VNIVERSIDAD NACIONAL
AVFNOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

POSGRADO EN CIENCIA E INGENIERIA DE LA COMPUTACION

"PROGRAMACION DE CONDUITS SOBRE LA PLATAFORMA PALM OS"

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS (COMPUTACION)

P R E S E N T A :

GABRIEL ANDRES ROJAS LEAL

DIRECTOR DE TESIS: ING. MARIO RODRIGUEZ MANZANERA

MEXICO, D. F. 2005

A ***Dios***, por que iluminas
mi camino y me haz protegido
durante todo el viaje.

A mi madre, ***Emma Guadalupe***,
sin ti este trabajo no estaría
terminado, gracias por ayudarme
a escribirlo, aún cuando no tenías
tiempo siempre hacías un espacio
para sentarte a transcribir mis
notas. Gracias mamá.

A ti, que siempre haz creído en mí,
eres la energía que alimenta los
deseos de seguir adelante.

A todos los que han dudado de mí y
han puesto piedras en el camino, son
el condimento que da sabor al éxito.

A mí...

"Faint" (Linkin Park)

I am a little bit of loneliness, a little bit of disregard
Handful of complaints, but I can't help the fact, that everyone can see these scars
I am what I want you to want, what I want you to feel
But it's like no matter what I do, I can't convince you, to just believe this is real
So I let go, watching you, turn your back like you always do
Face away and pretend that I'm not
But I'll be here 'cause you're all that I got

[Chorus]
(I can't feel the way I did before)
(Don't turn your back on me)
(I won't be ignored)
(Time won't heal this damage anymore)
(Don't turn your back on me)
(I won't be ignored)

I am a little bit insecure, a little unconfident
Cause you don't understand, I do what I can but sometimes I don't make sense
I am what you never wanna say, but I've never had a doubt
It's like no matter what I do, I can't convince you for once just to hear me out
So I let go, watching you, turn your back like you always do
Face away and pretend that I'm not
But I'll be here 'cause you're all that I've got

[Chorus]
(I can't feel the way I did before)
(Don't turn your back on me)
(I won't be ignored)
(Time won't heal this damage anymore)
(Don't turn your back on me)
(I won't be ignored)

(No)
(Hear me out now)
(You're gonna listen to me, like it or not)
(Right now)
(Hear me out now)
(You're gonna listen to me, like it or not)
(Right now)

(I can't feel the way I did before)
(Don't turn your back on me)
(I won't be ignored)

[Chorus]
(I can't feel the way I did before)
(Don't turn your back on me)
(I won't be ignored)
(Time won't heal this damage anymore)
(Don't turn your back on me)
(I won't be ignored)

I can't feel
Don't turn your back on me
I won't be ignored
Time won't heal
Don't turn your back on me
I won't be ignored

"Hay libros que no parecen escritos para que la gente aprenda, sino para que se enteren que el autor ha aprendido algo"
(Anónimo)

En los primeros años de las computadoras, cuando una simple calculadora era del tamaño de un baúl, las computadoras de bolsillo eran solamente ciencia ficción. Nadie imaginaba que en los 1990's estos dispositivos iban a desarrollarse tanto que no solo cabrían en un bolsillo si no que incluso dentro de un reloj de pulsera.

El presente trabajo es el resultado de una investigación acerca de los dispositivos portátiles y sus más comunes aplicaciones; tiene la finalidad de introducir al lector al extenso campo del cómputo móvil y de los asistentes personales digitales, mejor conocidos como PDA's, pone las bases necesarias para programar dichos dispositivos, además da un ejemplo bastante claro de cómo se hace un programa que corra sobre esta plataforma.

Este trabajo se enfoca en el desarrollo de las conexiones entre aplicaciones para dispositivos portátiles y computadoras de escritorio. Esta desarrollado en nuestro idioma, ya que casi toda la literatura en esta área se encuentra en el idioma inglés; podrá ser utilizado por el estudiante interesado como base para el aprendizaje de estas tecnologías y como guía para futuras investigaciones.

Incluye información sobre las nuevas tecnologías que se han ido incorporando recientemente a los PDA's las cuales además del manejo tradicional de una agenda electrónica permiten que estos dispositivos sean capaces de reproducir audio y video haciendo que ya no sea necesario portar con aparatos cuya función sea únicamente esa, llámese tocacintas, reproductores de CD's o TV's portátiles.

En los primeros dos capítulos parte se dan los conceptos generales acerca de esta tecnología, comenzando por su historia, desde los primeros asistentes personales y sus primeras aplicaciones, hasta la actualidad donde dos principales productores se disputan el mercado, que ya ha crecido bastante y aún le queda mucho camino por recorrer.

Uno de esos productores es la compañía Palm Inc., que perfeccionó esta tecnología y la llevó a las masas, colocándose así en la cima, tanto en desarrollo, como en ventas. Es por eso que se escoge su plataforma para desarrollar este proyecto, además los recursos de programación son muy accesibles para cualquier usuario, desde el más novel hasta el más experimentado.

Se analizan la mayoría de sus componentes, comenzando con el procesador, el cual es la adaptación de un antiguo procesador (68000) diseñado por Motorola, que tuvo bastante éxito con las primeras computadoras Apple Macintosh, y que ahora ha encontrado nueva vida debido a su gran versatilidad, tanto en desempeño, velocidad y ahorro de energía, que en estos dispositivos pasa a ser una prioridad.

Junto con la elección del procesador viene el sistema operativo que lo controla, en este caso la misma Palm Inc. ha desarrollado el Palm OS, que a lo largo de los años sigue siendo la mejor opción en cuanto a aprovechamiento de los recursos de hardware, que son muy limitados en comparación con una PC, además de ser un sistema de fácil uso para cualquier gente. Cabe mencionar que gran parte del éxito de la Palm se debe a las aplicaciones que corren sobre ella, desde las básicas como son la agenda, el directorio telefónico, la calculadora y el bloc de notas; hasta las más complejas como serían la reproducción de audio y video, la navegación en Internet e incluso la localización vía satélite en cualquier parte del mundo.

En el capitulo tres se mencionan las distintas formas de cómo se comunican estos dispositivos con otros PDA's, PC's y además con la Internet; donde la más importante es hacia y desde la PC, ya que por medio de esta se instalan aplicaciones, se actualizan datos y sobre todo se hacen los respaldos de la información contenida en los dispositivos en caso de que algún imprevisto haga perder dicha información, pudiendo ser reinsertada sin problema alguno.

En el capítulo cuatro se comienza con el uso del emulador de los sistemas Palm, esto permite realizar pruebas a cualquier aplicación sin la necesidad de utilizar directamente algún dispositivo real, así si surgen incompatibilidades, errores o simplemente no satisface al 100%, se pueden corregir de una forma más rápida y segura. Con este emulador se puede trabajar con cualquier modelo producido por Palm y algunas otras compañías que utilizan la misma plataforma solamente obteniendo la imagen (.ROM) del dispositivo deseado, ya sea directamente a través del cable de sincronización, o descargada del sitio web de Palm.

En el capítulo cinco se entra de lleno al diseño del conduit que nos sirve para transferir la información desde la PC hacia la Palm y viceversa.

Aquí se da un análisis de qué es y como funciona un conduit, además de las diferentes acciones que se pueden llevar a cabo entre la PC y la Palm. Todo esto se enfoca desde el entorno de programación Visual Basic, para que el lector vea que programar aplicaciones para este sistema no requiere forzosamente de un lenguaje muy complicado y que no es el único entorno de programación para esta plataforma de desarrollo.

Ya en el capítulo seis se da paso al desarrollo del conduit que servirá para transferir los datos originados en la aplicación de "Ventas" y cuya función será tener los registros actualizados de clientes, productos y precios de cada vendedor, junto con el inventario general. Esperando que con esto se cree un punto de partida para el desarrollo de futuras aplicaciones cada vez más orientadas hacia nuestro mercado evitando así la dependencia directa de grandes compañías, que como se ha visto tienen casi olvidado este gran mercado latinoamericano.

Junto con todo esto se da una breve introducción al entorno de desarrollo "Metrowerks CodeWarrior", el cual es la herramienta más utilizada para crear aplicaciones en C++ que funcionen sobre dicha plataforma Palm.

Se incluye una sección de acrónimos y un glosario que facilitan el entendimiento de muchos términos que pueden llagar a ser extraños para el lector que apenas se inicie en este campo.

Por último se menciona la bibliografía consultada durante todo este proyecto y que puede ser de gran ayuda para quién este interesado en seguir este camino.

"Los libros son las abejas que llevan el polen de una inteligencia a otra"
(Anónimo)

CAPITULO I

ASISTENTES PERSONALES DIGITALES

La idea de hacer una computadora del tamaño de la palma de la mano para guardar direcciones y números telefónicos, tomar notas y llevar seguimiento de las actividades diarias fue originada en los 90's, no obstante algunos organizadores computarizados de bolsillo ya estaban disponibles en los 80's. Uno de los primeros PDA's comerciales fue el Newton Message Pad de Apple Computer, un sistema demasiado grande, costoso y complicado, con un programa de reconocimiento de escritura muy primitivo.

En 1996, la Palm Pilot fue introducida, con un éxito total en el mercado de consumidores. Era pequeña y suficientemente ligera para caber en el bolsillo de una camisa, trabajaba durante varias semanas con baterías AAA, era fácil de usar y podía guardar miles de contactos, citas y notas. El día de hoy se pueden comprar dispositivos tipo Palm hechos por las más grandes compañías fabricantes de PC's (Hewlett-Packard, IBM, Compaq, Sony). Los PDA's han evolucionado en maquinas capaces de efectuar procesos, ejecutar juegos, tocar música y bajar información de Internet. Todas ellas tienen algo en común: Fueron diseñadas como complemento de una computadora.

Figura 1.1. Diferentes tipos de PDA's

Los PDA's se dividen en dos principales categorías: computadoras de mano y computadoras de bolsillo. Sus diferencias son el tamaño, la pantalla y el modo de inserción de datos. Comparadas con las computadoras de bolsillo, las computadoras de mano tienden a ser más grandes y pesadas. Tienen pantallas de cristal liquido (LCD) más grandes y utilizan un teclado miniatura, generalmente en combinación con tecnología de pantalla sensible para la inserción de datos. En cambio las computadoras de bolsillo son más pequeñas y ligeras, tienen LCD's más pequeños y se basan en tecnología de pantalla sensible/stylus y reconocimiento de escritura para la inserción de datos.

Independientemente del tipo de PDA, comparten las mismas características:

- Hardware
  - microprocesador
  - memoria de estado sólido
  - pantalla LCD
  - batería
  - dispositivo de entrada (botones en combinación con pantalla sensible o teclado)
  - puertos de entrada/salida

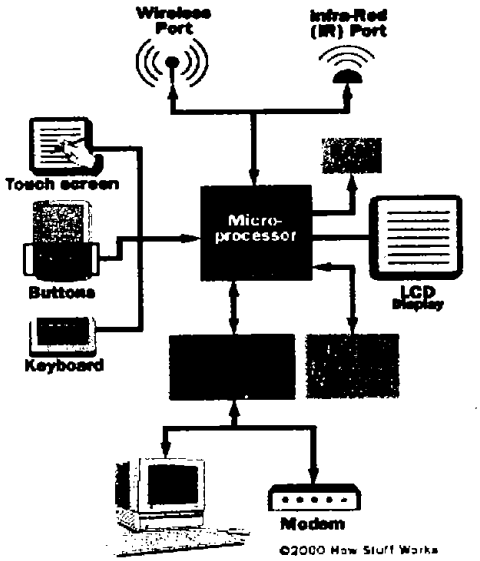

Figura 1.2. Interacción del microprocesador con el resto del hardware

- Software
  - Sistema Operativo
  - Software para PC

1.1 HARDWARE

1.1.1 Microprocesadores.

Como cualquier PC de escritorio o portátil, los PDA's están construidos alrededor de uno o más microprocesadores. Un microprocesador constituye el centro de procesamiento del PDA y coordina todas sus funciones de acuerdo a las instrucciones programadas en él. Los PDA's usan microprocesadores más pequeños y económicos, un ejemplo lo constituye el Motorola Dragonball o el Hitachi SH7709a, lo cuales tienden a ser más lentos que sus similares en las PC's (16-75MHz, comparado con los 1,000 MHz o más en PC's), aún así son adecuados para las tareas que se requiere realizar en los PDA's. Los beneficios de un tamaño pequeño y un precio bajo equilibran el costo del manejo de velocidades bajas de procesamiento.

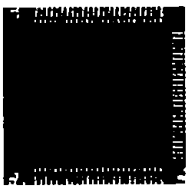

Figura 1.3. Vista superior del procesador Dragon Ball

1.1.2 Memoria.

Un PDA no necesita de un disco duro. Guarda sus programas básicos (el sistema operativo, el libro de direcciones, el calendario y el block de notas) en un chip de memoria de solo-lectura (FLASH EPROM) que permanece intacto aun cuando la maquina se apague La utilización de una ROM tiene otras ventajas como la de que al momento de encender el PDA todos los programas están inmediatamente disponibles y no se tiene uno que esperar a que sean cargados como ocurre con su contraparte en las PC's, en donde cualquier programa de usuario es cargado en memoria principal normalmente un RAM. Otra ventaja es no tener que salvar mediante el comando "Guardar" ya que toda información es grabada automáticamente en la memoria y aún cuando se apague el PDA los datos permanecen en esta, ya que el PDA continua alimentando la memoria con una pequeña cantidad de energía de la batería.

Todos los PDA's utilizan memoria de estado sólido; unas usan RAM estática y otras memoria de tipo Flash. Algunas incluso han incorporado formas removibles de memoria. Los PDA's generalmente vienen con mínimo 2 MB de memoria, en donde 1 de ellos puede albergar hasta 4,000 direcciones y 100 mensajes electrónicos, sin embargo, muchas aplicaciones toman más memoria, así los modelos más recientes de los PDA's vienen con mucho mayor capacidad (5 hasta 64 MB). Además, el sistema operativo Pocket PC ocupa más memoria así que los PDA's con este sistema operativo vienen con 16, 32 o 64 MB. En algunos modelos, la cantidad de memoria es escalable.

1.1.3 Pantalla LCD

Los PDA's utilizan un tipo especial de pantalla LCD, a diferencia de las pantallas LCD para PC de escritorio y portátiles, ya que en la PC son utilizan únicamente como dispositivos de salida, mientras que los PDA's la utilizan para entrada y salida de información. Dichas pantallas son más pequeñas que las de las computadoras portátiles, y pueden variar en tamaño. Todas las pantallas de los PDA's tienen las siguientes características:

- LCD, LCD extendido o CSTN
- resoluciones de 160x160, 240x320 píxeles
- blanco y negro (16 escalas de grises) o color (65,536 colores)
- matriz activa o pasiva (la matriz activa puede mostrar imágenes más nítidas y es más fácil de leer)
- reflectiva o con luz trasera (las pantallas con luz trasera son buenas para leer en condiciones de baja intensidad luminosa)

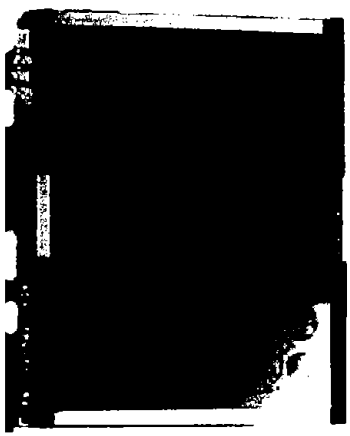

Figura 1.4. Pantalla TFT de cristal liquido

1.1.4 Baterías.

Los PDA's son alimentados por baterías alcalinas tamaño AAA, mientras que algunos otros utilizan baterías recargables (litio, níquel-cadmio). La vida de la batería depende del tipo de PDA y de su uso.. He aquí algunas de las cosas que pueden descargar las baterías:

- El sistema operativo (El sistema Pocket PC requiere mayor energía en virtud de sus mayores requerimientos de memoria).
- Mayor cantidad de memoria.
- Pantalla LCD a color.
- Grabado de voz.
- Reproducción de MP3.

La vida de la batería puede variar desde dos horas hasta dos meses dependiendo del modelo y sus características. La mayoría cuentan con un sistema de administración de energía para extender la vida útil o duración de la batería. Incluso si las baterías están tan bajas que ya no se puede encender el dispositivo (el cuál avisara con suficiente tiempo antes de que se apague), ahí generalmente suficiente energía para mantener la memoria alimentada. Si las baterías quedan completamente secas o si el usuario las retira, se cuenta con aproximadamente un minuto para reemplazarlas antes de que los capacitores dentro del dispositivo pierdan su carga. En este punto, la mayoría de los PDA's pierden toda su información, lo que hace necesario tener un respaldo en una PC de escritorio o portátil. Los PDA's también vienen con adaptadores de corriente que se utilizan en cualquier enchufe común o especiales para automóvil.

1.1.5 Dispositivo de entrada

Los PDA's varían en como uno introduce los datos y comandos. Las computadoras de mano utilizan comúnmente un teclado miniatura en combinación con una pantalla sensible. Las computadoras de bolsillo utilizan un lápiz (stylus) y la pantalla sensible exclusivamente en combinación con un programa de reconocimiento de escritura. Cada modelo también tiene algunos botones para activar pantallas o aplicaciones.

La pequeña pantalla de cuatro pulgadas sirve como un dispositivo tanto de entrada como de salida; en la parte superior de esta pantalla LCD se encuentra una sección de pantalla sensible la cual ejecuta programas al momento en que se le toca con un stylus o escribiendo datos sobre ella.

La pantalla es como un sándwich multicapas. Donde la parte superior tiene una hoja fina de plástico o vidrio con una cubierta resistente en su interior. Esta hoja flota sobre una fina capa de aceite no conductora, el cual descansa sobre otra fina hoja de vidrio que tiene el mismo acabado. Los bordes verticales y horizontales del vidrio se fijan a delgadas barras de tinta de plata.. Cuando una corriente directa es aplicada a cada par de barras se crea un campo eléctrico entre las mismas.

Cuando uno toca con el stylus la pantalla, la capa superior atraviesa el gel o aceite para hacer contacto con la capa inferior de vidrio (lo cual se llama "touchdown" o "contacto"). Esto causa un cambio en el campo eléctrico, el cual es registrado por el software controlador de la pantalla. Mediante el envío de corriente primero a las barras verticales y luego a las horizontales, la pantalla sensible obtiene las coordenadas (X , Y) del punto de contacto. El controlador barre la pantalla miles de veces por segundo y manda estos datos a cualquier aplicación que los necesite. En este proceso, el PDA sabe si el usuario esta tocando un icono en pantalla para lanzar un programa o si sé esta desplazando a través de la pantalla para insertar datos.

Mediante el uso del stylus, uno puede dibujar caracteres en la pantalla del dispositivo. Un programa dentro del PDA convierte los caracteres a letras y/o números. Sin embargo, estos dispositivos realmente no reconocen la escritura; en vez de eso uno tiene que ir escribiendo los números y letras uno a la vez. En los dispositivos Palm, el software que reconoce estos números y letras es conocido como Graffiti. Dicho programa requiere que cada letra sea insertada en un solo movimiento ininterrumpido, y el usuario usa un alfabeto especializado. Por ejemplo, para escribir la letra "A", el usuario tiene que dibujar una "Λ". La letra "F" luce como una "Γ"y así sucesivamente. Para ayudar al Graffiti a ser mas preciso, se deben de introducir las letras en una parte de la pantalla y los números en otra.

La desventaja del reconocimiento de escritura es que el usuario debe de aprender una nueva forma de escribir, es más lento que la escritura normal y el reconocimiento de caracteres es casi perfecto. Por otro lado, es sorprendente lo fácil que es aprender a usarlo y ver que en verdad funciona. Algunos PDA's permiten introducir datos en cualquier parte de la pantalla empleando algún otro programa de reconocimiento que no requiere un alfabeto especial, pero sigue siendo mejor dibujar las letras en cierta forma.

Si el usuario no se puede habituar al tipo de escritura del PDA, puede utilizar un teclado que aparece en pantalla. Luce como un teclado normal, excepto que uno toca las letras con el stylus. Algunos PDA's cuentan con un teclado plegable el cual se puede conectar al dispositivo, lo que es más práctico que escribir si el dispositivo es utilizado para mandar un correo electrónico.

Eventualmente, la mayoría de los PDA's irán incorporando tecnología de reconocimiento de voz, donde el usuario hablará a un micrófono incorporado mientras el software convertirá la voz en datos.

1.1.6 Dispositivos de entrada y salida

Debido a que los PDA's están diseñados para trabajar en conjunto con las computadoras de escritorio o portátiles, necesitan mantener la misma información. Si uno planeaa realizar una cita y la introduce en la PC de escritorio, uno tiene que transferir la información posteriormente al PDA; igualmente si el usuario apunta un teléfono en su PDA, este deberá ser transferido a la PC. También se necesita un respaldo de todo lo guardado en el PDA dentro de la PC por si se presentara el caso de que la batería se acabara y se pudiera perder toda la información. Así, que cualquier PDA debe ser capaz de comunicarse con una PC. La comunicación entre un PDA y una PC es referida como "sincronización de datos" o sincronización. Esto es llevado a cabo típicamente a través de un puerto serial o USB. Algunos de estos PDA's utilizan una base o atril (cradle) en donde se fijan mientras sé esta conectado a la PC.

Además de la comunicación a través de un cable, varios PDA's cuentan con un puerto de comunicaciones que utiliza luz infrarroja (IR) para enviar la información hacia una PC o a otro PDA. Otros PDA's ofrecen métodos inalámbricos para transferir desde y hacia una PC, a una red de computadoras o a algún proveedor de servicios de Internet como los que hay actualmente disponibles para los nuevos modelos de teléfonos celulares. Finalmente algunos PDA's incluyen un modem telefónico para transferir información.

1.2 SOFTWARE

1.2.1 Sistemas operativos.

Los sistemas operativos utilizados por los PDA's no son tan complejos como los usados en las PC's. Generalmente tienen menos instrucciones y ocupan menos memoria. Por ejemplo, el sistema operativo de Palm cabe en menos de 100k de memoria, que es menos del 1% de lo que ocupa Windows 98. Existen dos tipos de sistema operativo, el Palm OS (3Com) y el Pocket PC (anteriormente llamado Windows CE, Microsoft). Palm OS utiliza menos memoria y es más rápido, incluso muchos usuarios dicen que es más fácil de usar.

Pocket PC fácilmente soporta pantallas a color, gráficos, paquetes de Windows en miniatura (Word, Excel) y otros dispositivos (como reproductores de MP3 o de videos MPG), sin embargo toma más memoria y es más lento, además de complicado.

Actualmente Palm OS domina el mercado, pero el Pocket PC cada día le gana terreno. Otras compañías desarrollan software para ambos sistemas operativos.

1.2.2 Software de PC de escritorio o portátil

Al ser necesario sincronizar la información desde y hacia el PDA, se tiene que instalar un programa de sincronización (HotSync para Palm OS, ActiveSync para Pocket PC). Y también es frecuente el que se requiera tener versiones de agenda, calendario y otras importantes aplicaciones del PDA dentro de la PC y/o utilizar un manejador personal de información, como el Lotus Organizer o el Microsoft Outlook, que soporten el proceso de sincronización.

El PDA asigna a cada registro un único numero de identificación y anota la fecha de creación (Un registro puede ser una cita, un contacto, un memo, etc.). Cuando el usuario pulsa un botón en el PDA o en el Cradle, el software de sincronización compara los registros del PDA con los que hay en la PC y acepta los más recientes. La ventaja de la sincronización es que uno siempre tiene una copia de los datos, lo que puede ser un salvavidas si el PDA es dañado, robado o simplemente se le acaban las baterías.

CAPITULO II

PROCESADOR DRAGON BALL

El día de hoy, la Internet, la web, www, la supercarretera de la información constituyen términos familiares a millones de personas tanto en el mundo académico como en el empresarial. Sin embargo el uso de multimedios ligados a sistemas inalámbricos no resulta tan común como seria deseable. Si bien la unión e intersección de las tecnologías de audio, radio, cómputo e Internet es reciente este proyecto parte de la idea de que el lector tiene conocimientos técnicos sobre arquitectura y organización de computadoras, uso de protocolos de comunicación y de estándares, el modelo OSI e inclusive del propio Internet.

Con la aparición de microprocesadores y DSP's (Digital Signal Processors) cada vez más pequeños y poderosos la integración de tecnologías y la innovación de nuevas aplicaciones se ha dado como un proceso natural.

Dado que el conocimiento y creación de aplicaciones multimedia, requiere de contar con o establecerse bajo una plataforma determinada, nuestro primer interés es definir aquellas características básicas que deberán cumplir los elementos de cómputo además de los inalámbricos, en nuestro caso los microprocesadores incluidos en los dispositivos móviles y las interfases a dispositivos inalámbricos, a fin de estudiar finalmente el desarrollo de programas de aplicación embebidos en dichos elementos y dispositivos del tipo PALM.

2.1 ELECCIÓN DEL PROCESADOR

Requerimientos iniciales

- Alto nivel de integración en un solo chip.
- Buen rendimiento en lo referente a consumo de energía.
- Optimización para aplicaciones multimedia.
- Optimización para aplicaciones de tecnología Bluetooth
- Interfaces de alta velocidad hacia soluciones Bluetooth.
- Soporte para una gran variedad de aplicaciones.

Incluyendo los PDA's más populares, teléfonos inteligentes y dispositivos inalámbricos.

Considerando las características definidas como requerimientos iniciales, la selección del procesador tomara como base la quinta generación de microprocesadores. En nuestro caso consideraremos por sus características técnicas y por su disponibilidad e importancia en el mercado de dispositivos inalámbricos a la familia denominada Dragon Ball.

La elección del procesador Dragon Ball MX para el desarrollo de nuevos dispositivos portátiles se ha centrado en que principalmente provee una brecha a sus usuarios con relación a sus competidores más cercanos, en cuanto al rendimiento provisto por el nuevo núcleo ARM9 (conjunto de componentes) y a sus nuevas funciones totalmente integradas.

Estos microprocesadores cumplen con los requerimientos del mercado de dispositivos personales portátiles a través de la integración de periféricos inteligentes, de un núcleo avanzado de componentes y con altas capacidades en lo referente al ahorro y consumo de energía.

El nuevo procesador MC9328MX1 encapsulado en un arreglo de 256 pines utiliza el núcleo ARM920T el cual opera a velocidades de hasta 200 MHz. Dejando claro desde aquí que las bajas velocidades de reloj provistas en esta familia de componentes no representan algún impedimento en sus aplicaciones.

El procesador integra modularmente: un controlador de LCD( ), memoria RAM estática, soporte USB( ), un convertidor A/D y un controlador MMC/SD que permite manejar un poderosos conjunto de periféricos para enriquecer la experiencia multimedia.

Esta familia de procesadores es la primera en incorporar la tecnología Bluetooth para diversas aplicaciones. A fin de conocer sus bondades esta tecnología será descrita más adelante.

La selección de este procesador de quinta generación cubre fácilmente los requerimientos planteados: Presenta un alto nivel de integración en un chip, provee un buen rendimiento en un diseño de consumo de baja energía, utiliza la tecnología Bluetooth con interfaces de alta velocidad hacia otras soluciones Bluetooth lo que le permite dar un soporte para una gran variedad de aplicaciones multimedia, incluyendo los PDA's más populares, teléfonos inteligentes y la siguiente generación de dispositivos inalámbricos.

2.2 DIAGRAMA A BLOQUES

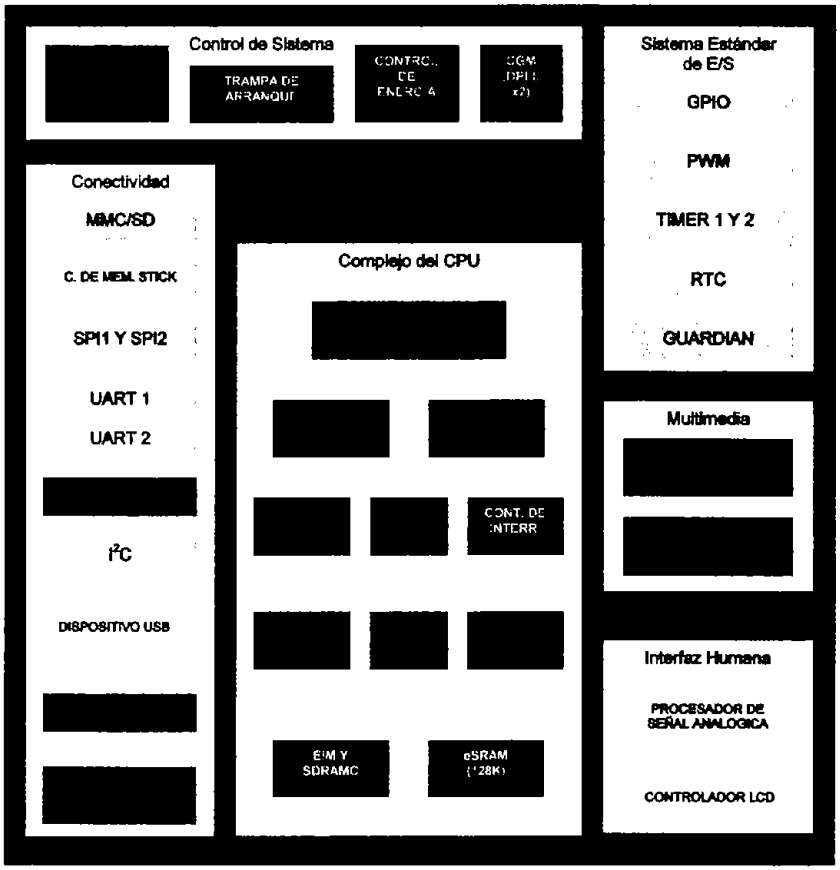

Figura 2.1. Diagrama Funcional a Bloques del MC9328MX1

2.3 ARQUITECTURA

A fin de poder dar soporte a una gran variedad de aplicaciones, la arquitectura del procesador MC9328MX1 se encuentra dividida modularmente en 6 partes.

A continuación explicaremos cada una de ellas, dado que es indispensable su conocimiento para el usuario que desee programar o entender la programación de aplicaciones:

2.3.1 Complejo del CPU.

En esta sección se encuentran los componentes nuevos y mejorados con los que este microprocesador alcanza mayores velocidades y capacidades.

2.3.1.1 El núcleo del procesador ARM920T.

El microprocesador MC9328MX1 utiliza como núcleo de componentes ARM920T, el cual presenta una velocidad máxima de procesamiento de 200 MHz; posee un cache de microinstrucciones y un cache de datos, cada uno de 16K palabras a disposición del programador.

Un motor ARM9 de alto desempeño RISC de 32 bits; un conjunto de instrucciones ARM Thumb de 16 bits para un nivel elevado de densidad de código; cuenta con un módulo de depuración de software embebido ICE JTAG 100% compatible con los procesadores ARM7TDMI; este nuevo núcleo ARM9TDMI incluye caches integrados, buffers de escritura y unidades de interfaz de bus, lo cual provee transparencia.

El CPU-cache; presenta una Arquitectura de Microcontrolador de Bus Avanzado (AMBA); el Bus de reloj incluye configuraciones asíncrona, síncrona y simple; Incluye un bloqueo de cache para soportar cargas mixtas en tiempo real y aplicaciones de usuario, por último contiene una Unidad de Manejo de Memoria Virtual (VMMU).

2.3.1.2 Bus de interfaces AHB a IP (AIPIs).

El microprocesador provee una interfaz de comunicación entre el bus AHB( ) de alta velocidad y el bus IP( ) de baja velocidad para periféricos esclavos lentos.

2.3.1.3 Modulo de interfaz externa (EIM)

Este módulo presenta, un selector de hasta 6 dispositivos externos, cada uno con 16 Mbytes de espacio de direccionamiento (el selector de chips para la ROM soporta un máximo de 32 Mbytes de espacio de direccionamiento); cuenta con protección por hardware; tamaño de puertos y estados de espera programables para cada chip seleccionado; también puede hacer selección de arranque de ROM interno/externo; un contador elegible de bus guardián; soporte para memoria flash Intel o AMD con enrutamiento de 32 bits de datos; un controlador de interrupciones para manejar un máximo de 64 fuentes de interrupción y capacidad de interrupción vectorizada con prioritización para 16 fuentes.

Este módulo representa una de las características notables que hacen que este procesador sea altamente apreciado por el programador, sus características en lo referente a capacidad de manejo de hasta 64 dispositivos diferentes capaces de interrumpir el proceso y por ende el tener 64 diferentes rutinas de interrupción, de las cuales 16 son vectorizadas permitiendo al programador direccionar vectorialmente y por hardware los dispositivos externos.

2.3.1.4 Controlador SDRAM (SDRAMC)

Este controlador presenta, un soporte para cuatro bancos de memoria de transferencia sencilla de 64 Mbits, 128 Mbits y 256 Mbit SDRAM( ); un soporte para memoria rápida flash con interfaz SDRAM Micron SyncFlash; una interfaz tipo PC100; la cual cuenta con una longitud extendida optimizada para la actualización del buffer de LCDC( ); Además, cuenta con software configurable para diferentes requerimientos de sistema; un temporizador de auto actualización y maquina de estados integrados; entrada y salida soportada por hardware actualizado por si mismo para mantener los datos validos durante el reinicio del sistema y en los modos de baja energía; finalmente un temporizador de auto apagado (Reloj de suspensión).

2.3.1.5 Controlador de acceso directo a memoria (DMAC)

Este controlador presenta, 11 canales de soporte de memoria lineal, memoria 2D, FIFO; soporte para puertos FIFO de 8, 16 o 32 bits de tamaño y transferencia de datos de tamaño de memoria; longitud configurable de DMA para cada canal hasta 16 palabras, 32 medias palabras o 64 bytes; control de la utilización del bus para un canal que no es disparado por una petición de DMA.

Transferencia completa de carga de datos o interrupciones para transferencia de errores provistas por un manejador de interrupciones; una señal de adquisición provista por los periféricos después de que la carga del DMA es completada.

2.3.1.6 Rango de operación de voltaje.

Los voltajes de operación del procesador son:

- Voltaje de I/O ( ) – 1.62 V a 1.98 V o 2.7 V a 3.3 V
- Voltaje interno lógico – 1.62 V a 1.98 V

2.3.1.7 Encapsulado MAPBGA de 256 pines.

Este microprocesador presenta un encapsulado de tipo MAPBGA de 256 pines con dimensiones de 14 mm x 14 mm x 1.3 mm y 0.8 mm de burbuja.

2.3.2 Sistema de control.

Este sistema se encarga del control de energía y del arranque del procesador.

2.3.2.1 Módulo generador de reloj (CGM) y módulo de control de energía.

Estos dos módulos contienen, PLL's (Phase Look Loops) anillos digitales de fase bloqueada (PLL's) y un controlador de reloj para la generación de los relojes internos;
Un MCUPLL que genera las frecuencias de reloj para el CPU utilizando un cristal de 32kHz, 32.768kHz o 38.4kHz;

El PLL de sistema genera las fases del reloj del sistema y el reloj de 48MHz para el USB utilizando cristales de 16MHz o cualquier cristal de 32kHz, 32.768kHz o 38.4kHz;
Soporte para los tres modos diferentes de consumo de energía: activo, espera y detenido.

2.3.2.2 Modo de trampa de arranque.

El cual permite al usuario inicializar al sistema y descargar programas o datos a la memoria de este a través del USART (Universal Synchronous Asynchronous Receiver Trassmiter); acepta la ejecución de comandos para ejecutar programas almacenados en memoria; soportar la operación memoria/registro lectura/escritura del tamaño de datos elegible entre byte, palabra o media-palabra y proveer un buffer de 16 bytes de instrucciones de almacenamiento y ejecución para el núcleo ARM920T.

