UNIVERSIDAD MICHOACANA DE SAN NICOLAS DE HIDALGO

ROBOT GUIADO VÍA PALM OS VÍA PUERTO SERIE

PROYECTO DE TESIS

Que para obtener el Título de

INGENIERO ELECTRICISTA

Presenta

Luis Cuauhtémoc García Puente

Asesor de Tesis

Dr. Félix Calderón Solorio

Febrero del 2007

Agradecimientos

Agradezco al Ingeniero Félix Jiménez Pérez por el apoyo incondicional a este trabajo de tesis, al laboratorio de electrónica y a mi asesor el Dr. Félix Calderón Solorio.

Dedicatoria

A mi padre por darme las armas necesarias para sacar adelante esta carrera, a mi madre por estar siempre al pendiente de mí, y a mis hermanos por haber estado con mis padres durante todo este tiempo estuve fuera de casa. A mis grandes amigos, que han sido como mi familia a lo largo de este bonito viaje que ha sido mi carrera: Lorena, Francisco, Oscar, Lucero, Ireri, Gerzaín, Mauricio, Juan, Abel, Jorge, Julio, Pedro. Además dedico este trabajo al Ingeniero Octavio García Pérez, por apoyar a mi familia y a mí en los momentos difíciles.

Resumen

Este trabajo describe el uso de las herramientas básicas, proporcionadas por Palm, para el desarrollo de aplicaciones para PDA. Una vez entendidos ciertos elementos del desarrollo de aplicaciones para PDA's, como son los recursos, archivos de cabecera, código fuente, etc., se realiza una sencilla aplicación para desplegar mensajes en la pantalla de la PDA.

Una herramienta importante para alcanzar el objetivo de controlar un robot, es el desarrollo de una aplicación, la cual establezca un envío de información al robot por medio del puerto serie. Para esto se desarrolla una aplicación, la cual cuenta con un campo de texto para introducir la información a mandar a través del puerto serie, y además contiene tres botones: una para la apertura del puerto, otro para el envió de la información contenida en el campo de texto, y un último para el cierre del puerto.

Se presentan dos pruebas realizadas a la aplicación para comprobar el buen desempeño de la misma. La primera prueba consiste en el envío de información al puerto serie de una computadora de escritorio y se presenta la información enviada por medio del software llamado Hyper Terminal con que cuenta Windows. La última prueba es la concerniente al control del robot, mandando ciertas órdenes en un formato especificado para el manejo del microcontrolador con que cuenta el robot.

Contenido

Lista de Figuras

Abreviaciones

OS: Operative System

PDA: Personal Digital Assistant

PODS: Palm OS Developer Suite

PORE: Palm OS Resource Editor

POSE: Palm OS Emulator

Capítulo 1

Introducción

El propósito principal de una PDA (de sus siglas en ingles *Personal Digital Assistant*) es trabajar como un organizador o agenda electrónica portátil, fácil de usar y capaz de compartir información con una PC, no como un reemplazo.

Las PDA's, también llamadas *handhelds* o *palmtops*, han evolucionado a través de los años. No solamente pueden manejar información personal, tales como contactos, citas, cosas por hacer. Los PDA's de hoy en día también pueden conectarse a Internet, actúan como sistemas de posicionamiento global GPS (de las siglas *Global Position System*), y correr software multimedia. Algo adicional, es que los últimos modelos de PDA's han sido combinados con teléfonos celulares, reproductores multimedia (figura 1.1) y otros artefactos electrónicos [How Stuff Works 2006].

Fig. 1.1 Las PDA's y sus diferentes aplicaciones electrónicas

Es importante el conocer un poco acerca de la historia de estos dispositivos. La primer PDA de la historia fue la no muy famosa NEWTON de *Apple Computing*, la cual es una agenda electrónica que funciona con un lápiz (*Stylus*) con el cual se escribe en la pantalla y

esta reconoce lo que la persona escribe. No tuvo mucho éxito por su costo y tamaño (se aproxima al de una *Handheld*. Una *Handheld* es un dispositivo electrónico de funciones similares a una computadora personal pero de un tamaño más compacto y portátil.

En febrero de 1998 se lanzó al mercado la Palm III la cual conservaba el uso de pilas AAA y por supuesto la sincronización con el PC e iluminación de la pantalla (*Back Light*), superando al modelo anterior con muchas innovaciones como lo son su forma ergonómica, *Stylus* de metal, el doble de memoria, un sistema operativo mejorado (Palm OS 3.0), y una característica muy especial, el puerto Infrarrojo (*IR port*) con el cual se le encontraron nuevas funciones a las Palm tales como el compartir información con otros dispositivos compatibles con el protocolo de transferencia vía IR.

En enero del 2006, Palm saca a la venta su nueva versión de celular con aplicaciones PDA, pero con la novedad de conexión con tecnología inalámbrica (*wireless*), toda una computadora personal en la palma de la mano. [Configura equipos 2006]

Otro aspecto importante acerca de la evolución de las PDA's han sido los cambios en los sistemas operativos de Palm OS. A lo largo de este trabajo de tesis se hace referencia y comentario acerca de 3 versiones de Palm OS, en particular importantes para este trabajo. La versión 4.0 de Palm OS, es con la que cuenta el PDA con que se trabajó, pero el software con el cual se desarrollan las aplicaciones habla solamente de dos sistemas operativos: Garnet y Cobalt. Garnet fue el nombre dado a la versión 5.4 de Palm OS y a la versión 6.0 fue llamada Cobalt. Cobalt hace uso de aplicaciones mas enfocadas a elementos multimedia, y no tiene compatibilidad con la versión de PDA utilizada. Garnet cuenta con ciertas aplicaciones nuevas y diferentes de la versión 4.0, pero cuenta con una herramienta para la compatibilidad de software que lo hace adecuado para lo creación de las aplicaciones a desarrollar. (Ver pág. 12 [Foster y Bachmann 2005])

1.1 Antecedentes

Existe software de libre distribución para PDA´s en la red, sin embargo, la necesidad de crear una aplicación capaz de establecer un intercambio de información a través del puerto serie llevó al desarrollo de este trabajo. Existen aplicaciones desarrolladas para computadoras personales, desarrolladas en anteriores trabajos de tesis para establecer

comunicación serial, pero debido al tamaño y peso que representan las computadoras se ha optado por el trabajar con las PDA´s.

El M.C. Félix Jiménez Pérez, profesor de la Facultad de Ingeniería Eléctrica de la UMSNH, ha desarrollado un robot que se desplaza por medio de dos ruedas acopladas a dos motores de CD. Dicho robot puede ser controlado de forma remota mediante un puerto serial.

La aplicación a desarrollar tendrá la posibilidad de enviar la información necesaria para el control del robot antes mencionado.

1.2 Objetivo

El objetivo principal de este trabajo será controlar un robot mediante el envío de información desde la PDA al microcontrolador del robot por medio del puerto serie con que cuenta el mismo.

Este trabajo propone el desarrollo de una aplicación de comunicación vía puerto serie para demostrar la gran ventaja que las PDA's tienen debido a su tamaño y peso, lo cual la hacen una gran herramienta tecnológica portátil a bajo costo.

1.3 Justificación

Existen ciertas aplicaciones de control por medio de software, en las cuales el elemento de control es más grande que el propio elemento a controlar. No es muy común ver un elemento de control más grande que el elemento a controlar. Es importante poder aplicar las tecnologías nuevas a los problemas reales de la vida laboral.

La ventaja más importante de las PDA's es su tamaño, en comparación con las LAPTOP's más pequeñas que se encuentran en el mercado, además de ser mucho mas económicas.

La PDA Palm m125 tiene un peso de 150 g contra 1860 g de una *Laptop* DELL 700M; en cuestión de dimensiones, la PDA tiene de Ancho: 7.9 cm, Alto: 12.2 cm y Profundo: 2.2 cm contra una *Laptop* que mide de Ancho: 29.7 cm, Alto 3.8 cm y Profundo: 21.6cm, realmente una gran diferencia en cuestión de espacio. El modelo de PDA Palm m105, tiene un precio en el mercado de 300 pesos. La única diferencia entre los modelos 125 y 105 es la posibilidad de expandir la memoria mediante una tarjeta tipo SD. La *Laptop* DELL

700M tiene un precio en el mercado de 14000 pesos. Sin embargo, las PDA no pueden competir en cuestión de memoria y velocidad con las *Laptops*, pero las características del modelo de PDA utilizado son suficientes para establecer comunicación serial.

Las PDA's cuentan con las suficientes herramientas para realizar la tarea de comunicación necesaria para este proyecto, además de que tiene la gran ventaja de ser de escaso tamaño y peso, cabe en el bolsillo.

1.4 Metodología

Este trabajo, esta organizado de la siguiente forma, para establecer la comunicación a través del puerto serie entre el robot y la PDA.

Se comenzará por desarrollar una aplicación que ilustre el manejo del software proporcionado por la compañía Palm, para familiarizarse con el ambiente de desarrollo de aplicaciones para Palm OS. A través del desarrollo de esta primera aplicación, se aclararán términos importantes, tales como lo que son los recursos, los archivos de cabecera, la declaración de los recursos, y algunas funciones importantes en C++.Todo esto se tratará en el capítulo 2.

Para el capítulo 3, una vez familiarizado con el desarrollo de aplicaciones sencillas, se desarrolla una segunda aplicación, la cual establezca una comunicación serial para el envío de información; esta aplicación contará con botones para la apertura y el cierre del puerto, así como para enviar ordenes al robot para su funcionamiento.

En el capítulo 4 se comprobará la correcta escritura en el puerto serie, para ello se hace uso de un software tipo Terminal para conectar una PC con la PDA, dicho software (Hyper Terminal) se encuentra en la mayoría de los sistemas operativos de Microsoft Windows.

Finalmente se realizará una prueba a la aplicación desarrollada para la comunicación serial, mediante el control de un robot de forma remota a través del puerto serie. Para el control solo será necesario el uso de la PDA, un cable de conexión serial adecuado para el modelo de PDA con que se esta trabajando y el dispositivo a controlar.

1.5 Descripción de los Capítulos

En el Capítulo 1 se presenta un pequeño vistazo a la historia de las PDA y su sistema operativo; se muestra como ha ido evolucionando este dispositivo electrónico con el paso de los años y su gran auge en esta última década. También se detallan los objetivos de realizar este trabajo y la metodología a seguir.

En el Capítulo 2 se desarrolla una aplicación para desplegar mensajes en la PDA, el desarrollo de esta aplicación servirá para familiarizar al lector con el software de desarrollo de aplicaciones proporcionado por la empresa Palm, así como con algunas definiciones de archivos y elementos a utilizar a través de este trabajo.

En el Capítulo 3 se crea una nueva aplicación para establecer un envío de información a través del puerto serie; se muestra como configurar el puerto serie, así como los métodos de apertura, escritura y cierre del mismo.

En el Capítulo 4 se realizan pruebas a la aplicación desarrollada en el capítulo 3, enviando información a través del puerto y desplegándolo en la pantalla de una computadora. Además se realiza una última prueba a la aplicación de comunicación serial, desarrollada a lo largo del capítulo 3, mediante el control de un pequeño robot que cuenta con un puerto serie para la comunicación con el microprocesador que lo controla.

En el Capítulo 5 se presentan las conclusiones obtenidas al realizar este trabajo, además de posibles trabajos futuros en base al mismo.

Capítulo 2

Para el desarrollo de las aplicaciones requeridas a lo largo de esta tesis, será necesario el desplegar mensajes para interactuar con el usuario. El desarrollo de una aplicación que despliegue mensajes en la pantalla de nuestro PDA será de mucha ayuda para la comprensión de los primeros pasos a seguir para el desarrollo de aplicaciones para Palm OS.

En este capítulo se presentarán las herramientas para el desarrollo de aplicaciones para Palm Os, así como la forma de probar el buen desempeño de la aplicación creada mediante el uso de una herramienta llamada emulador. Además se dará una breve descripción de las partes principales de las que consta nuestra nueva aplicación.

Desplegado de mensajes en Palm OS

Para este trabajo de tesis se trabajará con una PDA de la marca PALM, modelo m125 con las siguientes características:

- Software de Palm OS versión 4.0
- OS de gran eficacia y 8 MB de memoria
- Puerto infrarrojo
- Formatos de exportación e importación de archivos.

La primera aplicación a desarrollar es una interfaz gráfica, con un botón que lanza un mensaje de alerta al dar un clic sobre el; esta primera aplicación servirá para familiarizarse con algunas definiciones que serán de gran ayuda a lo largo de este trabajo.

En este capítulo se realizará un ejemplo para ilustrar todos los detalles del software con que se cuenta para desarrollar aplicaciones en Palm OS 4.0.

