSelect.controlID) {

```

```

        case MainGoButton:
            yourfunction();
            handled = true;
            break;
        default:
            break;
    }
    ....rest of generated code....
}
return handled;
}

```

When the application is launched and the “Go” button is pressed `yourfunction()` will be launched. When the function returns, `handled` will be set true so that event can be taken off the event stack. Buttons can also send you to different forms using the code presented in the Forms section of this guide. If you want the “Go” button to send you to the Functions Form then replace the `MainGoButton` case above with:

```

case MainGoButton:
    FrmGotoForm(FunctionsForm);
    handled = true;
    break;

```

If you want the “Go” button to draw the Functions Form on top of the current form then the case would be:

```

case MainGoButton:
    frmP = FrmInitForm (FunctionsForm);
    FrmDoDialog (frmP);
    FrmDeleteForm (frmP);
    handled = true;
    break;

```

Menus

Another way to launch events is to use menus. This can be done by simply changing the existing `MainFormDoCommand` generated.

```

static Boolean MainFormDoCommand(UINT16 command) {
    Boolean handled = false;
    FormPtr frmP;
    switch (command) {
        ...existing code generated by wizard ...
        case MenuMenuItem:
            yourfunction();
    }
    return handled;
}

```

Code for Controlling PPRK

After getting your new project to compile with some handy forms, menus, and buttons, you’re ready to start making those buttons send commands to the PPRK’s SV203 through the Palm’s serial port. If you have problems getting the files from the Carnegie Mellon website to compile directly, you may want to take the approach of harvesting their

code, adapting it if need be, and implementing it in your own project. Remember to give credit in your code where credit is due.

Communicating With the SV203

The first things you'll need are some send and receive functions for communicating with the SV203. A SV203 user's manual is supplied in the PPRK kit. In the back of it is a list of commands that can be used to control the SV203. All of your higher functions are built upon these commands. Luckily Greg Reshko of Carnegie Mellon University provided some nice functions that accept a command and a parameter called Rx and Rc in the examples on the web. You'll need to include SerialMgrOld.h for the serial functions to work. You'll also need to include this in AppStart(void):

```
SysLibFind("Serial Library", &portRef); // find serial library
SerOpen(portRef, 0, 9600); // open COM port
```

You'll need to include this in AppStop(void):

```
SerClose(portRef); //Close Serial Port
```

The examples are:

```
#include <SerialMgrOld.h>
.....
/*****
* Function: Tx
* Description: This function sends strings to the board. A list of possible strings to
* send is listed in the back of the SV203 manual
* Parameters: *command - The prefix of the command to be sent
* parameter - usually a number
* Example: You can send SV1 to 'talk to' servo 1
* Credit: Greg Reshko of Carnegie Mellon University Last modified: 4/03/2000
*****/
void Tx(char *command, int parameter) {
    Err error;
    char prm_buffer[]=" ";
    // parameter buffer
    StrPrintf(prm_buffer, "%d\r", parameter); // store param and '\r' into buffer
    SerSendWait(portRef, -1); // wait for port to clear
    SerSend(portRef, command, StrLen(command), &error); // send buffer
    SerSendWait(portRef, -1); // wait for port to clear
    SerSend(portRef, prm_buffer, StrLen(prm_buffer), &error); // send buffer
}
/*****
* Function: RC
* Description: This function gets strings from the board.
* Parameters: *buffer
* Credit: Greg Reshko of Carnegie Mellon University Last modified: 4/03/2000
*****/
void Rc(char *buffer) {
    Err error;
    // receive from portRef into incoming buffer using 5 bytes and 3ms timeout
    SerReceive(portRef, buffer, 4, 10, &error); // receiver buffer
    SerReceiveFlush(portRef, 5); // flush receive buffers
    SerSendFlush(portRef); // flush send buffers
}
```

Controlling the Servos

The send and receive functions allow for the creation of servo control functions. Notice the way the parameter is used. By physically modifying the servo, you can no longer tell the servo to move to an absolute position, but supplying different positions (that it will never reach) will act as a speed control. The servos were modified to think they were at the mid point of its range (128). Telling the servo to move to a position

greater than 128 will cause the servo to spin CCW and less than 128 CW. The servo will try to move quickly to positions far away (near 255 or 1) or slowly to closer positions (close to 128). The values 128 and 0 are used for stopping the servo. For example:

```

/*****
* Function: Servo
* Description: This function utilizes Tx to send the servo number and
*              velocity.
* Parameters: servo_number - ie. 1, 2, 3
*              velocity - 0 to 255
* Credit: Greg Reshko of Carnegie Mellon University Last modified: 4/03/2000
*****/
void Servo(int servo_number, int velocity) {
    Tx("SV", servo_number);
    Tx("M", velocity);
}

```

Reading the IR Rangers

The PPRK also comes outfitted with 3 IR range finders that output fairly reliable reading between 10 and 70 cm. The readings do fluctuate and some averaging of the values over time may want to be performed if your readings are critical. One again the Rx and Rc functions are utilized to poll the Analog to Digital ports. This principle can be used to interface the SV203 with any sensor with an analog output. An example of reading the IR sensors is:

```

/*****
* Function: IRDist
* Description: This function reads IR and returns distance in cm from
*              obstacle.
* Uses Tx, Rc
* Parameters: ir_number - the number of the IR being read
* Credit: Greg Reshko of Carnegie Mellon University Last modified: 4/03/2000
*****/
int IRDist(int ir_number) {
    int value=0;
    char incoming[]="999";
    Tx("AD", ir_number);
    Rc(incoming);
    // buffer correction -- replace all ASCII non-numbers with spaces
    for (int l=0; l<=2; l++)
        if ( incoming[l] == 10 )
            incoming[l]=' ';
    incoming[3] = '\0';
    if (incoming[0] != '1') // two digit number cut-off
        incoming[2] = '\0';
    value = StrAToI(incoming);
    return value;
}

```

Math Library

In order to use more complex math functions such as powers, log, natural logs, etc., you'll need to open the math library. You'll need two functions provided by Reshko.

```

/*****
* Function: OpenMathLib, CloseMathLib
* Description: This function opens and closes MathLib
* Parameters: None
* Credit: Greg Reshko of Carnegie Mellon University Last modified: 4/03/2000
*****/
void OpenMathLib() {
    Err error; // find and open MathLib library
    error = SysLibFind(MathLibName, &MathLibRef);
    if (error)
        error = SysLibLoad(LibType, MathLibCreator, &MathLibRef);
    ErrFatalDisplayIf(error, "Can't find MathLib");
    error = MathLibOpen(MathLibRef, MathLibVersion);
}

```

```

        ErrFatalDisplayIf(error, "Can't open MathLib");
    }
void CloseMathLib() {
    Err error;    // close MathLib library
    UInt16 usecount;
    error = MathLibClose(MathLibRef, &usecount);
    ErrFatalDisplayIf(error, "Can't close MathLib");
    if (usecount == 0)
        SysLibRemove(MathLibRef);
}

```

You'll then need to call on `OpenMathLib()` in `AppStart()` and `CloseMathLib()` in `AppStop()`.

Additional Functions

Of course you're probably going to want to have some loops in your program. For instance, a loop that constantly drives the robot and adjusts its trajectory based on the IR sensor readings. Unfortunately I haven't really nailed down the whole using `ctlSelectEvents` to cause a loop to stop, yet. However you can easily use Reshko's `SmartWait` function to watch for `penDownEvents`. The following functions show a function that watches for `penDownEvents` (when someone touches the screen) and an example loop that kicks out when the `penDownEvent` occurs.

```

void Message(int number) {
    FrmPtr frm = FrmGetActiveForm();
    char numbertext[]="          ";
    FrmCopyLabel(frm, MainMessageLabel, numbertext);
    StrIToA(numbertext, number);
    FrmCopyLabel(frm, MainMessageLabel, numbertext);
}
void Wait(int delay_ticks) {
    UInt32 start_time;
    start_time = TimGetTicks();
    while ( TimGetTicks() < (start_time + delay_ticks));
}
int SmartWait(int delay_ticks) {
    UInt32 start_time;
    EventPtr quit_event;
    start_time = TimGetTicks();
    while ( TimGetTicks() < (start_time + delay_ticks)) {
        EvtGetEvent(quit_event, 1); // get event using lms timeout
        if (quit_event->eType == penDownEvent)
            return 1; }
    return 0;
}
void Loop(void){
    int i=0;
    do {
        Message(i++);
        Wait(500);
    }while(SmartWait(1)==0);
    return;
}

```

You'll notice that some other functions were also included. The `Message(int number)` and `Wait(int delay_ticks)` functions can also be very helpful. The `Message` function allows for the displaying of number to the screen in a Label field named `MainMessageLabel`. This label can be created in Constructor and the name can be changed. The `Wait` function allows for the delaying of events. It uses the

`TimGetTicks()` function. One second is about one-thousand ticks.
`TimGetSecs()` could also be used to retrieve integer seconds instead of integer ticks.

It's fairly easy to harvest code from examples. Just make sure that you have all the needed header files are included, all of your functions have prototypes, and that any old variable type-declarations are updated (e.g. `ULong` to `UInt32`, `Ptr` to `MemPtr`).

Conclusions

You should now have all the tools you need to begin programming the Palm for controlling your PPRK. With a clean project, some buttons, and a little code harvesting getting your 'bot to run will be no problem.

I would like to acknowledge the great work of Greg Reshko of Carnegie Mellon University, and the help provided by Don Potter of the University of Georgia.