2.3.2.3 Características de manejo de energía.

El sistema cuenta con un sintetizador programable de reloj que usa cualquier cristal de 32 kHz, 32.768 kHz o 38.4 kHz para el control de frecuencias; capacidades de interrupción por baja energía y permite que cada módulo pueda ser apagado individualmente.

2.3.3 Sistema estándar de E/S.

Este sistema se encarga principalmente del control de E/S y de los contadores y temporizadores del procesador.

2.3.3.1 Dos contadores/temporizadores de 32 bits de propósito general.

Este dispositivo genera interrupciones en forma automática basándose en pines programables temporizadores de entrada/salida;
Cuenta con capacidad de entrada de captura con limite programable de disparo; y comparación de salida con modo programable.

2.3.3.2 Temporizador guardián.

Este temporizador puede ser programado desde 0.5 seg. hasta 64 seg. con una resolución de 0.5 seg.

2.3.3.3 Temporizador muestreador/reloj de tiempo real (RTC).

Su operación puede ser a 32.768kHz, 32kHz o 38.4kHz; cuenta con características completas de reloj: segundos, minutos, horas y días; es capaz de contar hasta 512 días; también integra una alarma programable diariamente con interruptor; además cuenta con interruptores de una vez por segundo, una vez por minuto, una vez por hora y una vez por día.

2.3.3.4 Módulo de modulación por amplitud de pulso (PWM)

Sus características son, FIFO 4 x 16 para minimizar la sobrecarga de interrupciones; tiene una resolución de 16 bits; además de generación de sonidos y melodías.

2.3.3.5 Puertos de E/S de propósito general (GPIO)

Cuenta con capacidad de interrupción; además de que integra 110 pines de E/S multiplexados en total con sus funciones mayormente dedicadas a la eficiencia de cada pin.

2.3.4 Interfaz humana.

Esta parte se encarga de la interacción entre el usuario y el sistema.

2.3.4.1 Controlador LCD (LCDC)

Sus características son:

- Tamaño de pantalla programable (máximo de 640 x 512 píxeles) con soporte para paneles monocromáticos, STN a color y TFT a color;
- Soporte para 4 bpp (bits por píxel) y 8 bpp para paneles de color pasivos;
- Soporte para 4 bpp, 8 bpp y 16 bpp para paneles TFT;
- Hasta 256 colores de salida de una paleta de 4096 para 8 bpp; en modo de color STN, la máxima profundidad de color es 8 bpp; en modo de B/N, la máxima profundidad es 4 bpp; hasta 16 escalas de gris salidas de 16 paletas;
- Cuenta con 128 kbytes de SRAM integrada para la actualización de pantalla y retención de datos; es capaz de manejar directamente los controladores de LCD de los fabricantes más populares incluyendo Motorola, Sharp, Hitachi y Toshiba; cuenta con soporte para un bus de datos para paneles TFT de tamaño de 12 o 16 bits;
- Cuenta con soporte para operaciones lógicas entre el cursor de hardware de color y el fondo; tiene la opción de utilizar la memoria del sistema como memoria de video.

2.3.4.2 Modulo de procesamiento de señal analógica (ASP)

Este módulo esta conformado por:

- Un CODEC de 16 bits (ADC de voz y DAC de voz) para procesamiento de voz;
- Un ADC (13 bits de resolución, 12 bits de exactitud) con 3 entradas para la pantalla sensible y detección de bajo voltaje;
- Un segundo ADC mejorado (13 bits de resolución, 12 bits de exactitud) para uso como un segundo;
- Un ADC con soporte para DMA; cuenta con circuitería mejorada para la pantalla sensible con manejo de baja energía.

2.3.5 Conectividad

2.3.5.1 Transmisores/receptores asíncronos universales (UART 1 y 2)

Estos dos UART's soportan la operación de transmision/recepcion serial de datos: 7 u 8 bits de datos, 1 o 2 bits de parada y paridad programable (par, impar o ninguna); tienen un rango programable de baudios hasta 1.00 MHz; cuentan con un FIFO de 32 bits en Transmisión (Tx) y FIFO de 32 bits de media-palabra en Recepción (Rx) que soporta auto baudio; además soporta el estándar IrDA 1.0.

2.3.5.2 Dos interfaces periféricas seriales (SPI)

Las características de estas dos interfaces son, SPI 1 configurable a maestro/esclavo únicamente; cuenta con transferencia programable de datos hasta 16 bits; además incluye una FIFO de 8 x 16 tanto para datos en Tx y Rx.

2.3.5.3 Dispositivo de bus serial universal (USB)

Este dispositivo cumple con la Especificación del Bus Serial Universal ver1.1; soporta hasta seis terminales lógicas; cuenta con soporte para encendido remoto; opera a velocidad total (12MHz).

2.3.5.4 Modulo controlador anfitrión para tarjetas multimedia y de seguridad digital (MMC/SD)

Es totalmente compatible con la Especificación de Sistema MMC versión 2.2 y con la Especificación de Tarjeta de Memoria SD 1.0 y la Especificación SD I/O 0.8e con 1 o 4 canales; hasta 10 tarjetas MMC y una SD son soportadas por el estándar (máxima transferencia de datos con un máximo de diez tarjetas); soporta el intercambio instantáneo de tarjetas; su transferencia de datos varia desde 20 Mbps hasta 80 Mbps.

2.3.5.5 Controlador anfitrión para memoria tipo stick (MSHC)

Cuenta con un buffer integrado tipo FIFO de 8 bytes para transmitir y recibir; además integra un circuito CRC; también soporta una fuente de reloj serial externa o interna; integra un comando de ejecución automática cuando una interrupción del Stick de Memoria es detectada (puede ser habilitado o deshabilitado); cuenta con dos pines integrados de propósito general para la detección de inserción o extracción del Stick de Memoria.

2.3.5.6 Modulo controlador para tarjeta inteligente (SIM)

Este módulo de interfaz de tarjeta inteligente cumple con la norma ISO7816; incluye una FIFO de recepción de 32 palabras; puede detectar la presencia de tarjeta SIM.

2.3.5.7 Módulo de interfaz serial síncrona y de sonido inter-IC (SSI/I^2S)

Soporta la interfaz genérica SSI para un chip de audio externo o un interprocesador de comunicación; soporta el estándar Philips de Sonido Inter-IC (I^2S) para un chip de audio digital externo.

2.3.5.8 Modulo de bus inter-IC (I^2C)

Este módulo soporta el estándar I^2C de Philips para el control digital externo; cuenta con el soporte para dispositivos de 3.3 V; cuenta con software programable de 1 a 64 diferentes frecuencias de reloj seriales; genera y detecta la señal de arranque y del bit de adquisición; además puede detectar si es bus esta ocupado.

2.3.5.9 Acelerador Bluetooth (BTA)

Cuenta con un motor de procesamiento de banda-base de bajo nivel; con un módulo coprocesador de selección de frecuencia, integra un buffer Rx y Tx de 32 palabras (16 bits); tiene un controlador programable de RF que soporta tres frentes (incluyendo SPI / controlador µWire); soporta transceptores externos de fabricantes como Motorola (MC13180) y Silicon Wave (SiW 1502); además incluye un temporizador de aplicaciones Bluetooth y otro de reinicio para soporte de baja energía.

2.3.6 Multimedia.

2.3.6.1 Puerto de video

Este puerto soporta un sensor externo de tipo CMOS de entrada de video.

2.3.6.2 Acelerador multimedia (MMA)

Las características de este acelerador es la adición de un MAC para la operación FIR y FFT donde las aplicaciones de MP3 ahorran entre 10% y 15% de MIPS en el CPU y además cuenta con un acelerador de hardware DCT/iDCT donde las aplicaciones de decodificación de MPEG4 ahorran aproximadamente 10% de MIPS en el CPU.

CAPITULO III

COMUNICACIONES CON LA PC

3.1 ADMINISTRADOR DE HOTSYNC

Los tipos de conexiones que el usuario puede utilizar entre la Palm y la PC pueden ser Local Serial, Local USB, Modem, Red e Infrarrojo pudiendo estar más de una opción seleccionada.

Dentro del Administrador de HotSync también encontramos las siguientes opciones:

Setup (configuración)- presenta una ventana de diálogo para configurar las preferencias del administrador y seleccionar el tipo de conexión.

Custom (personalizar)- presenta una ventana de diálogo(figura 3.2) mostrando todos los conduits disponibles en un perfil dado y permite al usuario modificar el tipo de sincronización para cada conduit.

Cuando el usuario selecciona un conduit y da click en el botón Change, otra ventana de diálogo como la que se muestra en la figura 3.1 aparece.

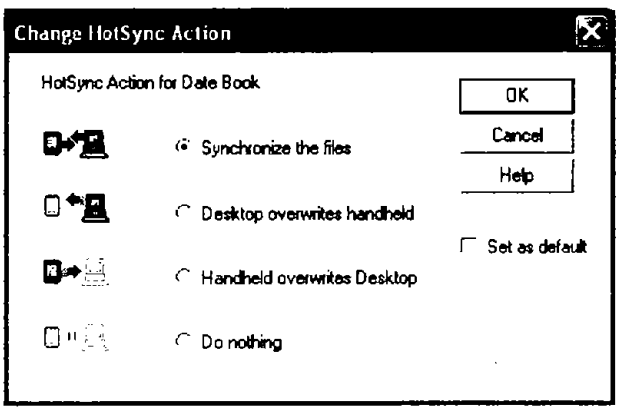

Figura 3.1. Ventana de diálogo de acciones de cambio en el HotSync.

Cuando el usuario cambia la selección actual y da click en el botón OK, la acción seleccionada es válida sólo para la siguiente sincronización y revierte los cambios previos para todas las sincronizaciones posteriores. Esta es una preferencia temporal, sin embargo, si el usuario selecciona la opción Set as default (predeterminar) habrá aplicado una preferencia permanente.

3.1.1 Primera sincronización

Proceso de sincronización. La primera vez que el usuario sincroniza sus datos, debe introducir su información en la PC cuando el Administrador de HotSync lo pida. Después de introducirlos y sincronizarlos dicho administrador reconoce al dispositivo y no vuelve a preguntar por esta información otra vez.

Durante la primera sincronización el administrador asigna al dispositivo un ID único de usuario. También transfiere a la Palm los datos que se introdujeron en la PC. Este proceso no vuelve a ocurrir en futuras sincronizaciones.

Esta primera operación debe ser realizada a través de una conexión local. Sincronizaciones posteriores pueden ser hechas con cualquier conexión válida.

3.1.2 Tipos de conexiones

El administrador de HotSync permite cuatro diferentes tipos de conexión entre la Palm y la PC. Cada una de estas conexiones provoca que el administrador de HotSync inicie actividades de sincronización de diferente manera, como se describe en la tabla 3.1.

Tabla 3.1. Tipos de conexión del HotSync.

Tipo de Conexión	Descripción de la Activación
Cable directo utilizando la base	El cable de la base conecta a la Palm con la PC utilizando un puerto serial o USB. Cuando el usuario presiona el botón de HotSync en la base, un circuito es cerrado en la Palm. Esto activa el cliente en el dispositivo el cual "despierta" al Administrador de HotSync en la PC.
Modem	La Palm está conectada a un modem que es marcado dentro de la PC. El usuario configura al HotSync en la PC para que utilice un modem y entonces seleccione Modem Sync en la Palm. La Palm marca a la PC y se conecta con el administrador de HotSync.
Red	El cable de la base esta conectado a una PC que se encuentra en una red de área local. La PC destino está también conectada a la red y el Administrador de HotSync está activo. Cuando el usuario presiona el botón de sincronización la conexión se realiza a través de la red.
Infrarrojo	Si la Palm tiene un puerto infrarrojo este puede sincronizarse con una PC equipada con un puerto infrarrojo estándar.

Cada conduit es diseñado individualmente para ser usado con los cuatro tipos de conexiones soportadas. De ahí entonces la sincronización por modem o red no soporta intervención del usuario, cualquier diseño debe evitar ventanas o interfases de usuario.

3.2 PROTOCOLOS UTILIZADOS

A través de estos protocolos la comunicación entre los PDA's será cada vez más recurrente debido a que incorporan mejoras que reducen la carga en la transmisión de datos y permiten una comunicación sencilla y segura.

3.2.1 La tecnología Bluetooth

La característica principal de la tecnología Bluetooth es que todas las computadoras y sus periféricos se comuniquen entre si, sin necesidad de cables, ya que mediante ondas de radio permite la comunicación en un entorno reducido entre cualquier dispositivo: computadoras, impresoras, auriculares, teléfonos, mandos a distancia, etc., que incorporen un pequeño circuito integrado de bajo costo. Este tipo de dispositivos ha empezado a inundar el mercado, como ya lo ha hecho la telefonía a través de la tecnología celular.

El nombre de "Bluetooth" viene del rey danés que unificó, cristianizó y unió los reinos de Noruega y Dinamarca en el siglo X, asemejándose a lo que actualmente hace esta tecnología.

Bluetooth es una especificación para la industria de la Informática y Telecomunicaciones que describe cómo se pueden interconectar dispositivos con características de dispositivos móviles, como lo son: teléfonos celulares, Asistentes Personales Digitales (PDA), computadoras y muchos otros dispositivos, ya sea en el hogar, en la oficina, en el automóvil, etc.

Bluetooth utiliza una conexión sin hilos de corto alcance, no necesita de visión directa entre los dispositivos que se conectan, ya que utiliza ondas de radio. En definitiva, lo que se consigue con Bluetooth es eliminar los cables, siempre que la distancia sea pequeña, dentro de lo que puede considerarse como un radio reducido o área local pequeña, pudiendo tratarse de una habitación, un despacho, el interior de un coche, etc.

3.2.1.1 Bluetooth, comparable con otras tecnologías, pero más avanzada.

La tecnología Bluetooth elimina el proceso de configuraciones complicadas de red, donde el usuario tenía que establecer numeros, grupos y nombres para que los dispositivos se pudieran comunicar entre sí.

Frente a otras tecnologías, como la de infrarrojos promovida por la IrDA o DECT y la propia 802.11 para LAN's, Bluetooth cuenta con el apoyo de la industria de las telecomunicaciones, lo que en cierta medida garantiza su éxito. Aunque ya existe un alto número de fabricantes que incorporan la interfaz IrDA (infrarrojos) en sus teléfonos entre ellos Ericsson, Motorola y Nokia, su uso resulta frustrante para muchos usuarios que tratan sin éxito de descargar información desde sus PC's o PDA's a sus teléfonos móviles, o viceversa.

Los dispositivos que incorporan Bluetooth se reconocen y se comunican de la misma forma que lo hace una computadora con su impresora; el canal permanece abierto y no requiere la intervención directa y constante del usuario cada vez que se quiere enviar algo.

3.2.1.2 El origen de la especificación Bluetooth

En 1994 la compañía Ericsson inició una investigación sobre la viabilidad de una interfaz vía radio, a un costo accesible y con un bajo consumo de energía, para la interconexión entre teléfonos móviles y otros accesorios, siempre con la intención de eliminar cables entre aparatos. La investigación partía de un proyecto que investigaba multicomunicadores conectados a una red celular, hasta que se llegó a un enlace de radio de corto alcance, llamado *MC link*. Conforme éste proyecto avanzaba se vio claro que éste tipo de enlace podía ser utilizado en un gran número de aplicaciones, ya que tenia como principal virtud el que se basaba en un chip de radio ya desarrollado relativamente económico.

Esta especificación surgió, a principio de 1998, de la colaboración de varias empresas líderes de la industria de las TIC: Ericsson, Nokia, Intel, IBM, Toshiba, Motorola y, más tarde, 3Com, Lucent, Microsoft y Motorola, que constituyeron el SIG (Special Interest Group), al que ya pertenecen mas de 2.000 empresas (Grupo de adeptos), que han adoptado esta tecnología para desarrollarla en sus propios productos, que empezaron a salir al mercado en el primer semestre del año 2001, sin tener que pagar regalías.

Una alternativa a Bluetooth, para aplicaciones en electrodomésticos (línea blanca) en el hogar, es Whitetooth -de ahí el nombre-, siendo una versión limitada de la primera, pero suficiente para el uso que se pretende dar. Esta tecnología es joven y aún tardará varios años en estar a punto.

3.2.1.3 Cómo funciona Bluetooth

Bluetooth permite que diversos aparatos se contacten entre ellos, dentro de un radio de acción de unos pocos metros. Para utilizarlo hay que equipar cada dispositivo con un microchip que transmite y recibe en la frecuencia de 2,4 GHz (2,402 y 2,480 MHz), una banda que está disponible para uso libre en todo el mundo, con algunas variaciones de

ancho de banda en países como España, Francia y Japón y que no necesita licencia ya que utilizan la banda ISM para uso comercial, la cual no la requiere: es decir, el FCC simplemente asigna la banda y establece las directrices de utilización, pero no se involucra ni decide sobre quién debe transmitir o recibir sobre esa banda. Además de los canales para datos, que admiten una velocidad de hasta 721 kbit/s, están disponibles tres canales de voz a 64 kbit/s. Con la nueva versión Bluetooth 2.0 se espera alcanzar hasta 4 Mbit/s y mayores distancias.

Los protocolos que se utilizan en una comunicación Bluetooth son similares a los que se emplean con tecnología de infrarrojos, por lo que no ha hecho falta desarrollar otros nuevos protocolos, pero mientras en una comunicación por infrarrojos se requiere un enlace visual entre dispositivos, con Bluetooth no es necesario, ya que emite en todas las direcciones e incluso atraviesa paredes.

Las conexiones son uno a uno con un rango máximo de 10 metros, aunque utilizando amplificadores, y con un consumo mayor de potencia, se puede llegar hasta los 100 metros, pero se introduce alguna distorsión en la señal. Al ser la emisión de señal de una potencia muy baja, se elimina cualquier peligro potencial para la salud.

3.2.1.4 Especificaciones del estándar

A continuación se listan las especificaciones principales de Bluetooth.

- Banda de Frecuencia: 2,4 GHz (Banda ISM).
- Potencia del transmisor: entre 1 y 100 mW, típica de 2,5 miliwatios.
- Canales máximos: hasta 3 de voz y 7 de datos por piconet.
- Velocidad de datos: hasta 720 kbit/s por piconet.
- Rango esperado del sistema: 10 metros (30 pies).
- Número de dispositivos: 8 por piconet y hasta 10 piconets.
- Tamaño del Módulo: 0,5 pulgadas cuadradas (9x9 mm).
- Interferencia: Bluetooth minimiza la interferencia potencial al emplear saltos rápidos en frecuencia =1.600 veces por segundo.

3.2.1.5 Definición de enlace físico

La especificación Bluetooth define dos tipos de enlace que permitan soportar aplicaciones multimedia:

- Enlace de Sincronización de Conexión Orientada (SCO)
- Enlace Asíncrono de Baja Conexión (ACL)

Los enlaces SCO soportan conexiones asimétricas, punto a punto, utilizadas normalmente en conexiones de voz, éstos enlaces están definidos en el canal, reservándose dos slots consecutivos (envío y retorno) en intervalos fijos. Los enlaces ACL soportan conmutaciones punto a punto simétricas y/o asimétricas, típicamente usadas en la transmisión de datos.

Un conjunto de paquetes se han definido para cada tipo de enlace físico:

- Para los enlaces SCO, existen tres tipos de slot simple, cada uno con una portadora a una velocidad de 64 kbit/s. La transmisión de voz se realiza sin ningún mecanismo de protección, pero si el intervalo de las señales en el enlace SCO disminuye, se puede seleccionar una velocidad de corrección de envío de 1/3 o 2/3.
- Para los enlaces ACL, se han definido el slot-1, slot-3, slot-5. Cualquiera de los datos pueden ser enviados protegidos o sin proteger con una velocidad de corrección de 2/3. La máxima velocidad de envío es de 721 kbit/s en una dirección y 57.6 kbit/s en la otra.

3.2.2 Protocolo de aplicación inalámbrico (WAP)

WAP es un protocolo promovido por el Foro WAP (liderado principalmente por Ericsson, Motorola y Nokia). Fue diseñado para entregar información a teléfonos celulares, que generalmente contienen pequeñas pantallas para presentar sólo unas cuantas líneas de texto. La arquitectura WAP ordena el software en los dispositivos portátiles y las aplicaciones y datos en un servidor Web. Una puerta de enlace es requerida para trasladar los protocolos y formatos de Internet desde y hacia el protocolo inalámbrico. Algunas puertas de enlace pueden ser incompatibles, para usar ciertos teléfonos la compañía debe tener los enlaces de la misma marca.

3.2.2.1 Transferencia de datos.

Los dispositivos WAP solamente pueden recibir datos sobre redes compatibles. Estos datos deben ser trasmitidos a los dispositivos sobre una red de área amplia inalámbrica que incorpore una puerta de enlace WAP compatible (diferentes fabricantes WAP tienen diferentes implementaciones).

3.2.2.2 Acceso de datos.

Las aplicaciones WAP sólo están disponibles cuando la cobertura de la red está disponible. Para acceder datos el usuario debe tener la cobertura de la red inalámbrica o algún otro tipo de conexión al servidor donde la aplicación y la base de datos residen. Como resultado, este protocolo es inapropiado para aplicaciones críticas (administración de clientes, administración de inventarios, etc.) que deben estar constantemente accesibles. Además, debido a que todos los procesos se llevan a cabo en la Web, las aplicaciones WAP pueden ser procesos tediosos que involucren múltiples decisiones o transacciones.

3.2.2.3 Interacción de datos.

Las aplicaciones WAP están diseñadas para pequeñas pantallas y uso limitado del usuario. La interacción del usuario con las aplicaciones WAP tienden a ser limitadas debido a que la mayoría de las aplicaciones están diseñadas para teléfonos inalámbricos con pequeñas pantallas que muestran unas cuantas líneas de texto suficientes para cincuenta caracteres o menos. Los teléfonos WAP también tienen pequeños teclados que hacen difícil la inserción de datos. El correo electrónico móvil, que es una aplicación potencialmente "asesina" para las redes inalámbricas, ha sido severamente restringido por el tedio de contestar los correos mediante el uso de estos pequeños teclados. Como resultado de estas limitaciones, los desarrolladores generalmente toman un acercamiento minimalista, presentando sólo una pieza de datos a la vez. En la mayoría de los casos se evaden los gráficos (excepto para mapas y aplicaciones geográficas) así como los elementos de la interfaz de usuario que podría crear una experiencia enriquecida y mejorar la legibilidad y facilidad de uso.

3.2.2.4 Desarrollo simplificado.

El desarrollo está generalmente dirigido a un dispositivo y servicio WAP en especifico. Los desarrolladores tienen que restringir sus aplicaciones a un dispositivo y servicio en específico o construir mas de una versión. Un gran número de fabricantes no se han adherido totalmente al estándar y han añadido mejoras y extensiones de su propiedad.

3.2.2.5 Seguridad inalámbrica.

El protocolo WAP incorpora una Capa de Seguridad de Transporte Inalámbrico (WTSL), que es un certificado digital que provee una fuerte seguridad industrial pero que es todavía un protocolo en desarrollo.

3.2.2.6 Pruebas futuras de aplicaciones inalámbricas.

Es difícil decir cual es el futuro del WAP. Hay personas en la industria que creen que se convertirá en un estándar obsoleto, otras que lo ven como una piedra angular hacia el XML y redes IP inalámbricas, y otras personas que lo ven siendo desplazado antes de que se establezca.

3.2.3 Web clipping

Web clipping es la mejor opción para aplicaciones de red que requieren una presentación enriquecida y mayor interacción con el usuario que las típicas aplicaciones WAP. Esta hecho a la medida para presentar múltiples piezas de información simultáneamente, como lo son catálogos en línea, guías de viajes, buscadores de bienes raíces o aplicaciones comerciales. Desarrollado por Palm e incluido en la plataforma Palm OS, es soportado por cada dispositivo que este diseñado para tal fin hecho por Palm. Actualmente existen más de cuatrocientas aplicaciones disponibles. Web clipping optimiza el HTML que es un lenguaje universal de presentación en las pequeñas pantallas de las Palm. También optimiza la transferencia de datos en redes inalámbricas mediante el almacenamiento estático de contenido HTML en la Palm y así minimizar la cantidad de datos que viajan sobre el enlace inalámbrico.

La arquitectura incluye una aplicación cliente que corre en la Palm, un servidor proxy y un servidor de contenido. La aplicación cliente está construida en HTML y es trasladada al formato de aplicación de Web clipping, el cual es un subconjunto del estándar HTML 3.2. El servidor proxy maneja la traslación entre el HTML del servidor de contenido y el HTML de Web clipping que la Palm entiende.

3.2.3.1 Transferencia de datos.

Las aplicaciones en Web clipping pueden enviar y recibir datos sobre la mayoría de redes inalámbricas existentes en el mundo. Además, mediante el uso del Kit de Internet Móvil de Palm, los datos pueden ser transferidos a aplicaciones Web clipping a través de un haz infrarrojo o un cable a un teléfono celular. Y aún cuando no haya disponibilidad de una red inalámbrica, estos datos pueden obtenidos a través de un modem conectado a la red telefónica.

3.2.3.2 Acceso de datos.

Como sucede con WAP, el acceso de datos solo puede ocurrir cuando el usuario tiene una conexión de red, debido a que la totalidad de los datos y la mayor parte del procesamiento residen en el servidor.

3.2.3.3 Interacción de datos.

Las aplicaciones para esta plataforma generalmente muestran más información en pantalla y tienen una mejor interacción con el usuario que las aplicaciones WAP. El tamaño de la pantalla es lo suficientemente grande para mostrar múltiples piezas de información a la vez.

3.2.3.4 Desarrollo simplificado.

Los desarrolladores Web que tengan experiencia en HTML pueden comenzar a desarrollar Web clipping con la mínima preparación. En algunos casos no es necesario algún cambio, solo la creación de la aplicación de búsqueda.

3.2.3.5 Seguridad inalámbrica.

En esta plataforma, el cifrado y autenticación entre la Palm y el servidor Web son realizados mediante una criptografía de curva elíptica, la cuál ofrece muy altos niveles de seguridad con llaves de tamaño pequeño.

3.2.3.6 Pruebas futuras de aplicaciones inalámbricas.

Mirando al futuro y basado en el pasado, el usuario puede confiar que las aplicaciones Web clipping correrán en futuros dispositivos Palm y en las redes inalámbricas dominantes.

CAPITULO IV

EMULADOR DE PALM OS

El Emulador de Palm OS (POSE) es un programa de computadora que simula la existencia del hardware de la plataforma Palm, esto significa que el hardware de la Palm es representado o simulado en software proveyendo la habilidad al usuario desarrollador de software de probar y depurar programas para la Palm en una computadora personal bajo cualquier S. O.

Cuando uno ejecuta una aplicación para la Palm con el POSE en la PC, el emulador ejecuta las instrucciones, actualiza la pantalla del dispositivo emulado, trabaja con registros especiales y maneja las interrupciones en la misma manera que lo hace el procesador físico real dentro de los dispositivos Palm. La diferencia es de que el emulador ejecuta estas instrucciones en nuestra PC pudiendo ofrecer características especiales al diseñador como el debugging o depuración de fallas y la operación paso a paso, instrucción por instrucción..

4.1 CARACTERÍSTICAS.

El POSE presenta una imagen en pantalla que luce exactamente como un dispositivo Palm.

El usuario puede seleccionar que versión y tipo de dispositivo Palm quiere emular. También puede especificar si se quiere que la imagen grafica el POSE tenga el doble de tamaño lo cual continuara mostrando una exacta representación de la Palm en la pantalla de la PC más fácil de ver.

El usuario puede utilizar el mouse de la PC justo como se utiliza el stylus en un dispositivo Palm. Incluso se puede utilizar el software de escritura Graffiti con el POSE y el mouse. También el simulador incluye atajos con el teclado que pueden utilizarse en la simulación.

El usuario puede utilizar el simulador para realizar depuración de las aplicaciones e incluso utilizar herramientas externas de depuración extensiva para sus aplicaciones. Cuando el usuario conecta el emulador con el Depurador de Palm el puede depurar de la misma manera que con el hardware real del dispositivo.

4.1.1 Características estándar.

El POSE emula exactamente el hardware de la Palm e incluye las siguientes características:

- Una replica exacta de la pantalla de la Palm, incluyendo el área de Graffiti y sus iconos circundantes

- Emulación del Stylus con el mouse de la PC

- Emulación de los botones físicos en la Palm, incluyendo:
  - Botón de on/off
  - Botones de aplicación
  - Botones de arriba y abajo
  - Botón de reinicio
  - Botón de HotSync

- Habilidad de agrandar la pantalla para una legibilidad y presentación mejoradas

- Pantalla con luz trasera

- Emulación de puertos de comunicaciones para modem y sincronización

4.1.2 Características de depuración

El emulador provee un gran número de características de depuración que ayudan al desarrollador a detectar problemas de código y operaciones inseguras en su aplicación. El POSE incluye las siguientes características, capacidades y habilidades de depuración:

- Uso de una herramienta de prueba automatizada llamada Gremlins, la cual genera eventos aleatorios repetidamente

- Soporte para depuradores externos, incluyendo el Depurador de Palm, el Depurador CodeWarrior de Metrowerks y gdb

- Monitoreo de acciones de las aplicaciones, incluyendo varios accesos a la memoria y actividades de bloques de memoria

- Registro de actividades de aplicaciones, incluyendo eventos manejados, llamadas a función y códigos del CPU ejecutados por la aplicación

- Perfilado del desempeño de la aplicación

CAPITULO V

CONSTRUCCIÓN DE CONDUITS PARA PALM

Conceptualmente, un conduit es la parte que mueve los datos desde y hacia la Palm, conectando la aplicación de la Palm con su contraparte en la PC. Un conduit es una pieza de software que corre en la PC cuando la Palm esta sincronizando bajo el control del administrador de HotSync.

Por diseño un conduit está dedicado a una sola aplicación en la Palm, el cual tendrá una o más bases de datos asociadas en la Palm. Hay que tener en cuenta que aunque el conduit es una pieza de software en la PC los datos de dicha aplicación pueden estar localizados donde sea (en la PC, en una base de datos relacional o incluso en la Internet). Una vez que uno ha decidido donde residirán los datos de nuestras aplicaciones es ahí cuando surge nuestra responsabilidad de construir un conduit capaz de entregar los datos hacia la Palm. El Kit de desarrollo de conduits provee un esquema para el desarrollo de conduits en ActiveX.

El alto nivel de la arquitectura del HotSync es mostrado en la figura 5.1.

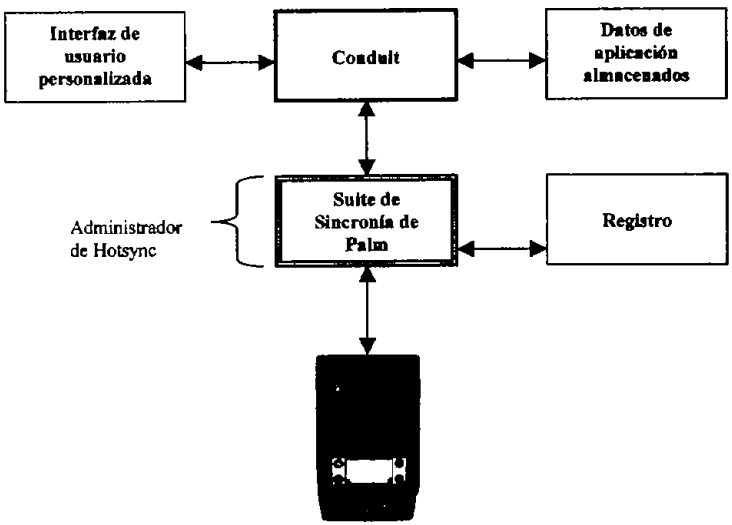

FIGURA 5.1. Vista de la arquitectura del HotSync.

El administrador de HotSync de la Palm tiene el control del proceso de sincronización. Mantiene una lista de los conduits configurados en el registro del sistema y maneja la interfaz con la Palm. El administrador de HotSync utiliza un conjunto de objetos COM (la Suite Sync API) para comunicarse con los conduits ActiveX. Los conduits manejan los datos específicos de una aplicación y suplen la interfaz de usuario si es apropiado.

5.1 APLICACIONES Y CONDUITS.

Antes de abordar los mecanismos de desarrollo de conduits vamos a revisar el concepto de sincronización de datos tal como se aplica hacia aplicaciones de Palm y bases de datos. Si la Palm es una extensión de la PC entonces es natural preguntarnos: ¿Cómo deberían fluir los datos entre la Palm y la PC? Esto dependerá en gran medida en el diseño y propósito de nuestra aplicación y sus bases de datos.

Programas de utilidades o entretenimiento rara vez utilizan un conduit, ya que no tienen datos en movimiento hacia y desde la PC. Estas aplicaciones usualmente guardan cualquier dato en la base de datos del sistema, el cual es automáticamente respaldado y restaurado por el administrador de HotSync.

El conduit unidireccional es el más típico de todos ya que transfiere los datos de la Palm hacia la PC o viceversa. Este tipo de conduit es útil en aplicaciones como cuestionarios, donde la Palm es utilizada principalmente como una herramienta remota de recolección de datos.

Las aplicaciones nativas para la Palm como la Agenda y la Lista de Tareas utilizan un conduit de tipo espejo donde los cambios en la PC y en la Palm son replicados en ambas direcciones. Este tipo de sincronización es tan importante que la Compañía ha documentado exactamente como un conduit de espejo debería ser.

Finalmente hay funciones del sistema como la instalación y el respaldo que utilizan conduits de propósito especial para realizar sus funciones. El conduit de respaldo por default sincroniza las bases de datos de aquellas aplicaciones que no tienen un conduit en específico o de aquellas bases de datos que si lo tienen pero que su tipo no es de "DATOS".

5.1.1 Tipos de conduits.

La Compañía Palm ha diseñado el administrador de HotSync y la interfaz de conduits para soportar una gran variedad de sincronización y replicación de datos necesitadas. Cada conduit registra su identificador único de su creador y el tipo de sincronización con el administrador de HotSync. Hay que notar que nuestro conduit debe soportar más de un tipo de sincronización; en ese caso, uno debería proveer una interfaz de usuario para permitir a dicho usuario personalizar el comportamiento de nuestro conduit.

He aquí los tipos de conduits soportados por el Kit de desarrollo de conduits:

- Rápido
  Realiza una sincronización rápida.

- Lento
  Realiza una sincronización lenta.

- HHaPC
  Copia la base de datos de la Palm hacia la PC y sobrescribe todos sus antiguos registros.

- PCaHH
  Copia la base de datos de la PC hacia la Palm y sobrescribe sus antiguos registros.

- Instalación
  Instala nuevas aplicaciones en la Palm (función de sistema).

- Respaldo
  Respalda la base de datos de la Palm en la PC (función de sistema).

- SinTarea
  No realiza alguna sincronización.

- Instalación de Perfil
  Realiza una descarga de perfil (función de sistema).

El administrador de HotSync de la PC mantiene el rastro de cuando el usuario sincronizó su Palm con la PC. Utiliza esta información para determinar a qué conduits llamar y qué tipo de sincronización realizará cada uno.

NOTA: El administrador de HotSync soporta múltiples usuarios en la misma PC. También soporta a un único usuario con múltiples Palms. El API del HotSync provee identificadores durante una sesión de HotSync para habilitar a una conduit que determine qué usuario y/o dispositivo está siendo sincronizado.

Si un conduit no soporta una interfaz de usuario, el administrador de HotSync utiliza el tipo de sincronización por default.

NOTA: Los conduits deberían siempre tener un Identificador de Usuario. De otro modo el usuario dará click en el botón de "cambio" en el cuadro de diálogo y nada pasará. No solamente esto es rudo sino que dejará al usuario preguntándose si algo está dañado.