Se requiere software para la implementación de aplicaciones llamado *Palm OS Developer Suite* (PODS), el cual esta disponible en la página de Palm [Palm OS 2006] y es totalmente gratuito.

Cabe mencionar que los nombres dados a los archivos creados pueden ser cambiados por el programador, solo es importante respetar las extensiones de dichos archivos.

Primeramente se creó un nuevo proyecto, (se nombró *control1*, pero el nombre dado al proyecto es decisión personal) y se siguieron los pasos que a continuación se detallan.

2.1 Un nuevo proyecto

Mediante el uso del PODS, lo primero que hay que hacer es ir a la barra de herramientas y en la opción de archivo (*File)* se realiza la selección para un nuevo archivo (*New*) como podemos observar en la figura 2.1. Esta opción ofrece una variedad de tipos de archivos que se pueden crear, para este caso se utilizó el *Managed Make 68K C/C++ Project*, debido a la compatibilidad del procesador con que cuenta la PDA.

Figura 2.1 Barra de herramientas para comenzar un proyecto de PODS

El sistema operativo con que cuenta la PDA utilizada es Palm OS 4.0. En el manual que se ha utilizado como guía, muestra dos tipos diferentes de código, para las versiones 5.4 (Garnet) y 6.0 (Cobalt) de Palm OS. La versión que más se asemeja a este sistema operativo es la 5.4. La serie 68 K de procesadores es el antecedente de la tecnología utilizada en las versiones Garnet y Cobalt: los procesadores ARM (por sus siglas en Ingles *Advanced RISC Machine*). Las versiones Garnet y Cobalt cuentan con una herramienta para la compatibilidad entre los diferentes tipos de procesadores, PACE (por sus siglas en inglés *Palm Application Compatibility Environment*), para emular un ambiente de 68 K. La opción de archivo *68 K PNO C/C++ Project* es útil para aplicaciones de procesamiento intensivo de datos, tales como juegos o codificación. *Protein C/C++ Project* es exclusiva para versiones Cobalt en adelante, y no hace uso de PACE.

Se debe dar un nombre al proyecto, en este caso por ejemplo se ha nombrado *control1*, pero esto es según lo desee el programador. La siguiente ventana, no debe modificarse por el momento ya que para la aplicación que se desarrolla son suficientes los valores por default, tales valores pueden observarse en la figura 2.2.

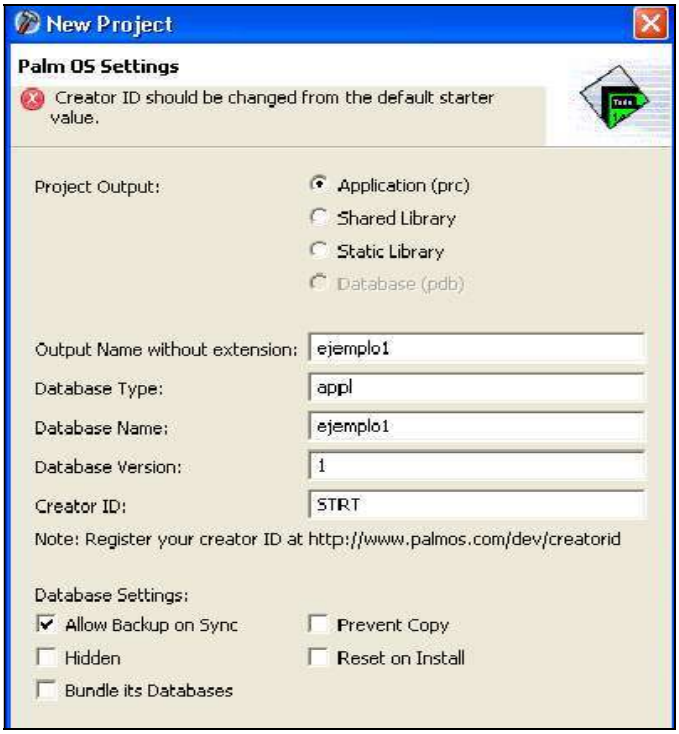

Figura 2.2 Ventana para ajuste de parámetros

La siguiente ventana, que se muestra en la figura 2.3, ofrece la posibilidad de seleccionar el contenido básico que tendrá el proyecto, y para la aplicación a desarrollar será suficiente la opción de *PilotMain Starter,* debido a que este tipo de proyecto da los archivos básicos para empezar a crear aplicaciones.

Se tienen otras 3 posibilidades para generar el código base del proyecto, las últimas dos (*simple aplication* y *sample aplication)* generan ejemplos de aplicaciones ya funcionales pero con tareas particulares, tales como desplegar un "hola mundo" o crear un sencillo juego de destreza. La primera opción de generación (*Source File Import Project*) crea solo un cascarón sin archivos, esta opción es útil para gente que desea importar algún código que ya haya creado con otro tipo de software.

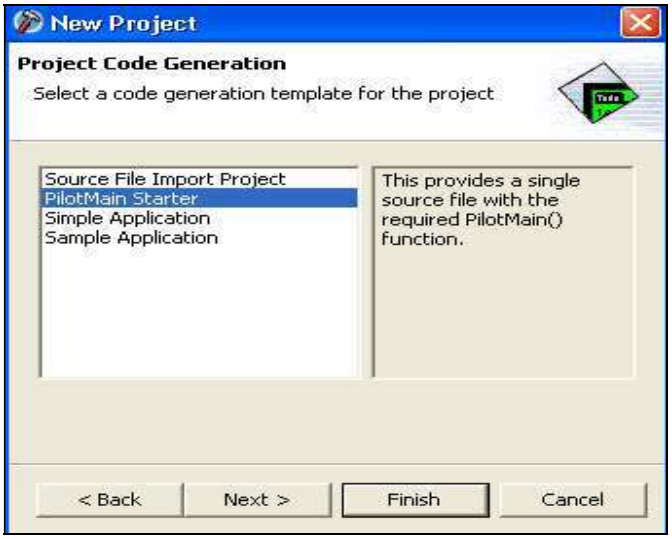

Figura 2.3 Ventana para la selección de las diferentes opciones de estructura

Esta opción de proyecto requiere de ciertos elementos que se agregarán a lo largo de la explicación de esta aplicación. Se debe crear una nueva carpeta para los recursos (botones, icono, formas, etc), será llamada "rsc" (el nombre dado a la carpeta es elección del programador), dicha opción se encuentra, en la opción nuevo (*new*) del menú archivo (*file*), en la barra de herramientas (figura 2.1). El directorio origen (*parent folder*) de la carpeta *rsc* será dentro del proyecto *control1*, esta opción aparecerá después de crear la carpeta, la cual se muestra en la figura 2.4.

Figura 2.4 Ventana para la creación de una nueva carpeta

2.2 El archivo de recursos (XRD)

El primer elemento que se debe crear, es el de los recursos del programa, para lo cual es necesario un nuevo archivo de recursos en blanco, así que nuevamente en la barra de herramientas (figura 2.1), seleccionamos un nuevo archivo de tipo *XRD*, como se puede observar en la figura 2.5. Se debe especificar el nombre del archivo (*AppResources.xrd* por ejemplo), y como directorio origen a la carpeta *rsc*, que fue creada (Ver capitulo 6 de [Foster y Bachmann 2005] para más detalles acerca de recursos).

Al finalizar la creación del archivo de recursos, *Palm OS Resource Editor* (PORE) es lanzado automáticamente.

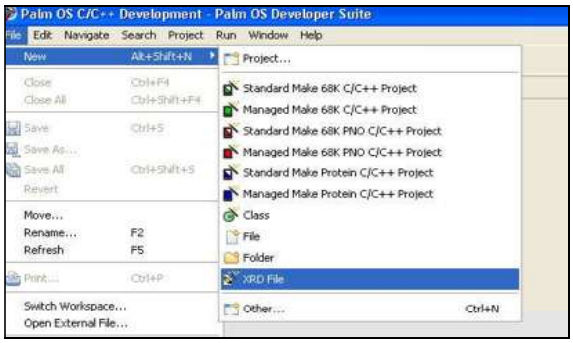

Figura 2.5 Barra de herramientas para la creación de un archivo de recursos vacío

Para esta aplicación se crearon 4 nuevos recursos:

- Form: pantalla
- Alert: mensaje de alerta
- App Icon Bitmap: icono
- App Icon Name String: etiqueta del icono.

En la siguiente sección se dan los detalles de dichos recursos.

2.2.1 El editor de recursos

El editor de recursos con que cuenta el PODS es el PORE, en la figura 2.6 se muestra la ventana principal del editor de recursos, y se obtiene automáticamente al instalar el PODS de forma completa.

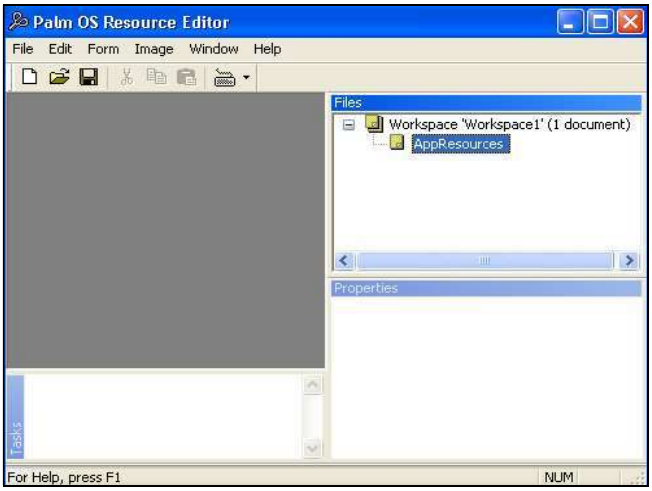

Figura 2.6 Ventana principal del editor de recursos

Para la creación de un nuevo recurso se debe ir a la barra de herramientas del PORE y seleccionar del menú de edición (*edit*) la opción de un nuevo recurso (*New Resource*), como se puede observar en la figura 2.7.

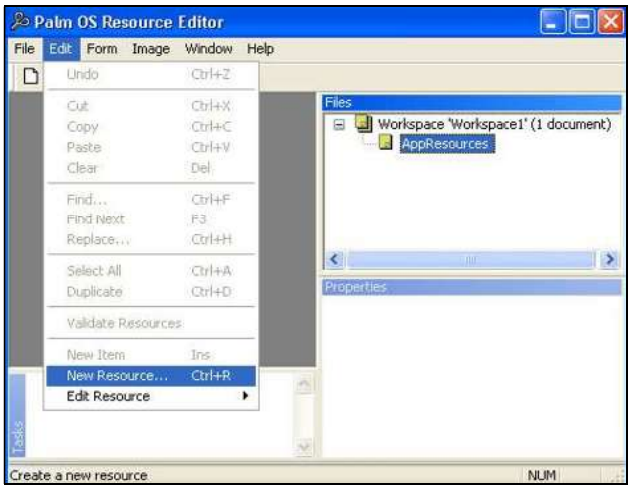

Figura 2.7 Barra de herramientas para la creación de un nuevo recurso

Pantalla

Para evitar confusiones a lo largo de este trabajo, se utilizará *FORM*, como significado de pantalla, (y viceversa) por la función que desempeña. Una pantalla, contiene todos los elementos gráficos que se despliegan durante la ejecución de una aplicación, tales como botones, mensajes de alerta, campos de texto, etc. En la página 18 de [Foster y Bachmann 2005] se dan mas detalles acerca de este recurso.

El primer recurso a crear será la pantalla. Siguiendo los pasos para la creación de un nuevo recurso antes descritos se selecciona un nuevo recurso del tipo pantalla, como puede verse en la figura 2.8.

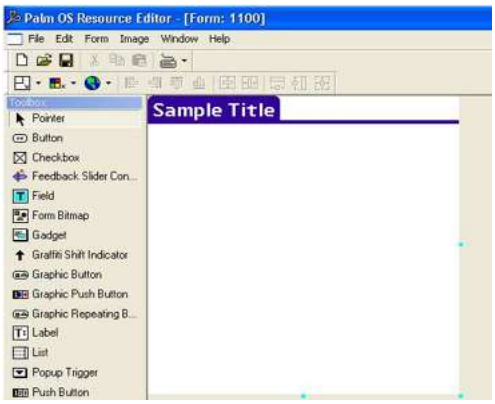

Figura 2.8 Ventana mostrada para la creación de una pantalla

A la pantalla principal de la aplicación se agregó un botón, para esto, en la parte izquierda de la ventana que aparece al crear la pantalla existe una barra de herramientas con una variedad de elementos que pueden agregarse a la ventana; se selecciona la opción de crear un nuevo botón. En la figura 2.9, se puede observar un botón creado para la pantalla principal.