5.1.2 Sincronización de espejo.

Como se mencionó anteriormente, la sincronización de imagen de espejo requiere que un conduit replique los cambios entre la PC y la Palm. Los conduits de imagen de espejo deben soportar sincronizaciones lenta y rápida. Cuando nuestro conduit es llamado a

hacer una sincronización rápida sólo necesita checar los registros que estén nuevos o corruptos en las bases de datos de las aplicaciones tanto en la Palm como en la PC. Si nuestro conduit es llamado para una sincronización lenta este debe checar cada registro contenido en la Palm y en la PC. El administrador de HotSync pide una sincronización lenta cada vez que determina que la última sincronización no fue con la actual PC. De otro modo pide una sincronización rápida.

El manejador de la base de datos de la Palm soporta banderas por cada registro que siguen cualquier cambio a los registros en la base de datos. Estas banderas son listadas en la Tabla 5.1. En orden para implementar exitosamente un conduit de imagen de espejo nuestra base de datos en la PC debe soportar alguna o todas estas banderas por registro.

BANDERA	SIGNIFICADO
Changed	Crea un nuevo registro o edita uno existente
Deleted	Borra el registro
Archived	Hace una copia en archive y después borra el registro
Secret	Marca el registro como privado

TABLA 5.1. Banderas de registro en la base de datos de la Palm.

El Kit de Desarrollo de Conduits tiene una sección en decisiones de diseño e intercambios que uno puede utilizar para evaluar el tipo de conduit que nuestra aplicación puede soportar.

Si el usuario ha experimentado con cualquiera de las aplicaciones de la Palm entonces él espera que el conduit replique los cambios de la misma forma. Nuestro conduit debe de emular este comportamiento para que nuestra aplicación sea lo más cercana a las demás.

La Tabla 5.2 muestra los posibles estados para cada registro en una aplicación. Los estados de registro en la PC se encuentra en la fila superior de la tabla y los estados de registro de la Palm se encuentra en la primera columna de la tabla.

	Sin Registro	Sin Cambio	Cambio	Nuevo	Borrar
Sin Registro				D → P	
Sin Cambio			D → P		Remueve DP
Cambio		P → D	Conflicto		P → D
Nuevo	P→ D				
Borrar		Remueve DP	D → P		

TABLA 5.2. Estados de Registro de un Conduit de Espejo.

La acción que nuestro conduit debe tomar es encontrada en la intersección de cada uno de los posibles estados y es explicada en la siguiente lista:

D → P

El registro de la PC reemplaza el registro en la Palm.

P → D

El registro de la Palm reemplaza el registro en la PC.

Remover (DP)

Borra los registros de la PC y/o de la Palm.

Conflicto

Sigue la aplicación de la regla para resolver un conflicto de sincronización.

Después de que el conduit toma la acción indicada, el registro en la PC se encuentra en el mismo estado que el registro en la Palm.

Un conflicto ocurre cuando un registro es cambiado simultáneamente en la Palm y en la PC. Palm recomienda que este conflicto debe ser resuelto haciendo el cambio en ambas direcciones. El usuario entonces puede editar o borrar los datos tanto en la Palm como en la PC. En la siguiente sincronización el conduit limpiará todos los cambios.

La Tabla 5.3 muestra como manejar los registros de la base de datos de Palm que son marcados para ser archivados. En esta tabla los estados de los registros en la PC son puestos en la primera fila, mientras que los estados de los registros en la Palm se encuentran en la primera columna.

	Borrar	Sin Cambio	Sin Registro	Cambio
Archivar	Archivar, Remueve DP	Archivar, Remueve DP	Archivar, Remueve DP	
Archivar, Cambio				Conflicto
Archivar, Sin Cambio				D → P

TABLA 5.3. Acciones de sincronización de espejo con la petición de archivado de la Palm.

Un conflicto surge si un registro ha sido cambiado en ambas bases de datos al mismo tiempo. En este caso, el conflicto es enterrado por que el usuario ha pedido que el registro sea archivado (esto significa que no quiere ser visto por un buen rato). He aquí como el KDC dice que debe ser manejado el conflicto:

> Si los cambios son idénticos, archivar ambos registros, de la PC y de la Palm.
> Si los cambios no son idénticos, no archivar el registro de la Palm; en vez de eso añadir el registro de la PC a la base de datos de la Palm y añadir el registro de Palm a la base de datos de la PC.

La Tabla 5.4 muestra que hacer cuando un registro de la PC ha sido marcado para ser archivado. El caso más común, en donde el registro de la Palm no ha sido cambiado, es manejado mediante el archivado del registro y la remoción del mismo de la Palm y de la PC.

	Archivar	Archivar, Cambio	Archivar, Sin Cambio
Borrar			
Sin Cambio	Archivar, Remueve DP		
Sin Registro			
Cambio		Conflicto	P → D

TABLA 5.4. Acciones de sincronización de espejo con la petición de archivado de la PC.

Un conduit de imagen de espejo debe de soportar también las sincronizaciones HHaPC y PCaHH. Recordando que este tipo de sincronizaciones hacen que los datos de la Palm sobrescriban los de la PC y viceversa. En este punto parecería que el conduit podría ser llamado para borrar ciegamente los datos del usuario, tanto en la PC como en la Palm. Pero hay que tener en cuenta que nuestro conduit es casi siempre llamado como resultado de la intervención del usuario.

Por ejemplo, un usuario accidentalmente borra una categoría entera de registros de datos en la Palm. En vez de sincronizar estos cambios y por consecuencia borrar los datos de la PC también, el usuario puede direccionar al conduit para sobrescribir todos los datos de la Palm y así restaurar efectivamente los registros borrados.

5.2 CATEGORÍAS

Un conduit debe sincronizar los cambios en los datos de categorías como en los datos de registros. Esto es porque una categoría es realmente un compuesto de datos y tiene tanto nombre como ID numérica, la lógica de sincronización es algo pesada. La Palm registra la lógica por default para los conduits nativos en el KDC. Estos se mencionan aquí:

1. Si la ID de categoría ha sido cambiada ya sea en la PC o en la Palm, pero el nombre es el mismo, actualiza todos los registros de la PC para usar la ID de categoría de la Palm.

2. Si el nombre de la categoría en la PC ha sido cambiado, pero la ID de la categoría es la misma, entonces actualiza el nombre de la categoría en la Palm. Nótese que esto permanece verdadero si el nuevo nombre de categoría en la PC no esta ya en uso en la Palm.

3. Si existe una categoría en la PC con un nuevo nombre e ID, y este no esta siendo usado en la Palm, entonces crea una nueva categoría en el dispositivo. Si el índice ya esta en uso, entonces asigna un nuevo índice desde la Palm y actualiza todos los registros con él.

Si la aplicación soporta las categorías hay que tener una cuidadosa consideración hacia el mapeo entre los nombres e identificadores de categorías en la Palm y en la PC. Y hay que estar atentos de que sólo hay soporte nativo en le sistema operativo de la Palm para sólo quince categorías activas.

5.3 DISEÑO DE CONDUITS

Hay importantes principios de diseño para un conduit. El primero y más importante, es que un conduit debe correr tan rápido como sea posible. Nadie quiere esperar mientras la sincronización trabaja. Haciendo honor a las banderas de sincronización rápida y lenta nos ayudará a optimizar el desempeño. Un conduit rápido también minimiza el uso del puerto serial de la Palm. Esto es importante, porque el uso de este puerto puede acabar con las baterías de la Palm muy rápidamente.

El segundo principio es el que un conduit debe siempre mover los datos de las aplicaciones en un modo natural entre la Palm y la PC. Nuestro usuario confía completamente en este comportamiento. El también confía en el conduit para restaurar todos los datos en caso de desastre. Si nuestro conduit no está correctamente implementado o tiene errores la aceptación del usuario en nuestra aplicación caerá bastante.

Un tercer principio es de que nuestro conduit debe ser capaz de correr sin la atención del usuario. Mientras que este es un buen principio de diseño para el software en general, es especialmente importante para un conduit. Esto es porque el usuario podría esta sincronizando desde una locación remota, comunicándose con el administrador de HotSync a través de un modem o una conexión de red. El usuario entonces no será capaz de responder algún comando o ventana de diálogo que el conduit podría mostrar en la PC. Esto causará que la sesión completa de HotSync se interrumpa o se acabe.

Si uno está construyendo un conduit de imagen de espejo entonces sigue la lógica mostrada anteriormente en las Tablas 5.2, 5.3 y 5.4 como vayan siendo requeridas en nuestra aplicación. Si nosotros no permitimos la edición de datos en la PC en nuestra aplicación entonces el conduit no debe soportarlo también.

El tener una interfaz de usuario y permitir que el mismo modifique el comportamiento natural del conduit, ayudará a lograr aceptación. Si esto es posible hay que permitir la carga o descarga de todos los registros de nuestra interfaz de usuario.

5.3.1 Cuando no usar un conduit

No todas las situaciones de sincronización requieren la implementación de un conduit. Es perfectamente apropiado el hacer una nueva base de datos desde el bosquejo y cargarla en el dispositivo si uno sabe que todo o la mayoría de los datos de nuestra aplicación han sido cambiados.

Simplemente hay que generar nuestra base de datos a partir de los nuevos datos y ponerlos en el directorio de instalación del HotSync. La base de datos será automáticamente transferida al dispositivo con la siguiente sincronización, sobrescribiendo cualquier dato existente en la Palm durante el proceso. Si nuestra aplicación tiene una base de datos que es de sólo lectura y la cual es periódicamente actualizada uno puede usar esta técnica. Si nuestro departamento de ventas produce una lista de precios la cual cambia mensualmente, entonces uno no necesita ejecutar el conduit a diario.

Si uno tiene el software para PC de la Palm instalado en la máquina de desarrollo como es requerido para correr el AppForge hay algunos puntos de los cuales deberíamos de estar al pendiente cuando instalamos el KDC. Por default nuestro sistema utiliza el administrador de HotSync que fue instalado con el software para PC de la Palm. Desafortunadamente los objetos ActiveX necesarios para desarrollar y depurar conduits no son instalados con versiones anteriores del software de PC para la Palm.

Si tenemos el privilegio de desarrollar nuestras aplicaciones y el software en computadoras por separado el siguiente paso es innecesario. De otro modo uno tendrá que modificar la configuración de la PC para que el nuevo manejador de HotSync corra durante el proceso de sincronización. Hay que utilizar la herramienta de edición del registro de Windows (RegEdit.exe) para alterar las dos llaves de registro como se muestran en el Ejemplo 1. Notar que la locación del manejador de HotSync actualizado depende de en donde instalamos el KDC en nuestro sistema.

Ejemplo 1: Parámetros de registro para el manejador de HotSync.

```
HKEY_CURRENT_USER\Software\U.S. Robotics\Pilot Desktop\Core
        HotSyncPath = C:\CDK401\Common\Bin\C4.01\HotSync.exe
        Path = C:\CDK401\Common\Bin\C4.01
```

5.3.2 Instalacion del kit de desarrollo de conduits.

El KDC es libremente descargable del sitio Web de Palm. Este kit contiene librerías y ligas de ejecución para Visual Basic y COM así como también para C y C++. El asistente de instalación pone todo en el directorio que uno seleccione.

Uno puede escoger instalar solamente las librerías COM, los ejemplos y la documentación utilizando la instalación personalizada. Se recomienda hacer una instalación completa, ya que el paquete contiene alguna documentación adicional que puede ser útil.

El directorio Com contiene el tutorial de Visual Basic y los proyectos de ejemplo, una herramienta de instalación y otro directorio que simplemente contiene enlaces a los documentos arriba mencionados. La mayoría de los proyectos de muestra son básicos que se enfocan en explicar una o dos partes del KDC. Hay también un ejemplo mas extenso SyncSamp, el cual reemplaza el conduit por default Memo e ilustra un conduit completo.

El directorio Common contiene los programas y librerías utilizados para desarrollar conduits. Este fólder también contiene la documentación tanto en formato PDF como en formato de Ayuda de Microsoft. Versiones anteriores del KDC son incluidas para compatibilidad atrasada; esto no nos ayudará, ya que sólo las más recientes versiones soportan el desarrollo en Visual Basic.

Debido a que cada conduit maneja exactamente una aplicación en la Palm uno debe introducir la ID del Creador única para la aplicación. (Debe haber solamente un conduit registrado para cada aplicación de la Palm, CondCfg.exe no aceptará registrar un segundo conduit para la misma aplicación o más precisamente una aplicación con la misma ID del Creador).

NOTA: Si no hay alguna aplicación en la Palm con esta ID, entonces el conduit no correrá. Uno debe instalar la aplicación primero.

Para nuestro ejemplo de conduit introducimos ***garl*** como la ID del Creador. Además introducimos ***garlDB*** en el campo de base de datos remota. Como veremos más adelante el administrador de HotSync provee al conduit con una lista más exacta de las bases de datos durante una sesión real de sincronización.

En la ventana de información anexa en la parte inferior de la ventana principal se selecciona la opción COM Conduit. Aquí se introduce la ruta completa al IDE de Visual Basic en el campo de cliente COM.

Los otros campos se dejan en blanco o con los valores por default. Estos ajustes son mostrados en la figura 5.2. Se presiona el botón de OK para registrar los nuevos valores y reiniciar el administrador de HotSync. Después se sale de la herramienta de configuración.

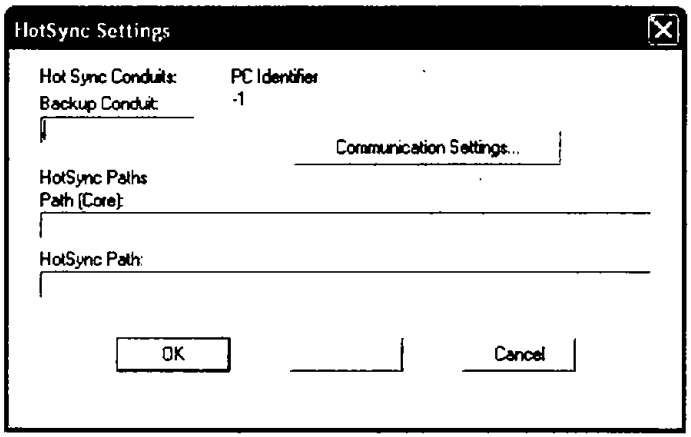

Figura 5.2. Configuración para la creación de un conduit.

Ahora se coloca la Palm en su base (cradle) y se presiona el botón de HotSync. Aunque el proceso de sincronización corre normalmente el IDE de VB no está activado. Esto es debido a que el administrador de HotSync no pudo encontrar una aplicación dentro de la Palm con la ID del Creador ***garl***.

Uno necesitará cargar la aplicación muestra e instalarla. Después de esto se debe realizar el HotSync una vez más. Esta vez el administrador de HotSync en la PC mostrará el mensaje Status: Synchronizing COMConduit y el IDE de VB se activará.

Ahora uno puede probar el ambiente abriendo el archivo SimpleDB.vbp que se encuentra en el directorio Tutorial. Este proyecto de VB abre la aplicación nativa Memo y lee todos los registros que concuerden con la búsqueda. Se introduce un punto de quiebre en la acción de click en botón y se presiona F8 para dar paso al programa. Notar que la Palm despliega Synchronizing Memo Pad, ya que la base de datos Memo ha sido abierta durante la sesión de HotSync. Ahora nos estamos comunicando como un conduit con la Palm desde VB.

Si el IDE de VB no aparece se debe poner al administrador de HotSync en modo de comandos. Primero se detiene al administrador de HotSync dando click con el botón derecho del mouse en el icono de la barra de tareas y seleccionando Exit. Lo siguiente es reiniciar el administrador de HotSync con la opción –v en la línea de comandos. Después de la sincronización uno puede revisar el registro de errores. Para hacer esto hay que darle click con el botón derecho del mouse en el icono y seleccionar View Log.

Si el proyecto no se conecta a la Palm o muestra otros errores, checar que las referencias del conduit están puestas correctamente. Esto se hace desde el IDE de VB escogiendo la opción de References en el menú de Project. Si estás referencias no se encuentran hay que registrarlas manualmente y reiniciar VB.

CAPITULO VI
CREACIÓN DE UN CONDUIT BÁSICO DE VENTAS

A partir de este punto se creará y se mostrará cómo usar un conduit real. Este conduit será utilizado por una aplicación de Ventas la cual está creada usando Metrowerks CodeWarrior. Este ejemplo demostrará que está envuelto en la creación de un conduit básico y que es lo que requiere para llegar al punto donde la sincronización está lista para comenzar.

6.1 VISIÓN GENERAL DE LOS CONDUITS.

Un conduit puede ser simple o complejo, dependiendo del trabajo que tiene que hacer. Independientemente de su complejidad uno lo crea de la misma forma: un conduit es un aditamento de PC hecho en un ambiente de desarrollo de PC.

Un conduit no es código que se ejecuta en una Palm, pero es una librería que se ejecuta durante la sincronización.

6.1.1 Desarrollo de conduits en windows.

Cuando el usuario está listo para desarrollar un conduit, utiliza el Kit de Desarrollo de Conduits (CDK) de Palm, el cual contiene una variedad de herramientas útiles y permite la elección de varios lenguajes para el desarrollo de conduits.

6.1.1.1 Uso de C++

El Kit de Desarrollo de Conduits versión 4.03 viene con una Suite de sincronización en C++ que requiere Visual C++ 6.0 o posterior sobre Windows 95 o posterior. Esta Suite puede ser utilizada también con Metrowerks CodeWarrior el cual se utilizará para crear el conduit de Ventas.

Cuando se instala el CDK, el usuario obtiene un Asistente para Conduits (Conduit Wizard) como un aditamento. Este asistente es una herramienta excelente la cual permite la creación de un nuevo conduit especificando una variedad de opciones sobre el tipo de conduit que se quiere. Este asistente entonces genera un proyecto junto con el código fuente preliminar el cual se usará como punto de partida.

6.1.2 Elementos requeridos en un conduit básico.

Este conduit contendrá algunos elementos esenciales que se explicarán a continuación:

6.1.2.1 Un mecanismo de registro

Un conduit no puede ser ejecutado a menos que ya haya sido registrado.

6.1.2.2 Seis puntos de entrada en C

Estos regresan el nombre del conduit, su número de versión, alguna otra información adicional sobre el mismo; dos son utilizados para configurarlo y el último sirve como una entrada hacia el proceso de sincronización.

6.1.2.3 Mensajes de registro

Se necesita proveer de mensajes de registro al usuario. Dentro de algunas cosas se debe mencionar si la sincronización fue exitosa o no.

6.1.3 Información de entrada necesaria para registrar un conduit

6.1.3.1 Entradas requeridas

Estas entradas son requeridas para registrar un conduit:

Conduit

El nombre de la Librería Dinámica (DLL) del conduit. Si esta entrada no incluye el directorio, el nombre debe ser encontrado en el directorio del HotSync o en la ruta actual; de otra forma debería incluir la ruta completa hacia la librería.

Creador (creator)

Es la ID del creador que consta de cuatro caracteres que identifica la(s) base(s) de datos de las cuales el conduit es responsable. El conduit será llamado durante una sincronización sólo si una aplicación con esta ID existe en la Palm.

Directorio (directory)

Esta cadena especifica el nombre del directorio donde están almacenados los conduits. Cada usuario tiene un subdirectorio dentro del directorio del HotSync. Dentro de cada uno de los directorios de los usuarios cada conduit tiene su propio directorio donde almacena sus archivos.

6.1.3.2 Entradas opcionales

Las entradas opcionales son más numerosas. Estas incluyen las siguientes:

Archivo (file)

Es una cadena que especifica un archivo (si esta no incluye un directorio, se asume que este se encuentra en el directorio de conduits). Esto tiene como fin que sea el archivo local con el cual el conduit sincronizará a la Palm. Sin embargo, el conduit no esta restringido a usar un solo archivo (algunos conduits pueden llegar a necesitar leer o escribir varios archivos en la PC).

Información

Esta es una cadena que provee información sobre el conduit. Puede ser utilizada para resolver conflictos. Si más de un conduit quiere manejar la misma ID de creador, una herramienta podría mostrar esta cadena y preguntar al usuario que conduit debería ser usado para sincronizar.

Nombre

Es el nombre del conduit visible al usuario. Si esta cadena se encuentra en el conduit puede ser usada en vez de llamar al punto de entrada GetConduitName.

Prioridad

Es un valor entre cero y cuatro que controla el orden relativo en que el conduit se ejecutará. Los conduits registrados con una prioridad más baja se ejecutarán antes que los conduits registrados con prioridades más altas. Si este valor no es fijado el Administrador de Sincronización utiliza un valor por omisión de 2 para la aplicación. Es difícil pensar en un caso en el cual el orden importe y será difícil encontrar un conduit que especifique algo diferente de 2.

DB remota

Es una cadena que especifica el nombre de una base de datos en la Palm. Es provista para ser usada en el conduit cuando este se ejecute; el conduit no está obligado a usarlo, sin embargo.

Nombre de Usuario

Es el nombre del usuario al cual se le ha instalado el conduit.

6.1.4 Puntos de entrada de un conduit

6.1.4.1 Puntos de entrada requeridos

long GetConduitName (char* tszName, WORD nLen)

Esta función regresa el nombre del conduit en pszName. El segundo parámetro es la longitud del buffer. Un conduit debería truncar su nombre, si fuese necesario, antes que dejar pasar el fin del buffer.

Como los otros puntos de entrada, el valor de retorno es 0 cuando no hay error o no-cero cuando lo hay.

DWORD GetConduitVersion ( )

Esta función regresa la versión del conduit como un valor de 4 bytes. La versión menor esta en el byte inferior. La versión mayor esta en el siguiente byte. Los otros dos bytes no son utilizados. Un conduit con versión 1.5 retornaría 0x15.

long OpenConduit(PROGRESSFN progress, CsyncProperties &sync)

Es a partir de este punto de entrada que el conduit realmente hace la sincronización. El primer parámetro, progress, es un apuntador a una función. Aunque la mayoría de los conduit no utilizan esta función el usuario puede mandarla llamar con una cadena de estado para mostrar su progreso dentro de la caja de diálogo del HotSync. El segundo parámetro es una estructura con información acerca de la sincronización. Los campos más importantes en esta estructura son mostrados en la tabla 6.1

Campo	Qué hace
m_SyncType	Computa los tipos de sincronización a ser realizados: • Copia de la Palm a la PC • Copia de la PC a la Palm • Sincronización rápida • Sincronización lenta • Nada hace
m_PathName	Contiene una cadena de la ruta completa al directorio de conduits dentro del directorio del usuario.
m_LocalName	Contiene el valor de la entrada de registro del Archivo.
m_UserName	Contiene el valor del usuario en la Palm.
m_RemoteName	Es un arreglo con los nombres de las bases de datos. Este arreglo está conformado con las bases de datos cuyo creador es el mismo del conduit. Si no se encuentran dichas bases de datos, el arreglo solo tiene una entrada copiada desde la entrada de registro RemoteDB.
m_RemoteDBList	Contiene la lista de todas las bases de datos en la Palm cuyo creador es el mismo del conduit.
m_nRemoteCount	Es un conteo de todas las bases de datos que hay en el arreglo m_RemoteName.

Tabla 6.1. Campos de Información sobre la Sincronización.

6.1.4.2 Puntos de entrada opcionales

long GetConduitInfo(ConduitInfoEnum infoType, void *pInArgs, void *pOut, DWORD *pdwOutSize)

Este punto de entrada es un camino amplio para regresar información desde el conduit, como una alternativa para seguir añadiendo más puntos de entrada. Este punto es llamado por el Administrador de Sincronización para que regrese el nombre del conduit. Sus parámetros son:

infoType

Puede ser eConduitName, eMfcVersion o eDefaultAction.

pInArgs

Utilizada para pasar información dentro de esta rutina. Su valor depende de infoType.

pOut

Utilizada para regresar información desde la rutina. Su valor depende de infoType.

pdwOutSize

Indica el tamaño del buffer pOut.

long ConfigureConduit(CsyncPreference& pref)

Esta rutina es llamada cuando el usuario quiere personalizar el conduit presionando el botón Change dentro de la ventana de diálogo de personalización del HotSync como se muestra en la figura 6.1. El conduit es responsable de mostrar una ventana de diálogo y guardar las opciones del usuario. El conduit responsable de una sincronización de tipo espejo despliega la ventana de diálogo que se muestra en la figura 6.2. El usuario escoge que acción debería pasar cuando ocurre una sincronización.

Conduits más específicos pueden tener distintas cosas que el usuario pueda configurar. En cualquier caso, la configuración del conduit debe permitir siempre al usuario hacer nada. De esta manera el usuario puede señalar y escoger que conduits activar.
Si este punto de entrada no está presente en el conduit al momento de presionar el botón change este hará nada.

Su parámetro es una estructura con los siguientes campos:

m_SyncType

Describe que tipo de operación ha sido escogida por el usuario: eFast, eSlow, eHHtoPC, ePCtoHH, eInstall, eBackup, eDoNothing o eProfileInstall.

m_SyncPref
Describe cuando esta operación aplica a la sincronización siguiente o a todas las sincronizaciones a partir de ahora.

long CfgConduit(ConduitCfgEnum cfgType, void *pArgs, DWORD *pdwArgsSize)
Este es un punto de entrada donde se configura el conduit .

6.2 EL REGISTRO DE HOTSYNC

El Kit de Desarrollo de Conduits provee varias rutinas para añadir al registro de HotSync. La principal es LogAddEntry.

Esta rutina se usa continuamente para añadir entradas al registro de HotSync. A continuación sus parámetros:

logstring
Es la cadena mostrada en el registro.

timestamp
De tipo Booleano, indica que la entrada de registro será fechada.

6.3 CÓDIGO DEL CONDUIT DE VENTAS

Como se había mencionado anteriormente, el Asistente para Conduits genera el código fuente del conduit, el cual incluye todo el código para los siguientes puntos de entrada: OpenConduit, GetConduitName, GetConduitVersion, ConfigureConduit, GetConduitInfo y CfgConduit. Más allá de eso, todo este código es casi funcional, con la excepción de OpenConduit. A continuación se muestra el código no útil que el Asistente genera para OpenConduit:

```
ExportFunc long OpenConduit(PROGRESSFN progress, CSyncProperties &sync)
{
   long retval = -1;

   if (pFn)
   {
        // TODO - create your own custom sync class, and run it
   }
   return (retval);
}
```

Desafortunadamente, si se usa de esta forma, la sincronización generará un error y mostrará el siguiente problema en el registro:

Conduit 'Ventas' Error: Unknown error. (FFFF)

Para evitar esto se necesitan hacer algunas cosas, incluyendo añadir el código necesario en OpenConduit para que este se ejecute correctamente. Se necesitan realizar los siguientes tres cambios:

1.- Añadir un archivo "include".

Primero se añade el archivo "include" el cual declara las API's para el registro de HotSync.

#include <HSLog.h>

2.- Modificar OpenConduit.

OpenConduit es pasado a una clase, CsyncProperties, que contiene información sobre la sincronización que se llevara a cabo. Analizando el campo m_SyncType de esta clase se puede observar que tipo de sincronización se realizará. El único tipo de sincronización que se puede manejar es eDoNothing (Hacer Nada). En este caso se debe escribir un mensaje apropiado para el registro y después regresar.

El siguiente código muestra todos los cambios realizados a OpenConduit:

```
ExportFunc long OpenConduit(PROGRESSFN progress, CSyncProperties &sync)
{
  long err =0;
  CONDHANDLE myConduitHandle =(CONDHANDLE)0;
  char conduitName [100 ];

  GetConduitName(conduitName,sizeof(conduitName));
  if (!progress)
    return -1;
  else if (sync.m_SyncType == eDoNothing) {
    LogAddFormattedEntry(slText,false,
      "%s - sync configured to Do Nothing",conduitName);
    return 0;
  } else if ((err = SyncRegisterConduit(myConduitHandle))==0) {
    LogAddEntry("",slSyncStarted, false);
    LogAddEntry(conduitName,err ? slSyncAborted : slSyncFinished,
      false);
    SyncUnRegisterConduit(myConduitHandle);
    return err;
  } else
    return err;
}
```

3.- Se actualizan las cadenas en el archivo de recursos.

Se encuentran un par de cadenas que deben de ser cambiadas en Sales.rc. a continuación se muestran los cambios en la tabla 6.2:

ID de Cadena	Se tiene	Se cambia
IDS_CONDUIT_NAME	TODO-Put conduit name	Ventas
IDS_SYNC_ACTION_TEXT	TODO-Put conduit action string here	HotSync action for Ventas

Tabla 6.2. Actualización de las cadenas en el archivo de recursos.

6.3.1 Registro del conduit

Ahora se tienen que realizar los cambios necesarios para que el Conduit trabaje, para esto hay que registrarlo primero:

1.- Se ejecuta el programa CondCfg.exe y se da un click en el botón Add, como se muestra en la figura 6.3

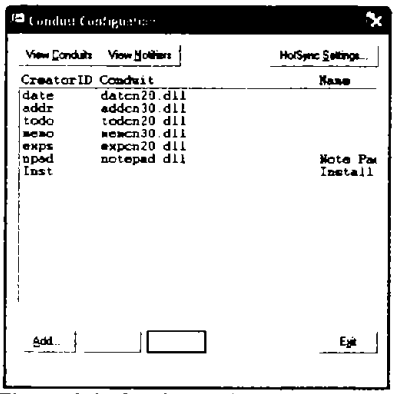

Figura 6.3. Configuración de Conduits.

2.- Se procede a llenar los campos de nombre del Conduit, ID del creador, directorio, archivo y base de datos remota; luego se procede a dar click en el botón OK como se muestra en la figura 6.4

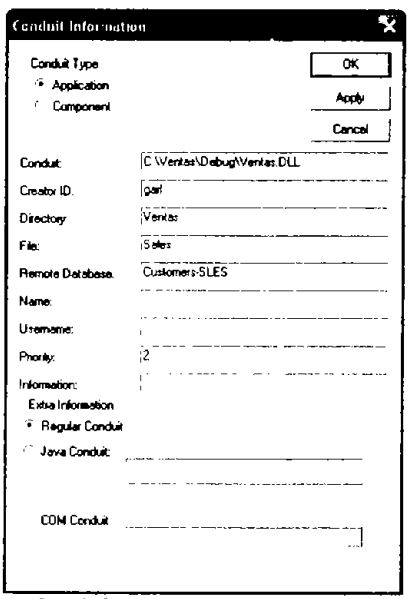

Figura 6.4. Información del Conduit de Ventas.

3.- A partir de este punto nuestro Conduit ya esta registrado y podemos trabajar con el como se muestra en la figura 6.5.

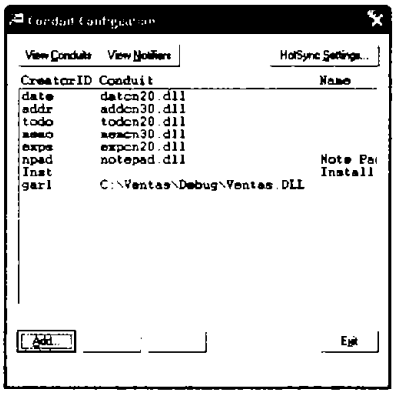

Figura 6.5. Conduit de Ventas Registrado.

6.3.2 Depuración de la fuente

Una vez que ha sido registrado el Conduit, se puede hacer depuración de su fuente. A continuación se explican los pasos necesarios:

1.- Cerrar la aplicación del HotSync (se escoge la opción Exit en el menú de HotSync que aparece en la bandeja del sistema de la barra de tareas).

2.- Desde el CodeWarrior se utiliza la herramienta Debug.

3.- Cuando se pide por un archivo para depurar se introduce:
C:\Ventas\Debug\Ventas.DLL

4.- Ahora se coloca un punto de rompimiento en alguna parte (por ejemplo dentro de OpenConduit). Cuando la ejecución alcance esta línea se detendrá permitiendo la depuración.

6.4 TRANSFERENCIA DE DATOS DESDE Y HACIA LA PALM UTILIZANDO EL CONDUIT DE VENTAS.

Hasta este punto se ha creado un Conduit Básico de Ventas el cual tiene como función Hacer Nada. A partir de aquí se verán las funcionalidades requeridas dentro del conduit para que este pueda soportar la transferencia de datos desde y hacia la Palm, así como algunas otras características útiles.

Paso a paso se irá modificando el código que conforma al Conduit Básico de Ventas de manera que este vaya incorporando las funciones de transferencia de ordenes de ventas y clientes desde la Palm, así como la transferencia de productos y clientes hacia el dispositivo. Además del manejo de registros borrados en la base de datos de clientes.

6.4.1 Donde almacenar los datos

Existe una importante demarcación a recordar cuando se decide donde almacenar los datos en la PC. Datos específicos a un usuario en particular deben ser almacenados en una locación privada, mientras que los datos que estén compartidos entre varios usuarios deberán ser almacenados en una locación grupal.

Por ejemplo, dentro de la Aplicación de Ventas utilizada aquí, cada usuario tiene su propia lista de clientes, pero tiene que obtener la lista de productos desde una locación general. El primer grupo de datos es específico a un usuario en particular; el segundo tipo es general. Ambos deben ser almacenados en locaciones distintas.

Datos específicos

Se recomienda almacenar estos datos dentro del directorio del conduit del usuario, el cual se encuentra en el directorio de HotSync.

Datos generales

Se deben almacenar dentro del directorio de la aplicación en la PC.

Se debe tener en mente que no es necesario tener esto almacenado localmente. Así como pueden ser almacenados en un escritorio en particular, es igual como si fueran almacenados en un servidor o en un sitio web.

6.4.2 Creación, apertura y cierre de bases de datos

La administración de la base de datos es completamente maneja por el conduit durante la sincronización.

6.4.2.1 Creación de la base de datos

Existe una llamada estándar utilizada por el Administrador de Sincronización para crear una base de datos:

```
long SyncCreateDB(CdbCreateDB& rDbStats)
```

Como todas las rutinas del Administrador de Sincronización, SyncCreateDB regresa el mensaje SYNCERR_NONE si es que ningún error ocurre; de otra manera regresará a un código de error. Esta rutina crea un nuevo registro o base de datos dentro de la Palm y después la abre. Se tiene este mismo control sobre la creación de la base de datos desde adentro del conduit que se tiene en la Palm.

El parámetro rDbStats contiene los siguientes importantes campos:

m_FileHandle
: Campo de salida. En un retorno exitoso este contiene un comando para la base de datos creada con acceso de lectura o escritura.

m_Creator
: ID del creador de la base de datos. Típicamente debería corresponder a la ID del creador de la aplicación.

m_Flags
: Atributos de la base de datos. Se escoge uno de los siguientes: eRecord para una base de datos estándar o eResource para una base de datos de recurso.

m_Type
: Es el tipo de la base de datos en formato de cuatro bytes.

m_CardNo
: Es la tarjeta de memoria donde la base de datos está almacenada. Se utiliza 0 si es que el dispositivo Palm tiene más de una tarjeta de memoria escribible.

m_Name
: Es el nombre de la base de datos.

m_Version
: Es la versión de la base de datos.

m_dwReserved
: Esta reservada para un uso futuro. Debe estar puesta a 0.

6.4.2.2 Apertura de la base de datos

La llamada del Administrador de Sincronización para abrir una base de datos remota es:

```
long SyncOpenDB(char *pname, int nCardNum, Byte& rHandle, Byte openMode)
```

Los valores para los cuatro parámetros son:

pName
Es el nombre de la base de datos.

nCardNum
Es la tarjeta de memoria donde está almacenada la base de datos.

rHandle
Parámetro de salida. En un retorno exitoso contiene un comando para la base de datos abierta.

openMode
Se utiliza (eDbRead | eDbShowSecret) para leer todos los registros, incluyendo los que son privados. Se utiliza (eDbRead | eDbWrite | eDbShowSecret) para habilitar la escritura y/o borrado de registros.

Se necesita cerrar cualquier base de datos que se abre; y sólo puede ser abierta una a la vez. Si se trata de abrir una nueva base de datos sin cerrar la anterior dará como resultado un error.

6.4.2.3 Cierre de la base de datos

La llamada del Administrador de Sincronización para cerrar una base de datos remota es SyncCloseDB y solamente tiene un parámetro: el comando que se crea cuando una base de datos es abierta o creada:

```
long SyncCloseDB(Byte fHandle)
```

6.4.3 Transferencia de datos hacia la Palm

Esto es comúnmente hecho con bases de datos que están actualizadas exclusivamente en la PC y que son transferidas rutinariamente a los dispositivos Palm donde no son modificadas, esto también puede ocurrir cuando el usuario escoge "Desktop overwrites handheld" en la ventana de diálogo de Configuración del HotSync como se muestra en la figura 6.6.

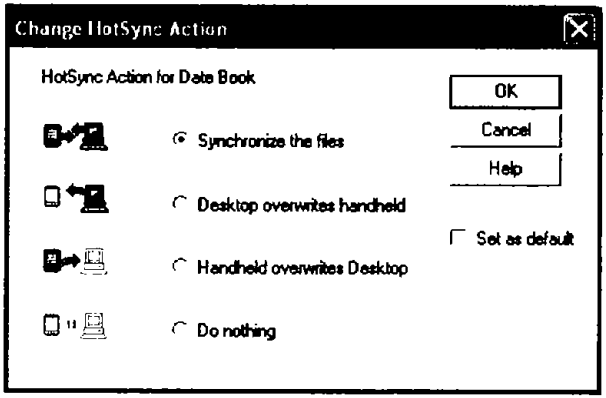

Figura 6.6. Elección de la Opción de Transferencia de Datos

Se necesita crear la base de datos si es que ésta no está ya creada. Se deben borrar los registros existentes antes de transferir los que están en la PC ya que no se desean los antiguos, sino los más recientes.

6.4.3.1 Borrado de registros existentes

Existen algunas rutinas distintas a escoger para remover registros de la Palm:

SyncDeleteRec
: Remueve un registro en específico.

SyncPurgeAllRecs
: Remueve todos los registros de la base de datos.

SyncPurgeAllRecsInCategory
: Remueve todos los registros de una categoría en específico.

SyncPurgeDeletedRecs
: Remueve todos los registros que han sido marcados como borrado o archivado.

Todos los anteriores remueven los registros completamente de la base de datos. En este caso en particular la llamada que se utilizará es SyncPurgeAllRecs.

6.4.3.2 Escritura de registros

Una vez que se tiene una limpia y vacía base de datos esta se puede llenar con registros frescos de la PC. Se pueden añadir o reemplazar registros con la llamada del Administrador de Sincronización SyncWriteRec:

```
long SyncWriteRec(CrawRecordInfo &rInfo)
```

El parámetro rInfo contiene bastantes campos importantes:

m_FileHandle
: Es un comando para la base de datos abierta.

m_RecId
: Campo de entrada/salida; esta es la ID del registro. Para añadir un nuevo registro se debe poner este campo a 0; de regreso el campo contiene la ID del nuevo registro. Para modificar un registro existente se debe poner en este campo la ID de uno de los registros en la base de datos. Si no concuerdan dichas ID's un error ocurrirá. Notar que cada vez que se añade un nuevo registro la Palm es la que asigna la ID única del mismo.

m_Attribs
: Son los atributos del registro.

m_CatId
: Es el índice de categorías del registro. Se utilizan valores del 0 al 14.

m_RecSize
: Es el número de bytes en el registro.

m_TotalBytes
: Es el número de bytes de datos en el buffer. Debe ser puesto al número de bytes en el registro para poder trabajar con los errores en algunas versiones del Administrador de Sincronización.

En la Palm, cuando el usuario crea un registro de base de datos se especifica la locación en la base de datos de ese registro. Cuando se utiliza un conduit no se tiene forma de especificar la locación exacta del registro. La rutina SyncWriteRec siempre añade nuevos registros al final de la base de datos. Además si se modifica un registro existente dicho registro es movido al final de la base de datos.

Una aplicación espera los registros de una manera ordenada. Si el conduit está escribiendo estos registros en una base de datos vacía, lo puede hacer de una manera ordenada. Si los está modificando estos no permanecerán en orden, en tal caso se utiliza una rutina que después que una sincronización ocurre ésta le dice a la aplicación que su base de datos ha sido cambiada y le da la oportunidad de ordenarla.

6.4.4 Transferencia de datos hacia la PC

Cuando se envían datos desde la Palm hacia la PC se tiene que leer a través de los registros de la base de datos remota y convertirlos en estructuras apropiadas en la PC. A continuación se explica el proceso paso a paso comenzando con las opciones que se tienen de la forma en que se leerán los registros.

6.4.4.1 Encontrando el número de registros

Con la rutina SyncGetDBRecordCount se encuentra el número de registros dentro de la base de datos:

```
long SyncGetDBRecordCount(BYTE fHandle, WORD &rNumRecs);
```

llamándola con:

```
WORD numRecords;
err = SyncGetDBRecordCount(rHandle, numRecords);
```

6.4.4.2 Lectura de registros

El usuario puede leer registros en una base de datos remota utilizando cualquiera de las siguientes estrategias:

- Iterando a través de cada registro, deteniéndose solamente en los que estén alterados
  Mediante el uso de la rutina SyncReadNextModifiedRec se puede iterar a través de los registros y detenerse solamente en aquellos que han sido modificados. Después los recupera de la base de datos remota si es que el bit de suciedad en los registros ha sido puesto.

  Una variación de esta rutina es SyncReadNextModifiedRecInCategory, la cual también filtra basada en la categoría de los registros. Esta función toma el índice de categorías como un parámetro adicional.

- Encontrando registros exactos vía la ID única del registro
  Algunas veces se desea leer registros basados en sus ID's únicas de registro. En tales casos se utiliza la rutina SyncReadRecordByID.

- Iterando a través de los registros de una base de datos desde el principio hasta el fin
  Utilizando la rutina SyncReadRecordByIndex se obtiene un registro basado en el número de registro. Esto se usa cuando se quiere leer una base de datos de principio a fin. Solamente tiene un parámetro, rInfo, el cual tiene el índice de registro como uno de sus campos.

6.4.4.3 Registros archivados y borrados

Para bases de datos que serán sincronizadas en ambos sentidos, la aplicación en la Palm no puede remover completamente un registro borrado solamente lo marca como tal . Cuando una sincronización ocurre, todos los registros marcados necesitan ser borrados de la base de datos en la PC.

Existe un par de formas donde el usuario puede borrar los registros marcados. Primero, hay que notar que se tienen dos diferentes formas para remover los registros de una base de datos: completamente borrarlos o solamente archivarlos. La figura 7.x muestra las dos posibles ventanas de diálogo que el usuario puede seleccionar cuando la opción para borrar un registro está dada. Escogiendo "Save archive copy on PC" significa que el registro es marcado como borrado hasta la siguiente sincronización, en este punto es guardado el registro en un archivo y después borrado de la base de datos. El no seleccionar la opción "Save archive copy on PC" significa que el registro es marcado como borrado hasta la siguientes sincronización y esta vez es completamente borrado de la base de datos.

6.4.4.3.1 Archivado de registros

Se deben crear por separado los ficheros de archivo y anexar los registros archivados ahí. Esto se utiliza en situaciones en las cuales el usuario no desea que los registros estén desordenando la aplicación en la Palm o en la PC, pero si desea que estos estén disponibles si es necesario. Es recomendable crear un fichero de archivos por separado para cada categoría.

6.4.4.3.2 Borrado de registros

Una vez que los registros que han sido marcados para archivado ya lo están y cualquier registro marcado para ser borrado ya ha sido removido de su archivo correspondiente en la PC, dichos registros deben ser completamente borrados de la Palm utilizando la rutina SyncPurgeDeletedRex, de la cual su sintaxis es:

```
err = SyncPurgeDeletedRex(rHandle);
```

6.5 EL CONDUIT DE VENTAS

Con toda esta información sobre el rol del conduit para mover datos desde y hacia la Palm, es momento de retomar el conduit de ventas. Se extenderá este conduit de manera que él mismo soporte las acciones de "PC sobrescribe a la Palm" y "Palm sobrescribe a la PC".

Para este conduit se tiene que definir qué significa cada tipo de sobrescritura. A continuación se muestra la lógica que se piensa da sentido a la aplicación de ventas:

PC sobrescribe a la Palm

Las bases de datos de productos y clientes son completamente sobrescritas desde la PC; nada pasa a su orden.

Palm sobrescribe a la PC

Los productos son ignorados ya que no pueden ser cambiados en la Palm. Las bases de datos de clientes y ordenes son copiadas a la PC. Cualquier cliente archivado es anexado a un fichero por separado; los clientes borrados son removidos de la Palm.

El resultado final de esto es que la base de datos de clientes es transferida tanto hacia como desde la Palm. La base de datos de productos es enviada a la Palm y la base de datos de las órdenes es enviada a la PC. La figura 7.x muestra como se mueve cada tipo de datos.

6.5.1 Formato utilizado para guardar datos en la PC

Se almacenan los datos en la PC como archivos de texto delimitados por tabuladores.

Los clientes serán almacenados en un archivo nombrado Clientes.txt dentro del directorio del usuario junto con el conduit de ventas. Cada línea en el archivo es de la forma:

```
Customer ID<tab>Name<tab>Address<tab>City<tab>Phone
```

Las órdenes serán almacenadas en un archivo llamado Ordenes.txt dentro del mismo directorio. Cada orden es almacenada así:

```
ORDER Customer ID
quantity<tab>Product ID
quantity<tab>Product ID
...
quantity<tab>Product ID
```

Las órdenes se siguen una tras otra dentro del archivo.
Los productos son almacenados en el archivo Productos.txt y comienza con las categorías seguida por los productos:

```
Name of Category 0
Name of Category 1
…
Name of last Category
<línea en blanco>
Product ID<tab>Name<tab>Category Number<tab>Price
…
Product ID<tab>Name<tab>Category Number<tab>Price
```

6.5.2 Modificación de la rutina OpenConduit

A continuación se muestran las modificaciones necesarias a la función OpenConduit para que pueda manejar la copia de datos desde la Palm hacia la PC (eHHtoPC) y desde la PC hacia la Palm (ePCtoHH).

```
ExportFunc long OpenConduit (PROGRESSFN progress, CSyncProperties &sync)
{
  long err = 0;
  CONDHANDLE myConduitHandle = (CONDHANDLE) 0;
  char conduitName[100];

  GetConduitName(conduitName, sizeof(conduitName));

  if (!progress)
    return -1;
  else if (sync.m_SyncType == eDoNothing) {
    LogAddFormattedEntry(slText, false,
       "%s - sync configured to Do Nothing", conduitName);
    return 0;
  } else if ((err = SyncRegisterConduit(myConduitHandle)) == 0) {
    LogAddEntry("", slSyncStarted, false);

    if (sync.m_SyncType == eHHtoPC) {
       if ((err = CopyOrdersFromHH(sync)) != 0
         goto exit;
       if ((err = CopyCustomersFromHH(sync)) != 0
         goto exit;
    } else if (sync.m_SyncType == ePCtoHH) {
      if ((err = CopyProdustsAndCategoriesToHH(sync)) != 0
        goto exit;
       if ((err = CopyCustomersToHH(sync)) != 0
        goto exit;
    } else if (sync.m_SyncType == eFast || sync.m_SyncType == eSlow) {
       LogAddEntry("Sales conduit doesn't (yet) support fast or slow. Choose
       "Desktop overwrites Handheld or Handheld overwrites Desktop" "from the
       custom menu", slWarning, false);
    }
exit:

       LogAddEntry(conduitName, err ? slSyncAborted : slSyncFinished,
           false);
       SyncUnRegisterConduit(myConduitHandle);
      return err;
    } else
       return err;
}
```

Con esta modificación se pueden copiar datos en una sola dirección, ya sea de la Palm hacia la PC o de la PC hacia la Palm. En este punto no es posible hacer una transferencia bidireccional al mismo tiempo.

6.5.3 Código general

Además del código que se acaba de añadir, se necesita definir las bases de datos, crear las variables globales y adicionar algunas estructuras de datos.

6.5.3.1 Definir bases de datos

Estas definiciones se pueden copiar directamente del código de ventas utilizado para la aplicación:

```
#define salesCreator        'SLES'
#define salesVersion        0
#define customerDBType      'DATA'
#define customerDBName      "Customers-SLES"
#define orderDBType         'Ordr'
#define orderDBName         "Orders-SLES"
#define productDBType       'Prod'
#define productDBName       "Products-SLES"
```

6.5.3.2 Globales

Se leen y escriben los registros mediante el uso de un buffer global. Se dimensiona tan grande como sea posible en el Palm OS:

```
#define  kMaxRecordSize    65535
char     gBigBuffer[kMaxRecordSize];
```

6.5.3.3 Estructuras de datos

Se tienen estructuras que necesitan corresponder exactamente a las que hay en la Palm. Después se pueden utilizar las mismas para leer y escribir datos en la Palm.

```
struct Item {
    unsigned long   productID;
    unsigned long   quantity;
};

struct PackedOrder {
    long            customerID;
    unsigned short  numItems;
    Item            items[1];
};

struct PackedCustomer{
    long customerID;
    char name[1];
};
```

```
struct PackedProduct {
    unsigned long   productID;
    unsigned long   price;  // in cents
    char    name[1];
};

#define kCategoryNameLength 15
typedef char    CategoryName[kCategoryNameLength + 1];

struct PackedCategories {
    unsigned short   numCategories;
    CategoryName     names[1];
};
```

Enseguida se añaden estructuras para almacenar datos en memoria. Para esto se utilizan los constructores y destructores y hacer la tarea más fácil:

```
struct Customer {
    Customer() { name = address = city = phone = 0;}
    ~Customer() {delete [] name; delete [] address; delete [] city;
        delete [] phone; };
    long customerID;
    char *name;
    char *address;
    char *city;
    char *phone;
};

struct Categories {
    Categories(int num) { numCategories = num;
        names = new CategoryName[num];}
    ~Categories() {delete [] names;Ç;
    unsigned short numCategories;
    CategoryName *names;
};

struct Order {
    Order(unsigned short num) { numItems = num;
       items = new Item[numItems];};
    ~Order() { delete [] items;};
    CategoryName *names;
};

struct Product {
    Product() {name = 0;};
    ~Product() {delete [] name;};

    unsigned long   productID;
    unsigned long   price;  // in cents
    unsigned char   category:4;
    char    *name;
};
```

6.5.4 Transferencia de datos hacia la Palm

Para transferir datos hacia la Palm se necesitan tener en cuenta los siguientes puntos:

1. Tener una copia de la lista de clientes en la Palm. Si la base de datos no existe, entonces se tendrá que crear.
2. Una vez que la base de datos está abierta, se necesitan leer los registros.
3. Se repite este procedimiento con los productos.

6.5.4.1 Transferencia de clientes hacia la Palm

Para realizar esta tarea se utiliza la función CopyCustomersToHH, como a continuación se ejemplifica:

```
int CopyCustomers ToHH(CSyncProperties &sync)
{
    FILE *fp = NULL;
    BYTE rHandle;
    int err;
    bool dbOpen = false;

    if ((err = SyncOpendDB(customerDBName, 0, rHandle, eDbWrite | eDbRead
      | eDbShowSecret)) != 0) {
        LogAddEntry("SyncOpenDB failed", slWarning, false);
        if (err == SYNCERR_FILE_NOT_FOUND)
        {
            CDbCreateDB dbInfo;
            memset(&dbInfo, 0, sizeof(dbInfo));
            dbInfo.m_Creator  = salesCreator;
            dbInfo.m_Flags    = eRecord;
            dbInfo.m_CardNo   = 0;
            dbInfo.m_Type     = customerDBType;
            strcpy(dbInfo.m_Name, customerDBName);

            if ((err = SyncCreateDB(dbInfo)) !=0)
            {
                LogAddEntry("SyncCreateDB failed", slWarning, false);
                goto exit;
            }
            rHandle = dbInfo.m_FileHandle;
        } else
            goto exit;
    }
    dbOpen = true;

    char    buffer[BIG_PATH *2];
    strcpy(buffer, sync.m_PathName);
    strcat(buffer, "Customers.txt");

    if ((fp = fopen(buffer, "r")) == NULL) {
        err = 1;
        LogAddFormattedEntry(slWarning, false, "fopen(%s) failed",
            buffer);
        goto exit;
    }

    Customer *c;
    while (c = ReadCustomer(fp)) {
        CRawRecordInfo recordInfo;
        recordInfo.m_FileHandle = rHandle;
        recordInfo.m_RecId = 0;
        recordInfo.m_pBytes = (unsigned char *) gBigBuffer;
        recordInfo.m_Attribs = 0;
```

```
        recordInfo.m_CatId = 0;
        recordInfo.m_RecSize = CustomerToRawRecord(gBigBuffer,
            sizeof(gBigBuffer), c);
        recordInfo.m_dwReserved = 0;

        if ((err = SyncWriteRec(recordInfo)) !=0 {
            delete c;
            LogAddEntry("SyncWriteRec failed", slWarning, false);
            goto exit;
        }
        delete c;
    }
exit:
    if (fp)
        fclose(fp);
    if (dbOpen)
        if ((err = SyncCloseDB(rHandle)) != 0)
            LogAddEntry("SyncDBClose failed", slWarning, false);
    return err;
}
```

Se trata de abrir la lista de clientes dentro de la Palm, si esta no existe se le crea utilizando la función SyncCreateDB. Enseguida se abre el archivo Clientes.txt, que contiene dicha lista de clientes; a continuación se borran todos los registros existentes de la base de datos en la Palm. Después se comienza a leer a cada cliente (utilizando ReadCustomer) y se guarda (utilizando SyncWriteRec) en la base de datos.

La función ReadCustomer lee al cliente desde un archivo de texto, retornando 0 si es que ya no hay más clientes; tal como se muestra a continuación:

```
Customer *ReadCustomer(FILE *fp)

{
    const char *separator = "\t";
    if (fgets(gBigBuffer, sizeof(gBigBuffer), fp) == NULL)
        return 0;
    char *customerID = strtok(gBigBuffer, separator);
    char *name = strtik(NULL, separator);
    char *address = strtok(NULL, separator);
    char *city = strtok(NULL, separator);
    char *phone = strtok(NULL, separator);

    if (!address)
        address = "";
    if (!city)
        city = "";
    if (!phone)
        phone = "";
    if (customerID && name) {
        Customer *c = new Customer;
        c->customerID = atol(customerID);
        c->name = new char[strlen(name) + 1];
        strcpy(c->name, name);
        c->address = new char[strlen(address) + 1];
        strcpy(c->address, address);
        c->city = new char[strlen(city) + 1];
        strcpy(c->city, city);
        c->phone = new char[strlen(phone) + 1];
```

```
        strcpy(c->phone, phone);
        return c;
    }else
        return 0;
}
```

La función CustomerToRawRecord convierte un registro que este en memoria en un registro con formato para la Palm; retorna el número de bytes que ha escrito.

```
int CustomerToRawRecord(void *buf, int bugLength, Customer *c)
{
    PackedCustomer *cp = (PackedCustomer *) buf;
    cp->customerID = SyncHostToHHDWord(c->customerID);
    char *s = cp->name;
    strcpy(s, c->name);
    s += strlen(s) + 1;
    strcpy(s, c->address);
    s += strlen(s) + 1;
    strcpy(s, c->city);
    s += strlen(s) + 1;
    strcpy(s, c->phone);
    s += strlen(s) + 1;
    return s - (char *) buf;
}
```

6.5.4.2 Transferencia de productos hacia la Palm

En esta parte el uso de la función CopyProductsAndCategoriesToHH actualiza la base de datos de los productos que se encuentra en la Palm desde el archivo Productos.txt que esta en la PC:

```
int CopyProductsAndCategoriesToHH(CSncProperties &sync)

{
    FILE *fp = NULL;
    BYTE rHandle;
    int err;
    bool dbOpen = false;

    char    buffer[BIG_PATH *2];
    strcpy(buffer, sync.m_PathName);
    strcat(buffer, "Products.txt");

    if ((fp = fopen(buffer, "r")) == NULL) {
        err = 1;
        LogAddFormattedEntry(slWarning, false, "fopen(%s) failed",
            buffer);
        goto exit;
    }

    if ((err= SyncOpenDB(productDBName, 0, rHandle,
        eDbWrite | eDbRead | eDbShowSecret)) != 0) {
        if (err == SYNCERR_FILE_NOT_FOUND)
        {
            CDbCreateDB dbInfo;
            memset(&dbInfo, 0, sizeof(dbInfo));
            dbInfo.m_Creator = salesCreator;
```

```
            dbInfo.m_Flags    = eRecord;
            dbInfo.m_CardNo   = 0;
            dbInfo.m_Type     = productDBType;
            strcpy(dbInfo.m_Name, productDBName);

            if ((err = SyncCreateDB(dbInfo)) != 0)
            {
                LogAddEntry("SyncCreateDB failed", slWarning, false);
                goto exit;
            }
            rHandle = dbInfo.m_FileHandle;
        } else
                goto exit;
        }
        dbOpen = true;

        if ((err = SyncPurgeAllRecs(rHandle)) != 0) {
            LogAddEntry("SyncPurgeAllRecs failed", slWarning, false);
            goto exit;
        }
        Categories *c;
        if (c = ReadCategories(fp)) {
            CDbGenInfo rInfo;

            rInfo.m_pBytes = (unsigned char *) gBigBuffer;
            rInfo.m_TotalBytes = CategoriesToRawRecord(gBigBuffer,
                sizeof(gBigBuffer), c);
            rInfo.m_BytesRead = rInfo.m_TotalBytes;
            rInfo.m_dwReserved = 0;
            if ((err = SyncWriteDBAppInfoBlock(rHandle, rInfo)) !=0) {
                delete c;
                LogAddEntry("SyncWriteDBAppInfoBlock failed", slWarning,
                  false);
                goto exit;
            }
            delete c;
        }
        Product *p;
        while (p = ReadProduct(fp)) {
            CRawRecordInfo recordInfo;
            recordInfo.m_FileHandle = rHandle;
            recordInfo.m_RecId = 0;
            recordInfo.m_pBytes = (unsigned char *) gBigBuffer;
            recordInfo.m_Attribs = 0;
            recordInfo.m_CatId = p->category;
            recordInfo.m_RecSize = ProductToRawRecord(gBigBuffer,
                sizeof(gBigBuffer), p);
            recordInfo.m_dwReserved = 0;

            if ((err = SyncWriteRec(recordInfo)) !=0) {
                delete p;
                LogAddEntry("SyncWriteRec failed", slWarning, false);
                got exit;
            }
        }

exit:

    if (fp)
        fclose(fp);

    if (dpOpen)
        if ((err = SynCloseDB(rHandle)) != 0)
    return err;
}
```

Para leer las categorías de los productos se utiliza la función ReadCategories la cual lee el archivo Productos.txt hasta que encuentra una línea en blanco; para realizar esto se utiliza el código siguiente:

```
#define kMaxCategories 15
Categories *ReadCategories(FILE *file)
{

	const char *separator = "\n";
	int numCategories = 0;
	Categories *c = new Categories(kMaxCategories);
	for (int i = 0; i < kMaxCategories ; i++) {
		if (fgets(gBigBuffer, sizeof(gBigBuffer), fp) == NULL)
			break;

		if (gBigBuffer[strlen(gBigBuffer) - 1] == '\n')
			gBigBuffer[strlen(gBigBuffer) - 1] = '\0';
		if (gBigBuffer[0] == '\0')
			break;

		strncpy(c->names[i], gBigBuffer, kCategoryNameLength);
		c->names[i][kCategoryNameLength] = '\0';
	}

	c->numCategories = i;
	return c;
}
```

La función ReadProducts lee los productos que se encuentran en el mismo archivo, para realizar esto se añaden las siguientes líneas de código:

```
Product *ReadProduct(FILE *fp)
{
	const char *separator = "\t";
	if (fgets(gBigBuffer, sizeof(gBigBuffer), fp) == NULL)
		return 0;


	char *productID = strtok(gBigBuffer, separator);
	char *name = strtok(NULL, separator);
	char *categoryNumber = strtok(NULL, separator);
	char *price = strtok(NULL, separator);

	if (productID && name && categoryNumber) {
		Product *p = new Product;
		p->productID = atol(productID);
		p->name = new char[strlen(name) + 1];
		strcpy(p->name, name);
		p->category = (unsigned char) atoi(categoruNumber);
		p->price = (long) (atof(price) * 100); // Convertir a cents.
		return p;
	} else
		return 0;
}
```

La función CategoriesToRawRecord, mostrada a continuación, escribe las categorías en el formato que la Palm necesita:

```
int CategoriesToRawRecord(void *buf, int bufLength, Categories *c)
{
	PackedCategories *pc = (PackedCategories *) buf;
	pc->numCategories = SyncHostToHHWord(c->numCategories);
	char *s = (char *) pc->names;
	for (int i = 0; i < c->numCategories; i++) {
		memcpy(s, c->names[i], sizeof(CategoryName));
		s += sizeof(CategoryName);
	}
	return s - (char *) buf;
}
```

La función ProductToRawRecord es similar, solo que antes hay que intercambiar la ID del producto con el precio; para realizar esto se tiene el siguiente código:

```
int ProductToRawRecord(void *buf, int buflength, Product *p)
{
	PackedProduct *pp = (PAckedProduct *) buf;
	pp->ProductID = SyncHostToHHWord(p->productID);
	pp->price = SyncHostToHHWord(p->price);
	strcpy(pp->name, p->name);
	return offsetof(PackedProduct, name) + strlen(pp->name) + 1;
}
```

Con esto se completa la parte del código necesario para que el conduit pueda transferir los datos hacia la Palm. Sin embargo, como el orden de la función SyncWriteRec no ha sido definido, los datos deberán se reordenados dentro de sus archivos. En este caso se ordenarán por su ID dentro de la función PilotMain que viene incluida en el código de la Aplicación de Ventas; a continuación se muestra el código necesario:

```
} else if (cmd == sysAppLaunchCmdSyncNotify) {
	DmOpenRef	db;

	db= DmOpenDatabaseByTypeCreator(customerDBType, salesCreator,
		dmModeReadWrite);
	if (db) {
		DmInsertionSort(db, DmComparF *) CompareIDFunc, 0);
		DmCloseDatabase(db);
	} else
		error = DmGetLastErr();
	db= DmOpenDatabaseByTypeCreator(productDBType, salesCreator,
		dmModeReadWrite);
	if (db) {
		DmInsertionSort(db, (DmComparF *) CompareIDFunc, 0);
		DmCloseDatabase(db);
	} else
		error = DmGetLastErr();

}
```

6.5.5 Transferencia de datos hacia la PC

Para realizar esta tarea se necesita seguir los mismos pasos que vieron anteriormente, pero en vez de transferir los datos a la Palm estos se moverán hacia la PC. A continuación se enlistan los puntos a seguir:

1. Se copian las ordenes de compra desde la Palm hacia la PC, abriendo la base de datos y leyendo los registros.
2. Realizando la conversión propia.
3. Enviando los datos en camino a la PC.
4. Repitiendo estos mismos pasos con los clientes.

6.5.5.1 Transferencia de ordenes hacia la PC

Para transferir dichas ordenes de compra desde la Palm hacia la PC solamente se tiene que agregar el siguiente código dentro del conduit:

```
int CopyOrdersFromHH(CSyncProperties &sync)
{
	FILE *fp = NULL;
	BYTE rHandle;
	int err;
	bool dbOPen = false;
	int i;

	if ((err = SyncOpenDB(orderDBName, 0, rHandle,
		eDbRead | eDbShowSecret )) != 0) {
		LogAddEntry("SyncOpenDB failed", slWarning, false);
		goto exit;
	}
	dbOPen = true;

	char   buffer[BIG_PATH *2];
	strcpy(buffer, sync.m_PathName);
	strcat(buffer, "Orders.txt");

	if ((fp = fopen(buffer, "w")) == NULL) {
		LogAddFormatedEntry(slWarning, false, "fopen(%s) failed",
			buffer);
		goto exit;
	}

	WORD recordCount;
	if ((err = SyncGetDBRecordCount(rHandle, recordCount)) !=0) {
		LogAddEntry("SyncGetDBRecordCount failed", slWarning, false);
		goto exit;
	}

	CRawRecordInfo recordInfo;
	recordInfo.m_FileHandle = rHandle;

	for (i = 0; i < recordCount; i++) {
		recordInfo.m_RecIndex = i;
		recordInfo.m_TotalBytes = (unsigned short) sizeof(gBigBuffer);
		recordInfo.m_pBytes = (unsigned char *) gBigBuffer;
		recordInfo.m_dwReserved = 0;
```

```
            if ((err = SyncReadRecordByIndex(recordInfo)) !=0) {
                  LogAddEntry("SyncReadRecordByIndex failed", slWarning,
false);
                  goto exit;
            }
            Order *o = RawRecordToOrder (recordInfo.m_pBytes);
            if ((err = WriteOrderToFile(fp, o)) != 0) {
                  LogAddEntry("WriteOrderToFile failed", slWarning, false);
                  delete o;
                  goto exit;
            }
            delete o;
      }
exit:
      if (fp)
            fclose(fp);

      if (dbOpen)
            if ((err = SyncCloseDB(rHandle)) != 0)
                  LogAddEntry("SyncDBClosefailed", slWarning, false);
      return err;
}
```

Este código abre la base de datos de las ordenes de compra, en modo de solo lectura, ya que no puede realizar algún cambio. Enseguida crea el archivo Ordenes.txt y encuentra el número de registro dentro de la base utilizando la función SyncGetDBRecordCount. Luego lee registro por registro usando SyncReadRecordByIndex. después con la función RawRecordToOrder convierte los datos de la Palm en formato para almacenados en memoria:

```
Order *RawRecordToOrder(void *p)
{
      PackedOrder *po = (PacketOrder *) p;
      unsigned short numItems = SyncHHToHostWord(po->numItems);
      Order *o = new Order(numItems);
      o->customerID = SyncHHToHostDWord(po->numItems);
      for (int i = 0; i < o->numItems; i++) {
        o->items[i].productID = SyncHHToHostDWord(po->items[i].productID);
        o->items[i].quantity = SyncHHToHostDWord(po->items[i].quantity);
      }
      return o;
}
```

Por último, las ordenes son guardadas en archivo mediante el uso de WriteOrderToFile:

```
int WriteOrderToFile(FILE *fp, const Order *o)
{
      int result;

      if ((result = fprintf(fp, "ORDER %ld\n", o->customerID)) < 0)
            return result;
      for (int i = 0; i < o->numItems; i++) {
            if ((result = fprintf(fp, %ld %ld\n", o->items[i].quantity,
                  o->items[i].productID)) < 0)
                  return result;
      }
      return 0;
}
```

6.5.5.2 Transferencia de clientes hacia la PC

El transferir clientes hacia la PC es un poco más complicado que con las ordenes de compra, ya que la Palm soporta el borrado y almacenado de clientes. Con el siguiente código se verá la manera de mover la base de datos desde la Palm hacia la PC:

```
int CopyCustomersFromHH(CSyncProperties &sync)
{
	FILE *fp = NULL;
	FILE *archivefp = NULL;
	BYTE rHandle;
	int err;
	bool dbOpen = false;
	int i;

	if ((err = SyncOpenDB(customerDBName, 0, rHandle,
		eDbWrite | eDbRead | eDbShowSecret)) != 0) {
		LogAddEntry("SyncOpenDB failed", slWarning, false);
		goto exit;

	}
	dbOpen = true;

	char	buffer[BIG_PATH *2];
	strcpy(buffer, sync.m_PathName);
	strcat(buffer, "Customers.txt");

	if ((fp = fopen(buffer, "w")) == NULL) {
		LogAddFormattedEntry(slWarning, false, "fopen(%s) failed",
			buffer);
		goto exit;

	}

	strcpy(buffer, sync.m_PathName);
	strcat(buffer, "CustomersArchive.txt");

	if ((archivefp = fopen(buffer, "a")) == NULL) {
		LogAddFormattedEntry(slWarning, false, "fopen(%s) failed",
			buffer);
		goto exit;

	}

	WORD recordCount;
	if ((err = SyncGetDBRecordCount(rHandle, recordCount)) !=0) {
		LogAddEntry("SyncGetDBRecordCount failed", slWarning, false);
		goto exit;

	}

	CRawRecordInfo recordInfo;
	recordInfo.m_FileHandle = rHandle;

	for (i = 0; i < recordCount; i++) {

		recordInfo.m_RecIndex = i;
		recordInfo.m_TotalBytes = (unsigned short) sizeof(gBigBuffer);
		recordInfo.m_pBytes = (unsigned char *) gBigBuffer;
		recordInfo.m_dwReserved = 0;
```

```
            if ((err = SyncReadRecordByIndex(recordInfo)) !=0) {

                  LogAddEntry("SyncReadRecordByIndex failed", slWarning,
                    false);
                  goto exit;
            }

            FILE *fileToWriteTo;
            if (recordInfo.m_Attribs & eRecAttrDeleted)
                  fileToWriteTo = archivefp;
            else if (recordInfo.m:_Attribs & eRecAttrDeleted)
                  continue;   // Saltar registros borrados.
            else
                  fileToWriteTo = fp;

            Customer *c = RawRecordToCustomer(recordInfo.m_pBytes);
            if ((err = WriteCustomerToFile(fileToWriteTo, c)) != 0) {
                  delete c;
                  LogAddEntry("WriteCustomerToFile failed", slWarning, false);
                  goto exit;
            }
            delete c;
      }

      if ((err = SyncPurgeDeletedRecs(rHandle)) != 0)
            LogAddEntry("SyncPurgeDeletedRecs failed", slWarning, false);

exit:
      if (fp)
            fclose(fp);

      if (archivefp)
            fclose(archivefp);

      if (dbOpen)
            if ((err = SyncCloseDB(rHandle)) != 0)
                  LogAddEntry("SyncDBClose failed", slWarning, false);
      return err;
}
```

Después de leer cada registro con la función SyncReadRecordByIndex, se examinan sus atributos (utilizando m_Attribs). Si el bit de archivo esta activo, entonces se escribe el registro en un archivo diferente (se pueden anexar a ArchivoClientes.txt):

```
      File *fileToWriteTo;

      if (recordInfo.m_Attribs & eRecAttrArchived)
        fileToWriteTo = archivefp;
```

Si el bit de borrado esta activo, entonces se salta este registro:

```
      else if (recordInfo.m_Attribs & eRecAttrDeleted) continue;
```

Una vez que se ha hecho la iteración a través de todos los registros se proceden a remover los archivados y borrados en la Palm usando la función SyncPurgeDeletedRecs. Para poder hacer esto se necesita abrir la base de datos con la función eDbWrite la cual permitirá la escritura.

Una vez hecho esto se necesita poner a los registros de clientes a donde pertenecen. Utilizando la siguiente rutina dichos clientes son agregados al archivo dado:

```
int WriteCustomerToFile(FILE *fp, const Customer *c)
{
    int result;

    if ((result = fprint(fp, "%d\t%s\t%s\t%s\t%s\n", c->customerID,
        c->name, c->address, c->city, c->phone)) < 0)
        return result;
}
```

Por último se muestra el código que convierte un registro en la Palm en un cliente:

```
Customer *RawRecordToCustomer(void *rec)
{
    Customer *c = new Customer;
    PackedCustomer *pc = (PackedCustomer *) rec;
    c->customerID = pc->customerID;
    char * p = (char *) pc->name;
    c->name = new char[strlen(p)+1];
    strcpy(c->name, p);
    p += strlen(p) + 1;
    c->address = new char[strlen(p)+1];
    strcpy(c->address, p);
    p += strlen(p)+1;
    c->city = new char[strlen(p)+1];
    strcpy(c->city, p);
    p += strlen(p) + 1;
    c->phone = new char[strlen(p)+1];
    strcpy(c->phone, p);
    return c;
}
```

Con todo este código, que se vio anteriormente, puesto en su lugar el Conduit de Ventas ahora ya es capaz de transferir datos desde y hacia la Palm cuantas veces se necesite.