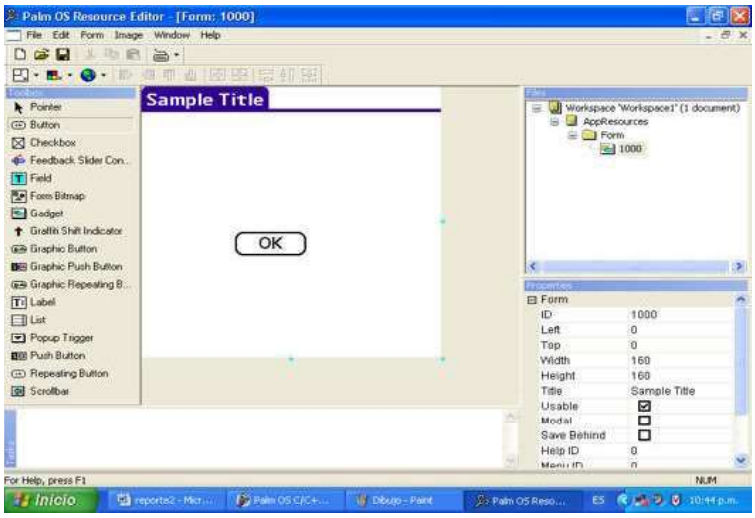

Figura 2.9 Ventana presentada durante la creación de un botón

Para darle forma al botón, se debe arrastrar manteniendo presionado el ratón, en la posición de la pantalla donde se quiera localizar el botón.

El elemento creado tiene ciertas características o propiedades (*properties*) por default, las cuales pueden editarse en la parte inferior derecha de la ventana (figura 2.9). Si se requiere modificar las propiedades de algún elemento que se haya creado solo debe tenerse cuidado de seleccionar el elemento que desea modificarse, dando clic izquierdo sobre el objeto.

El tamaño y la posición del botón también pueden ser modificados con seleccionar la imagen dibujada del botón y estirarlo o moverlo según plazca.

Mensaje de Alerta

El siguiente recurso a crear, es un mensaje de alerta (*Alert*), que se utilizará para desplegar un pequeño mensaje al dar clic sobre el botón.

Repitiendo los pasos para la creación de un nuevo recurso, como se realizó para la creación de la ventana, ahora se selecciona un nuevo recurso del tipo mensaje de alerta. En la figura 2.10 puede verse un mensaje de alerta creado con el editor de recursos.

Figura 2.10 Mensaje de alerta creado en el editor de recursos

El editor de recursos crea automáticamente un botón, el cual cierra el mensaje al presionarlo; no es necesario escribir código para el manejo de este botón.

Si se desea realizar alguna modificación a cualquier recurso previamente creado, en la parte superior derecha de la pantalla del PORE, se encuentra una ventana de nombre *files*, en esta se encuentran una serie de carpetas que contienen en su interior los recursos creados, y con dar doble clic sobre el archivo puede hacerse modificaciones al recurso. Otra forma de

movilizarse hacia algún recurso creado, es mediante la barra de herramientas, con la opción de ventana (window), ya que esta da un listado de los recursos creados.
. En la figura 2.7 podemos observar que esta opción se encuentra antes de la opción de ayuda (help) del PORE.

Icono

Para poder crear un icono que aparezca en la lista de aplicaciones del PDA, se necesitan dos nuevos recursos: *App Icon Bitmap* y *App Icon Name String*.

El *App Icon Bitmap* sirve para crear el icono que aparecerá en la pantalla principal del PDA. Siguiendo los pasos para la creación de un nuevo recurso (figura 2.7) debe seleccionarse un nuevo recurso de tipo icono (*App Icon Bitmap*).

Existen ciertas propiedades que deben ajustarse, del lado izquierdo de la ventana se encuentra el listado de propiedades que se editaran (figura 2.11). Lo primero, serán las dimensiones del nuevo mapa de bits (Bitmap), para lo cual hay que dar clic en el botón de ajustes (*Set…*). Ajustando el ancho (Width) y el alto (Height), los valores dados por default son 22 para alto y ancho, pero puede ser redimensionado por el programador si así lo desea. Una vez ajustadas las dimensiones se presiona OK.

Un mapa de bits, es una imagen para mostrar en forma de icono. Para crear un nuevo mapa de bits para el icono se presiona el botón para agregar un nuevo elemento (*Add…*), y será del tipo *1x, Black & White*; solo es necesario dar clic en la casilla de verificación y a continuación presionar OK. Posiblemente ya se encuentren por default estas propiedades. En caso de que la pantalla de la PDA tenga alta resolución y una gama de colores diferentes de los grises con los que cuenta los modelos de PDA con que se están trabajando en este trabajo de tesis, es posible utilizar otras opciones de mapas de bits. Ahora aparecerá una pequeña área de dibujo y en la parte derecha de la ventana una paleta de colores (*Color Palette*), en el caso concerniente a los modelos de PDA utilizados, solo serán los colores blanco y negro. En la parte superior de la ventana se encuentran una serie de herramientas de dibujo, de las cuales se utilizará la indicada para crear una elipse rellena de color (*Filed Elipse Tool*). El usuario puede usar alguna otra herramienta de dibujo, si así lo desea. Dando clic sobre la herramienta a utilizar y de la paleta de colores se selecciona el color negro y sobre el área de dibujo se comienza a dibujar el icono. En la figura 2.11 se muestran las ventanas desplegadas asociadas a los respectivos botones y herramientas que

se utilizaron para la creación del icono. El programador puede diseñar su propio icono de acuerdo a sus gustos y necesidades.

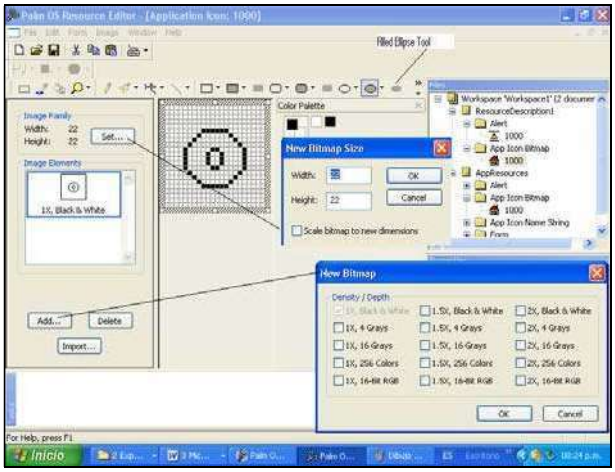

Figura 2.11 Ventanas desplegadas por el PORE durante la creación de un icono

Etiqueta del icono

Ahora solo falta crear una etiqueta de texto que presente el nombre de la aplicación, para esto se crea un nuevo recurso de tipo *App Icon Name String*.

Al crear el nuevo recurso, aparecerá una ventana como la mostrada en la figura 2.12, con el siguiente código:

```
<APP_ICON_NAME_RESOURCE RESOURCE_ID="1000">
        <TEXT> "Sample Application" </TEXT>
</APP_ICON_NAME_RESOURCE>
```

Solo debe cambiarse la frase *Sample Application* que se encuentra entre comillas, por la palabra que describa a la aplicación ó el nombre dado a la misma.

En la figura 2.12 puede observarse el código modificado en el editor de recursos, reemplazando las líneas antes mencionadas.

Figura 2.12 Ventana del editor de recursos para la creación de una etiqueta

Se debe tomar en cuenta el poco espacio que se tiene en la pantalla del PDA, se tiene un espacio de 9 caracteres para las etiquetas. Al sobrepasar este numero, el texto faltante se sustituye por puntos suspensivos en la pantalla de la PDA. En la figura 2.13, puede observarse el icono y su respectiva etiqueta en la pantalla de una PDA.

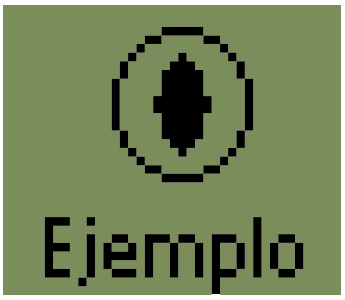

Figura 2.13 Icono y etiqueta en la pantalla de un PDA

2.3 El archivo de cabecera

Ahora se necesita declarar todos los recursos que se han creado, mediante un archivo de cabecera que se creará a continuación. Un archivo de cabecera (*header*), es aquel donde se declaran los recursos a utilizar dentro de una aplicación. Mediante el PODS debe crearse un nuevo archivo desde la barra de herramientas (figura 2.1) en el menú archivo (*file*), con el comando nuevo (*new*). Se selecciona la creación de un nuevo archivo de texto y debe especificarse el directorio origen del archivo de cabecera. La opción de un archivo de texto aparece como "*file"* en el menú de opciones que se despliega. La ubicación del archivo será *ejemplo/src* y puede ser escrito en el campo de texto que se encuentra en la parte superior de la ventana (*Enter or select the parent fólder:*) como se puede observar en la figura 2.4. El nombre del archivo de cabecera es desición del programador, solo la extensión del

archivo deberá ser ".h", ya que es la extensión correspondiente a un archivo de cabecera. Por ejemplo, nosotros lo nombraremos *AppResources.h.*

Se declaran los recursos mediante la instrucción **#define**, seguido del nombre que se asignará al recurso y por ultimo el ID (por sus siglas del inglés *identifier*). El ID es un numero de identificación generado por el PORE, y para conocerlo se debe regresar a revisar el editor de recursos. Puede regresarse al editor de recursos dando doble clic sobre el archivo, el cual podemos encontrar en la parte izquierda de la pantalla del PODS.

Ya en el editor de recursos, puede encontrarse los ID de los recursos en la ventana de propiedades que se encuentra en la parte inferior derecha de la ventana, (tal como se puede observar en la figura 2.9) y tiene el nombre de *ID* o *Resource ID*. Los nombres declarados en el archivo de cabecera, deberán coincidir con los nombres de los recursos que se utilicen en el código fuente de la aplicación (archivo C).

El código del archivo *AppResources.h* quedará de la siguiente forma:

```
#define FormPrincipal   1000
#define BotonPrincipal  1000
#define MensajeAlerta   1000
#define Icono           1000
```

Los números de identificación pueden coincidir entre si, siempre y cuando sean diferentes tipos de recursos, por ejemplo dos botones no pueden tener un ID con valor de 1000, pero puede coincidir que un botón y un mensaje de alerta coincidan en este número.

2.4 Código fuente

Ahora va a crearse el código para la aplicación, para lo cual se modificará el archivo *AppMain.c*, ya que el cascaron creado por el PODS solo contiene ciertas funciones básicas.

Recordando un poco, la opción seleccionada para la generación del proyecto (*PilotMain Starter*) generó ajustes básicos y archivos prediseñados, y un sencillo archivo tipo C/C++ con la función *PilotMain()* codificada. Ahora queda agregar el código necesario para el manejo del evento del botón de la pantalla principal de la aplicación.

En la primera parte del código, se encuentran dos directrices para la inclusión de datos y declaraciones de recursos a utilizar, mediante el comando *#include*

```
#include <PalmOS.h>
#include "AppResources.h"
```

El archivo PalmOS.h es para incluir librerías y ciertos archivos necesarios para la ejecución del programa.

AppResources.h define los recursos que se crearon previamente para la aplicación, tales como botón, señal de alerta, icono, etc.

La función principal que Palm OS ejecuta, en la aplicación, es la función *PilotMain*, como sucede con la función *main* en *ansi* C, la cual luce de la siguiente manera en *control1:*

```
UInt32 PilotMain(UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
{
	Err error = errNone;
	switch (cmd) {
		case sysAppLaunchCmdNormalLaunch:
			if ((error = ComienzaAplicacion()) == 0) {
				ProcesaEventoAplic();
				DetenAplicacion();
			}
			break;
		default:break;
	}
	return error;
}
```

Los nombres que se darán a las funciones descritas a continuación, son elección del programador, pero es una buena idea que el nombre de la función dé una idea acerca de lo que esta parte de código realiza.

Durante una ejecución normal del programa, la primera función llamada es *ComienzaAplicacion* y luce de la siguiente manera:

```
static Err ComienzaAplicacion(void)
{
	FrmGotoForm(FormPrincipal);
	return errNone;
}
```

La función *ComienzaAplicacion*, tiene la responsabilidad de llamar al recurso FormPrincipal, el cual contiene la interfaz de la aplicación. La función *FrmGotoForm* cierra la ventana (*form*) principal de la PDA, para poder cargar y abrir la pantalla principal. Regresando un poco al *PilotMain,* si no existe ningún error durante la ejecución, la siguiente función a llamar es *ProcesaEventoAplic*.

```
static void ProcesaEventoAplic(void)
{
      EventType   event;
      do {
            EvtGetEvent(&event, evtWaitForever);
            if (SysHandleEvent(&event))
                  continue;
            if (ManejaEventoAplicacion(&event))
                  continue;
            FrmDispatchEvent(&event);
      } while (event.eType != appStopEvent);
}
```

La función *ProcesaEventoAplic* es la encargada del procesamiento de los eventos recibidos por la aplicación. Los eventos son organizados mediante una estructura de datos tipo cola [Tenenbaum, Langssam y Augenstein 1993], lo cual permite ir procesando un evento a la vez.

La función *ProcesaEventoAplic* procesa tres tipos diferentes de eventos o casos, los cuales son despachados en el siguiente orden:

1. `SysHandleEvent` se encarga de los eventos del sistema, tales como presionar el botón de encendido de la PDA.
2. `ManejaEventoAplicacion` carga los recursos necesarios para nuestra pantalla principal.
3. `FrmDispatchEvent` maneja la acción que debe realizar nuestra aplicación.

Una vez que el evento ha sido manejado o tratado, el siguiente evento que se encuentra en la cola pasa a ser analizado, esto sucede consecutivamente mientras no llegue a la cola un evento de tipo *appStopEvent* para detener la ejecución del método y regresar al *PilotMain*.

Los primeros dos tipos de despachadores de eventos son parte de PalmOS, pero el *ManejaEventoAplicacion*, debe ser creado por cuenta propia, el programador debe tener en

cuenta las acciones que deben ser realizadas por la aplicación y los recursos que deben cargarse.

El código de *ManejaEventoAplicacion* se muestra a continuación:

```
static Boolean ManejaEventoAplicacion(EventType* pEvent)
{
	UInt16		formId;
	FormType*	pForm;
	Boolean		handled = false;

	if (pEvent->eType == frmLoadEvent) {
		// Load the form resource.
		formId = pEvent->data.frmLoad.formID;

		pForm = FrmInitForm(formId);
		FrmSetActiveForm(pForm);
		switch (formId)
		 {
			case FormPrincipal:
				FrmSetEventHandler(pForm,ManejaEventoFormPrinc);
				break;
			default:break;
		}
		handled = true;
	}
	return handled;
}
```

La función *ManejaEventoAplicacion*, tiene dos tareas primordiales:

- Inicializar el recurso pantalla
- Manejar ciertos eventos del recurso pantalla

La inicialización del recurso pantalla, se realiza mediante dos funciones:

- *FrmInitForm(formId)*
- *FrmSetActiveForm(pForm)*

El primero de ellos carga en la memoria el recurso y lo inicializa, regresando un apuntador a dicho recurso. La siguiente función toma ese apuntador y hace que el recurso pantalla se

vuelva activo. Una pantalla activa puede desplegar recursos tales como mensajes, botones, campos, etc. En Palm OS solo puede haber una pantalla activa a la vez.

Una vez inicializada y activa la pantalla, es necesario manejar los eventos de dicho recurso. En este caso solo se cuenta con una pantalla, por lo cual solo es necesario un manejador de eventos; en aplicaciones mas complejas, con mas de una pantalla, debe existir un manejador para cada pantalla existente en la aplicación.

La función encargada de manejar los eventos de la pantalla principal es *ManejoEventoFormPrinc*. Esta función no es muy complicada, debido a que solo manejaremos el evento del botón creado, y se detalla a continuación:

```
static Boolean ManejaEventoFormPrinc(EventType* pEvent)
{
	Boolean	handled = false;
	FormType*	pForm;
	switch (pEvent->eType) {
		case frmOpenEvent:
			pForm = FrmGetActiveForm();
			FrmDrawForm(pForm);
			handled = true;
			break;
		case ctlSelectEvent:
	switch(pEvent->data.ctlSelect.controlID)
			{
				case BotonPrincipal:
				DespliegaMensaje(MensajeAlerta);
				handled = true;
				break;
				default:break;
			}
			break;
		default:break;
	}

	return handled;
}
```

La aplicación que se esta desarrollando hace un llamado a una función muy importante y necesaria para la presentación del mensaje mediante el recurso mensaje de alerta, tal función

es: *DespliegaMensaje*; y es llamada en caso de dar clic sobre el botón que se creó en la pantalla principal.

La función *void DespliegaMensaje(UInt16 alertID)*, recibe como parámetro el número de identificación (Id) del recurso Alerta, y realiza una llamada dentro de si a la función *FrmCustomAlert(alertID, NULL, NULL, NULL)*, la cual sirve para desplegar el mensaje en el correspondiente recurso mensaje de Alerta.

Puede observarse que la función descrita anteriormente recibe 4 parámetros, para el caso de la aplicación desarrollada, los últimos 3 parámetros serán *NULL* ya que no son de utilidad para la aplicación. En caso de tener curiosidad acerca de estos tres últimos parámetros, sírvase de consultar *Palm OS 68K API Documentation*, el cual se encuentra dentro de la ayuda que viene incluida dentro del PODS.

La manera en la cual esta aplicación cierra todos los eventos, es mediante la función *DetenAplicacion*, la cual cierra todas las pantallas activas en la aplicación

Después de una breve descripción del código (el código completo se encuentra en el apéndice A), es hora de ver como trabaja la aplicación, para lo cual se utilizará la herramienta de emulación con la que cuenta PODS: *Palm OS Emulator* (POSE).

2.5 El emulador

Un emulador es un programa que funciona como una verdadera Palm. Esta aplicación le puede servir para visualizar en la computadora el funcionamiento de una Palm y hacer las mismas cosas que haría en una verdadera, como instalar aplicaciones, llevar una agenda, tener sus contactos, etc. [Todo Palm 2006]

Después de haber creado los archivos, recursos y código, es tiempo de observar como trabajará la aplicación en la PDA, para esto, primero se visualizará por medio del POSE, para después descargar la aplicación a la PDA.

Cada vez que se modifique el código, es necesario recargar el proyecto. Esto puede hacerse de dos formas, presionando F5, o en la barra de herramientas del menú archivo seleccionando la opción “refrescar” (*Refresh*).

De la barra de herramientas, se selecciona la opción de “correr” (*RUN...)* del menú del mismo nombre, como podemos observar en la figura 2.14.

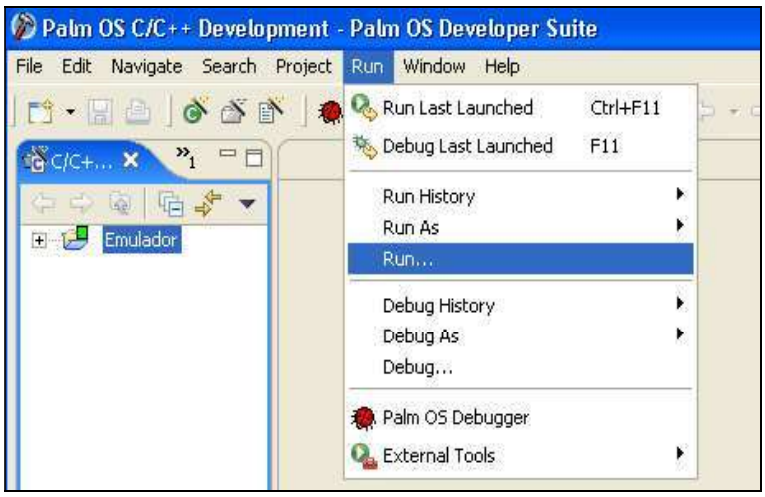

Figura 2.14 Despliegue de la barra de herramientas para emular la aplicación

Para emular la aplicación, debe crearse una nueva configuración; se debe seleccionar el icono con la figura de una PDA y presionar el botón NUEVO (figura 2.15). Si se realizaron correctamente los pasos anteriores, debe aparecer una nueva aplicación para Palm OS con en el nombre del proyecto (en este caso control1), como se puede observar en la figura 2.15.

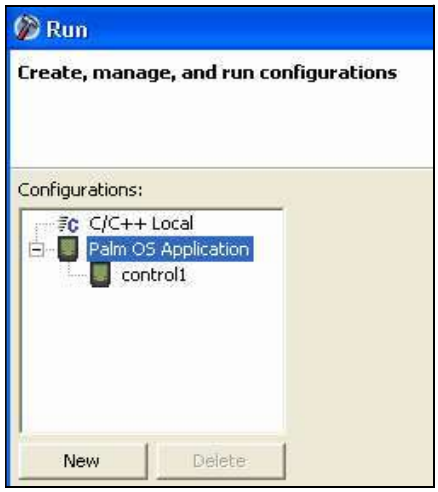

Figura 2.15 Ventana para la creación, manejo, y ejecución de una nueva configuración

Debe seleccionarse la opción con el nombre del proyecto, para desplegar algunas configuraciones; para la emulación de la aplicación se selecciona como destino de la aplicación, al emulador de Palm OS (figura 2.16). Por el momento, no es necesario

modificar los valores de los demás ajustes, los valores dados por default son suficientes para la emulación.

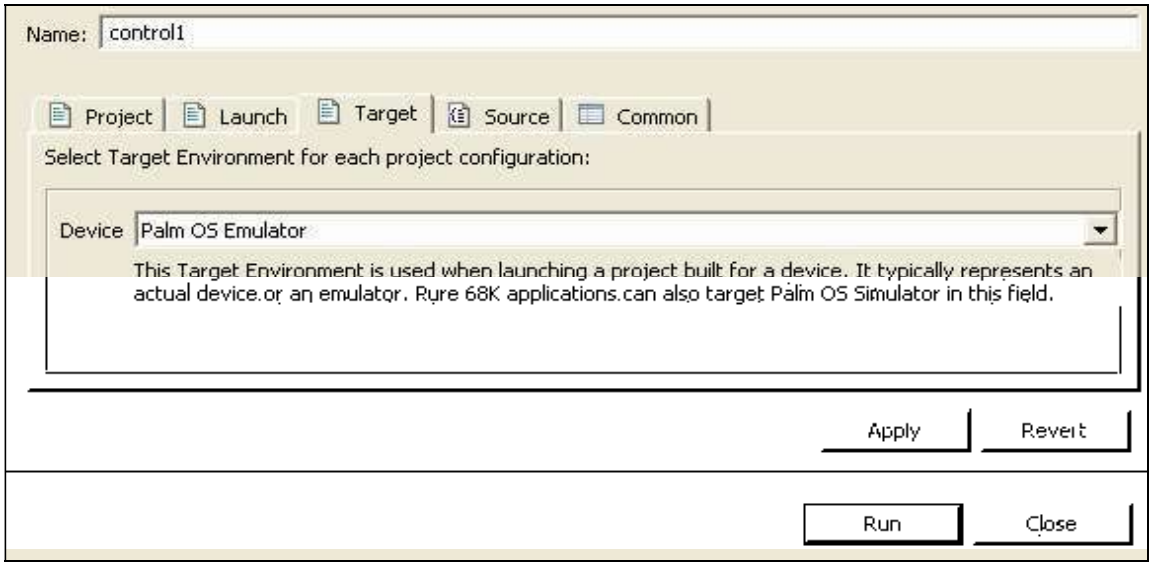

Figura 2.16 Ventana de ajustes para la nueva configuración

Se presiona el botón “correr”, situado en la parte inferior derecha de la misma ventana (figura 2.16). La primera vez que es lanzado el emulador, aparece una ventana para ajustar nuestra nueva sesión de emulación.

Son cuatro los ajustes que deben cambiarse en la ventana antes mencionada y son los siguientes:

- ROM File es el archivo que contiene el software del sistema Palm OS
- Device es el modelo de la PDA (Palm m105).
- Skin es el tipo de carátula.
- RAM size es el tamaño de memoria RAM con que cuenta la PDA, en este caso será de 8Mb (8192Kb).

En la figura 2.17 se muestran los ajustes necesarios para la emulación de la aplicación.

Figura 2.17 Ventana de ajustes para el emulador

La ubicación de los archivos ROM, puede cambiar dependiendo de la locación donde se instaló el PODS, aunque regularmente es la siguiente:
C:\Archivos de programa\PalmSource\Palm OS Developer Suite\PalmOSTools\Palm OS Emulator\ROMS

Después de ajustar estos parámetros, se presiona el botón OK, esto lanzará dos ventanas, una con el emulador y otra que nos indica que la aplicación se esta lanzando (*Launching*). La ventana de lanzamiento dice que si se quiere detener el proceso presione cancelar, o que cuando la ventana del emulador haya sido lanzada presione OK para continuar con la emulación, como puede observarse en la figura 2.18.