CONCLUSIONES

Se puede concluir ahora que con todo lo visto anteriormente el lector tiene ahora las bases necesarias para poder desarrollar aplicaciones y sus respectivas conexiones que funcionen sobre los dispositivos Palm, sin tener que preocuparse de que no llegue a ser compatible con productos futuros, ya que la plataforma de programación seguirá el mismo camino con algunas modificaciones futuras, pero que no serán difíciles de adaptar y permitir así una migración de un sistema a otro sin complicación alguna.

Aunque dichos productos ofrezcan mayor espacio y velocidad de procesamiento, hay que seguir el camino que ha permitido este gran avance, utilizando la menor cantidad de líneas de código, ya que entre menos tiempo tarde y ocupe poco espacio será menor el consumo de energía y por ende mayor la duración de la batería.

En esta era donde la información es la piedra angular debemos tener un buen control sobre ella, ya que la perdida o alteración de la misma puede acarrear graves consecuencias. De ahí la importancia de siempre estar en sincronía con el resto de la sociedad. De igual forma ocurre con estos asistentes personales, los cuales deben ser siempre respaldados y actualizados en una unidad de almacenamiento permanente y es aquí donde cobra suma importancia la utilización de estos pequeños programas llamados "conduits" que llevan a cabo dichas tareas de transferencia de datos entre un dispositivo y otro.

Aún cuando se este llegando al punto donde ya no es tan necesario hacer grandes respaldos de toda la información contenida en una Palm se deben seguir utilizando estos conduits, porque nunca se sabe que podría pasar, desde un simple borrado involuntario hasta un corto circuito o un robo del dispositivo donde muchas veces lo que vale más es la información contenida que el mismo aparato.

Finalmente este trabajo puede ser usado como punto de partida para proyectos similares que cada vez alcancen límites más altos y que vayan enriqueciendo nuestra cultura informática; así poco a poco ir reduciendo nuestro rezago tecnológico en este campo con respecto de otras naciones.

Conforme el mundo se ha ido haciendo "más pequeño" gracias a todos estos avances tecnológicos y a la movilidad que tiene la gente, cada vez es más evidente que países se van quedando atrás y aunque en el nuestro se tiene la intención de avanzar, pero no los recursos suficientes, depende de nosotros hacer un mayor esfuerzo para que futuras generaciones sigan formándose y continúen enriqueciendo nuestra cultura, que en algún punto de la historia de la humanidad fue, y puede volver a ser, una de las más avanzadas.

Estamos en una época donde la vida no es fácil de comprender y de estudiar sin la utilización de una computadora. Cada nuevo descubrimiento o invento está íntimamente relacionado con la utilización de una computadora y cada día es imprescindible que este tipo de dispositivos sean más veloces, portátiles y con mayor capacidad de almacenamiento, ya que toda investigación que es realizada actualmente no se hace en un lugar fijo, sino que al contrario, cada día que pasa estas actividades se realizan en distintos lugares al mismo tiempo lo cual hace imposible estar desplazando equipo grande y pesado de un lugar a otro con la rapidez y facilidad que se desearía.

Además cabe mencionar que dichos dispositivos deben tener la capacidad de trasmitir la información a cortas y grandes distancias para así poder compartirla con otros dispositivos y lograr avances en conjunto con mayor rapidez. Es por eso que el desarrollo de estas nuevas tecnologías portables es de suma importancia para el desarrollo del conocimiento humano, tanto ahora como en el futuro mediato.

ANEXOS

CÓDIGO FUENTE DEL PROGRAMA DE VENTAS

A continuación se presenta de forma integra el código fuente del Programa de Ventas, el cual se utiliza como punto de partida para la creación del Conduit que transfiere la información de dicha aplicación desde la PC hacia la Palm y viceversa.

Código Fuente

Sales.c

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan
    neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)"
by O'Reilly.

Permission granted to use this file however you see fit.
*/
#define DO_NOT_ALLOW_ACCESS_TO_INTERNALS_OF_STRUCTS
#include <BuildDefines.h>
#ifdef DEBUG_BUILD
#define ERROR_CHECK_LEVEL ERROR_CHECK_FULL
#endif
#include <PalmOS.h>

#include "Utils.h"
#include "Exchange.h"
#include "Customers.h"
#include "Customer.h"
#include "Common.h"
#include "Item.h"
#include "Order.h"
#include "SalesRsc.h"
#include "Data.h"


#define kSalesPrefID     0x00
#define kSalesVersion    0x00
#define kCustomerDBType  'DATA'
#define kCustomerDBName  "Customers-SLES"
#define kOrderDBType     'Ordr'
#define kOrderDBName     "Orders-SLES"
#define kProductDBType   'Prod'
#define kProductDBName   "Products-SLES"



Order       *gCurrentOrder;
UInt16      gCurrentOrderIndex;
UInt16      gCurrentCategory = 0;
UInt16      gNumCategories;
privateRecordViewEnum     gPrivateRecordStatus;

typedef struct {
  Boolean     saveBackup;
} SalesPreferenceType;



void DeleteNthItem(UInt16 itemNumber)
{
  UInt16    newNumItems;
```

```
  ErrNonFatalDisplayIf(itemNumber >= gCurrentOrder->numItems,
    "bad itemNumber");

  // move items from itemNumber+1..numItems down 1 to
  // itemNumber .. numItems - 1
  if (itemNumber < gCurrentOrder->numItems - 1)
    DmWrite(gCurrentOrder,
      OffsetOf(Order, items[itemNumber]),
      &gCurrentOrder->items[itemNumber+1],
      (gCurrentOrder->numItems - itemNumber - 1) * sizeof(Item));

  // decrement numItems;
  newNumItems = gCurrentOrder->numItems - 1;
  DmWrite(gCurrentOrder,
    OffsetOf(Order, numItems), &newNumItems, sizeof(newNumItems));

  // resize the pointer smaller. We could use MemPtrRecoverHandle,
  // MemHandleUnlock, MemHandleResize, MemHandleLock.
  // However, MemPtrResize will always work
  // as long as your are making a chunk smaller.  Thanks, Bob!
  MemPtrResize(gCurrentOrder,
    OffsetOf(Order, items[gCurrentOrder->numItems]));
}

static
void InitializeCustomers(void)
{
  Customer  c1 = {1, "Joe's toys-1", "123 Main St." ,"Anytown",
    "(123) 456-7890"};
  Customer  c2 = {2, "Bucket of Toys-2", "" ,"", ""};
  Customer  c3 = {3, "Toys we be-3", "" ,"", ""};
  Customer  c4 = {4, "a", "" ,"", ""};
  Customer  c5 = {5, "b", "" ,"", ""};
  Customer  c6 = {6, "c", "" ,"", ""};
  Customer  c7 = {7, "d", "" ,"", ""};
  Customer *customers[7];
  UInt16  numCustomers = sizeof(customers) / sizeof(customers[0]);
  UInt16  i;

  customers[0] = &c1;
  customers[1] = &c2;
  customers[2] = &c3;
  customers[3] = &c4;
  customers[4] = &c5;
  customers[5] = &c6;
  customers[6] = &c7;
  for (i = 0; i < numCustomers; i++) {
    UInt16  index = dmMaxRecordIndex;
    MemHandle h = DmNewRecord(gCustomerDB, &index, 1);
    if (h) {
      PackCustomer(customers[i], h);
      DmReleaseRecord(gCustomerDB, index, true);
    }
  }
}

static
void InitializeProducts(void)
{
#define kMaxPerCategory 4
#define kNumCategories 3
  Product prod1 = {125, 253 ,"GI-Joe"};
  Product prod2 = {135, 1122 ,"Barbie"};
  Product prod3 = {145, 752 ,"Ken"};
  Product prod4 = {9,   852 ,"Skipper"};
```

```
Product prod5 = {126, 253 ,"Kite"};
Product prod6 = {127, 350 , "Silly-Putty"};
Product prod7 = {138, 650 ,"Yo-yo"};
Product prod8 = {199, 950 ,"Legos"};
Product prod9 = {120, 999 ,"Monopoly"};
Product prod10= {129, 888 , "Yahtzee"};
Product prod11= {10, 899 ,  "Life"};
Product prod12= {20, 1199 ,"Battleship"};
Product *products[kNumCategories][kMaxPerCategory];
UInt16  i;
UInt16  j;
MemHandle h;


products[0][0] = &prod1;
products[0][1] = &prod2;
products[0][2] = &prod3;
products[0][3] = &prod4;
products[1][0] = &prod5;
products[1][1] = &prod6;
products[1][2] = &prod7;
products[1][3] = &prod8;
products[2][0] = &prod9;
products[2][1] = &prod10;
products[2][2] = &prod11;
products[2][3] = &prod12;
for (i = 0; i < kNumCategories; i++) {
  for (j = 0; j < kMaxPerCategory && products[i][j]->name; j++) {
    UInt16      index;
    PackedProduct findRecord;
    MemHandle     h;

    findRecord.productID = products[i][j]->productID;
    index = DmFindSortPosition(gProductDB, &findRecord, 0,
      (DmComparF* ) CompareIDFunc, 0);
    h = DmNewRecord(gProductDB, &index, 1);
    if (h) {
      UInt16  attr;
      // Set the category of the new record to the category it
      // belongs in.
      DmRecordInfo(gProductDB, index, &attr, NULL, NULL);
      attr &= (UInt16) ~dmRecAttrCategoryMask;
      attr |= (UInt16) i;      // category is kept in low bits of attr

      DmSetRecordInfo(gProductDB, index, &attr, NULL);
      PackProduct(products[i][j], h);
      DmReleaseRecord(gProductDB, index, true);
    }
  }
}

h = DmNewHandle(gProductDB,
  OffsetOf(CategoriesStruct, names[kNumCategories]));
if (h) {
  char  *categories[] = {"Dolls", "Toys", "Games"};
  CategoriesStruct  *c = (CategoriesStruct *) MemHandleLock(h);
  LocalID       dbID;
  LocalID       appInfoID;
  UInt16        cardNo;
  UInt16        num = kNumCategories;
  Err         err;

  DmWrite(c, OffsetOf(CategoriesStruct, numCategories), &num,
    sizeof(num));
  for (i = 0; i < kNumCategories; i++)
    DmStrCopy(c,
```

```
        OffsetOf(CategoriesStruct, names[i]), categories[i]);
        MemHandleUnlock(h);
    appInfoID = MemHandleToLocalID( h);
    err = DmOpenDatabaseInfo(gProductDB, &dbID, NULL, NULL,
      &cardNo, NULL);
    if (err == errNone) {
      err = DmSetDatabaseInfo(cardNo, dbID, NULL, NULL, NULL, NULL,
        NULL, NULL, NULL, &appInfoID, NULL, NULL, NULL);
      ErrNonFatalDisplayIf(err, "DmSetDatabaseInfo failed");
    }
  }
}

static
void InitializeOrders(void)
{
  Item item1 =  {125, 253};
  Item item2 =  {145, 999};
  Item item3 =  {135, 888};
  Item item4 =  {9, 777};
  Item item5 =  {10, 6};
  Item item6 =  {20, 5};
  Item item7 =  {120, 5};
  Item item8 =  {126,  3};
  Item item9 =  {127,  45};
  Item item10 = {129, 66};
  Item item11 = {138, 75};
  Item item12 = {199, 23};
  Item items[12];
  MemHandle h;
  Order   *order;
  UInt16  recordNum;
  UInt16  numItems = sizeof(items) / sizeof(items[0]);

  items[0] =  item1;
  items[1] =  item2;
  items[2] =  item3;
  items[3] =  item4;
  items[4] =  item5;
  items[5] =  item6;
  items[6] =  item7;
  items[7] =  item8;
  items[8] =  item9;
  items[9] =  item10;
  items[10] = item11;
  items[11] = item12;

  order= GetOrCreateOrderForCustomer(1, &recordNum);

  // write numItems
  DmWrite(order, OffsetOf(Order, numItems), &numItems, sizeof(numItems));

  // resize to hold more items
  h = MemPtrRecoverHandle(order);
  MemHandleUnlock(h);
  MemHandleResize(h, OffsetOf(Order, items) + sizeof(Item) * numItems);

  order = (Order *) MemHandleLock(h);

  // write new items
  DmWrite(order, OffsetOf(Order, items), items, sizeof(items));

  // done with it
  MemHandleUnlock(h);
  DmReleaseRecord(gOrderDB, recordNum, true);
}
```

```
static
Err AppStart(void)
{
  SalesPreferenceType prefs;
  UInt16              prefsSize;
  UInt16              mode = dmModeReadWrite;
  Err                 err = errNone;
  CategoriesStruct    *c;
  Boolean             created;

  // Read the preferences / saved-state information.  There is only one
  // version of the preferences so don't worry about multiple
  // versions.
  prefsSize = sizeof(SalesPreferenceType);
  if (PrefGetAppPreferences(kSalesCreator, kSalesPrefID, &prefs,
    &prefsSize, true) == noPreferenceFound) {
    // POSE doesn't send sysAppLaunchCmdSyncNotify launch code so
    // we may not already be registered
    ExgRegisterData(kSalesCreator, exgRegExtensionID, "cst");
  } else {
     gSaveBackup = prefs.saveBackup;
  }

  // Determime if secret records should be shown.
  if (Has35FeatureSet())
    gPrivateRecordStatus =
      (privateRecordViewEnum) PrefGetPreference(prefShowPrivateRecords);
  else {
    if (PrefGetPreference(prefHidePrivateRecordsV33))
      gPrivateRecordStatus = hidePrivateRecords;
    else
      gPrivateRecordStatus = showPrivateRecords;
  }
  if (gPrivateRecordStatus != hidePrivateRecords)
    mode |= dmModeShowSecret;

  // Find the Customer database.  If it doesn't exist, create it.
  OpenOrCreateDB(&gCustomerDB, kCustomerDBType, kSalesCreator, mode,
    0, kCustomerDBName, &created);
  if (created)
    InitializeCustomers();

  // Find the Order database.  If it doesn't exist, create it.
  OpenOrCreateDB(&gOrderDB, kOrderDBType, kSalesCreator, mode,
    0, kOrderDBName, &created);
  if (created)
    InitializeOrders();

  // Find the Product database.  If it doesn't exist, create it.
  OpenOrCreateDB(&gProductDB, kProductDBType, kSalesCreator, mode,
    0, kProductDBName, &created);
  if (created)
    InitializeProducts();

  c = (CategoriesStruct *) GetLockedAppInfo(gProductDB);
  if (c) {
    gNumCategories = c->numCategories;
    MemPtrUnlock(c);
  }

  return err;
}


static
```

```
void AppStop(void)
{
  SalesPreferenceType prefs;

  // Write the preferences / saved-state information.
  prefs.saveBackup = gSaveBackup;

  PrefSetAppPreferences(kSalesCreator, kSalesPrefID, kSalesVersion, &prefs,
    sizeof(SalesPreferenceType), true);

  // Close all open forms,  this will force any unsaved data to
  // be written to the database.
  FrmCloseAllForms();

#ifdef DEBUG_BUILD
  CheckDatabases(false);
#endif
  // Close the databases.
  DmCloseDatabase(gCustomerDB);
  DmCloseDatabase(gOrderDB);
  DmCloseDatabase(gProductDB);
}

void SelectACategory(ListPtr list, UInt16 newCategory)
{
  UInt16    numItems;

  gCurrentCategory = newCategory;
  numItems = DmNumRecordsInCategory(gProductDB, gCurrentCategory) +
    (gNumCategories + 1);
  LstSetHeight(list, (Int16) numItems);
  LstSetListChoices(list, NULL, (Int16) numItems);
}


void DrawOneProductInList(Int16 itemNumber, RectanglePtr bounds,
  Char **text)
{
#pragma unused(text)
  void *p = NULL;
  FontID  curFont;
  Boolean setFont = false;
  const char  *toDraw = "";
  Int16   seekAmount = itemNumber;
  UInt16  index = 0;

  if (itemNumber == gCurrentCategory) {
    curFont = FntSetFont(boldFont);
    setFont = true;
  }
  if (itemNumber == gNumCategories)
    toDraw = "---";
  else if (itemNumber < gNumCategories) {
    CategoriesStruct *c = (CategoriesStruct *) GetLockedAppInfo(gProductDB);

    if (c) {
      toDraw = c->names[itemNumber];
      p = c;
    }
  } else {
    MemHandle h;
    DmSeekRecordInCategory(gProductDB, &index,
      seekAmount - (gNumCategories + 1), dmSeekForward, gCurrentCategory);
    h = DmQueryRecord(gProductDB, index);
    if (h) {
      PackedProduct *packedProduct = (PackedProduct *) MemHandleLock(h);
```

```
      Product s;
      UnpackProduct(&s, packedProduct);
      toDraw = s.name;
      p = packedProduct;
    }
  }
  DrawCharsToFitWidth(toDraw, bounds);
  if (p)
    MemPtrUnlock(p);
  if (setFont)
    FntSetFont(curFont);
}


// returns false if no product selected, true if one is selected
// (in which case productID and name are set)
Boolean HandleProductPopupSelect(EventPtr event, UInt32 *newProductID,
  Char *productName)
{
  ListPtr     list = event->data.popSelect.listP;
  ControlPtr  control = event->data.popSelect.controlP;

  ErrNonFatalDisplayIf(event->eType != popSelectEvent,
    "wrong kind of event");
  if (event->data.popSelect.selection < (gNumCategories + 1)) {
    if (event->data.popSelect.selection < gNumCategories)
      SelectACategory(list, (UInt16) event->data.popSelect.selection);
    LstSetSelection(list, (Int16) gCurrentCategory);
    CtlHitControl(control);
    return false;
  } else {
    UInt16        index = 0;
    MemHandle     h;
    PackedProduct   *packedProduct;
    Product     s;

    DmSeekRecordInCategory(gProductDB, &index,
      event->data.popSelect.selection - (gNumCategories + 1),
      dmSeekForward, gCurrentCategory);
    ErrNonFatalDisplayIf(DmGetLastErr(), "Can't seek to product");
    h = DmQueryRecord(gProductDB, index);

    ErrNonFatalDisplayIf(!h, "Can't get record");
    packedProduct = (PackedProduct *) MemHandleLock(h);
    UnpackProduct(&s, packedProduct);
    ErrNonFatalDisplayIf(StrLen(s.name) >= kMaxProductNameLength,
      "product name too long");
    StrNCopy(productName, s.name, kMaxProductNameLength);
    productName[kMaxProductNameLength-1] = '\0';  // just in case too long
    MemHandleUnlock(h);
    *newProductID = s.productID;
    return true;
  }
}

void HandleProductPopupEnter(UInt16 controlID, UInt16 listID,
  UInt32 productID)
{
  FormPtr form = FrmGetActiveForm();
  UInt16 listIndex = FrmGetObjectIndex(form, listID);
  ListPtr list = (ListPtr) FrmGetObjectPtr(form, listIndex);


  if (productID == 0) {
    SelectACategory(list, gCurrentCategory);
    LstSetSelection(list, (Int16) gCurrentCategory);
```

```
  } else {
    MemHandle h;
    Product   p;
    UInt16    attr;
    UInt16    index;
    Coord     dontCare;
    Coord     x;
    Coord     y;

    h = GetProductFromProductID(productID, &p, &index);
    ErrNonFatalDisplayIf(!h, "can't get product for existing item");
    DmRecordInfo(gProductDB, index, &attr, NULL, NULL);
    MemHandleUnlock(h);

    SelectACategory(list, attr & dmRecAttrCategoryMask);
    LstSetSelection(list,
      (Int16) (DmPositionInCategory(gProductDB, index, gCurrentCategory) +
      gNumCategories + 1));

    // make list top match trigger top
    FrmGetObjectPosition(form, listIndex, &x, &dontCare);
    FrmGetObjectPosition(form, FrmGetObjectIndex(form, controlID),
      &dontCare, &y);
    FrmSetObjectPosition(form, listIndex, x, y);
  }
}

static
void Search(FindParamsPtr findParams)
{
  Err           err;
  UInt16        pos;
  UInt16        fieldNum;
  UInt16        cardNo = 0;
  UInt16        recordNum;
  Char          *header;
  Boolean       done;
  MemHandle     recordH;
  MemHandle     headerH;
  LocalID       dbID;
  DmOpenRef     dbP;
  RectangleType r;
  DmSearchStateType searchState;

  // unless told otherwise, there are no more items to be found
  findParams->more = false;

  // Find the application's data file.
  err = DmGetNextDatabaseByTypeCreator(true, &searchState,
    kCustomerDBType, kSalesCreator, true, &cardNo, &dbID);
  if (err)
    return;

  // Open the expense database.
  dbP = DmOpenDatabase(cardNo, dbID, findParams->dbAccesMode);
  if (! dbP)
    return;

  // Display the heading line.
  headerH = DmGetResource(strRsc, FindHeaderString);
  header = (Char *) MemHandleLock(headerH);
  done = FindDrawHeader(findParams, header);
  MemHandleUnlock(headerH);
  if (done) {
    findParams->more = true;
  }
```

```
else {
    // Search all the fields; start from the last record searched.
  recordNum = findParams->recordNum;
  for(;;) {
    Boolean match = false;
    Customer    customer;

    // Because applications can take a long time to finish a find
    // users like to be able to stop the find.  Stop the find
    // if an event is pending. This stops if the user does
    // something with the device.  Because this call slows down
    // the search we perform it every so many records instead of
    // every record.  The response time should still be short
    // without introducing much extra work to the search.

    // Note that in the implementation below, if the next 16th
    // record is secret the check doesn't happen.  Generally
    // this shouldn't be a problem since if most of the records
    // are secret then the search won't take long anyways!
    if ((recordNum & 0x000f) == 0 &&      // every 16th record
      EvtSysEventAvail(true)) {
      // Stop the search process.
      findParams->more = true;
      break;
    }

    recordH = DmQueryNextInCategory(dbP, &recordNum,
      dmAllCategories);
    // Have we run out of records?
    if (! recordH)
      break;

    // Search each of the fields of the customer

    UnpackCustomer(&customer, (PackedCustomer *) MemHandleLock(recordH));

    if ((match = FindStrInStr(customer.name,
      findParams->strToFind, &pos)) != false)
      fieldNum = CustomerNameField;
    else if ((match = FindStrInStr(customer.address,
      findParams->strToFind, &pos)) != false)
      fieldNum = CustomerAddressField;
    else if ((match = FindStrInStr(customer.city,
      findParams->strToFind, &pos)) != false)
      fieldNum = CustomerCityField;
    else if ((match = FindStrInStr(customer.phone,
      findParams->strToFind, &pos)) != false)
      fieldNum = CustomerPhoneField;

    if (match) {
      done = FindSaveMatch(findParams, recordNum, pos, fieldNum, 0,
        cardNo, dbID);
      if (!done) {
        //Get the bounds of the region where we will draw the results.
        FindGetLineBounds(findParams, &r);

        // Display the title of the description.
        DrawCharsToFitWidth(customer.name, &r);

        findParams->lineNumber++;
      }
    }
    MemHandleUnlock(recordH);
    if (done)
      break;
    recordNum++;
```

```
    }
  }
  DmCloseDatabase(dbP);
}

static
void GoToItem (GoToParamsPtr goToParams, Boolean launchingApp)
{
  EventType   event;
  UInt16    recordNum = goToParams->recordNum;

   // If the current record is blank, then it will be deleted, so we'll
   // save the record's unique id to find the record index again, after all
   // the forms are closed.
  if (! launchingApp) {
    UInt32    uniqueID;

    DmRecordInfo(gCustomerDB, recordNum, NULL, &uniqueID, NULL);
    FrmCloseAllForms();
    DmFindRecordByID(gCustomerDB, uniqueID, &recordNum);
  }

  FrmGotoForm(CustomersForm);

  // Send an event to select the matching text.
  MemSet (&event, 0, sizeof(EventType));

  event.eType = frmGotoEvent;
  event.data.frmGoto.formID = CustomersForm;
  event.data.frmGoto.recordNum = goToParams->recordNum;
  event.data.frmGoto.matchPos = goToParams->matchPos;
  event.data.frmGoto.matchLen = (UInt16) goToParams->searchStrLen;
  event.data.frmGoto.matchFieldNum = goToParams->matchFieldNum;
  event.data.frmGoto.matchCustom = goToParams->matchCustom;
  EvtAddEventToQueue(&event);
}

static
Boolean AppHandleEvent(EventPtr event)
{
  FormPtr frm;
  UInt16    formId;
  Boolean handled = false;

  switch (event->eType) {
  case frmLoadEvent:
    // Load the form resource specified in  event then activate the form.
    formId = event->data.frmLoad.formID;
    frm = FrmInitForm(formId);
    FrmSetActiveForm(frm);

    // Set the event handler for the form.  The handler of the currently
    // active form is called by FrmDispatchEvent each time it receives
    // an event.
    switch (formId)
    {
    case OrderForm:
      FrmSetEventHandler(frm, OrderHandleEvent);
      break;

    case CustomersForm:
      FrmSetEventHandler(frm, CustomersHandleEvent);
      break;

    }
    handled = true;
```

```
    break;

  case menuEvent:
    if (event->data.menu.itemID == OptionsAboutSales) {
      FrmAlert(AboutBoxAlert);
      handled = true;
    }
    break;
  }
  return handled;
}


static
void AppEventLoop(void)
{
  EventType event;
  UInt16      error;

  do {
    EvtGetEvent(&event, evtWaitForever);
    if (! SysHandleEvent(&event))
      if (! MenuHandleEvent(0, &event, &error))
        if (! AppHandleEvent(&event))
          FrmDispatchEvent(&event);
  } while (event.eType != appStopEvent);
}

UInt32 PilotMain(UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
{
  Err error = errNone;

  error = RomVersionCompatible(sysMakeROMVersion(3, 0, 0, 0, 0),
    launchFlags);
  if (error)
    return error;

  if (cmd == sysAppLaunchCmdNormalLaunch)
  {
    error = AppStart();
    if (error == errNone)
    {
      FrmGotoForm(CustomersForm);
      AppEventLoop();

      AppStop();
    }
  }
  // Launch code sent to running app before sysAppLaunchCmdFind
  // or other action codes that will cause data searches or manipulation.
  else if (cmd == sysAppLaunchCmdSaveData) {
    FrmSaveAllForms();
  }
  else if (cmd == sysAppLaunchCmdFind) {
    Search((FindParamsPtr)cmdPBP);
  }
  // This launch code might be sent to the app when it's already running
  else if (cmd == sysAppLaunchCmdGoTo) {
    Boolean launched;
    launched = (Boolean) (launchFlags & sysAppLaunchFlagNewGlobals);

    if (launched) {
      error = AppStart();
      if (error == errNone) {
        GoToItem((GoToParamsPtr) cmdPBP, launched);
        AppEventLoop();
```

```
        AppStop();
      }
    } else {
      GoToItem((GoToParamsPtr) cmdPBP, launched);
    }
  } else if (cmd == sysAppLaunchCmdExgReceiveData) {
    DmOpenRef dbP;

    // if our app is not active, we need to open the database
    // The subcall flag is used to determine whether we are active
    if (launchFlags & sysAppLaunchFlagSubCall) {
      dbP = gCustomerDB;

      // save any data we may be editing.
      FrmSaveAllForms();

      error = ReceiveSentData(dbP, (ExgSocketPtr) cmdPBP);
    } else {
      dbP = DmOpenDatabaseByTypeCreator(kCustomerDBType, kSalesCreator,
        dmModeReadWrite);
      if (dbP) {
        error = ReceiveSentData(dbP, (ExgSocketPtr) cmdPBP);

        DmCloseDatabase(dbP);
      }
    }
  } else if (cmd == sysAppLaunchCmdSyncNotify) {
    DmOpenRef db;

    ExgRegisterData(kSalesCreator, exgRegExtensionID, "cst");

    // after a sync, we aren't guaranteed the order of any changed
    // databases.
    // We'll just resort the products and customer which could have changed.
    // we're going to do an insertion sort because the databases
    // should be almost completely sorted (and an insertion sort is
    // quicker on an almost-sorted database than a quicksort).
    // Since the current implementation of the Sync Manager creates new
    // records at the end of the database, our database are probably sorted.
    db= DmOpenDatabaseByTypeCreator(kCustomerDBType, kSalesCreator,
      dmModeReadWrite);
    if (db) {
      DmInsertionSort(db, (DmComparF *) CompareIDFunc, 0);

      DmCloseDatabase(db);
    } else
      error = DmGetLastErr();
    db= DmOpenDatabaseByTypeCreator(kProductDBType, kSalesCreator,
      dmModeReadWrite);
    if (db) {
      DmInsertionSort(db, (DmComparF *) CompareIDFunc, 0);
      DmCloseDatabase(db);
    } else
      error = DmGetLastErr();
  }

  return error;
}
```

Customer.c

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan
    neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)"
by O'Reilly.

Permission granted to use this file however you see fit.
*/
#define DO_NOT_ALLOW_ACCESS_TO_INTERNALS_OF_STRUCTS
#include <BuildDefines.h>
#ifdef DEBUG_BUILD
#define ERROR_CHECK_LEVEL ERROR_CHECK_FULL
#endif
#include <PalmOS.h>
#include "Customer.h"
#include "Utils.h"
#include "SalesRsc.h"
#include "Common.h"

Boolean     gSaveBackup = true;

Boolean AskDeleteCustomer(void)
{
  FormPtr form = FrmInitForm(DeleteCustomerForm);
  UInt16  hitButton;
  UInt16  ctlIndex;

  // Set the "save backup" checkbox to its previous setting.
  ctlIndex = FrmGetObjectIndex(form, DeleteCustomerSaveBackupCheckbox);
  FrmSetControlValue(form, ctlIndex, gSaveBackup);

  hitButton = FrmDoDialog(form);
  if (hitButton == DeleteCustomerOKButton)
    gSaveBackup = (Boolean) FrmGetControlValue(form, ctlIndex);
  FrmDeleteForm(form);

  return (Boolean) (hitButton == DeleteCustomerOKButton);
}

static
Boolean CustomerHandleEvent(EventPtr event)
{
  if (event->eType == ctlSelectEvent &&
    event->data.ctlSelect.controlID == CustomerDeleteButton) {
    if (!AskDeleteCustomer())
      return true;  // don't bail out if they cancel the delete dialog
  }
  return false;
}

// isNew is true if this is a brand-new customer
void EditCustomerWithSelection(UInt16 recordNumber, Boolean isNew,
  Boolean *deleted, EventPtr event)

{
  FormPtr    form;
  UInt16     hitButton;
  Boolean    dirty = false;
  Boolean    isEmpty = false;
  ControlPtr privateCheckbox;
  UInt16     attributes;
  Boolean    isSecret;
```

```
  FieldPtr    nameField;
  FieldPtr    addressField;
  FieldPtr    cityField;
  FieldPtr    phoneField;
  Customer    theCustomer;
  UInt16      offset = OffsetOf(PackedCustomer, name);
  MemHandle   customerHandle;

#ifdef DEBUG_BUILD
  CheckDatabases(true);
#endif
  customerHandle = DmGetRecord(gCustomerDB, recordNumber);
  *deleted = false;
  DmRecordInfo(gCustomerDB, recordNumber, &attributes, NULL, NULL);
  isSecret = (Boolean) ((attributes & dmRecAttrSecret) == dmRecAttrSecret);

  form = FrmInitForm(CustomerForm);
  FrmSetEventHandler(form, CustomerHandleEvent);

  UnpackCustomer(&theCustomer,
    (PackedCustomer *) MemHandleLock(customerHandle));

  nameField = (FieldPtr) GetObjectFromForm(form, CustomerNameField);
  addressField = (FieldPtr) GetObjectFromForm(form, CustomerAddressField);
  cityField = (FieldPtr) GetObjectFromForm(form, CustomerCityField);
  phoneField = (FieldPtr) GetObjectFromForm(form, CustomerPhoneField);

  SetFieldTextFromStr(nameField,    theCustomer.name, false);
  SetFieldTextFromStr(addressField, theCustomer.address, false);
  SetFieldTextFromStr(cityField,    theCustomer.city, false);
  SetFieldTextFromStr(phoneField,   theCustomer.phone, false);

  FrmDrawForm(form); // must do now, because pre-3.5, focus needed to be set
                     // *after* a call to FrmDrawForm
  // select one of the fields
  if (event && event->data.frmGoto.matchFieldNum) {
    FieldPtr selectedField =
      (FieldPtr)  GetObjectFromForm(form, event->data.frmGoto.matchFieldNum);
    FldSetScrollPosition(selectedField, event->data.frmGoto.matchPos);
    FrmSetFocus(form,
      FrmGetObjectIndex(form, event->data.frmGoto.matchFieldNum));
    FldSetSelection(selectedField, event->data.frmGoto.matchPos,
      event->data.frmGoto.matchPos + event->data.frmGoto.matchLen);
  } else {
    FrmSetFocus(form, FrmGetObjectIndex(form, CustomerNameField));
    FldSetSelection(nameField, 0, FldGetTextLength(nameField));
  }

  // unlock the customer
  MemHandleUnlock(customerHandle);

  privateCheckbox =
    (ControlPtr) GetObjectFromForm(form, CustomerPrivateCheckbox);
  CtlSetValue(privateCheckbox, isSecret);

  hitButton = FrmDoDialog(form);

  if (hitButton == CustomerOKButton) {
    dirty = FldDirty(nameField) || FldDirty(addressField) ||
      FldDirty(cityField) || FldDirty(phoneField);
    if (dirty) {
      theCustomer.name = FldGetTextPtr(nameField);
      if (!theCustomer.name)
        theCustomer.name = "";
      theCustomer.address = FldGetTextPtr(addressField);
      if (!theCustomer.address)
```

```
        theCustomer.address = "";
      theCustomer.city = FldGetTextPtr(cityField);
      if (!theCustomer.city)
        theCustomer.city = "";
      theCustomer.phone = FldGetTextPtr(phoneField);
      if (!theCustomer.phone)
        theCustomer.phone = "";
    }
    isEmpty = StrCompare(theCustomer.name, "") == 0 &&
      StrCompare(theCustomer.address, "") == 0 &&
      StrCompare(theCustomer.city, "") == 0 &&
      StrCompare(theCustomer.phone, "") == 0;
    if (dirty)
      PackCustomer(&theCustomer, customerHandle);
    if (CtlGetValue(privateCheckbox) != isSecret) {
      dirty = true;
      if (CtlGetValue(privateCheckbox)) {
        attributes |= dmRecAttrSecret;
        // tell user how to hide or mask private records
        if (gPrivateRecordStatus == showPrivateRecords)
          FrmAlert(privateRecordInfoAlert);
      } else
        attributes &= (UInt8) ~dmRecAttrSecret;
      DmSetRecordInfo(gCustomerDB, recordNumber, &attributes, NULL);
    }
  }
  FrmDeleteForm(form);

  DmReleaseRecord(gCustomerDB, recordNumber, dirty);
  if (hitButton == CustomerDeleteButton) {
    *deleted = true;
    if (isNew && !gSaveBackup)
      DmRemoveRecord(gCustomerDB, recordNumber);
    else {
      if (gSaveBackup)  // Need to archive it on PC
        DmArchiveRecord(gCustomerDB, recordNumber);
      else
        DmDeleteRecord(gCustomerDB, recordNumber);
      // Deleted records are stored at the end of the database
      DmMoveRecord(gCustomerDB, recordNumber, DmNumRecords(gCustomerDB));
    }
  }
  else if (hitButton == CustomerOKButton && isNew && isEmpty) {
    *deleted = true;
    DmRemoveRecord(gCustomerDB, recordNumber);
  }
  else if (hitButton == CustomerCancelButton && isNew) {
    *deleted = true;
    DmRemoveRecord(gCustomerDB, recordNumber);
  }
#ifdef DEBUG_BUILD
  CheckDatabases(true);
#endif
}

// returns true if the customer was changed/deleted
void EditCustomer(UInt16 recordNumber, Boolean isNew, Boolean *deleted)
{
  EditCustomerWithSelection(recordNumber, isNew, deleted, NULL);
}
```

Customers.c

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan
    neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)"
by O'Reilly.

Permission granted to use this file however you see fit.
*/
#define DO_NOT_ALLOW_ACCESS_TO_INTERNALS_OF_STRUCTS
#include <BuildDefines.h>
#ifdef DEBUG_BUILD
#define ERROR_CHECK_LEVEL ERROR_CHECK_FULL
#endif
#include <PalmOS.h>
#include <TxtGlue.h>
#include "Customers.h"
#include "Utils.h"
#include "Common.h"
#include "Customer.h"
#include "Exchange.h"
#include "SalesRsc.h"

static
Int32 GetCustomerIDForNthCustomer(UInt16 itemNumber)
{
  Int32          customerID;
  UInt16         index = 0;
  MemHandle      h;
  PackedCustomer *packedCustomer;

  // must do seek to skip over secret records
  DmSeekRecordInCategory(gCustomerDB, &index, itemNumber, dmSeekForward,
    dmAllCategories);
  h = DmQueryRecord(gCustomerDB, index);
  ErrNonFatalDisplayIf(!h,
    "can't get customer in GetCustomerIDForNthCustomer");
  packedCustomer = (PackedCustomer *) MemHandleLock(h);
  customerID = packedCustomer->customerID;
  MemHandleUnlock(h);

  return customerID;
}
static
void DrawOneCustomerInListWithFont(Int16 itemNumber, RectanglePtr bounds, Char
**text)
{
#pragma unused(text)
  MemHandle h;
  UInt16  index = 0;

  // must do seek to skip over secret records
  DmSeekRecordInCategory(gCustomerDB, &index,  (UInt16) itemNumber,
    dmSeekForward, dmAllCategories);
  if (gPrivateRecordStatus == maskPrivateRecords) {
    UInt16 attr;

    DmRecordInfo(gCustomerDB, index, &attr, NULL, NULL);
    if (attr & dmRecAttrSecret) {
      // show gray rectangle
      const CustomPatternType grayPattern =
        {0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55};
      CustomPatternType origPattern;
```

```
      MemHandle bitmapHandle;
      RectangleType grayBounds = *bounds;

      // draw lock icon
      grayBounds.extent.x -= SecLockWidth;
      bitmapHandle = DmGetResource(bitmapRsc, SecLockBitmap);
      if (bitmapHandle) {
        BitmapPtr bitmap = (BitmapPtr) MemHandleLock(bitmapHandle);

          // right-hand side, centered vertically
        WinDrawBitmap(bitmap, grayBounds.topLeft.x + grayBounds.extent.x,
          grayBounds.topLeft.y + ((grayBounds.extent.y - SecLockHeight) / 2));
        MemHandleUnlock(bitmapHandle);
        DmReleaseResource(bitmapHandle);
      }
      // draw gray box
      WinGetPattern(&origPattern);
      WinSetPattern(&grayPattern);
      WinFillRectangle(&grayBounds, 0);
      WinSetPattern(&origPattern);
      return;
    }
  }
  h = DmQueryRecord(gCustomerDB, index);
  if (h) {
    FontID  curFont;
    Boolean setFont = false;
    PackedCustomer  *packedCustomer = (PackedCustomer *) MemHandleLock(h);

    if (!OrderExistsForCustomer(packedCustomer->customerID)) {
      setFont = true;
      curFont = FntSetFont(boldFont);
    }
    DrawCharsToFitWidth(packedCustomer->name, bounds);
    MemHandleUnlock(h);

    if (setFont)
      FntSetFont(curFont);
  }
}
static
void InitNumberCustomers(void)
{
  ListPtr list = (ListPtr)
    GetObjectFromActiveForm(CustomersCustomersList);
    // if we use DmNumRecords, we'll count the deleted records too

  UInt16  numCustomers =
    DmNumRecordsInCategory(gCustomerDB, dmAllCategories);

  LstSetListChoices(list, NULL, (Int16) numCustomers);
}
static
void RedrawCustomersAfterChange(void)
{
  ListPtr list = (ListPtr) GetObjectFromActiveForm(CustomersCustomersList);

  LstEraseList(list); // erase list *before* updating num items
  InitNumberCustomers();
  LstDrawList(list);  // draw list *after* updating num items
}
static
void CustomersFormOpen(FormPtr form)
{
  ListPtr list = (ListPtr) GetObjectFromForm(form, CustomersCustomersList);
```

```
  InitNumberCustomers();
  LstSetDrawFunction(list, DrawOneCustomerInListWithFont);
  LstSetSelection(list, noListSelection);

  if (sysGetROMVerMajor(GetRomVersion()) >= 4)
    FrmSetMenu(form,  CustomersWithSendMenuBar);
}

static
Boolean CustomersHandleMenuEvent(UInt16 menuID)
{
  Boolean handled = false;
  UInt16  recordNumber = 0; //add at beginning. Will maintain sorting by
                            // customer ID becuase temporary customer IDs are
                            // assigned more and more negative
  MemHandle customerHandle;
  Int32 newCustomerID;

  switch (menuID) {
  case CustomerNewCustomer:
    newCustomerID = GetLowestCustomerID() - 1;
    customerHandle = DmNewRecord(gCustomerDB, &recordNumber, 1);
    if (!customerHandle) {
      FrmAlert(DeviceIsFullAlert);
    } else {
      Customer  theCustomer;
      Boolean   deleted;

      theCustomer.customerID = newCustomerID;
      theCustomer.name = theCustomer.address = theCustomer.city =
        theCustomer.phone = "";
      PackCustomer(&theCustomer, customerHandle);
      DmReleaseRecord(gCustomerDB, recordNumber, 1);
      EditCustomer(recordNumber, true, &deleted);
      RedrawCustomersAfterChange();
    }
    handled = true;
    break;

  case CustomerBeamAllCustomers:
    SendAllCustomers(exgBeamPrefix);
    handled = true;
    break;

  case CustomerSendAllCustomers:
    SendAllCustomers(exgSendPrefix);
    handled = true;
    break;
  }
  return handled;
}

Boolean CustomersHandleEvent(EventPtr event)
{
  Boolean   handled = false;
  Boolean   deleted;
  FormPtr   form;
  UInt16    listIndex;

  switch (event->eType)
  {
  case lstSelectEvent:
    listIndex = (UInt16) event->data.lstSelect.selection;

      LstSetSelection(event->data.lstSelect.pList, noListSelection);
    if (gPrivateRecordStatus == maskPrivateRecords) {
```

```
      UInt16 attr;
      UInt16  index = 0;

        // must do seek to skip over deleted records
        DmSeekRecordInCategory(gCustomerDB, &index,  listIndex,
          dmSeekForward, dmAllCategories);
        DmRecordInfo(gCustomerDB, index, &attr, NULL, NULL);
       if (attr & dmRecAttrSecret) {
         if (!SecVerifyPW (showPrivateRecords)) {
             handled = true;
             break;
           } else {
             // We only want to unmask this one record, so restore the
preference.
             PrefSetPreference (prefShowPrivateRecords, maskPrivateRecords);
           }
         }
       }
     OpenCustomerWithID(GetCustomerIDForNthCustomer(listIndex));
     handled = true;
     break;

   case menuEvent:
     handled = CustomersHandleMenuEvent(event->data.menu.itemID);
     break;

   case keyDownEvent:
     if (TxtGlueCharIsVirtual(event->data.keyDown.modifiers,
       event->data.keyDown.chr) &&
       event->data.keyDown.chr == vchrPageUp ||
       event->data.keyDown.chr == vchrPageDown) {
       ListPtr list = (ListPtr) GetObjectFromActiveForm(CustomersCustomersList);
       WinDirectionType  d;

       if (event->data.keyDown.chr == vchrPageUp)
         d = winUp;
       else
         d = winDown;
       LstScrollList(list, d, 1);
     }
     handled = true;
     break;

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

    case frmGotoEvent:
      EditCustomerWithSelection(event->data.frmGoto.recordNum, false,
        &deleted, event);
      RedrawCustomersAfterChange();
      handled = true;
      break;

    case frmCloseEvent:
#ifdef DEBUG_BUILD
      CheckDatabases(true);
#endif
      handled = false;
      break;
   }
   return handled;
}
```

Data.c

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)" by O'Reilly.

Permission granted to use this file however you see fit.
*/
#define DO_NOT_ALLOW_ACCESS_TO_INTERNALS_OF_STRUCTS
#include <BuildDefines.h>
#ifdef DEBUG_BUILD
#define ERROR_CHECK_LEVEL ERROR_CHECK_FULL
#endif
#include <PalmOS.h>
#include "Data.h"

DmOpenRef   gCustomerDB;
DmOpenRef   gOrderDB;
DmOpenRef   gProductDB;

#ifdef DEBUG_BUILD
void CheckDatabases(Boolean justCustomer)
{
  UInt8 highest;
  UInt32  count;
  UInt32  busy;

  return;
  DmGetDatabaseLockState(gCustomerDB, &highest, &count, &busy);
  if (highest || busy)
    ErrFatalDisplay("customer db not quiescent");
  if (justCustomer)
    return;
  DmGetDatabaseLockState(gOrderDB, &highest, &count, &busy);
  if (highest || busy)
    ErrFatalDisplay("order db not quiescent");
  DmGetDatabaseLockState(gProductDB, &highest, &count, &busy);
  if (highest || busy)
    ErrFatalDisplay("product db not quiescent");
}
#endif

Int16 CompareIDFunc(Int32 *p1, Int32 *p2, Int16 i,
  SortRecordInfoPtr s1, SortRecordInfoPtr s2, MemHandle appInfoH)
{
#pragma unused(i, s1, s2, appInfoH)
  // can't just return *p1 - *p2 because that's a Int32 which may overflow
  // our return type of Int16.  Therefore, we do the comparison ourself
  // and check
  if (*p1 < *p2)
    return -1;
  else if (*p1 > *p2)
    return 1;
  else
    return 0;
}

void PackCustomer(Customer *customer, MemHandle customerDBEntry)
{
  // figure out necessary size
  UInt16    length = 0;
  Char    *s;
  UInt16    offset = 0;
```

```
  length = (UInt16) (sizeof(customer->customerID) + StrLen(customer->name) +
    StrLen(customer->address) + StrLen(customer->city) +
    StrLen(customer->phone) + 4); // 4 for string terminators

  // resize the MemHandle
  if (MemHandleResize(customerDBEntry, length) == 0) {
    // copy the fields
    s = (Char *) MemHandleLock(customerDBEntry);
    offset = 0;
    DmWrite(s, offset, &customer->customerID,
      sizeof(customer->customerID));
    offset += sizeof(customer->customerID);
    DmStrCopy(s, offset, customer->name);
    offset += (UInt16) StrLen(customer->name) + 1;
    DmStrCopy(s, offset, customer->address);
    offset += (UInt16) StrLen(customer->address) + 1;
    DmStrCopy(s, offset, customer->city);
    offset += (UInt16) StrLen(customer->city) + 1;
    DmStrCopy(s, offset, customer->phone);
    MemHandleUnlock(customerDBEntry);
  }
}

// packedCustomer must remain locked while customer is in use
void UnpackCustomer(Customer *customer,
  const PackedCustomer *packedCustomer)
{
  const char *s = packedCustomer->name;
  customer->customerID = packedCustomer->customerID;
  customer->name = s;
  s += StrLen(s) + 1;
  customer->address = s;
  s += StrLen(s) + 1;
  customer->city = s;
  s += StrLen(s) + 1;
  customer->phone = s;
  s += StrLen(s) + 1;
#ifdef DEBUG_BUILD
  if (StrLen(customer->name) > 80 || StrLen(customer->address) > 80 ||
      StrLen(customer->city) > 80 || StrLen(customer->phone) > 80)
      ErrFatalDisplay("field too long");
#endif
}

// returns a customer ID no higher than any existing one.
// This is used so that new customers get negative customer IDs.
Int32 GetLowestCustomerID()
{
  UInt16    cardNo;
  LocalID   dbID;
  UInt16    mode;
  DmOpenRef dbP = gCustomerDB;
  Boolean   databaseReopened;
  Int32   result;

  DmOpenDatabaseInfo(dbP, &dbID, NULL, &mode, &cardNo, NULL);

  // want *all* customers
  if (!(mode & dmModeShowSecret))
  {
    dbP = DmOpenDatabase(cardNo, dbID, dmModeReadOnly | dmModeShowSecret);
    databaseReopened = true;
  } else
    databaseReopened = false;
```

```
  // we must use categories so we don't count deleted
  if (DmNumRecordsInCategory(dbP, dmAllCategories) == 0)
    result = 0;
  else {
    // the first record won't be deleted because deleted are at the end
    MemHandle h = DmQueryRecord(dbP, 0);
    PackedCustomer *c;

    c = (PackedCustomer *) MemHandleLock(h);
    result = c->customerID;
    if (result > 0)
      result = 0;
    MemHandleUnlock(h);
  }
  if (databaseReopened)
    DmCloseDatabase(dbP);
  return result;
}

void PackProduct(Product *product, MemHandle productDBEntry)
{
  // figure out necessary size
  UInt16    length = 0;
  Char    *s;
  UInt16    offset = 0;

  length = (UInt16) (sizeof(product->productID) + sizeof(product->price) +
    StrLen(product->name) + 1);

  // resize the MemHandle
  if (MemHandleResize(productDBEntry, length) == errNone) {
    // copy the fields
    s = (Char *) MemHandleLock(productDBEntry);
    DmWrite(s, OffsetOf(PackedProduct, productID), &product->productID,
      sizeof(product->productID));
    DmWrite(s, OffsetOf(PackedProduct, price), &product->price,
      sizeof(product->price));
    DmStrCopy(s, OffsetOf(PackedProduct, name), product->name);
    MemHandleUnlock(productDBEntry);
  }
}

// packedProduct must remain locked while product is in use
void UnpackProduct(Product *product,
  const PackedProduct *packedProduct)
{
  product->productID = packedProduct->productID;
  product->price = packedProduct->price;
  product->name = packedProduct->name;
}

// if successful, returns the product, and the locked MemHandle
MemHandle GetProductFromProductID(UInt32 productID, Product *theProduct, UInt16
*indexPtr)
{
  UInt16        index;
  MemHandle     foundHandle = 0;
  PackedProduct findRecord;

  findRecord.productID = productID;
  index = DmFindSortPosition(gProductDB, &findRecord, NULL, (DmComparF *)
CompareIDFunc, 0);
  if (index > 0) {
    PackedProduct *p;
    MemHandle     h;
```

```
    index--;
    h = DmQueryRecord(gProductDB, index);
    p = (PackedProduct *) MemHandleLock(h);
    if (p->productID == productID) {
      if (theProduct)
        UnpackProduct(theProduct, p);
      else
        MemHandleUnlock(h);
      if (indexPtr)
        *indexPtr = index;
      return h;
    }
    MemHandleUnlock(h);
  }
  return NULL;
}

void GetProductNameFromProductID(UInt32 productID, Char *productName)
{
  Product p;
  MemHandle h = GetProductFromProductID(productID, &p, NULL);

  *productName = '\0';
  ErrNonFatalDisplayIf(!h, "can't get product");
  if (h) {
    ErrNonFatalDisplayIf(StrLen(p.name) >= kMaxProductNameLength, "product name
too long");
    StrCopy(productName, p.name);
    MemHandleUnlock(h);
  }
}

// open a database. If it doesn't exist, create it.
Err OpenOrCreateDB(DmOpenRef *dbP, UInt32 type, UInt32 creator,
  UInt16 mode, UInt16 cardNo, char *name, Boolean *created)
{
  Err err = errNone;

  *created = false;
  *dbP = DmOpenDatabaseByTypeCreator(type, creator, mode);
  if (! *dbP)
  {
    err = DmGetLastErr();
    if (err == dmErrCantFind)
      err = DmCreateDatabase(cardNo, name, creator, type, false);
    if (err != errNone)
      return err;
    *created = true;

    *dbP = DmOpenDatabaseByTypeCreator(type, creator, mode);
    if (! *dbP)
      return DmGetLastErr();
  }
  return err;
}
```

Item.c

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)" by O'Reilly.

Permission granted to use this file however you see fit.
*/
#define DO_NOT_ALLOW_ACCESS_TO_INTERNALS_OF_STRUCTS
#include <BuildDefines.h>
#ifdef DEBUG_BUILD
#define ERROR_CHECK_LEVEL ERROR_CHECK_FULL
#endif

#include <PalmOS.h>
#include "Item.h"
#include "Common.h"
#include "Utils.h"
#include "SalesRsc.h"

static Item         *gItem;
static UInt32       gEditedProductID;
static Char         gProductName[kMaxProductNameLength];

static
void ItemFormOpen(void)
{
  ListPtr list;
  FormPtr form = FrmGetFormPtr(ItemForm);
  ControlPtr  control;
  FieldPtr field = (FieldPtr) GetObjectFromForm(form, ItemQuantityField);
  char  quantityString[kMaxNumericStringLength];

  StrIToA(quantityString, (Int32) gItem->quantity);
  SetFieldTextFromStr(field, quantityString, false);

  FrmSetFocus(form, FrmGetObjectIndex(form, ItemQuantityField));
  FldSetSelection(field, 0, (UInt16) StrLen(quantityString));

  list = (ListPtr) GetObjectFromForm(form, ItemProductsList);
  LstSetDrawFunction(list, DrawOneProductInList);

  gEditedProductID = gItem->productID;
  control = (ControlPtr) GetObjectFromForm(form, ItemProductPopTrigger);
  if (gItem->productID) {
    Product p;
    MemHandle h;

    h = GetProductFromProductID(gItem->productID, &p, NULL);
    ErrNonFatalDisplayIf(!h, "can't get product for existing item");

    ErrNonFatalDisplayIf(StrLen(p.name) >= kMaxProductNameLength,
      "product name too long");
    StrCopy(gProductName, p.name);
    CtlSetLabel(control, gProductName);
    MemHandleUnlock(h);
  } else
    CtlSetLabel(control, kUnknownProductName);
}

static
Boolean ItemHandleEvent(EventPtr event)
{
  Boolean   handled = false;
```

```
  FieldPtr  fld;

  switch (event->eType) {
    case ctlSelectEvent:
      switch (event->data.ctlSelect.controlID) {
      case ItemOKButton:
        {
          char  *textPtr;
          UInt32  quantity;

          fld = (FieldPtr) GetObjectFromActiveForm(ItemQuantityField);
          textPtr = FldGetTextPtr(fld);
          ErrNonFatalDisplayIf(!textPtr, "No quantity text");
          quantity = (UInt32) StrAToI(textPtr);
          gItem->quantity = quantity;
          gItem->productID = gEditedProductID;
      }
      break;

      case ItemCancelButton:
        break;

      case ItemDeleteButton:
        if (FrmAlert(DeleteItemAlert) != DeleteItemOK)
          handled = true; // don't allow further processing
        break;
      }
      break;

      case ctlEnterEvent:
        if (event->data.ctlEnter.controlID == ItemProductPopTrigger)
          HandleProductPopupEnter(ItemProductPopTrigger, ItemProductsList,
            gEditedProductID);
        // handled = false;
        break;

      case popSelectEvent:
        if (event->data.ctlEnter.controlID == ItemProductPopTrigger){
          if (HandleProductPopupSelect(event, &gEditedProductID,
            gProductName))
            CtlSetLabel(event->data.popSelect.controlP, gProductName);
        handled = true;
        break;
      }
      break;
    }
  return handled;
}
Boolean EditItem(Item *item, Boolean *deleted)
{
  FormPtr frm = FrmInitForm(ItemForm);
  UInt16  hitButton;

  gItem = item;
  FrmSetEventHandler(frm, ItemHandleEvent);
  ItemFormOpen();

  hitButton = FrmDoDialog(frm);
  FrmDeleteForm(frm);
  if (hitButton == ItemCancelButton)
    return false;
  else {
    *deleted = (Boolean) (hitButton == ItemDeleteButton);
    return true;
  }
}
```

Order.c

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)" by O'Reilly.

Permission granted to use this file however you see fit.
*/
#define DO_NOT_ALLOW_ACCESS_TO_INTERNALS_OF_STRUCTS
#include <BuildDefines.h>
#ifdef DEBUG_BUILD
#define ERROR_CHECK_LEVEL ERROR_CHECK_FULL
#endif
#include <PalmOS.h>
#include <TxtGlue.h>
#include "Order.h"
#include "Item.h"
#include "Common.h"
#include "Utils.h"
#include "SalesRsc.h"
#include "Customer.h"
#include "Exchange.h"

// Columns in the table
#define kProductIDColumn     0
#define kProductNameColumn   1
#define kQuantityColumn      2


static Char gProductNames[OrderItemLastPopup - OrderItemFirstPopup +
1][kMaxProductNameLength];
static UInt16       gTopVisibleItem;      // which item  is in the first row
static Boolean      gRowSelected = false; // true if something is selected
static UInt16       gCurrentSelectedItemIndex;  // which item  is selected
(meaningful
                                                  // only if gRowSelected is true)
static Boolean      gCurrentOrderChanged;
static Char         *gFormTitle = NULL;

// returns record number for order, if it exists, or where it
// should be inserted
UInt16 OrderRecordNumber(Int32 customerID, Boolean *orderExists)
{
  Order   findRecord;
  UInt16  recordNumber;

  *orderExists = false;
  findRecord.customerID = customerID;
  recordNumber = DmFindSortPosition(gOrderDB, &findRecord, 0,
    (DmComparF *) CompareIDFunc, 0);

  if (recordNumber > 0) {
    Order *order;
    MemHandle theOrderHandle;
    Boolean foundIt;

    theOrderHandle = DmQueryRecord(gOrderDB, recordNumber - 1);
    ErrNonFatalDisplayIf(!theOrderHandle, "DMGetRecord failed!");

    order = (Order *) MemHandleLock(theOrderHandle);
    foundIt = (Boolean) (order->customerID == customerID);
    MemHandleUnlock(theOrderHandle);
    if (foundIt) {
```

```
      *orderExists = true;
      return recordNumber - 1;
    }
  }
  return recordNumber;
}

void OpenCustomerWithID(Int32 customerID)
{
  if ((gCurrentOrder =  GetOrCreateOrderForCustomer(
    customerID, &gCurrentOrderIndex)) != NULL)
    FrmGotoForm(OrderForm);
}

Boolean OrderExistsForCustomer(Int32 customerID)
{
  Boolean orderExists;

  OrderRecordNumber(customerID, &orderExists);
  return orderExists;
}

// returns true if successful. itemNumber is location at which it was
// added
static
Boolean AddNewItem(UInt16 *itemNumber)
{
  MemHandle theOrderHandle;
  UInt16  numItems;
  Item  newItem = {0, 0};
  MemHandle oldHandle;

  ErrFatalDisplayIf(!gCurrentOrder, "no current order");
  theOrderHandle = MemPtrRecoverHandle(gCurrentOrder);
  MemHandleUnlock(theOrderHandle);

  oldHandle = theOrderHandle;
  theOrderHandle = DmResizeRecord(gOrderDB, gCurrentOrderIndex,
    MemHandleSize(theOrderHandle) + sizeof(Item));
  if (!theOrderHandle) {
    gCurrentOrder = (Order *) MemHandleLock(oldHandle);
    FrmAlert(DeviceIsFullAlert);
    return false;
  }
  gCurrentOrder = (Order *) MemHandleLock(theOrderHandle);
  numItems = gCurrentOrder->numItems + 1;
  DmWrite(gCurrentOrder, OffsetOf(Order, numItems), &numItems,
    sizeof(numItems));
  *itemNumber = gCurrentOrder->numItems - 1;
  DmWrite(gCurrentOrder, OffsetOf(Order, items[*itemNumber]), &newItem,
    sizeof(newItem));
  gCurrentOrderChanged = true;
  return true;
}

Order *GetOrCreateOrderForCustomer(Int32 customerID,
  UInt16 *recordNumPtr)
{
  MemHandle theOrderHandle;
  Order   *order;
  Boolean exists;

  *recordNumPtr = OrderRecordNumber(customerID, &exists);
  if (exists) {
    theOrderHandle = DmGetRecord(gOrderDB, *recordNumPtr);
    ErrNonFatalDisplayIf(!theOrderHandle, "DMGetRecord failed!");
```

```
    order = (Order *) MemHandleLock(theOrderHandle);
  } else {
    Order o;
    theOrderHandle = DmNewRecord(gOrderDB, recordNumPtr, sizeof(Order));
    if (!theOrderHandle) {
      FrmAlert(DeviceIsFullAlert);
      return NULL;
    }
    o.numItems = 0;
    o.customerID = customerID;
    order = (Order *) MemHandleLock(theOrderHandle);
    DmWrite(order, 0, &o, sizeof(o));
  }
  return order;
}

static
UInt16 GetRecordNumberForCustomer(Int32 customerID)
{
  UInt16       recordNumber;
  PackedCustomer  findRecord;
  PackedCustomer  *packedCustomer;
  MemHandle      theCustomerHandle;

  findRecord.customerID = customerID;
  recordNumber = DmFindSortPosition(gCustomerDB, &findRecord, 0,
    (DmComparF *) CompareIDFunc, 0);

  ErrNonFatalDisplayIf(recordNumber == 0, "Can't find customer");

  theCustomerHandle = DmQueryRecord(gCustomerDB, recordNumber - 1);
  ErrNonFatalDisplayIf(!theCustomerHandle, "DMGetRecord failed!");

  packedCustomer = (PackedCustomer *) MemHandleLock(theCustomerHandle);
  ErrNonFatalDisplayIf(packedCustomer->customerID != customerID,
    "Can't find customer");
  MemHandleUnlock(theCustomerHandle);
  return recordNumber - 1;
}

static
void ShowHideIndex(FormPtr form, UInt16 objIndex, Boolean show)
{
  if (show)
    FrmShowObject(form, objIndex);
  else
    FrmHideObject(form, objIndex);
}

static
void ShowHideRow(FormPtr form, UInt16 row, Boolean show)
{
  ShowHideIndex(form, FrmGetObjectIndex(form, row + OrderItemFirstField),
    show);
  ShowHideIndex(form, FrmGetObjectIndex(form, row + OrderItemFirstPopup),
    show);
}

FieldPtr  *gFields;
UInt16    gNumRows;
UInt16    gNumVisibleRows;

static
FieldPtr GetFieldForRow(UInt16 row)
{
  return gFields[row];
```

```
}


static
UInt16 RowNumberToPopupID(UInt16 rowNumber)
{
  return rowNumber + OrderItemFirstPopup;
}


static
ControlPtr GetPopupForRow(UInt16 rowNumber)
{
  return (ControlPtr) GetObjectFromActiveForm(RowNumberToPopupID(rowNumber));
}

static
void InitRow(UInt16 row, UInt16 itemIndex, Boolean redraw)
{
  Char  buffer[kMaxNumericStringLength + 1];
  ControlPtr  popup;
  UInt32      productID;

  StrIToA(buffer, (Int32) gCurrentOrder->items[itemIndex].quantity);
  SetFieldTextFromStr(GetFieldForRow(row), buffer, redraw);

  popup = GetPopupForRow(row);
  productID = gCurrentOrder->items[itemIndex].productID;
  if (productID)
    GetProductNameFromProductID(productID, gProductNames[row]);
  else
    StrCopy(gProductNames[row], kUnknownProductName);
  CtlSetLabel(popup, gProductNames[row]);
}

static
UInt16 ItemNumberToRowNumber(UInt16 itemNumber)
{
  return itemNumber - gTopVisibleItem;
}

static
UInt16 RowNumberToItemNumber(UInt16 rowNumber)
{
  return rowNumber + gTopVisibleItem;
}

static
FieldPtr GetCurrentField(void)
{
  if (gRowSelected) {
    FieldPtr field =
GetFieldForRow(ItemNumberToRowNumber(gCurrentSelectedItemIndex));
    ErrNonFatalDisplayIf(field == NULL, "row selected, but no current field");
    return field;
  }
  else
    return NULL;
}
static
void SaveCurrentField(void)
{
  FieldPtr field;
  Boolean  dirty;

  ErrNonFatalDisplayIf(!gRowSelected, "a row should be selected!");
```

```
  field = GetCurrentField();

  dirty = FldDirty(field);

  if (dirty) {
    Char  *textP = FldGetTextPtr(field);
    UInt32 newQuantity = 0;

    if (textP)
      newQuantity = (UInt32) StrAToI(textP);
    DmWrite(gCurrentOrder,
      OffsetOf(Order, items[gCurrentSelectedItemIndex].quantity),
      &newQuantity, sizeof(newQuantity));
    gCurrentOrderChanged = true;
  }
}

static
UInt16 GetLastPossibleTopItem()
{
  if (gCurrentOrder->numItems > gNumRows)
    return gCurrentOrder->numItems - gNumRows;
  else
    return 0;
}

static
void DeselectRow(Boolean removeFocus)
{
  if (removeFocus)
    FrmSetFocus(FrmGetActiveForm(), noFocus);
  gRowSelected = false;
}

static void SelectItem(UInt16 itemNumber);

static
void LoadFields(Boolean redraw)
{
  FormPtr   form = FrmGetActiveForm();
  UInt16    row;
  UInt16    lastPossibleTopItem = GetLastPossibleTopItem();
  Boolean   hadFocus = false;
  UInt16    focusedItem;

  // If we have a currently selected item, make sure that it is visible
  if (gRowSelected)
    if (gCurrentSelectedItemIndex < gTopVisibleItem ||
      gCurrentSelectedItemIndex >= gTopVisibleItem + gNumRows) {

      hadFocus = true;
      focusedItem = RowNumberToItemNumber(gCurrentSelectedItemIndex);
      DeselectRow(true);
      gTopVisibleItem = gCurrentSelectedItemIndex;
  }

  // scroll up as necessary to display an entire page of info
  if (gTopVisibleItem > lastPossibleTopItem)
    gTopVisibleItem = lastPossibleTopItem;

  gNumVisibleRows = gCurrentOrder->numItems - gTopVisibleItem;
  if (gNumVisibleRows > gNumRows)
    gNumVisibleRows = gNumRows;
  for (row = 0; row < gNumVisibleRows; row++) {
      ShowHideRow(form, row, true);
```

```
      InitRow(row, row + gTopVisibleItem, redraw);
  }
  for (row = gNumVisibleRows; row < gNumRows; row++)
      ShowHideRow(form, row, false);
  SclSetScrollBar(
    (ScrollBarPtr) GetObjectFromForm(form, OrderScrollbarScrollBar),
    (Int16) gTopVisibleItem, 0, (Int16) lastPossibleTopItem,
    (Int16) gNumRows - 1);

  if (hadFocus)
    SelectItem(focusedItem);
}

static
UInt16 RowNumberToFieldID(UInt16 rowNumber)
{
  return rowNumber + OrderItemFirstField;
}

static
void SelectItem(UInt16 itemNumber)
{
  FormPtr form;
  UInt16  fieldIndex;
  FieldPtr  field;

  ErrFatalDisplayIf(gRowSelected, "row already selected");
  gRowSelected = true;
  gCurrentSelectedItemIndex = itemNumber;
  if (itemNumber < gTopVisibleItem ||
    itemNumber >= gTopVisibleItem + gNumVisibleRows) {
    UInt16    lastPossibleTopItem = GetLastPossibleTopItem();

    gTopVisibleItem = itemNumber;
    if (gTopVisibleItem < lastPossibleTopItem)
      gTopVisibleItem = lastPossibleTopItem;
    LoadFields(true);
  }
  form = FrmGetActiveForm();
  fieldIndex = FrmGetObjectIndex(form,
    RowNumberToFieldID(ItemNumberToRowNumber(itemNumber)));
  FrmSetFocus(form, fieldIndex);
  field = GetCurrentField();
    // select all the text for easy typing
  FldSetSelection(field, 0, FldGetTextLength(field));
}

static
Boolean OrderDeselectRowAndDeleteIfEmptyHelper(Boolean removeFocus)
{
  Boolean   empty;

  if (!gRowSelected)
    return false;

  SaveCurrentField();

  // If the item ID is 0, delete the item.

  empty = (Boolean)
    (gCurrentOrder->items[gCurrentSelectedItemIndex].productID == 0);

  if (empty) {
    gCurrentOrderChanged = true;
    DeleteNthItem(gCurrentSelectedItemIndex);
```

```
    LoadFields(true);
  }
  DeselectRow(removeFocus);

  return empty;
}

// returns true if some item (other than that in index) matches the
// given product ID. If true, return sthe index that matches.
static Boolean ProductIDExistsInOrder(UInt32 productID, UInt16 *index)
{
  UInt16  i;

  for (i = 0; i < gCurrentOrder->numItems; i++) {
    if (gCurrentOrder->items[i].productID == productID && i != *index) {
      *index = i;
      return true;
    }
  }
  return false;
}

static
Boolean OrderDeselectRowAndDeleteIfEmpty(void)
{
  return OrderDeselectRowAndDeleteIfEmptyHelper(true);
}

static
Boolean OrderDeselectRowAndDeleteIfEmptyButDontRemoveFocus(void)
{
  return OrderDeselectRowAndDeleteIfEmptyHelper(false);
}

static
void OrderScrollRows(Int16 numRows)
{
  Int32 newVisible = gTopVisibleItem + (Int32) numRows;

  if (newVisible < 0)
    gTopVisibleItem = 0;
  else
    gTopVisibleItem = (UInt16) newVisible;
  LoadFields(true);
}

static
void OrderSetTitle()
{
  MemHandle customerHandle;
  Customer  theCustomer;
  FormPtr   form = FrmGetActiveForm();
  Char      *newTitle;
  MemHandle titleParamHandle;

  customerHandle = DmQueryRecord(gCustomerDB,
    GetRecordNumberForCustomer(gCurrentOrder->customerID));
  UnpackCustomer(&theCustomer,
    (PackedCustomer *) MemHandleLock(customerHandle));
  titleParamHandle = DmGetResource(strRsc, OrderTitleString);
  newTitle = TxtGlueParamString((Char *) MemHandleLock(titleParamHandle),
    theCustomer.name, NULL, NULL, NULL);
  MemHandleUnlock(titleParamHandle);
  ErrFatalDisplayIf(!newTitle, "can't allocate memory");
  MemHandleUnlock(customerHandle);
  FrmSetTitle(form, newTitle);
```

```
  // don't free until *after* calling FrmSetTitle
  // (because until then form still has the old title)
  if (gFormTitle)
    MemPtrFree(gFormTitle);
  gFormTitle = newTitle;
}

static
void OrderFormOpen(FormPtr form)
{
  UInt16    i;

  gNumRows = OrderItemLastField - OrderItemFirstField + 1;

  gFields = (FieldPtr *) MemPtrNew(sizeof(FieldPtr) * gNumRows);
  ErrFatalDisplayIf(!gFields, "can't allocate memory");
  for (i = 0; i < gNumRows; i++)
    gFields[i] = (FieldPtr) GetObjectFromForm(form, OrderItemFirstField + i);
  gRowSelected = false;
  gCurrentOrderChanged = false;
  gTopVisibleItem = 0;
  LoadFields(false);
  LstSetDrawFunction((ListPtr) GetObjectFromForm(form, OrderProductsList),
    DrawOneProductInList);

  OrderSetTitle();
}

static
void OrderFormClose(void)
{
  UInt16    numItems;
  MemHandle theOrderHandle;
  OrderDeselectRowAndDeleteIfEmpty();
  numItems = gCurrentOrder->numItems;
  // unlock the order
  theOrderHandle = MemPtrRecoverHandle(gCurrentOrder);
  MemHandleUnlock(theOrderHandle);

  // delete Order if it is empty; release it back to the database otherwise
  if (numItems == 0)
    DmRemoveRecord(gOrderDB, gCurrentOrderIndex);
  else
    DmReleaseRecord(gOrderDB, gCurrentOrderIndex, gCurrentOrderChanged);

  MemPtrFree(gFields);

  // free parameterized handle
  MemPtrFree(gFormTitle);
  gFormTitle = NULL;
#ifdef DEBUG_BUILD
  CheckDatabases(false);
#endif
}

static
Boolean OrderHandleMenuEvent(UInt16 menuID)
{
  Boolean handled = false;
  UInt16  zero = 0;

  switch (menuID) {
  case RecordDeleteItem:
    if (!gRowSelected)
      FrmAlert(NoItemSelectedAlert);
    else {
```

```
      UInt16 itemNumToDelete = gCurrentSelectedItemIndex;
      MenuEraseStatus(0); // because we'll turn off insertion point when
                          // deselecting; command bar will then restore it!
      if (OrderDeselectRowAndDeleteIfEmpty()) {
        // it was an empty row anyway, don't bother them
      } else if (FrmAlert(DeleteItemAlert) == DeleteItemOK) {
        DeleteNthItem(itemNumToDelete);
        gCurrentOrderChanged = true;
        LoadFields(true);
      }
    }
    handled = true;
    break;

  case RecordCustomerDetails:
    {
      Boolean deleted;

      EditCustomer(GetRecordNumberForCustomer(gCurrentOrder->customerID),
        false, &deleted);
      if (deleted) {
        // OrderCloseForm will remove order associated with this customer as
        // long as we set the numItems to 0
        OrderDeselectRowAndDeleteIfEmpty(); // this may decrement numItems
        DmWrite(gCurrentOrder, OffsetOf(Order, numItems), &zero,
          sizeof(zero));
          FrmGotoForm(CustomersForm);
      }
      else
        OrderSetTitle();
    }
    handled = true;
    break;

  case RecordBeamCustomer:
    SendCustomer(GetRecordNumberForCustomer(
      gCurrentOrder->customerID),  exgBeamPrefix);
    handled = true;
    break;

  case RecordSendCustomer:
    SendCustomer(GetRecordNumberForCustomer(
      gCurrentOrder->customerID), exgSendPrefix);
    handled = true;
    break;

  case RecordDeleteCustomer:
    if (AskDeleteCustomer()) {
      UInt16  recordNumber =
        GetRecordNumberForCustomer(gCurrentOrder->customerID);

      if (gSaveBackup) {  // archive it on PC
        DmArchiveRecord(gCustomerDB, recordNumber);
      } else {
        DmDeleteRecord(gCustomerDB, recordNumber);
      }
      // Deleted records are stored at the end of the database
      DmMoveRecord(gCustomerDB, recordNumber, DmNumRecords(gCustomerDB));

      // OrderCloseForm will remove order associated with this customer as
      // long as we set the numItems to 0
      OrderDeselectRowAndDeleteIfEmpty(); //  this may decrement numItems
      DmWrite(gCurrentOrder, OffsetOf(Order, numItems), &zero,
        sizeof(zero));
      FrmGotoForm(CustomersForm);
    }
```

```
    break;
  }
  return handled;
}


static
Boolean OrderHandleKey(EventPtr event)
{
  UInt16  c = event->data.keyDown.chr;
  Boolean handled = false;

  if (TxtGlueCharIsVirtual(event->data.keyDown.modifiers,
    event->data.keyDown.chr)) {
    // bottom-to-top screen gesture can cause this, depending on
    // configuration in Prefs/Buttons/Pen
    if (c == sendDataChr)
      handled = OrderHandleMenuEvent(RecordBeamCustomer);

    else if (c == vchrPageUp || c == vchrPageDown) {
      Int16 numRowsToScroll = (Int16) gNumRows;

      OrderDeselectRowAndDeleteIfEmpty();
      if (c == vchrPageUp)
        numRowsToScroll = -numRowsToScroll;
      OrderScrollRows(numRowsToScroll);
      handled = true;
    }
  } else if (c == linefeedChr) {
    // The return character takes us out of edit mode.
    OrderDeselectRowAndDeleteIfEmpty();
    handled = true;
  } else if (!gRowSelected && TxtGlueCharIsAlNum(c) &&
    !TxtGlueCharIsAlpha(c)) {
    // we can't use TxtGlueCharIsDigit(c) in 4.0 SDK
    // because the macro is broken
    UInt16  itemNumber;

    if (AddNewItem(&itemNumber)) {
      SelectItem(itemNumber);
      // handled = false; // pass it through to the field
    }
  }
  return handled;
}

Boolean OrderHandleEvent(EventPtr event)
{
  Boolean   handled = false;
  FormPtr   form;
  FieldPtr  field;
  Char      productName[kMaxProductNameLength];

  switch (event->eType) {
  case ctlEnterEvent:
    if (event->data.ctlSelect.controlID >= OrderItemFirstPopup &&
      event->data.ctlSelect.controlID <= OrderItemLastPopup) {
      UInt16 row = event->data.ctlSelect.controlID - OrderItemFirstPopup;
      if (! (gRowSelected && RowNumberToItemNumber(row) ==
gCurrentSelectedItemIndex)) {
        OrderDeselectRowAndDeleteIfEmpty();
        SelectItem(RowNumberToItemNumber(row));
      }
      HandleProductPopupEnter(event->data.ctlSelect.controlID,
OrderProductsList,
        gCurrentOrder->items[gCurrentSelectedItemIndex].productID);
```

```
      //handled = false;  // still want the popup to occur
    }
    break;

  case popSelectEvent:
    if (event->data.popSelect.controlID >= OrderItemFirstPopup &&
      event->data.popSelect.controlID <= OrderItemLastPopup) {
      UInt32  productID;
      UInt16    row;
      Char    newProductName[kMaxProductNameLength];

      row = event->data.popSelect.controlID - OrderItemFirstPopup;
      if (HandleProductPopupSelect(event, &productID, newProductName)) {
        UInt16  oldItemIndex = RowNumberToItemNumber(row);
        UInt16  itemIndex = oldItemIndex;

        SaveCurrentField();
        if (ProductIDExistsInOrder(productID, &itemIndex)) {
          GetProductNameFromProductID(productID,
            productName);
          if (FrmCustomAlert(ProductExistsAlert, productName, NULL, NULL)
            != ProductExistsCancel) {
            UInt32  newItemTotal;
            DeselectRow(true);

            newItemTotal = gCurrentOrder->items[itemIndex].quantity +
              gCurrentOrder->items[oldItemIndex].quantity;
            DmWrite(gCurrentOrder, OffsetOf(Order, items[itemIndex].quantity),
              &newItemTotal, sizeof(newItemTotal));
            DeleteNthItem(oldItemIndex);
            LoadFields(true); // remove deleted row from screen
            if (oldItemIndex < itemIndex)
              itemIndex--;  // because we've removed an item before this one
            SelectItem(itemIndex);
          }
        } else {
          DmWrite(gCurrentOrder,
            OffsetOf(Order,
            items[gCurrentSelectedItemIndex].productID),
            &productID,
            sizeof(productID));
          StrCopy(gProductNames[row], newProductName);
          CtlSetLabel(event->data.popSelect.controlP, gProductNames[row]);
        }
        gCurrentOrderChanged = true;
      }
      handled = true;
    }
    break;

  case menuCmdBarOpenEvent:
    field = GetCurrentField();
    if (field)
      FldHandleEvent(field, event);
    if (gRowSelected) {
      // Add Beam and delete
      MenuCmdBarAddButton(menuCmdBarOnRight, BarDeleteBitmap,
        menuCmdBarResultMenuItem, RecordDeleteItem, 0);
      MenuCmdBarAddButton(menuCmdBarOnLeft, BarBeamBitmap,
        menuCmdBarResultMenuItem, RecordBeamCustomer, 0);
    }
    // field buttons have already been added
    event->data.menuCmdBarOpen.preventFieldButtons = true;
    // don't set handled to true; this event must fall through to the system.
    break;
```

```
  case ctlSelectEvent:
    switch (event->data.ctlSelect.controlID) {
      case OrderNewButton:
      {
        UInt16  itemNumber;
        OrderDeselectRowAndDeleteIfEmpty();
        if (AddNewItem(&itemNumber)) {
          SelectItem(itemNumber);
        }
    }
    handled = true;
      break;
    case OrderDoneButton:
      FrmGotoForm(CustomersForm);
    handled = true;
      break;
    case OrderDetailsButton:
      if (!gRowSelected)
        FrmAlert(NoItemSelectedAlert);
      else {
        UInt16  indexToShow = gCurrentSelectedItemIndex;
        Boolean deleteItem;
        Item    item;

        SaveCurrentField();
        item = gCurrentOrder->items[gCurrentSelectedItemIndex];
        if (EditItem(&item, &deleteItem)) {
          gCurrentOrderChanged = true;
          if (deleteItem) {
            DeleteNthItem(gCurrentSelectedItemIndex);
            DeselectRow(true);
            LoadFields(true);
          } else {
            UInt16  oldItemIndex = gCurrentSelectedItemIndex;
            UInt16  itemIndex = oldItemIndex;

            if (ProductIDExistsInOrder(item.productID, &itemIndex)) {
              GetProductNameFromProductID(item.productID,
                productName);
              if (FrmCustomAlert(ProductExistsAlert, productName, NULL, NULL)
                != ProductExistsCancel) {
                UInt32  newItemTotal;

                DeselectRow(true);
                newItemTotal = gCurrentOrder->items[itemIndex].quantity +
                  item.quantity;
                DmWrite(gCurrentOrder, OffsetOf(Order,
items[itemIndex].quantity),
                  &newItemTotal, sizeof(newItemTotal));
                DeleteNthItem(oldItemIndex);
                LoadFields(true); // remove deleted row from screen
                if (oldItemIndex < itemIndex)
                  itemIndex--;  // because we've removed an item before this one
                SelectItem(itemIndex);
              }
            } else {
              DmWrite(gCurrentOrder, OffsetOf(Order,
items[gCurrentSelectedItemIndex]), &item,
                sizeof(item));
              LoadFields(true);
            }
          }
        }
      }
      handled = true;
      break;
```

```
    }
    break;

 case fldEnterEvent:
  {
      // we can't remove the focus because the focus has already changed!
      // However, if item is already selected, do nothing
      UInt16 itemNumber =
        RowNumberToItemNumber(event->data.fldEnter.fieldID -
OrderItemFirstField);
      if (! (gRowSelected && gCurrentSelectedItemIndex == itemNumber)) {
        OrderDeselectRowAndDeleteIfEmptyButDontRemoveFocus();
        SelectItem(itemNumber);
      }
    }
  break;

  case keyDownEvent:
    handled = OrderHandleKey(event);
    break;

  case sclRepeatEvent:
    OrderDeselectRowAndDeleteIfEmpty();
    OrderScrollRows(event->data.sclRepeat.newValue -
      event->data.sclRepeat.value);
    handled = false;  // scrollbar needs to handle the event, too
    break;

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

case frmCloseEvent:
  OrderFormClose();
    handled = false;
    break;

  case menuOpenEvent:
    // send only supported on 4.0 and later
    if (sysGetROMVerMajor(GetRomVersion()) >= 4) {
      MemHandle h = DmGetResource(strRsc, SendCustomerString);
      if (h) {
        MenuAddItem(RecordBeamCustomer, RecordSendCustomer, '\0',
          (Char *) MemHandleLock(h));
        MemHandleUnlock(h);
      }
    }
    handled = true;
    break;

  case frmSaveEvent:
    OrderDeselectRowAndDeleteIfEmpty();
    handled = false;

  case menuEvent:
    handled = OrderHandleMenuEvent(event->data.menu.itemID);
  }
  return handled;
}
```