Figura 2.18 Ventana para cancelar o continuar la emulación

Si los ajustes fueron los correctos, deberá aparecer un emulador con la imagen de la PDA y desplegando en su pantalla la aplicación que se diseñó, como puede observarse en las figuras 2.19 y 2.20.

Figura 2.19 Ventana del emulador desplegando el botón creado en la aplicación

Figura 2.20 Venta del emulador desplegando mensaje de alerta

Para cerrar el emulador, solo se da clic derecho sobre el emulador, y presiona salir. Lo anterior, lanzará una ventana, la cual permite salvar la sesión de emulación. En caso de salvar el archivo, se creará uno nuevo de tipo sesión de emulador Palm, tal y como se muestra en la figura 2.21.

Figura 2.21 Icono de archivo de sesión de emulador

Este nuevo tipo de archivo, permite lanzar el emulador con solo abrir el archivo tipo sesión, sin necesidad de abrir el PODS. En caso de no salvar la sesión de emulación, se tendrá que seguir todos los pasos que se han detallado en esta sección, cada vez que se quiera lanzar el emulador.

2.6 Conclusiones

Con la creación de la aplicación descrita en este capítulo, pueden ponerse en práctica el uso de las herramientas proporcionadas por la empresa Palm para el desarrollo de aplicaciones para el sistema operativo Palm OS con que cuentan sus equipos. No es necesario el desarrollo de una aplicación muy compleja para mostrar el uso de este nuevo software, sin embargo es necesario hacer uso de referencias extras, aparte de este trabajo, tales como son manuales y documentos de referencia tales como API.

El aspecto de la programación en C++, no representa una gran limitante para programadores que solo están familiarizados con lenguajes de programación diferentes. Solo se debe tener especial cuidado con el uso de apuntadores en lenguaje C++; para esto existe mucha bibliografía útil en aclarar las dudas concernientes a esta parte de la programación. Como ejemplo, puede consultar el capítulo 7 de [Deitel y Deitel 1995].

El programador puede darle la forma y aspecto que guste a sus aplicaciones diseñadas, en este trabajo solo se dio una sugerencia de cómo hacer uso de las herramientas para el desarrolla de software para PDA's. Es importante la curiosidad para poder explotar al máximo todas las ventajas que ofrece un software como es el PODS. Se sugiere en gran medida el consultar manuales del tipo de [Foster y Bachmann 2005], para referencia a temas que no se cubren en esta trabajo, concernientes al diseño de aplicaciones para PDA's.

Capitulo 3

En este capítulo se desarrollará una aplicación que establezca una comunicación a través del puerto serie. Se dan algunas definiciones necesarias para la comprensión de una comunicación serial. Se aclararán ciertas configuraciones necesarias para establecer una correcta comunicación.

Comunicación Serial

Un puerto serie es un interfase de comunicaciones entre ordenadores y periféricos en donde la información es transmitida bit a bit enviando un solo bit a la vez. (En contraste con el puerto paralelo que envía varios bites a la vez) [Wikipedia 2006].

Dado que el objetivo principal del presente trabajo de tesis es desarrollar una aplicación para la comunicación entre el PDA y algún elemento de control, por medio del puerto serie, en este capítulo se creará un nuevo proyecto para ejemplificar las principales herramientas de Palm OS para establecer una comunicación serial.

Las acciones a realizar para establecer la comunicación serial, pueden ser resumidas de la siguiente manera:

- Configuración del puerto
- Apertura del puerto
- Escritura en el puerto
- Cierre del puerto

Se retomará parte del código propuesto en el capítulo 2 para el desarrollo de esta nueva aplicación. Las funciones que no tendrán cambios a lo largo del desarrollo de esta aplicación, son los siguientes:

- ComienzaAplicacion
- DespliegaMensaje
- ManejaEventoAplic
- ProcesaEventoAplic
- PilotMain

Se dará detalle de los cambios efectuados a los métodos requeridos a lo largo de este capítulo.

3.1 Configuración del puerto

Para establecer una comunicación, en este caso a través del puerto serie, debe haber una compatibilidad en la configuración, tanto del elemento receptor como el emisor. Existen ciertas configuraciones dadas por default por el sistema operativo Palm OS, las cuales se enlistan a continuación para poder ajustar el elemento receptor, el cual en este caso será el software de Terminal llamado "*Hyper Terminal*" con la que cuenta la mayoría de las versiones de Windows:

- Bits de datos: 8
- Paridad: ninguno
- Bits de parada: 1
- Control de flujo: ninguno

Estos son los parámetros que se deben ajustar en la Terminal, mas adelante, cuando se realicen las pruebas a la aplicación. Es posible cambiar la configuración del puerto de la PDA, agregando algo de código, detallado en la página 694 de [Foster y Bachmann 2005], en este caso la configuración por default será suficiente.

Se deben declarar algunas variables globales, necesarias para la configuración del puerto, tales son: el número de identificación del puerto y una variable de tipo booleano que indica el estado del puerto, ya sea abierto ó cerrado.

La variable que indica el ID del puerto debe tomar el valor de 0x8000, este valor es el indicado para trabajar con el puerto serial y es compatible con el procesador de la PDA. Para mayor información acerca del número de identificación del puerto, puede consultar la página 678 de [Foster y Bachmann 2005].

La variable que indica el estado del puerto, tomará valores de verdadero si se encuentra abierto el puerto y falso si es que se encuentra cerrado. Una buena opción, será declarar estas variables enseguida de incluir las librerías necesarias para el manejo del puerto, tal como se muestra a continuación:

```
#include <PalmOS.h>
#include "AppResources.h"
```

```
#include "SerialMgr.h"
#include "SystemResources.h"
//variables globales
UInt16 DireccionPuerto = 0x8000;
//verdadero si el puerto esta abierto
//falso si el puerto esta cerrado
Boolean EstadoPuerto;
```

La librería que contiene las funciones para el manejo del puerto, se encuentra en el archivo SerialMgr.h. Este archivo es parte de las librerías que proporciona el PODS para el control de las funciones del puerto serie.

3.2 Apertura del puerto

La función para la apertura del puerto debe especificar la velocidad a la cual se va a llevar acabo la comunicación en bits por segundo. A continuación se puede observar la función creada para la apertura del puerto:

```
static Err AbrePuerto (void)
{
      Err error = 0;
      // Open the serial port with an initial baud rate of
      // 9600.
      error = SrmOpen(serPortCradlePort, 9600, &DireccionPuerto);
      ErrNonFatalDisplayIf(error == serErrBadPort,"serErrBadPort");
      switch (error)
      {
            case errNone:
            break;
            case serErrAlreadyOpen:
            SrmClose(DireccionPuerto);
            DespliegaMensaje(AlertaPuertoOcupado);
            return error;
            break;
            default:
            DespliegaMensaje(AlertaPuertoAbierto);
            return error;
            break;
```

```
        }
        EstadoPuerto = true;
// limpia el Puerto en caso de haber basura en el
        SrmReceiveFlush(DireccionPuerto, 100);
        return error;
}
```

La función principal de este código es *SrmOpen*, la cual recibe como primer argumento, un identificador del puerto, los cuales están especificados en la página 678 de [Foster y Bachmann 2005], y para este caso, tomará el valor de *serPortCradlePort*, esta variable ya se encuentra definida en las librerías que se incluyeron al principio del código. Como segundo parámetro recibe la velocidad de transmisión dada en bits por segundo; una velocidad de 9600 es adecuada para la aplicación. El último parámetro que recibe esta función es el número del puerto, el cual declaramos como variable global con el nombre de *DireccionPuerto* al principio del código.

En caso de que el puerto ya se encuentre previamente abierto, la aplicación hace un llamado a la función *SrmClose*, ya definido en las librerías de Palm OS incluidas, para cerrar el puerto y poder desplegar un mensaje de alerta adecuado. En cualquier otro caso de error al momento de la apertura, la aplicación despliega un mensaje de error general (*AlertaPuertoAbierto*). Los mensajes de alerta *AlertaPuertoOcupado* y *AlertaPuertoAbierto* son recursos que deben crearse y declararse como previamente se explico en las secciones 2.2 y 2.3 de este trabajo.

Si no ocurre ningún error durante la apertura del puerto, la variable global *EstadoPuerto* tomará un valor de verdadero, lo cual indica que el puerto ha sido abierto y se encuentra listo para la escritura.

Por último, en la función *AbrePuerto*, se hace uso de la función *SrmReceiveFlush* la cual limpia el puerto en caso de existir basura en él y recibe como parámetros el número del puerto, así como un intervalo de tiempo. En caso de encontrar información, basura dentro del puerto, la cuenta es reiniciada; en caso de cumplirse el tiempo indicado, el método llega a su fin y regresa el tipo de error en caso de haberlo. Este intervalo de tiempo es dado en *ticks*. Los *ticks* son una medida de tiempo especificada por el Palm OS y tienen una equivalencia de alrededor de 100 *ticks* por cada segundo.

3.3 Escritura del puerto

Una vez que el puerto ha sido debidamente configurado y abierto, se encuentra en condiciones para su escritura; en el caso de esta aplicación se hace uso de cinco botones que mandaran cada uno un carácter diferente necesario para cada orden que se ejecutará por el robot. La función principal a utilizar para llevar acabo la escritura del puerto, es *SrmSend* y recibe cuatro parámetros. El primero de ellos, es la variable global que contiene el número de puerto; el segundo de ellos, es un apuntador al contenido que se desea enviar a través del puerto; el tercero de ellos debe indicar la longitud en bytes del elemento a enviar; si el método encuentra un error durante su ejecución, este es regresado por medio del cuarto parámetro.

La estructura del código creado para la escritura del puerto se muestra a continuación:

```
static void EscribePuerto (int opcion)
{
	Err error;
	Char *text = NULL;

	if (EstadoPuerto == false)
	return;

	if(opcion == 1)
	{
		text = "A";
	}
	else if(opcion == 2)
	{
		text = "R";
	}
	else if(opcion == 3)
	{
		text = "D";
	}
	else if(opcion == 4)
	{
		text = "I";
	}
	else if(opcion == 5)
	{
		text = "P";
	}


	if (text)
	{
		SrmSend(DireccionPuerto, text, StrLen(text), &error);
	}
}
```

La función desarrollada en este capítulo para la escritura del puerto (EscribePuerto), recibe como parámetro un entero que nos indicará cual de los cinco botones fue presionado y por ende que información será enviada a través del puerto mediante una serie de condicionales de tipo *if else*.
El apuntador de la información que deseamos enviar hará referencia a los caracteres constantes que se enviaran dependiendo del botón presionado.

Para obtener el tercer parámetro de la función *SrmSend*, la longitud de la información a enviar, será útil la función *StrLen*. *StrLen* recibe como parámetro un apuntador a la información que deseamos enviar.

Es oportuno mencionar que la aplicación no cuenta con una protección en caso de que el puerto no haya sido debidamente abierto antes de la escritura. Si el puerto se encuentra cerrado y el usuario intenta enviar información a través del puerto, no sucederá nada en lo absoluto. Esto no genera ningún error o problema, solo se debe prestar atención en abrir el puerto antes de mandar información a través de el.

3.4 Cierre del puerto

El mantener abierto el puerto, sin requerirlo, provocará la descarga de baterías de la PDA, por lo cual es necesario cerciorarse de cerrar el puerto una vez cerrada la aplicación, o en caso de no requerir que el puerto se encuentre abierto.

El código de la función creada para el cierre del puerto serie se muestra a continuación:

```
static void CierraPuerto (void)
{
      Err error;
      error = SrmSendWait(DireccionPuerto);
      ErrNonFatalDisplayIf(error == serErrBadPort,"SrmClose: bad port");
      if (error == serErrTimeOut)
      DespliegaMensaje(AlertaPuertoTiempofin);
      SrmClose(DireccionPuerto);
      EstadoPuerto = false;
}
```

La función principal del código mostrado, es la función *SrmClose*, mencionada previamente en la sección 3.2, el cual se encarga del cierre del puerto. En caso de

presentarse un error durante el cierre del puerto, la aplicación despliega un mensaje de alerta (*AlertaPuertoTiempofin*), el cual debe ser declarado previamente, tal como se explicó previamente en las secciones 2.2 y 2.3 de este trabajo. Una vez cerrado el puerto, la variable global que indica el estado del puerto, toma un valor de falso, de lo cual se deduce que el puerto se encuentra cerrado.

Una vez terminado el uso de la aplicación, es importante cerciorarse que el puerto se encuentre debidamente cerrado, para esto, es útil realizar una modificación a la función *DetenAplicacion*, agregando la función *CierraPuerto* para el cierre del puerto una vez cerrada a aplicación, como se puede observar a continuación:

```
static void DetenAplicacion(void)
{
      // Close all the open forms.
      FrmCloseAllForms();
      CierraPuerto();
}
```

3.5 Uso del puerto serie

Para el empleo de las funciones previamente creadas, se hará uso de siete botones, los cuales deben ser creados y declarados previamente.

La creación de un nuevo recurso se explica en la sección 2.2.1. El objetivo es desarrollar una aplicación similar a la que se muestra a continuación en la figura 3.1.

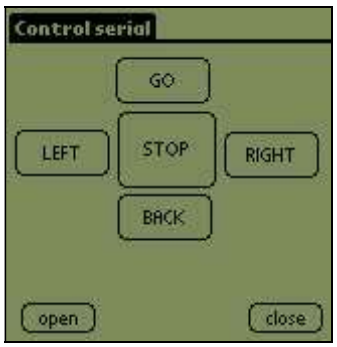

Figura 3.1 Ventana principal de aplicación para comunicación serial

La figura 3.1 puede servirle al programador como guía para la creación de los recursos necesarios para el desarrollo de esta aplicación. El programador tiene la libertad de darle la apariencia que desee a sus aplicaciones, solo es importante ser conciente del código fuente dado en C++ para el buen desempeño de la aplicación.

El uso de las funciones de apertura, escritura y cierre del puerto, debe ser incluido en la función *ManejaEventoFormPrinc*, mediante el uso, en este caso particular, de botones.
La función *ManejaEventoFormPrinc* fue previamente discutida en la sección 2.4 de este trabajo y debe ser modificada de la siguiente forma:

```
static Boolean ManejaEventoFormPrinc(EventType* pEvent)
{
      Boolean     handled = false;
      FormType*   pForm;

      switch (pEvent->eType) {

            case menuEvent:
                  return MainFormDoCommand(pEvent->data.menu.itemID);

            case frmOpenEvent:
                  pForm = FrmGetActiveForm();
                  FrmDrawForm(pForm);
                  handled = true;
                  break;

            case ctlSelectEvent:
      switch(pEvent->data.ctlSelect.controlID)

                  {

                        case BotonAbrePuerto:
                        AbrePuerto();
                        handled = true;
                        break;

                        case BotonCierraPuerto:
                        CierraPuerto();
                        handled = true;
                        break;

                        case adelante:
                        EscribePuerto(1);
                        handled = true;
                        break;

                        case atras:
                        EscribePuerto(2);
                        handled = true;
                        break;
```

```
                        case derecha:
                        EscribePuerto(3);
                        handled = true;
                        break;

                        case izquierda:
                        EscribePuerto(4);
                        handled = true;
                        break;

                        case paro:
                        EscribePuerto(5);
                        handled = true;
                        break;

                        handled = true;
                        break;

                        default:
                        break;
                    }
                   break;

              default:break;
        }

        return handled;
}
```

La única modificación realizada al método presentado en la sección 2.4, es el manejo de los eventos de los botones para la apertura, escritura y cierre del puerto.

El código completo se muestra en el Apéndice C y D de este trabajo. Una vez realizado la creación de los recursos, el archivo de cabecera con la declaración de los recursos y el código fuente de esta aplicación, es posible utilizar el emulador, tal y como se detalló en la sección 2.5 de este trabajo. La emulación de la aplicación, servirá para corroborar que todo ha sido creado de forma correcta; el siguiente paso es realizar una prueba real a la aplicación, estableciendo una comunicación serial con algún dispositivo ó software.

Para finalizar, debe descargarse la aplicación al PDA. Para esto puede utilizarse el software sugerido, llamado Pilot Install. Cabe aclarar que existen mas herramientas para la descarga de aplicaciones para PDA's, se tiene la libertad de utilizar alguna otra si ya se tiene experiencia en el manejo de la misma. El uso de Pilot Install para la descarga de aplicaciones, se detalla en el Apéndice B de este trabajo.

La forma en que trabaja la aplicación es la siguiente:

- Se presiona el botón para la apertura del puerto

- Se presiona uno de los botones para enviar alguna orden al robot
- En caso de no querer enviar más órdenes, se presiona el botón de cierre del puerto.

3.6 Conclusiones

El diseño de la aplicación desarrollada en el presente capítulo muestra los pasos para codificar y enviar información a través del puerto serie. Deben tomarse en cuenta ciertos aspectos cuando uno se encuentra desarrollando algún trabajo innovador, en el caso de esta aplicación, se debe ser conciente del ahorro de energía en este tipo de dispositivos (PDA), ya que son alimentados, generalmente por baterías. A todo esto, se debe tener cierto cuidado en el cierre del puerto, de caso contrario significará un desperdicio de energía.

Esta aplicación puede utilizarse en forma real para muchos propósitos de control, uno de ellos, podría ser el sustituir el uso de una computadora de escritorio que claramente representa un mayor peso y tamaño, por el uso de un PDA portátil en el control de un pequeño robot, que por sus dimensiones haría gran contraste con las dimensiones de una computadora de escritorio.

Una gran ventaja del puerto serial, puede ser el hecho de que a pesar de las nuevas tecnologías en desarrollo, todavía tiene una larga vida en las aplicaciones industriales debido a su sencillo protocolo de comunicación, por ejemplo, con microcontroladores

Capitulo 4

Finalmente en este capítulo se harán las pruebas necesarias a la aplicación desarrollada en el capítulo previo. Se dará una breve explicación de cómo utilizar software para establecer comunicación serial con una computadora personal como primera prueba. Se realizará la prueba final a la aplicación desarrollada mediante el control de un pequeño robot, estableciendo una comunicación entre la PDA y el microcontrolador del robot.

Pruebas y Resultados

Para realizar una prueba real a la aplicación desarrollada, es factible el uso de la Hyper Terminal con la cual cuenta la mayoría de los sistemas operativos Windows de Microsoft.

Finalmente se hará la tan esperada prueba al robot, con lo cual quedará totalmente demostrado el buen funcionamiento de la aplicación desarrollada.

4.1 La Hyper Terminal

Lo primero es conectar físicamente el cable de conexión serial apto para el modelo de PDA con que se esta trabajando, al puerto serie COM1 de la computadora personal. Cabe aclarar, que la última generación de computadoras en el mercado, ya no cuentan con este periférico. Lo siguiente es configurar la conexión de la Hyper Terminal, con los parámetros que se establecieron para esta aplicación. Por lo general, la Hyper Terminal, se encuentra ubicada en la lista de programas, la categoría de accesorios, y después la subcategoría de comunicaciones. Para Windows XP, podemos observarlo en la figura 4.1.

Figura 4.1 Ubicación de la Hyper Terminal

Debe crearse una nueva configuración de conexión, con un nombre y un icono que la identifique, en este caso puede ser “serial”, como se muestra en la figura 4.2.

Figura 4.2 Ventana para la creación de una nueva configuración de conexión

La siguiente ventana importante es la que cuestiona acerca del periférico al cual se desea conectar, en este caso será el puerto serie COM1 de la computadora. Dicha ventana se muestra en la figura 4.3.

Figura 4.3 Ventana para la conexión al puerto serie

En la siguiente ventana, se deben ajustar los parámetros de conexión; recordando que se estableció una velocidad de transmisión de 9600 bits por segundo y que los parámetros de conexión dados por default por el Palm OS fueron:

- Bits de datos: 8
- Paridad: ninguno
- Bits de parada: 1
- Control de flujo: ninguno

Se pueden ajustar estos parámetros de conexión en la ventana, como se puede observar en la figura 4.4.

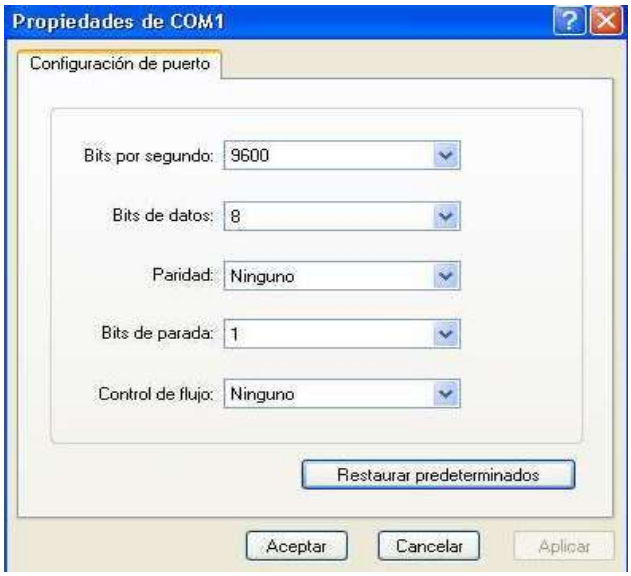

Figura 4.4 Ventana para ajustar los parámetros de conexión

Una vez configurada la Hyper Terminal, solo debe seleccionarse la aplicación para la comunicación serial, descrita y creada en el capítulo 3 de este trabajo e instalada en la PDA con la cual se esta trabajando.

Se debe presionar el botón para la apertura del puerto; a continuación presionar uno de los botones para enviar una orden y la mandará al puerto COM1 y aparecerá en la pantalla de la Hyper Terminal, tal como puede observarse en la figura 4.5.

Figura 4.5 Ventana de Hyper Terminal presentando una prueba a la aplicación creada

En caso de no estar enviando ninguna información a través del puerto, es conveniente presionar el botón de cierre del puerto, esto con el objetivo de ahorrar la energía de las baterías. Al cerrar la aplicación del PDA, el puerto es cerrado automáticamente.

4.2. Control de un Robot mediante el puerto serie

Para esta prueba se hará uso de un robot creado en el laboratorio de electrónica de la Facultad de Ingeniería Eléctrica de la Universidad Michoacana de San Nicolás de Hidalgo, desarrollado por el M.C. Félix Jiménez Pérez.

Dentro del trabajo de aplicaciones para PDA, no se describirá la construcción del robot, o la manera en que se programa el microcontrolador del mismo; solo se comentará el tipo de información necesaria para el control del robot. Dicha información será mandada a través del puerto serie mediante la aplicación desarrollada a lo largo del capítulo 3.

El robot tiene unas dimensiones de 40cm de altura, 40cm de ancho y 20cm de altura aproximadamente y pesa alrededor de 3 kilogramos. En la figura 4.6 se puede observar al robot a controlar. Se observa que el robot cuenta con dos ruedas para desplazarse, dichas ruedas giran por medio de dos motores de CD acoplados a cada una de ellas. El robot y el microcontrolador son energizados mediante 1 batería industrial recargable de 12 volts a 2.3 amperes.

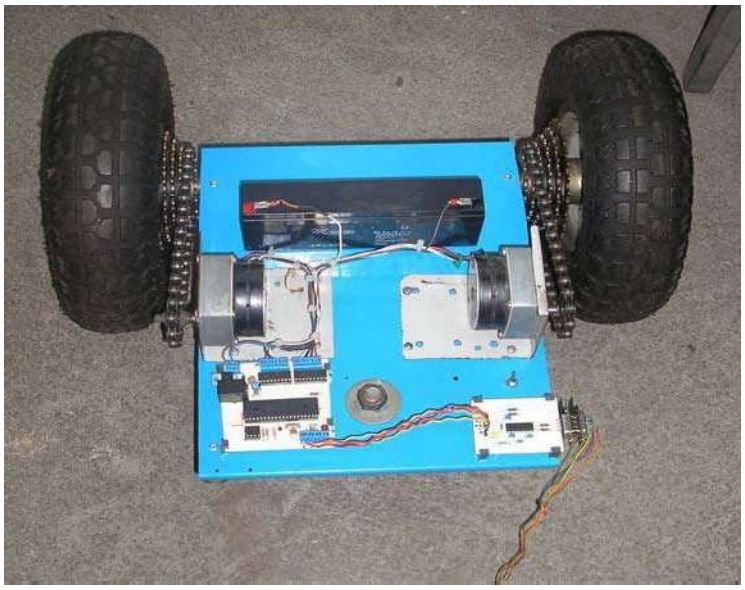

Figura 4.6 Robot a controlar por puerto serie

El robot puede ser controlado de forma remota. Para el control del robot se cuenta con un puerto serie al cual debe mandarse ciertas instrucciones para hacer actuar a los motores que mueven las ruedas.

El código del programa cargado al microcontrolador no será tratado en este trabajo, solo se describirá el formato de las instrucciones necesarias para controlar el robot.

El microcontrolador recibe una señal de 1 bit en código ASCII (este código es el que enumera los caracteres que se pueden desplegar en la computadora). El robot recibe una letra por el puerto serie e interpreta los siguientes datos en el puerto serie como las siguientes órdenes:

- A : adelante
- R : atrás
- D : derecha
- I : izquierda
- P : paro

Estas órdenes se envían usando la aplicación diseñada en el capítulo 3 (figura 4.7), siguiendo los pasos descritos en la sección 3.5.