Exchange.c

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)" by O'Reilly.

Permission granted to use this file however you see fit.
*/
#define DO_NOT_ALLOW_ACCESS_TO_INTERNALS_OF_STRUCTS
#include <BuildDefines.h>
#ifdef DEBUG_BUILD
#define ERROR_CHECK_LEVEL ERROR_CHECK_FULL
#endif
#include <PalmOS.h>
#include "Exchange.h"
#include "Common.h"

#define kAllCustomersName "All.cst"

// read at most numBytes into a new record.
// Don't use very much dynamic RAM or stack space--another app is running
static
Err ReadIntoNewRecord(DmOpenRef db, ExgSocketPtr socketPtr,
  UInt32 numBytes, UInt16 *indexPtr)
{
  char  buffer[100];
  Err   err;
  UInt16  index = 0;
  UInt32  bytesReceived;
  MemHandle recHandle = NULL;
  Char  *recPtr;
  UInt32  recSize = 0;
  Boolean allocatedRecord = false;

  do {
    UInt32  numBytesToRead = sizeof(buffer);

    if (numBytesToRead > numBytes)
      numBytesToRead = numBytes;
    bytesReceived = ExgReceive(socketPtr, buffer, numBytesToRead, &err);
    numBytes -= bytesReceived;
    if (err == errNone) {
      if (!recHandle)
        recHandle = DmNewRecord(db, &index, bytesReceived);
      else

        recHandle = DmResizeRecord(db, index, recSize + bytesReceived);
      if (!recHandle) {
        err = DmGetLastErr();
        break;
      }
      allocatedRecord = true;
      recPtr = (Char *) MemHandleLock(recHandle);
      err = DmWrite(recPtr, recSize, buffer, bytesReceived);
      MemHandleUnlock(recHandle);
      recSize += bytesReceived;
    }
  } while (err == errNone && bytesReceived > 0 && numBytes > 0);

  if (recHandle) {
    DmReleaseRecord(db, index, true);
  }
  if (err != errNone && allocatedRecord)
```

```
    DmRemoveRecord(db, index);

  *indexPtr = index;
  return err;
}

Err ReceiveSentData(DmOpenRef db, ExgSocketPtr socketPtr)
{
  Err   err;
  UInt16    index;
  Boolean isSingleCustomer = true;
  Int32 newCustomerID = GetLowestCustomerID() - 1;

  // we have all customer if it has a name like
  // "foo:all customers"
  // Otherwise, it's a single customer.
  if (socketPtr->name) {
    Char *colonLocation;

    colonLocation = StrChr(socketPtr->name, ':');
    if (colonLocation &&
      StrCompare(colonLocation + 1, kAllCustomersName) == 0)
      isSingleCustomer = false;
  }
  err = ExgAccept(socketPtr);
  if (err == errNone) {
    if (isSingleCustomer) {
      // one customer
      err = ReadIntoNewRecord(db, socketPtr, 0xffffffff, &index);

      // must assign a new unique customer ID
      if (err == errNone) {
        MemHandle h = DmGetRecord(db, index);
        DmWrite(MemHandleLock(h), OffsetOf(Customer, customerID),
          &newCustomerID, sizeof(newCustomerID));
        MemHandleUnlock(h);
        DmReleaseRecord(db, index, true);
      }
    } else {
      // all customers
      UInt16  numRecords;

      ExgReceive(socketPtr, &numRecords, sizeof(numRecords), &err);
      while (err == errNone && numRecords-- > 0) {
        UInt16 recordSize;

        ExgReceive(socketPtr, &recordSize, sizeof(recordSize), &err);
        if (err == errNone) {
          err = ReadIntoNewRecord(db, socketPtr, recordSize, &index);
          // must assign a new unique customer ID
          if (err == errNone) {
            MemHandle h = DmGetRecord(db, index);
            DmWrite(MemHandleLock(h),
              OffsetOf(Customer, customerID),
              &newCustomerID, sizeof(newCustomerID));
            newCustomerID--;
            MemHandleUnlock(h);
            DmReleaseRecord(db, index, true);
          }
        }
      }
    }
    err = ExgDisconnect(socketPtr, err);
  }
```

```
  if (err == errNone) {
    DmRecordInfo(db, index, NULL, &socketPtr->goToParams.uniqueID, NULL);
    DmOpenDatabaseInfo(db, &socketPtr->goToParams.dbID,
      NULL, NULL, &socketPtr->goToParams.dbCardNo, NULL);
    socketPtr->goToParams.recordNum = index;
    socketPtr->goToCreator = kSalesCreator;
  }
  return err;
}

static
Err SendBytes(ExgSocketPtr s, void *buffer, UInt32 bytesToSend)
{
  Err err = errNone;

  while (err == errNone && bytesToSend > 0) {
    UInt32 bytesSent = ExgSend(s, buffer, bytesToSend, &err);
    bytesToSend -= bytesSent;
    buffer = ((char *) buffer) + bytesSent;
  }
  return err;
}

void SendCustomer(UInt16 recordNumber, const Char *scheme)
{
  ExgSocketType s;
  MemHandle     theRecord = DmQueryRecord(gCustomerDB, recordNumber);
  PackedCustomer  *thePackedCustomer;
  Err       err;
  Char      name[50];

  thePackedCustomer = (PackedCustomer *) MemHandleLock(theRecord);
  MemSet(&s, sizeof(s), 0);
  s.description = thePackedCustomer->name;
  StrPrintF(name, "%s%s", scheme, "customer.cst");
  s.name = name;

  err = ExgPut(&s);
  if (err == errNone)
    err = SendBytes(&s, thePackedCustomer, MemHandleSize(theRecord));
  MemHandleUnlock(theRecord);
  err = ExgDisconnect(&s, err);
}

void SendAllCustomers(const Char *scheme)
{
  DmOpenRef dbP = gCustomerDB;
  UInt16    mode;
  LocalID   dbID;
  UInt16    cardNo;
  Boolean   databaseReopened;
  UInt16    numCustomers;

  // If the database was opened to show secret records, reopen it to not
  // see secret records.  The idea is that secret records are not sent when
  // a category is sent.  They must be explicitly sent one by one.
  DmOpenDatabaseInfo(dbP, &dbID, NULL, &mode, &cardNo, NULL);
  if (mode & dmModeShowSecret) {
    dbP = DmOpenDatabase(cardNo, dbID, dmModeReadOnly);
    databaseReopened = true;
  } else
    databaseReopened = false;

  // We should send  because there's at least one record to send.
  if ((numCustomers = DmNumRecordsInCategory(dbP, dmAllCategories)) > 0) {
    ExgSocketType s;
```

```
    MemHandle      recHandle;
    Err            err;
    UInt16         index;
    Char           name[50];
    UInt16         i;

    MemSet(&s, sizeof(s), 0);
    s.description = "All customers";
    StrPrintF(name, "%s%s", scheme, kAllCustomersName);
    s.name = name;

    err = ExgPut(&s);
    if (err == errNone)
      err = SendBytes(&s, &numCustomers, sizeof(numCustomers));

    // iterate through customers backward because we know we'll be adding
    // them at the beginning of the database when received. This way,
    // they'll end up in the right order when received
    for (i = 0,index = dmMaxRecordIndex; err == errNone && i < numCustomers;
      i++, index--) {
      UInt16  numberToSeek = 0;

      err = DmSeekRecordInCategory(dbP, &index, numberToSeek,
        dmSeekBackward, dmAllCategories);
      if (err == errNone) {
        UInt16 recordSize;

        recHandle = DmQueryRecord(dbP, index);
        ErrNonFatalDisplayIf(!recHandle, "Couldn't query record");
        recordSize = (UInt16) MemHandleSize(recHandle);
        err = SendBytes(&s, &recordSize, sizeof(recordSize));
        if (err == errNone) {
          PackedCustomer *theRecord;

          theRecord = (PackedCustomer *) MemHandleLock(recHandle);
          err = SendBytes(&s, theRecord, MemHandleSize(recHandle));
          MemHandleUnlock(recHandle);
        }
      }
    }
    err = ExgDisconnect(&s, err);
  } else
    FrmAlert(NoDataToBeamAlert);

  if (databaseReopened)
    DmCloseDatabase(dbP);
}
```

Utils.c

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)" by O'Reilly.

Permission granted to use this file however you see fit.
*/
#define DO_NOT_ALLOW_ACCESS_TO_INTERNALS_OF_STRUCTS
#include <BuildDefines.h>
#ifdef DEBUG_BUILD
#define ERROR_CHECK_LEVEL ERROR_CHECK_FULL
#endif
#include <PalmOS.h>
#include "Utils.h"
#include "Common.h"
#include "SalesRsc.h"

Err SetFieldTextFromStr(FieldPtr field, const Char *s, Boolean redraw)
{
   MemHandle      h;

   h = FldGetTextHandle(field);
   if (h) {
     Err  err;

     FldSetTextHandle(field, NULL);
     err = MemHandleResize(h, StrLen(s) + 1);
     if (err != errNone) {
       FldSetTextHandle(field, h);  // restore handle
       return err;
     }
   } else {
     h = MemHandleNew(StrLen(s) + 1);
     if (!h)
       return memErrNotEnoughSpace;
   }
   // at this point, we have a handle of the correct size

   // copy the string to the locked handle.
   StrCopy((Char *) MemHandleLock(h), s);
   // unlock the string handle.
   MemHandleUnlock(h);

   FldSetTextHandle(field, h);
   if (redraw)
     FldDrawField(field);
   return errNone;
}

// draw strings at top of rectangle r, but don't overwrite
// right-edge of r
void DrawCharsToFitWidth(const Char *s, RectanglePtr r)
{
  Int16 stringLength = StrLen(s);
  Int16 pixelWidth = r->extent.x;
  Boolean truncate;

  // FntCharsInWidth will update stringLength to the
  // maximum without exceeding the width
  FntCharsInWidth(s, &pixelWidth, &stringLength, &truncate);
  WinDrawChars(s, stringLength, r->topLeft.x, r->topLeft.y);
}
```

```
void *GetLockedAppInfo(DmOpenRef db)
{
  UInt16  cardNo;
  LocalID dbID;
  LocalID appInfoID;
  Err     err;

  if ((err = DmOpenDatabaseInfo(db, &dbID, NULL, NULL,
    &cardNo, NULL)) != errNone)
    return NULL;
  if ((err = DmDatabaseInfo(cardNo, dbID, NULL, NULL, NULL, NULL, NULL,
    NULL, NULL, &appInfoID, NULL, NULL, NULL)) != errNone)
    return NULL;
  return MemLocalIDToLockedPtr(appInfoID, cardNo);
}

UInt32 GetRomVersion()
{
  UInt32 romVersion;

  // The system records the version number in a feature.  A feature is a
  // piece of information which can be looked up by a creator and feature
  // number.
  FtrGet(sysFtrCreator, sysFtrNumROMVersion, &romVersion);
  return romVersion;
}


Err RomVersionCompatible(UInt32 requiredVersion, UInt16 launchFlags)
{
  UInt32 romVersion = GetRomVersion();

  // See if we're on in minimum required version of the ROM or later.
  if (romVersion < requiredVersion)
    {
    // If the user launched the app from the launcher, explain
    // why the app shouldn't run.  If the app was contacted for
    // something else, like it was asked to find a string by the
    // system find, then don't bother the user with a warning dialog.
    // These flags tell how the app was launched to decided if a
    // warning should be displayed.
    if ((launchFlags &
      (sysAppLaunchFlagNewGlobals | sysAppLaunchFlagUIApp))
      == (sysAppLaunchFlagNewGlobals | sysAppLaunchFlagUIApp)) {
      FrmAlert(RomIncompatibleAlert);

      // Pilot 1.0 will continuously relaunch this app unless we switch
      // to another safe one.  The sysFileCDefaultApp is
      // considered "safe".
      if (sysGetROMVerMajor(romVersion) < 2) {
        AppLaunchWithCommand(sysFileCDefaultApp,
          sysAppLaunchCmdNormalLaunch, NULL);
      }
    }
    return sysErrRomIncompatible;
  }
  return errNone;
}

Boolean Has35FeatureSet()
{
  return (Boolean) (GetRomVersion() > 0x03503000);  // from Palm OS Reference
}

UInt16 GetObjectIndexInActiveForm(UInt16 objectID)
```

```
{
  return FrmGetObjectIndex(FrmGetActiveForm(), objectID);
}

void *GetObjectFromForm(FormPtr form, UInt16 objectID)
{
  return FrmGetObjectPtr(form,
    FrmGetObjectIndex(form, objectID));
}

void *GetObjectFromActiveForm(UInt16 objectID)
{
  return GetObjectFromForm(FrmGetActiveForm(), objectID);
}
```

Headers

SalesRsc.h

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)" by O'Reilly.

Permission granted to use this file however you see fit.
*/
#define CustomersForm                           1000
#define CustomersCustomersList                  1002

#define OrderForm                               1100
#define OrderNewButton                          1102
#define OrderDetailsButton                      1103
#define OrderDoneButton                         1104
#define OrderProductsList                       1106
#define OrderScrollbarScrollBar                 1105
#define OrderItemsTable                         1101
#define OrderItemFirstField                     1111
#define OrderItem1Field                         1111
#define OrderItem2Field                         1112
#define OrderItem3Field                         1113
#define OrderItem4Field                         1114
#define OrderItem5Field                         1115
#define OrderItem6Field                         1116
#define OrderItem7Field                         1117
#define OrderItemLastField                      1117

#define kFirstFieldTop                          20

#define OrderItemFirstPopup                     1121
#define OrderItem1Popup                         1121
#define OrderItem2Popup                         1122
#define OrderItem3Popup                         1123
#define OrderItem4Popup                         1124
#define OrderItem5Popup                         1125
#define OrderItem6Popup                         1126
#define OrderItem7Popup                         1127
#define OrderItemLastPopup                      1127

#define ItemForm                                1200
#define ItemOKButton                            1204
#define ItemDeleteButton                        1205
#define ItemCancelButton                        1206
#define ItemQuantityField                       1203
#define ItemProductsList                        1210
#define ItemProductPopTrigger                   1209

#define DeleteCustomerForm                      1400
#define DeleteCustomerOKButton                  1404
#define DeleteCustomerCancelButton              1405
#define DeleteCustomerSaveBackupCheckbox        1403

#define CustomerForm                            1300
#define CustomerOKButton                        1303
#define CustomerCancelButton                    1304
#define CustomerDeleteButton                    1305
#define CustomerPrivateCheckbox                 1310
#define CustomerNameField                       1302
#define CustomerAddressField                    1307
#define CustomerCityField                       1309
```

```
#define CustomerPhoneField                      1313

#define RomIncompatibleAlert                    1001

#define DeleteItemAlert                         1201
#define DeleteItemOK                            0
#define DeleteItemCancel                        1

#define NoItemSelectedAlert                     1000

#define AboutBoxAlert                           1100

#define ProductExistsAlert                      1200
#define ProductExistsAdd                        0
#define ProductExistsCancel                     1

#define DeviceIsFullAlert                       1300

#define DeleteCustomerHelpString                1400
#define FindHeaderString                        1000
#define ItemHelpString                          1001
#define CustomerHelpString                      1003
#define SendCustomerString                      1004
#define OrderTitleString                        1005

#define CustomersWithSendMenuBar        1000
#define CustomersNoSendMenuBar          1100
#define OrderWithSendMenuBar            1200
#define OrderMenuBar                    1300
#define DialogWithInputFieldMenuBar     1400

#define CustomerNewCustomer             2001
#define CustomerBeamAllCustomers        2002
#define CustomerSendAllCustomers        2003

#define OptionsAboutSales               2101

#define RecordDeleteItem                2201
#define RecordDeleteCustomer            2202
#define RecordCustomerDetails           2203
#define RecordBeamCustomer              2204
#define RecordSendCustomer              2205

#define EditUndo                        10000
#define EditCut                         10001
#define EditCopy                        10002
#define EditPaste                       10003
#define EditSelectAll                   10004
#define EditSeparator                   10005
#define EditKeyboard                    10006
#define EditGrafitti                    10007

#define kDefaultButtonHeight 12
#define kInterButtonWidth 6
#define kDefaultButtonBottom  159
#define kDefaultButtonLeft    1
#define kDefaultButtonInModalLeft 5
#define kDefaultButtonInModalBottomMargin 5
#define kGSIFromBottom  10
#define kGSIInModalLeft     148
#define kGSILeft  152
#define kGSITop     150

#define kItemFormHeight 118
#define kDeleteCustomerFormHeight 95
#define kCustomerFormHeight       120
```

Customer.h

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan
    neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)"
by O'Reilly.

Permission granted to use this file however you see fit.
*/
#ifndef __CUSTOMER_H__
#define __CUSTOMER_H__

#include <PalmOS.h>

extern Boolean      gSaveBackup;

// returns true if the customer was changed/deleted
void EditCustomer(UInt16 recordNumber, Boolean isNew, Boolean *deleted);
void EditCustomerWithSelection(UInt16 recordNumber, Boolean isNew,
  Boolean *deleted, EventPtr event);
Boolean AskDeleteCustomer(void);

#endif
```

Customers.h

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan
    neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)"
by O'Reilly.

Permission granted to use this file however you see fit.
*/
#ifndef __CUSTOMERS_H__
#define __CUSTOMERS_H__

#include <PalmOS.h>

Boolean CustomersHandleEvent(EventPtr event);

#endif
```

Data.h

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan
    neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)"
by O'Reilly.

Permission granted to use this file however you see fit.
*/
#ifndef __DATA_H__
#define __DATA_H__

/*
Three databases:
  Customer: (sorted by order to visit. New customers added at beginning).
    customer ID # (unique. Permanent assignment by desktop (positive values). If
the
      Handheld creates a customer, it assigns a temporary negative unique #
    name: null-terminated string
    address: null-terminated string
    city: null-terminated string
    phone: null-terminated string

  Order: (sort order irrelevant)
    customerID
    numProducts
    products: array of
      product ID #
      quantity

  Product each product has a category (sorted by product id #)
    Product ID # (assigned by desktop, unique)
    Price
    ProductName: null-terminated string

  In the app info block of the product database:
    Num categories: short
    array of
      category names
*/
typedef struct {
  UInt32  productID;
  UInt32  quantity;
} Item;

typedef struct {
  Int32 customerID;
  UInt16  numItems;
  Item  items[1]; // this array will actually be numItems long.
} Order;

typedef struct {
  Int32   customerID;
  const char *name;
  const char *address;
  const char *city;
  const char *phone;
} Customer;

typedef struct {
  Int32 customerID;
  char  name[1];  // actually may be longer than 1
```

```
} PackedCustomer;

#define kMaxProductNameLength 40

typedef struct {
  UInt32  productID;
  UInt32  price;  // in cents
  const char  *name;
} Product;

typedef struct {
  UInt32  productID;
  UInt32  price;  // in cents
  char  name[1];  // actually may be longer than 1, but no more than
          //  kMaxProductNameLength (including null terminator)
} PackedProduct;

typedef char  CategoryName[16];

typedef struct {
  UInt16      numCategories;
  CategoryName  names[1];
} CategoriesStruct;


extern DmOpenRef    gCustomerDB;
extern DmOpenRef    gProductDB;
extern DmOpenRef    gOrderDB;

Int16 CompareIDFunc(Int32 *p1, Int32 *p2, Int16 i,
  SortRecordInfoPtr s1, SortRecordInfoPtr s2, MemHandle appInfoH);
MemHandle GetProductFromProductID(UInt32 productID, Product *theProduct, UInt16
*indexPtr);
void GetProductNameFromProductID(UInt32 productID, Char *productName);
void PackCustomer(Customer *customer, MemHandle customerDBEntry);
void UnpackCustomer(Customer *customer,
  const PackedCustomer *packedCustomer);
void PackProduct(Product *product, MemHandle productDBEntry);
void UnpackProduct(Product *product,
  const PackedProduct *packedProduct);
Int32 GetLowestCustomerID();
Err OpenOrCreateDB(DmOpenRef *dbP, UInt32 type, UInt32 creator,
  UInt16 mode, UInt16 cardNo, char *name, Boolean *created);

#ifdef DEBUG_BUILD
void CheckDatabases(Boolean justCustomer);
#endif
#endif
```

Item.h

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan
    neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)"
by O'Reilly.

Permission granted to use this file however you see fit.
*/
#ifndef __ITEM_H__
#define __ITEM_H__

#include <PalmOS.h>
#include "Data.h"

// return true if edited, false otherwise
Boolean EditItem(Item *item, Boolean *deleted);

#endif
```

Order.h

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan
    neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)"
by O'Reilly.

Permission granted to use this file however you see fit.
*/
#ifndef __ORDER_H__

#define __ORDER_H__

#include <PalmOS.h>
#include "Common.h"

Boolean OrderHandleEvent(EventPtr event);
UInt16 OrderRecordNumber(Int32 customerID, Boolean *orderExists);
Boolean OrderExistsForCustomer(Int32 customerID);
Order *GetOrCreateOrderForCustomer(Int32 customerID,
  UInt16 *recordNumPtr);

#endif
```

Common.h

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan
    neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)"
by O'Reilly.

Permission granted to use this file however you see fit.
*/
#ifndef __COMMON_H__
#define __COMMON_H__

#include <PalmOS.h>
#include "Data.h"

#define kMaxNumericStringLength   6
#define kSalesCreator   'SLES'
#define kUnknownProductName "-Product-"

extern privateRecordViewEnum      gPrivateRecordStatus;
extern UInt16       gCurrentCategory;
extern UInt16       gNumCategories;
extern Order        *gCurrentOrder;
extern UInt16       gCurrentOrderIndex;

void OpenCustomerWithID(Int32 customerID);
Boolean OrderExistsForCustomer(Int32 customerID);
Boolean HandleProductPopupSelect(EventPtr event,
  UInt32 *newProductID, Char *productName);
void HandleProductPopupEnter(UInt16 triggerID, UInt16 listID, UInt32 productID);

void SelectACategory(ListPtr list, UInt16 newCategory);
void DrawOneProductInList(Int16 itemNumber, RectanglePtr bounds,
  Char **text);
void DeleteNthItem(UInt16 itemNumber);

#endif
```

Exchange.h

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan
    neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)"
by O'Reilly.

Permission granted to use this file however you see fit.
*/
#ifndef __EXCHANGE_H__
#define __EXCHANGE_H__


#include <PalmOS.h>

void SendCustomer(UInt16 recordNumber, const Char *scheme);
void SendAllCustomers(const Char *scheme);
Err ReceiveSentData(DmOpenRef db, ExgSocketPtr socketPtr);

#endif
```

Utils.h

```
/*
 Copyright (c) 1998-2001, Neil Rhodes and Julie McKeehan
    neil@pobox.com
 All rights reserved.

From the book "Palm OS Programming (2nd edition)"
by O'Reilly.

Permission granted to use this file however you see fit.
*/
#ifndef __UTILS_H__

#define __UTILS_H__

#include <PalmOS.h>
Err SetFieldTextFromStr(FieldPtr field, const Char *strP, Boolean redraw);

void DrawCharsToFitWidth(const Char *s, RectanglePtr r);

void *GetLockedAppInfo(DmOpenRef db);

Err RomVersionCompatible(UInt32 requiredVersion, UInt16 launchFlags);

Boolean Has35FeatureSet();

UInt32 GetRomVersion();
UInt16 GetObjectIndexInActiveForm(UInt16 objectID);

void *GetObjectFromActiveForm(UInt16 objectID);
void *GetObjectFromForm(FormPtr form, UInt16 objectID);


#endif
```

CWDebugDefines.h

```
/* CWDebugDefines.h */
#define DEBUG_BUILD
```

Archivos de herramientas PRC

Makefile

```
CC = m68k-palmos-gcc
DEBUGCFLAGS= -g -DDEBUG_BUILD
RELEASECFLAGS= -O2
# change following from DEBUGCFLAGS to RELEASECFLAGS for a no-debug build
CFLAGS = -palmos4.0 $(DEBUGCFLAGS)
LDFLAGS=-lPalmOSGlue
#change the following line to change the name of the built project
# make sure you also change the name of the .def file to match
APP=Sales

SRCDIR=Src/
OUTPUTDIR=GCC/
RCPFILE=Sales.rcp

OBJS=$(OUTPUTDIR)Customer.o \
     $(OUTPUTDIR)Customers.o \
     $(OUTPUTDIR)Data.o \
     $(OUTPUTDIR)Exchange.o \
     $(OUTPUTDIR)Item.o \
     $(OUTPUTDIR)Order.o \
     $(OUTPUTDIR)Sales.o \
     $(OUTPUTDIR)Utils.o


$(OUTPUTDIR)$(APP).prc: $(OUTPUTDIR)$(APP) $(OUTPUTDIR)bin.stamp $(APP).def
       build-prc $(APP).def $(OUTPUTDIR)$(APP) -o $(OUTPUTDIR)$(APP).prc
$(OUTPUTDIR)*.bin

$(OUTPUTDIR)$(APP): $(OBJS)
       $(CC) -o $@ $(OBJS) $(CFLAGS) $(LDFLAGS)

$(OUTPUTDIR)%.o: $(SRCDIR)%.c
       $(CC) $(CFLAGS) -c $< -o $@

$(OBJS): $(SRCDIR)SalesRsc.h $(SRCDIR)Utils.h $(SRCDIR)Common.h $(SRCDIR)Data.h

$(OUTPUTDIR)Customer.o: \
       $(SRCDIR)Customer.h

$(OUTPUTDIR)Customers.o: \
       $(SRCDIR)Customer.h  \
       $(SRCDIR)Customers.h  \
       $(SRCDIR)Exchange.h

$(OUTPUTDIR)Exchange.o: \
       $(SRCDIR)Exchange.h

$(OUTPUTDIR)Item.o: \
       $(SRCDIR)Item.h

$(OUTPUTDIR)Order.o: \
       $(SRCDIR)Customer.h  \
       $(SRCDIR)Exchange.h  \
       $(SRCDIR)Item.h

$(OUTPUTDIR)Sales.o: \
       $(SRCDIR)Customer.h  \
       $(SRCDIR)Customers.h  \
       $(SRCDIR)Exchange.h  \
       $(SRCDIR)Item.h  \
```

```
        $(SRCDIR)Order.h


$(OUTPUTDIR)bin.stamp: $(SRCDIR)$(RCPFILE) $(SRCDIR)SalesRsc.h \
      $(SRCDIR)Sales1.bmp $(SRCDIR)Sales2.bmp $(SRCDIR)Sales4.bmp \
      $(SRCDIR)Sales8.bmp
        ( cd $(OUTPUTDIR); rm *.bin; pilrc -allowEditID -I \
             ../$(SRCDIR) ../$(SRCDIR)$(RCPFILE))
        touch $(OUTPUTDIR)/bin.stamp


clean:
        rm -f $(OBJS) $(OUTPUTDIR)$(APP) $(OUTPUTDIR)$(APP).prc \
          $(OUTPUTDIR)*.bin $(OUTPUTDIR)bin.stamp
```

Sales.def

```
application { "SalesApp-SLES" "SLES" backup }
```

PROGRAMACIÓN EN METROWERKS CODEWARRIOR

Metrowerks CodeWarrior es un entorno de programación capaz de desarrollar herramientas que puedan correr en distintas plataformas, ya sean PC's con Windows o estaciones de trabajo con Solaris, permitiendo así el mejor costo/beneficio para sus sistemas de desarrollo evitando el uso de sistemas propietarios y/o embebidos.