Figura 4.7 Aplicación para comunicación serial descargada a PDA

Para que el robot se desplace hacia delante, el motor de la derecha (viendo de frente al robot) gira en sentido contrario de las manecillas del reloj y el motor de la izquierda gira en sentido de las manecillas del reloj.

Para que el robot se desplace hacia atrás, se invierte el sentido de giro de los dos motores que mueven a las ruedas.

Con la orden de dar vuelta a la derecha, el motor izquierdo y derecho (viendo el robot de frente) giran en sentido contrario a las manecillas del reloj.

Para dar vuelta a la izquierda, el motor izquierdo y derecho (viendo el robot de frente) giran en sentido de las manecillas del reloj.

Con la orden de paro, los motores del robot se detienen por completo.

Cabe señalar que el robot se desplaza a una velocidad única, no se programó para controlar la velocidad del robot.

Se construyó un adaptador (figura 4.8) para la conexión entre el microcontrolador y el cable de conexión serial de la PDA, ya que los dos tenían conectores tipo hembra. Este problema puede ser resuelto comprando un adaptador en cualquier tienda de electrónica, pero se construyo por razones de economía.

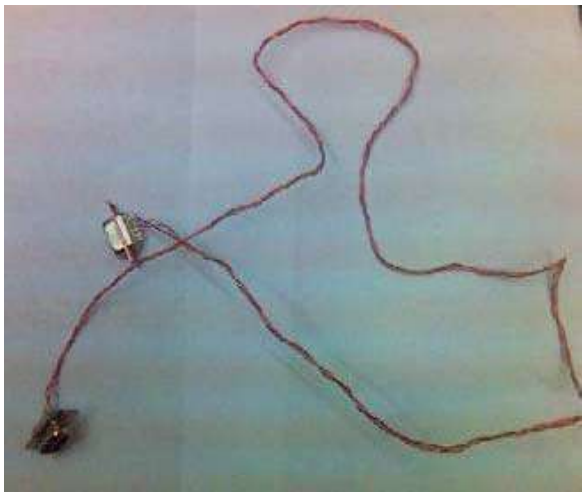

Figura 4.8 Adaptador para el puerto serie

En la figura 4.9 se muestra a la PDA acoplada al microcontrolador del robot por medio del adaptador y el cable de conexión serial, listos para la prueba.

Figura 4.9 Prueba de la aplicación serial

El funcionamiento del robot en cuestión no esta puesto en tela de evaluación, el objetivo de esta prueba fue demostrar el buen funcionamiento de la aplicación desarrollada en el capítulo 3 y lograr los objetivos planteados al principio de este trabajo.

4.3. Conclusiones

Es posible, con el uso de esta aplicación, realizar una comunicación con algún dispositivo que cuente con un puerto serial. Se puede observar por medio de las pruebas realizadas con la Hyper Terminal, el buen desempeño de la aplicación desarrollada a través de este trabajo y es interesante el explotar las ventajas con la que cuenta la PDA por su tamaño y el hecho de caber en el bolsillo. La aplicación desarrollada en el capítulo 3 de este trabajo puede ser ajustada para realizar distintas tares de comunicación mediante el puerto serie, en el caso del control del robot, solo fue necesario conocer el formato de la información que se requería para el control del mismo. Quedan abiertas las puertas para trabajos futuros, ya sea de control o comunicación, y seria interesante una actualización de la comunicación serial al protocolo de comunicación mediante USB (de las siglas en ingles Universal Serial Bus).

Capitulo 5

Conclusiones y Trabajos Futuros

5.1 Conclusiones

La realización de esta tesis, implicó el uso de un nuevo lenguaje de programación, aclarando que se requeriría de un curso extra para dominar el uso de C++, pero ese no era el objetivo de este trabajo; así como el manejo de nuevas herramientas para el diseño de aplicaciones para Palm OS, todo esto con el objetivo de aprovechar las ventajas que presentan las PDA, frente a las computadoras personales en cuestión de portabilidad y costos. El uso de la Hyper Terminal aclara la duda del funcionamiento de la aplicación desarrollada.

El control del robot presentado en la última sección del Capítulo 4, demostró que la aplicación tiene la característica de poder realizar diferentes tareas y no se cierra a un solo fin u objetivo. En la industrial, el uso del puerto serie para la comunicación con ciertos dispositivos de control, como es el caso del PLC (Controladores Lógicos Programables), es algo común. Se debe tener muy en cuenta la gran ventaja de los dispositivos PDA para el control o comunicación de dispositivos electrónicos, debido a sus dimensiones y a que con el paso del tiempo van creciendo las características de memoria y velocidad de procesamiento de datos. Es cierto que una PDA nunca tendrá las mismas características que una computadora de escritorio de la misma generación, pero se debe evaluar las necesidades de velocidad y memoria requeridas para resolver algún problema de la vida real. Queda demostrado que las características con que cuenta el modelo de PDA utilizado en este trabajo de tesis fueron suficientes para cumplir con el objetivo del control del robot propuesto.

El desarrollo de trabajos basados en las nuevas tecnologías, abre las puertas al desarrollo de trabajos cada vez más interesantes y complejos. Posiblemente el presente trabajo no muestre una aplicación muy innovadora y posiblemente existan otras herramientas de comunicación, pero el objetivo es sacar provecho a los dispositivos PDA para su aplicación a problemas reales en los cuales el tamaño del elemento controlador pueda significar la diferencia entre resolver el problema y no hacerlo.

5.2 Trabajos Futuros

El uso del puerto USB es cada vez mas común en la vida cotidiana, algunas modelos recientes de computadoras personales y de escritorio han reemplazado sus puertos serie y paralelo por puertos USB. Esto no significa que se este descontinuado el uso del puerto serie, al menos no en el área industrial. En concreto, se sugiere la implementación del protocolo de comunicación mediante el puerto USB.

En el desarrollo de la aplicación de comunicación serial solo se programo la parte referente a la escritura del puerto, ya que para el control del robot solo era necesario mandar información al microcontrolador del robot. Sin embargo, para que exista una verdadera comunicación entre dos elementos debe haber una retroalimentación de información. Para esto es necesario implementar una nueva función para la lectura del puerto y así poder obtener información concerniente al estado del elemento a controlar. La lectura del puerto es una función más complicada, pero no imposible. Si no se implementó en este trabajo fue debido a que no se requería esta función para el control del robot, y el desarrollar esta función implica el dedicar algo más de tiempo.

Posiblemente sea algo tedioso el tener que seguir todos los pasos de abrir, escribir y cerrar el puerto. Podría resumirse todos estos pasos con el uso de un solo botón que realice dichas funciones en el orden requerido, sin embargo, esto no es tan sencillo como se comenta, ya que de cerrar el puerto antes de mandar todo la información se reflejaría en una posible perdida de información. Es necesario hacer pruebas para el cálculo del tiempo necesario para el óptimo envío de la información.

En la prueba hecha al robot no se toca el tema de la programación hecha al microcontrolador, este tipo de programación puede hacerse en lenguaje C de programación, por lo cual no será tarea difícil el hacer modificaciones al programa del robot para poder incluir nuevas funciones o pulir con las que ya cuenta el robot.

APENDICE A

Código fuente para desplegado de mensajes en Palm OS

```
#include <PalmOS.h>
#include "AppResources.h"

static Err ComienzaAplicacion(void)
{
      FrmGotoForm(FormPrincipal);
      return errNone;
}

static void DetenAplicacion(void)
{
      FrmCloseAllForms();
}

 void DespliegaMensaje(UInt16 alertID)
{
      FrmCustomAlert(alertID, NULL, NULL, NULL);
}

static Boolean ManejaEventoFormPrinc(EventType* pEvent)
{
      Boolean    handled = false;
      FormType*  pForm;
      switch (pEvent->eType) {
            case frmOpenEvent:
                  pForm = FrmGetActiveForm();
                  FrmDrawForm(pForm);
                  handled = true;
                  break;
            case ctlSelectEvent:
      switch(pEvent->data.ctlSelect.controlID)
                  {
```

```
                    case BotonPrincipal:
                    DespliegaMensaje(MensajeAlerta);
                    handled = true;
                    break;
                    default:break;
               }
               break;
          default:break;
     }
     return handled;
}

static Boolean ManejaEventoAplicacion(EventType* pEvent)
{
     UInt16          formId;
     FormType*  pForm;
     Boolean         handled = false;
     if (pEvent->eType == frmLoadEvent) {
          formId = pEvent->data.frmLoad.formID;
          pForm = FrmInitForm(formId);
          FrmSetActiveForm(pForm);
          switch (formId) {
               case FormPrincipal:
     FrmSetEventHandler(pForm,ManejaEventoFormPrinc);
                    break;
               default:break;
          }
          handled = true;
     }
     return handled;
}

static void ProcesaEventoAplic(void)
{
     EventType  event;
```

```
      do {
            EvtGetEvent(&event, evtWaitForever);
            if (SysHandleEvent(&event))
                  continue;
            if (ManejaEventoAplicacion(&event))
                  continue;
            FrmDispatchEvent(&event);
      } while (event.eType != appStopEvent);
}

UInt32 PilotMain(UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
{
      Err error = errNone;
      switch (cmd) {
            case sysAppLaunchCmdNormalLaunch:
                  if ((error = ComienzaAplicacion()) == 0) {
                        ProcesaEventoAplic();
                        DetenAplicacion();
                  }
                  break;
            default:break;
      }
      return error;
}
```

APENDICE B

Descarga de aplicaciones para Palm OS mediante Pilot Install

Para la transmisión de la aplicación hacia la Palm, el software que se utilizó fue el Pilot Install, versión 4.9.0.0 desarrollado por la compañía ENVICON y es de distribución gratuita [Freewarepalm 2006].

Es posible descargar las aplicaciones por medio de otros tipos de software, sin embargo el que aquí se presenta, además de ser gratuito es fácil de usar. En la figura B1 se muestra la ventana principal del software.

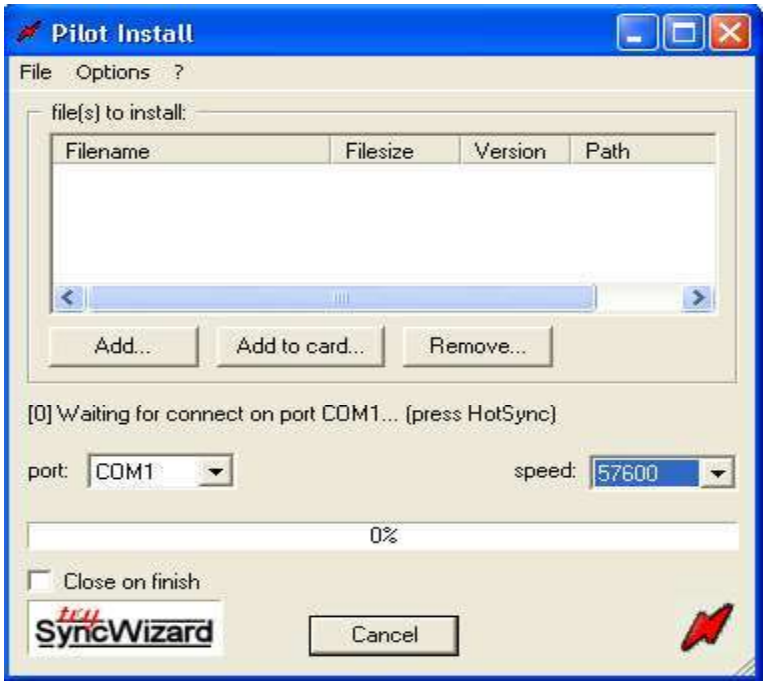

Figura B1 Ventana principal de Pilot Install

El primer paso, es seleccionar la ubicación donde se encuentra la aplicación que se quiere descargar. Los archivos que deben descargarse a la PDA son de tipo *palm-program*.

Las carpetas donde se encuentran los proyectos creados en PODS se encuentran, por lo general en C:\Archivos de programa\PalmSource\Palm OS Developer Suite\workspace\. La ubicación puede cambiar dependiendo de donde se haya instalado PODS.

Después de seleccionar la carpeta con el nombre del proyecto, se debe abrir la carpeta con el nombre *Debug*. En esta ubicación se encuentra el archivo que se debe descargar, y tiene

el nombre del proyecto. La opción para seleccionar la ubicación del archivo, se encuentra en el menú File de la barra de herramientas. Aquí, del menú *Add*, se selecciona la opción de *Select*… y se especifica la ubicación del archivo. Esto desplegará una ventana para seleccionar el archivo, como se puede observar en la figura B2.

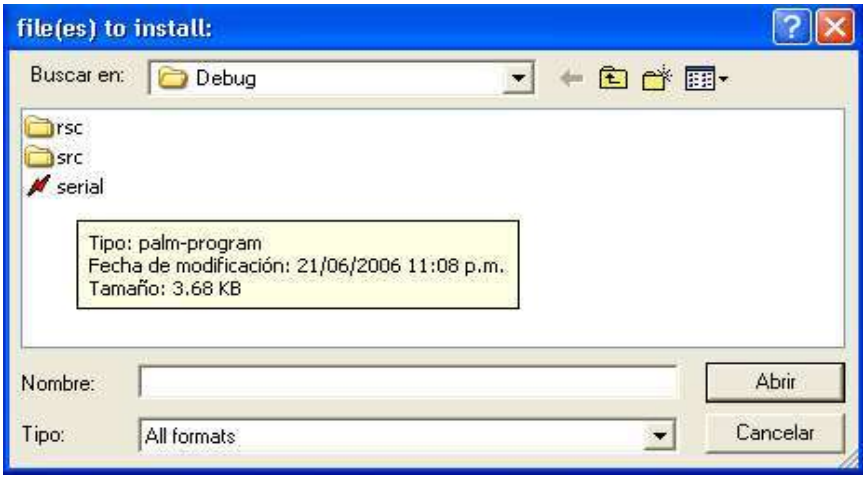

Figura B2 Ventana para instalar archivos de programa

Una vez seleccionado el archivo a descargar a la PDA, la ventana principal muestra el archivo a descargar e informa que el software se encuentra esperando la conexión con la PDA, como se muestra en la figura B3.

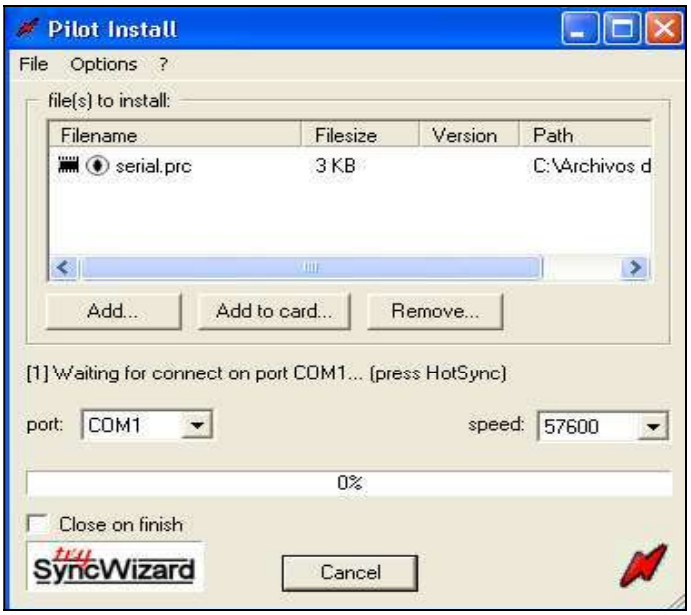

Figura B3 Ventana principal de Pilot Install esperando la conexión

Lo siguiente es seleccionar la aplicación llamada HotSync, la cual se encuentra instalada por default en la PDA; esta aplicación, establece una conexión entre la PDA y el puerto serie, en este caso, para descargar la aplicación creada. En la figura B4, se muestra el icono de la aplicación HotSync, que aparece en la ventana principal del PDA.

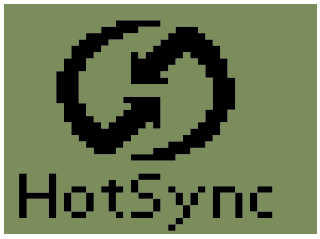

Figura B4 Icono de la Aplicación HotSync

Una vez seleccionada la aplicación HotSync, el tipo de conexión a realizar, será local mediante el cable o base de sincronización (cable serie) correspondiente al modelo de PDA con que se esta trabajando. Una vez ajustada el tipo de conexión a establecer, se presiona el botón que tiene la figura del icono de la figura B4, el cual se encuentra en el centro de la pantalla, como se puede observar en la figura B5.

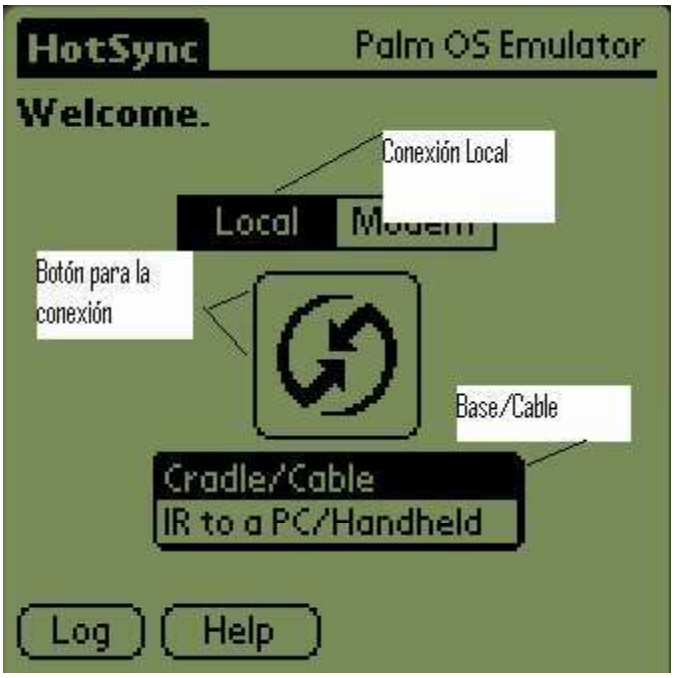

Figura B5 Ventana principal del HotSync

Después de presionar el botón de conexión aparece una pantalla de espera en la PDA, mientras dura la descarga. Al terminar la descarga, reaparece la ventana principal del HotSync, lo cual nos indica que se realizó la descarga. Al regresar a la ventana principal del PDA se puede observar la nueva aplicación descargada, con el Icono y la etiqueta diseñada por el usuario, tal como se describió en la sección 2.2.1.

Apéndice C

Código fuente para aplicación de conexión serial

```
#include <PalmOS.h>
#include "AppResources.h"
#include "SerialMgr.h"
#include "SystemResources.h"

UInt16 DireccionPuerto = 0x8000;
Boolean EstadoPuerto;

static Err ComienzaAplicacion(void)
{
      FrmGotoForm(FormPrincipal);
      return errNone;
}

void DespliegaMensaje(UInt16 alertID)
{
      FrmAlert(alertID);
}

static Err AbrePuerto (void)
{
      Err error = 0;
      error = SrmOpen(serPortCradlePort, 9600, &DireccionPuerto);
      ErrNonFatalDisplayIf(error == serErrBadPort,"serErrBadPort");
      switch (error)
      {
            case errNone:
            break;
            case serErrAlreadyOpen:
            SrmClose(DireccionPuerto);
            DespliegaMensaje(AlertaPuertoOcupado);
            return error;
            break;
            default:
            DespliegaMensaje(AlertaPuertoAbierto);
            return error;
            break;
      }
      EstadoPuerto = true;
      SrmReceiveFlush(DireccionPuerto, 100);
      return error;
}


static void EscribePuerto (int opcion)
{
      Err error;
```

```
      Char *text = NULL;
      if (EstadoPuerto == false)
      return;
      if(opcion == 1)
      {
            text = "A";
      }
      else if(opcion == 2)
      {
            text = "R";
      }
      else if(opcion == 3)
      {
            text = "D";
      }
      else if(opcion == 4)
      {
            text = "I";
      }
      else if(opcion == 5)
      {
            text = "P";
      }
      if (text)
      {
            SrmSend(DireccionPuerto, text, StrLen(text), &error);
      }
}

static void CierraPuerto (void)
{
      Err error;
      error = SrmSendWait(DireccionPuerto);
ErrNonFatalDisplayIf(error == serErrBadPort,"SrmClose:bad port");
      if (error == serErrTimeOut)
      DespliegaMensaje(AlertaPuertoTiempofin);
      SrmClose(DireccionPuerto);
      EstadoPuerto = false;
}

static void DetenAplicacion(void)
{
      FrmCloseAllForms();
      CierraPuerto();
}


static Boolean ManejaEventoFormPrinc(EventType* pEvent)
{
      Boolean    handled = false;
      FormType*  pForm;
      switch (pEvent->eType) {
```

```
      case frmOpenEvent:
            pForm = FrmGetActiveForm();
            FrmDrawForm(pForm);
            handled = true;
            break;
      case ctlSelectEvent:
switch(pEvent->data.ctlSelect.controlID)
            {
                  case BotonAbrePuerto:
                  AbrePuerto();
                  handled = true;
                  break;

                  case BotonCierraPuerto:
                  CierraPuerto();
                  handled = true;
                  break;

                  case adelante:
                  EscribePuerto(1);
                  handled = true;
                  break;

                  case atras:
                  EscribePuerto(2);
                  handled = true;
                  break;

                  case derecha:
                  EscribePuerto(3);
                  handled = true;
                  break;

                  case izquierda:
                  EscribePuerto(4);
                  handled = true;
                  break;

                  case paro:
                  EscribePuerto(5);
                  handled = true;
                  break;

                  handled = true;
                  break;
                  default:break;
            }
            break;
      default:
            break;
}
return handled;
```

```
}

static Boolean ManejaEventoAplicacion(EventType* pEvent)
{
      UInt16          formId;
      FormType*  pForm;
      Boolean         handled = false;
      if (pEvent->eType == frmLoadEvent) {
            formId = pEvent->data.frmLoad.formID;
            pForm = FrmInitForm(formId);
            FrmSetActiveForm(pForm);
            switch (formId) {
                  case FormPrincipal:
                  FrmSetEventHandler(pForm,ManejaEventoFormPrinc);
                        break;
                  default:break;
            }
            handled = true;
      }
      return handled;
}
static void ProcesaEventoAplic(void)
{
      EventType  event;
      do {
            EvtGetEvent(&event, evtWaitForever);
            if (SysHandleEvent(&event))
                  continue;
            if (ManejaEventoAplicacion(&event))
                  continue;
            FrmDispatchEvent(&event);
      } while (event.eType != appStopEvent);
}

UInt32 PilotMain(UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
{
      Err error = errNone;
      switch (cmd) {
            case sysAppLaunchCmdNormalLaunch:
                  if ((error = ComienzaAplicacion()) == 0) {
                        ProcesaEventoAplic();
                        DetenAplicacion();
                  }
                  break;
            default:break;
      }
      return error;
}
```

Apéndice D

Archivo de cabecera para aplicación de conexión serial

```
#define FormPrincipal 1000

//abre el puerto para la conexion
#define BotonAbrePuerto         1001

//cierra el puerto
#define BotonCierraPuerto  1002

//botones de control
#define adelante 1000
#define atras 1006
#define izquierda 1003
#define derecha 1005
#define paro 1007

//mensajes de alerta
#define Alerta                  1000
#define AlertaPuertoOcupado           1100
#define AlertaPuertoAbierto           1200
#define AlertaPuertoTiempofin   1300
#define AlertaPuertoEnvio       1400
#define AlertaPuertoError  1500

//icono de la aplicación
#define Icono        1000
```

Referencias

[Configura equipos 2006]
Configura equipos. Algo de historia. 30 de Mayo de 2006, http://www.configurarequipos.com/doc75.html

[Deitel y Deitel 1995]
H. M. Deitel / P. J. Deitel, Como programar en C/C++, México: Prentice Hall, 1995

[Tenenbaum, Langssam y Augenstein 1993]
Aaron M Tenenbaum, Yedidyah Langsam, Moshe A, Augenstein, Estructuras de datos en C, México: Prentice Hall, 1993

[Freewarepalm 2006]
Freewarepalm.com. Utilities. 15 de Junio de 2006, http://www.freewarepalm.com/utilities/simplyinstall-pilotinstall.shtml
L

[Foster y Bachmann 2005]
Lonnon R. Foster y Glen Bachmann, Professional Palm OS Programming, Indianapolis: Wiley Publishing, Inc, 2005

[How Stuff Works 2006]
How Staff Works. How PDA's works. 30 de Mayo de 2006 http://www. howstuffworks.com/pda.htm

[Palm OS 2006]
Palm OS. Tools. 14 de Abril de 2006. www.palmos.com/dev/ tools/dev_suite.html

[Todo Palm 2006]
Todo Palm. Emulador. 15 de junio de 2006, http://www.todopalm.cl/ aprendiendo/emulador.asp

[Wikipedia 2006]
Wikipedia. Puerto Serie. 10 de junio de 2006, es.wikipedia.org/wiki/Puerto_serial