Las herramientas de CodeWarrior pueden ser utilizadas para escribir software para distintos mercados específicos, incluyendo suscriptores inalámbricos, sistemas de transporte, dispositivos para el consumidor, redes y comunicaciones. CodeWarrior puede abarcar el mercado de proyectos que requieren programas eficientes para dispositivos pequeños y de bajo consumo de energía. Es por eso que la mayoría del software diseñado para la plataforma Palm OS ha sido escrita en CodeWarrior.

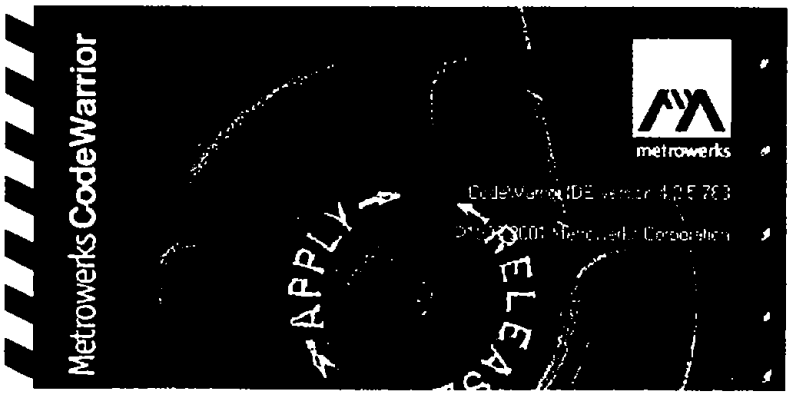

Figura 1. Pantalla de información sobre Metrowerks CodeWarrior

Instalación de Metrowerks Codewarrior

Para instalar Metrowerks CodeWarrior versión 8.0 para la Plataforma Palm OS es necesario descargar el archivo de instalación el cuál se encuentra en www.metrowerks.com y que tiene un tamaño de 36.4 Mbytes.

Al momento de descargar el programa es recomendable registrarse en el sitio para así obtener la clave de uso del software, la cual será requerida para completar la instalación.

Una vez que ha sido descargado el archivo se procede a ejecutarlo desde la ubicación donde fue guardado, lo cual nos muestra la figura 2.

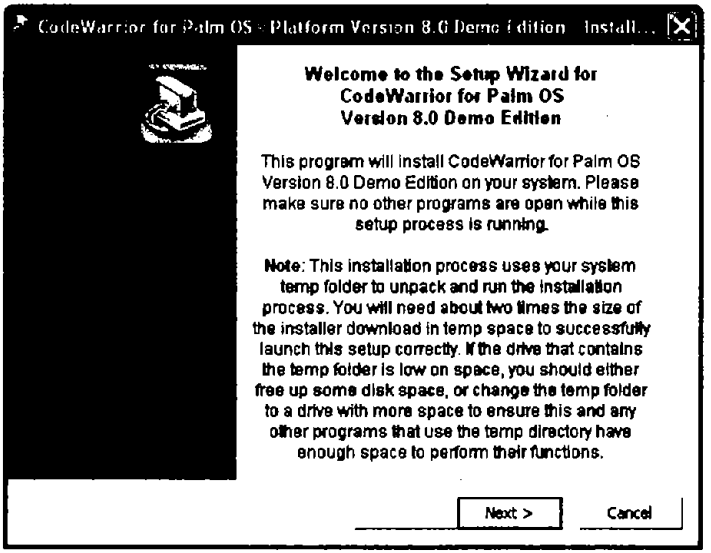

Figura 2. Pantalla de bienvenida del asistente de instalación de Metrowerks CodeWarrior.

Después se procede a dar clic en el botón "Next" lo cual nos pasa a las siguientes ventanas donde el usuario acepta el contrato de uso del software y especifica la asociación de extensión de archivos con CodeWarrior, tal como se ilustra en las figuras 3 y 4.

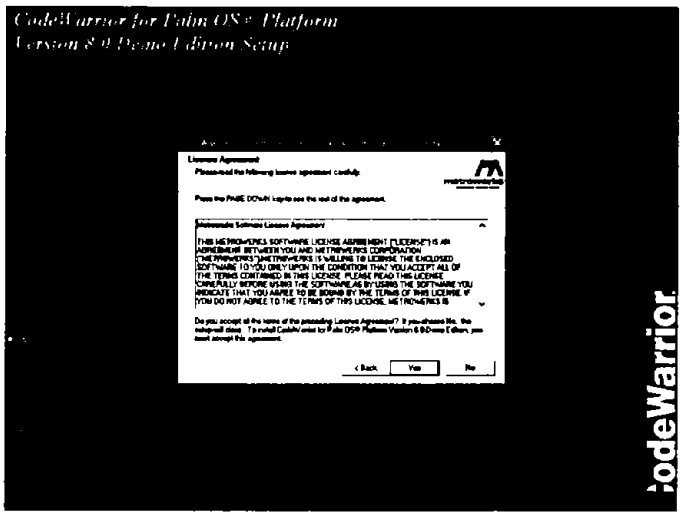

Figura 3. Contrato de uso de licencia del producto.

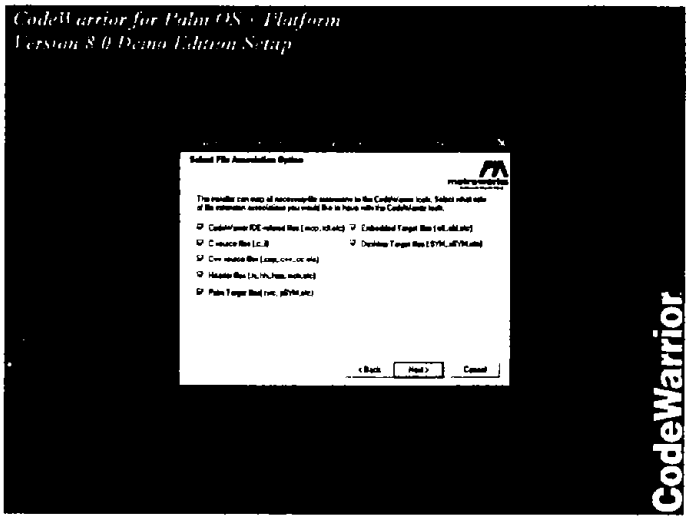

Figura 4. Selección del tipo de archivos asociados con el programa.

Una vez terminada la instalación se requerirá la clave de uso del producto y se reiniciara el sistema para efectuar los cambios realizados por el programa. Después de esto el programa está listo para ser utilizado.

Ambiente de desarrollo integrado (IDE)

Las herramientas de CodeWarrior funcionan en diferentes plataformas y pueden generar código de máquina para una gran variedad de sistemas de escritorio y embebidos.

En este conjunto de herramientas el Editor, Compilador, Montador, Depurador y otros módulos de software operan dentro de un Ambiente de Desarrollo Integrado (IDE). El IDE examina el control y ejecución de estas herramientas y les provee una interfaz que es consistente y predecible.

La operación de las herramientas de desarrollo es unida, o sea que no son programas que corran por separado. El usuario puede fácilmente cambiar entre el Editor, Compilador, Montador y Depurador simplemente señalándolo y seleccionándolo o mediante un comando.

La tabla 1 muestra qué es un IDE:

¿Qué es un IDE?		
• El IDE consiste de herramientas que son usadas a través del proceso de desarrollo de software.		
Administrador de Proyectos	Compilador	Editor
Ensamblador	Motor de Búsqueda	Montador
Navegador de Código Fuente	Depurador	Diseñador GUI
• Las herramientas son completamente unidas e integradas.		
• Un ambiente único es provisto para el desarrollo de software. o Operación consistente o Libre movimiento a través de todas las herramientas o Diseño sin modos		

Tabla 1. ¿Qué es un IDE?

La figura 5 describe el modelo conceptual de la arquitectura IDE de CodeWarrior. Los componentes IDE son interdependientes uno de otro y trabajan en acuerdo para mover al usuario rápidamente a través de las fases de edición, construcción y depuración del ciclo de desarrollo.

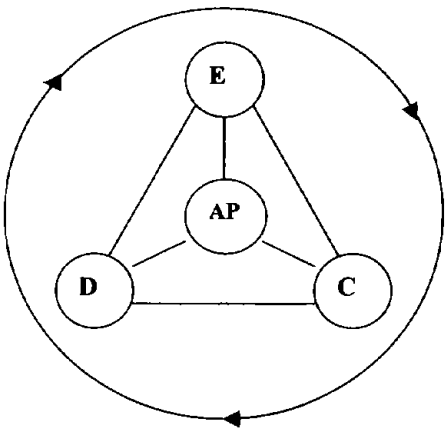

Figura 5. Interacción entre todos los componentes del IDE.

La fuerza y facilidad de uso de CodeWarrior esta centrada alrededor del Administrador de Proyectos el cual orquesta la operación de las herramientas. Este Administrador de Proyectos también maneja funciones vitales de vigilancia, como es rastrear archivos modificados.

La figura del tetraedro es una buena representación de la arquitectura IDE. Cada punto representa una herramienta o combinación de herramientas. Cada punto se comunica con los demás puntos, por ejemplo, el Compilador transfiere información sobre los errores de sintaxis que detecta y los regresa al Editor. El Administrador de Proyectos puede dejar al Editor abierto con el archivo fuente apropiado con el punto de inserción posicionado en la primera línea errónea del código fuente.

Componentes IDE

Administrador de proyectos.

El Administrador de Proyectos puede manejar proyectos tan pequeños como "Hola Mundo" y tan grandes y complejos como el mismo CodeWarrior. El usuario puede utilizar el CodeWarrior para administrar cualquier tipo de proyectos, no sólo código fuente.

Editor de código fuente.

El Editor es un ambiente completo de edición que puede manejar cualquier archivo de texto. Sintaxis claramente identificable por colores marca palabras clave para cada uno de los lenguajes soportados.

Motor de búsqueda.

El Motor de Búsqueda busca cualquier tipo de texto o grupo de archivos.

Navegador de fuente.

El Navegador de Fuente es una poderosa herramienta de navegación y edición para todos los lenguajes soportados.

Sistema construido.

El Sistema Construido incluye todos los compiladores, montadores y ensambladores para una plataforma dada en particular. Esta arquitectura de inserción permite a terceros escribir sus propias herramientas que se integran con el IDE de CodeWarrior. Este mexicanismo es similar a como terceros insertan mejoras a un navegador Web.

Depurador.

El Depurador permite al usuario ver el código fuente así como está o ensamblado. Mediante el uso del Depurador el usuario puede colocar los puntos de quiebre, ver el cambio de variables específicas y moverse línea por línea a través del programa.

Diseñador.

El Diseñador es una herramienta de desarrollo visual que ayuda al usuario a crear una interfaz completa de usuario. Incluso genera el código para dicha interfaz.

Ambientes de ejecución para Codewarrior

CodeWarrior funciona en múltiples sistemas operativos incluyendo Windows, Mac OS, Solaris y Linux, junto con otros que serán añadidos en el futuro cercano. La interfaz luce y funciona idénticamente en cualquiera de estos ambientes. Existen por supuesto diferencias mínimas en la apariencia de la pantalla debido al sistema operativo donde funciona, pero sus capacidades funcionales son efectivamente las mismas.

Uno de los más grandes beneficios del acercamiento de la plataforma múltiple de CodeWarrior es el de que los archivos de proyecto son completamente transferibles de un sistema operativo a otro. Por ejemplo, si el usuario ha estado escribiendo un proyecto para la arquitectura StarCore de Motorola utilizando las herramientas basadas en Windows, dicho proyecto podría ser transferido a un sistema Solaris para su prueba final sin tener la necesidad de convertir dicho proyecto en cualquier sentido.

Sistema de construcción de Codewarrior

Respondiendo a los comandos del administrador de proyectos, los compiladores del lenguaje utilizan una Representación Intermedia (IR) la cual es residente en memoria y sintacticamente completa. La IR es una expresión independiente del hardware de la lógica de los programas y puede ser mejorada por un optimizador. Los generadores de código de respaldo leen la IR y producen el código de máquina específico.

Los montadores utilizan este código de máquina para crear ejecutables y librerías diseñadas para funcionar en el hardware destino. Pre y post inserciones pueden ser añadidas. El soporte es provisto para usar las tradicionales herramientas de la línea de comandos. Los componentes de montador y compilador son inserciones modulares que permiten a los desarrolladores crear herramientas personalizadas o propietarias.

Los usuarios obtienen beneficios significativos de esta arquitectura. Por ejemplo, un módulo en el compilador de C/C++ transfiere a un módulo inmediato en toda la generación de códigos.

Las Optimizaciones en la IR significan que cualquier nuevo generador de código comienza su transferencia utilizando código altamente optimizado. Debido a que cada compilador utiliza el mismo módulo de salida, utilizar un nuevo procesador no requiere cambios en el compilador en el código fuente, esto hace la transferencia de código más simple.

Usos del Codewarrior

¿Que es un objetivo?

- Una plataforma objetivo es el chip o sistema operativo seleccionado para desarrollar.
  - Ejemplos de procesador son PowerPC, MIPS, x86 y 68K.
  - Ejemplos de sistema operativo son Windows, Mac OS y Neutrino.
- Un objetivo de construcción es un archivo, librería o opción de construcción para un programa.
  - Ejemplos de objetivo de construcción son depuración, versión y optimización.
- Un objetivo de construcción es análogo a producción de un archivo.

En CodeWarrior, la palabra "objetivo" puede referirse a dos diferentes cosas.
Primero, la plataforma objetivo es el sistema operativo específico o arquitectura de chip donde el usuario quiere ejecutar su código. Segundo, el objetivo de construcción refiere a las librerías específicas y archivos fuente que se usan y qué opciones se fijan.

Cada objetivo de construcción puede ser fijado completamente diferente. Por ejemplo, el usuario puede tener por separado las construcciones de depuración y de versión. La construcción de depuración podría tener todas las optimizaciones del compilador apagadas y la generación de código depurado encendida, como es la generación de tablas de símbolos y podría usar las librerías del Depurador para construir el código ejecutable. La construcción de versión podría tener todas las optimizaciones del compilador encendidas y la generación de código depurado apagada y podría usar librerías optimizadas para generar el código ejecutable.

Las funciones de un objetivo de construcción como es Makefile especifican los archivos, el compilador e instrucciones del montador, así como las librerías necesarias para generar el programa.

Especificaciones del objetivo

Las especificaciones del objetivo son las opciones que son únicas a un objetivo de construcción, estas incluyen las rutas de acceso a archivos, el tipo de montador a usar y qué plataforma se utiliza, la información de depuración, las opciones de lenguaje, las opciones de compilador y más. Estas especificaciones son cambiadas visualmente a través de un panel disponible desde el menú Edit, como se ilustra en la figura 6.

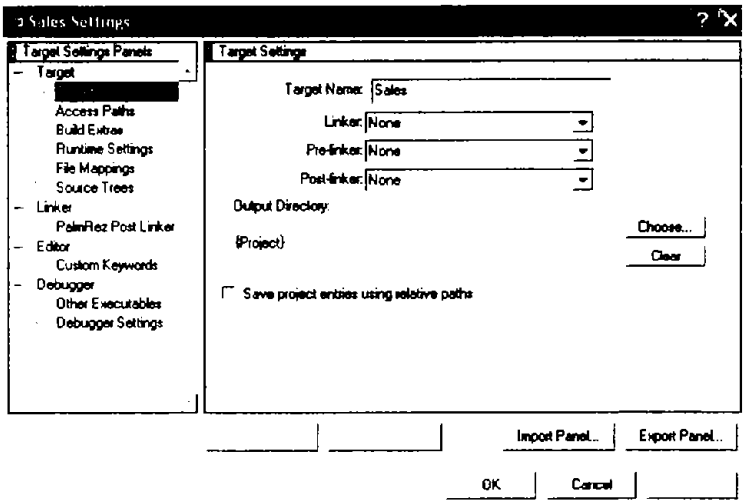

Figura 6. Especificaciones de un objetivo.

Preferencias de IDE

Las preferencias de IDE son especificaciones que son globales a todo el ambiente de desarrollo de CodeWarrior y afectan a todos los proyectos subsecuentes y objetivos de construcción. Estas preferencias incluyen coloreado de la sintaxis para el Editor, manejo de llaves, manejo de depuración y más. Estas especificaciones son cambiadas visualmente a través de un panel en el menú Edit, como se ilustra la figura 7.

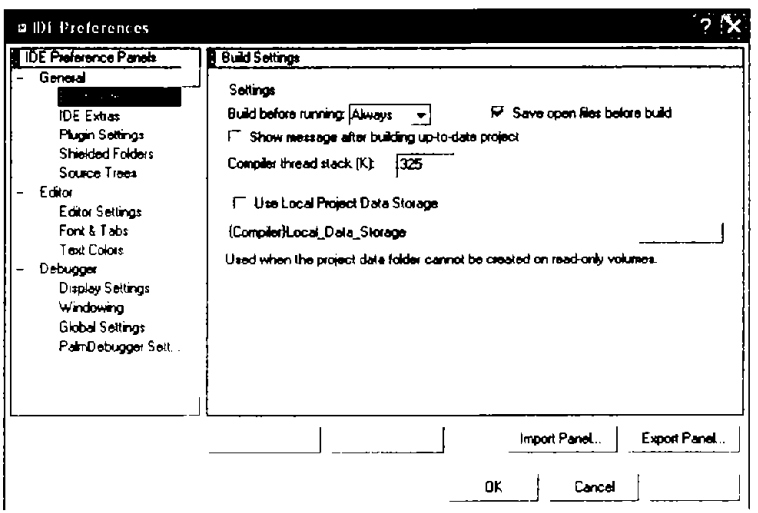

Figura 7. Preferencias de IDE.

Creación de un proyecto a partir de "un sobre"

El término "sobre" significa una plantilla para un nuevo proyecto. Un proyecto en CodeWarrior es un "contenedor" para todos los archivos de código fuente librerías y opciones para el compilador/montador. Esta información es usada para crear un programa.

Un proyecto "sobre" es simplemente un archivo de proyector estándar que sirve como una "plantilla" para los nuevos proyectos a crear. CodeWarrior siempre hace un duplicado de este "sobre" para utilizarse en cada proyecto.

Si la ventana de New Fólder esta activa, todos los archivos del "sobre" de proyecto aparecerán en nuevo directorio el cual tendrá el mismo nombre que el archivo de proyecto. Si dicha ventana esta inactiva, todos los archivos aparecerán en el directorio que el usuario especifique. Notar que en este caso existe el riesgo de colisión de nombres si el "sobre" contiene algún archivo que concuerde con el nombre de un archivo existente en el directorio.

Para sistemas integrados existen muchas posibles combinaciones de código de inicialización, librerías, controladores seriales y hardware objetivo a los cuales se les pueden proveer proyectos "sobre". De ahí que se sugiere tener precaución cuando se utilice el "sobre" predeterminado.

La ventana de proyecto

La ventana de proyecto contiene la información de sólo un proyecto de CodeWarrior a la vez, sin embargo, el usuario puede tener múltiples proyectos abiertos simultáneamente.

Esta ventana contiene la barra de herramientas y el menú de objetivos la parte baja de la ventana muestra los totales del código y datos cuando son compilados en un objetivo de construcción.

Los botones en la parte superior de la ventana son utilizados para seleccionar diferentes vistas del proyecto. La vista de archivos muestra todos los archivos y librerías para cada objetivo de construcción en el proyecto. La vista del montador habilita al usuario para cambiar el orden en el cual los archivos son montados, una característica que es importante para algunas plataformas y la vista de objetivos permite al usuario cambiar el objetivo de construcción actual y crear objetivos dependientes.

El botón de diseño sólo aparece si existe un diseño RAD en el proyecto. La vista de diseño integra el Desarrollo Rápido de Aplicación (RAD) dentro del IDE, las cuales son usadas para crear una interfaz visual del código. Esta versión de herramientas sólo soporta el desarrollo en Java.

Vista de archivos

La vista de archivos muestra todos los archivos, incluso aquellos que no son parte del objetivo de construcción actual cuyo nombre es mostrado en la barra de herramientas de la ventana de proyecto.

Los archivos pertenecen a los objetivos de construcción, no a los proyectos. El remover un archivo de la vista de archivos implica que el usuario lo ha removido del proyecto esto es equivalente a removerlo de todos los objetivos de construcción.

La columna de estado-control-versión solamente aparece cuando se está utilizando el Sistema de Control de Versión (VCS) con el proyecto.

Dar click con el botón derecho sobre un archivo dentro de la ventana de proyecto es un buen atajo para acceder a los comandos comunes que son utilizados para trabajar con un archivo en particular, como lo es el borrado y el precompilado. Cuando la vista de objetivo está activa, el dar click con el botón derecho del mouse sobre un nombre de objetivo muestra los comandos relacionados con dichos objetivos, como lo es la opción de crearle su propio código.

Si se añade un archivo a un proyecto utilizando los comandos de menú, el Administrador de Proyectos coloca a dicho archivo relacionado con el primer elemento seleccionado en la vista de archivos. Si el primer elemento seleccionado es un archivo, el archivo añadido aparece antes del elemento seleccionado. Si el primer elemento seleccionado es un grupo y si dicho grupo no tiene archivos seleccionados, el archivo añadido aparece al final de, pero dentro, del grupo. Si no hay nada seleccionado, el archivo añadido aparece al final de la lista de archivos dentro de la vista de archivos.

Arrastrar varios archivos o incluso un directorio completo es la manera más rápida de añadir archivos a un proyecto. Si el usuario arrastra un directorio a la ventana de proyecto, CodeWarrior crea un arreglo de archivos que refleja el arreglo de directorios.

Si el proyecto contiene más de un objetivo de construcción, por ejemplo el 68K o el PowerPC el programa preguntará al usuario para identificar el objetivo al cual el desea añadir el archivo. Se pueden añadir archivos a uno o más objetivos como sea apropiado. Si el proyecto contiene solamente un objetivo de construcción, los archivos son automáticamente añadidos a dicho objetivo.

La manera más rápida para remover un archivo es usando el menú contextual, esto puede ser usado para remover uno o más archivos seleccionados a través del teclado o del mouse.

Notar que el borrar un archivo del proyecto no borra a este del disco duro de la computadora. Solamente remueve la referencia a dicho archivo de la ventana del proyecto. El archivo permanece en el disco sin algún cambio.

Creación de un grupo

Los grupos proveen una manera de organizar los archivos de proyecto en un modo lógico, estos no son sinónimos con los directorios reales. Sin embargo, el usuario puede reflejar una estructura de directorios mediante el uso de grupos si así lo desea. La manera de organizar los proyectos depende estrictamente del usuario, ya que no existen reglas estrictas, ni flexibles.

Para la creación de un nuevo grupo, se debe ir al menú Project y después a la opción Create New Group; el usuario entonces puede añadir archivos manualmente a este grupo. Si se arrastra un directorio hacia la ventana del proyecto, el Administrador de Proyectos crea un nuevo grupo utilizando el nombre del directorio como el nombre del grupo y automáticamente añade todos los archivos hacia ese directorio.

Trabajo con grupos

Trabajar con grupos es similar a trabajar con archivos, el usuario puede mover los grupos con solo arrastrarlos y puede anidar grupos como subgrupos. Si se necesita renombrar a un grupo por cualquier razón, con solo dar doble click sobre el nombre del mismo y escribir el nuevo nombre es suficiente.

Para examinar un grupo cerrado, se utilizan los controles de jerarquía para abrirlo. Si el usuario quiere ver que archivos se encuentran en un grupo cuando este está cerrado o se quiere abrir un archivo en un grupo cerrado, se utiliza el menú de archivo tal como se muestra a continuación.

Para borrar un grupo, primero se selecciona y después se escoge la opción Delete en el menú Edit, o simplemente se utiliza el menú contextual que aparece al darle click con el botón derecho sobre el grupo. Si el usuario quiere remover el grupo pero quiere preservar su contenido, simplemente tiene que arrastrar los archivos fuera del grupo antes de removerlo.

Configuración del proyecto predeterminado

El proyecto predeterminado es típicamente el primer proyecto que el usuario abre. Es el objetivo de cualquier comando que el usuario ordena al IDE para ejecutar. El usuario puede cambiar este objetivo a un proyecto diferente mediante el uso del comando Set Default Project que se encuentra en el menú proyecto.

La configuración del proyecto predeterminado solo es necesaria si el IDE encuentra una situación ambigua donde no esta claro qué opciones controlan un archivo; por ejemplo, un archivo que es usado en múltiples proyectos abiertos, pero el proyecto original donde fue abierto fue subsecuentemente cerrado. Si el usuario entonces utiliza el comando Compile para este archivo, CodeWarrior utilizará las opciones del proyecto predeterminado para compilar dicho archivo.

Si el usuario cierra el proyecto predeterminado cuando otros proyectos están abiertos, el proyecto que quede encabezando la lista se convertirá en el proyecto predeterminado.

Ventana del inspector de proyecto

Esta ventana es utilizada para ver y modificar los diferentes bits de información en uno o más archivos; la mayoría de estas opciones pueden ser vistas o cambiadas dentro de la ventana de proyecto, aunque solamente se puede utilizar para un archivo o grupo de archivos a la vez.

Esta ventana permite al usuario ver la estructura de directorios completa para cualquier archivo, o la ruta de acceso utilizada para acceder al archivo de interés y a los objetivos de construcción que utiliza.

Creación de proyectos estacionarios

Cuando es posible, la creación de nuevos proyectos utilizando uno de los proyectos estacionarios ya provistos ahorra tiempo y ayuda a prevenir errores. Permite modificar el proyecto para que concuerde con un proyecto ideal. Se puede hacer cualquier tipo de cambios que se necesite, como por ejemplo, añadir librerías diferentes o adicionales, archivos fuente, opciones de objetivo y objetivos.

Una vez que el proyecto es creado se debe compilar para estar seguro que todo funciona bien. Se debe cerrar el proyecto y mover el directorio del proyecto al directorio estacionario de CodeWarrior. Después se debe borrar la carpeta de datos del proyecto estacionario, ya que no es requerida más.

Notar que el Administrador de Proyectos mantiene un registro de los archivos recordando los directorios en los cuales los archivos están situados, esto se le conoce como rutas de acceso. Una ruta de acceso absoluta es una ruta precisa desde el directorio raíz hasta el directorio con los archivos. Si el proyecto estacionario incluye rutas de acceso absolutas, éstas no trabajarán debido a que el nuevo proyecto probablemente esté en un a locación diferente o en un sistema con un diferente árbol de directorios.

Objetivos

El término "objetivo" tiene bastantes significados dentro de CodeWarrior, es importante distinguir entre los variados "objetivos" y recordar qué hacen. Para evitar la confusión, el término objetivo de construcción se refiere a una lista de archivos, librerías y opciones utilizadas para crear un programa, esto es análogo a la creación de un archivo donde se dirige la generación de código aplicando las opciones específicas en el proceso de construcción. Un archivo de proyecto puede tener uno o más objetivos de construcción.

El término "Plataforma Objetivo" se refiere al sistema operativo especifico o al microprocesador en donde se quiere que el programa se ejecute; por ejemplo, una plataforma objetivo puede ser un programa QNX/Neutrino corriendo en una PowerPC o una aplicación DSP que se ejecute sin los beneficios de un sistema operativo de tiempo real. Una plataforma objetivo es escogida como parte de la configuración de los objetivos de construcción.

Utilización de la ventana de configuración

La ventana de Configuración de Objetivos controla todas las opciones para los objetivos de construcción activos. El usuario puede obtener esta ventana utilizando uno de los siguientes métodos: (1) Seleccionando en el menú Edit la opción Target Name y Settings en donde Target Name es el nombre del objetivo de construcción activo, (2) dando click en el botón Target Settings en la ventana de proyecto, o (3) dando doble click en un objetivo en la vista de Objetivos de la ventana de proyecto.

Para modificar las opciones de los objetivos de construcción, primero se tiene que seleccionar un panel de la ventana de configuración de objetivos, tal como se muestra en la figura. Mientras más el usuario se haga familiar con CodeWarrior aprenderá qué paneles necesita cambiar más frecuentemente.

Una vez que han sido realizados estos cambios, se debe apretar el botón de guardar. Si se cierra la ventana antes de salvar dichos cambios, CodeWarrior preguntará si el usuario desea salvarlos.

El botón de revertir es útil después de que el usuario ha realizado los cambios pero antes de que los salve. Después de que se ha dado click en el botón de guardar, dichos cambios son aceptados y el botón de revertir queda deshabilitado. Se utilizan las opciones de fábrica para restaurar los paneles activos a sus valores internos predeterminados.

El usuario utiliza varios paneles de opciones para configurar un objetivo de construcción. Algunas de estas opciones que el usuario puede ajustar son la plataforma objetivo, su compilador, montador y opciones de optimización.

Selección de la plataforma objetivo

La plataforma objetivo es seleccionada utilizando el menú del montador dentro del panel de opciones de los objetivos. La acción de este montador determina la plataforma objetivo, otros paneles son utilizados para configurar el compilador, montador y otras opciones de construcción que convierten al objetivo de construcción. La tabla siguiente muestra algunos de los significados de la palabra "objetivo".

Creación de un objetivo

Para crear un nuevo objetivo de construcción se tiene que seleccionar la vista de objetivo dentro de la ventana de proyecto. Después se tiene que escoger la opción de crear nuevo objetivo y nombrarlo en la ventana que aparece así como se muestra en la figura.

Existen varios escenarios en los cuales se querrán utilizar múltiples objetivos de construcción. Por ejemplo, se puede tener un objetivo de construcción que esté configurado para la depuración y otro para la distribución, eso es, optimizado sin código de depuración.

El objetivo clon es una copia exacta de un objetivo existente, incluyendo qué archivos debe utilizar y qué configuración de construcción posee. Esta característica es útil para crear un objetivo de construcción base con el cual trabajar y cambiar como sea requerido.

Borrado de un objetivo

Para borrar un objetivo de construcción se siguen los mismos pasos que para borrar cualquier archivo o grupo dentro de la ventana de proyecto. La forma más rápida de borrar es dar click con el botón derecho del mouse sobre el objetivo y seleccionar la opción Delete. Cualquier archivo que solamente esté ligado al objetivo borrado también será removido del proyecto. Si el archivo está compartido con objetivos de construcción, permanece en el proyecto.

Configurando el objetivo predeterminado

Cada proyecto contiene un objetivo activo por omisión el cual aparece en el menú objetivo dentro de la ventana de proyecto. En la vista de objetivo, el objetivo predeterminado incluye una flecha negra en su icono.

El usuario puede modificar este objetivo predeterminado de muchísimas maneras, tal como se muestra a continuación.

- Técnicas disponibles
  - Project → Set Default Target
  - Barra de herramientas Proyecto, menú del objetivo actual
  - Vista de objetivo de la ventana de proyectos, icono del objetivo
- Realimentación
  - Opción marcada en el submenú de objetivo
  - El nombre aparece en el menú del objetivo actual
  - El icono muestra una flecha en la vista de objetivos

Adición de un archivo a un objetivo

Los archivos que se encuentran ya dentro del proyecto no pueden ser añadidos al mismo otra vez. Los archivos existentes pueden ser añadidos al objetivo actual de la siguiente manera; el usuario puede dar click en la columna de objetivo sobre el archivo dentro de la vista de archivos, o se puede utilizar el inspector de proyectos. Notar que la acción de dar click sobre la columna de objetivo para un grupo añade todos los archivos y subgrupos al grupo del objetivo de construcción actual.

Borrado de un archivo de un objetivo

La eliminación de un archivo de un objetivo de construcción puede ser hecho de la misma manera como se añadió un archivo: de igual forma se da click en la columna de objetivo o se utiliza el inspector de proyecto. Otro método que el usuario puede utilizar , es borrar el archivo del proyecto en sí.

Si sólo existe un objetivo que utilice este archivo o un sólo objetivo de construcción en el proyecto, el archivo es removido del proyecto también.

El eliminar un archivo del proyecto lo remueve de todos los objetivos. Notar que, esto se debe hacer con mucho cuidado ya que esto no puede ser deshecho. Si se quiere reintegrar el archivo al proyecto, este se debe añadir al mismo a mano.

Adición de nuevos tipos de archivos

El panel de Mapeo de Archivos, que esta disponible a través de la ventana de Configuración de Objetivos, permite al usuario cambiar como CodeWarrior trata a un archivo en particular. También permite a CodeWarrior reconozca diferentes tipos de archivo y no solamente archivos fuente y librerías.

No hay un límite en el tipo de archivos que pueden ser reconocidos dentro del Administrador de Proyecto. Algunos ejemplos incluyen notas, imágenes, documentos y hojas de cálculo. El único requisito es de que cada archivo utilice una extensión única. Esto es crítico para Windows.

Las opciones de los objetivos son específicas para un objetivo de construcción. Añadir un nuevo tipo de archivo, por ejemplo .jpg, sólo a ese objetivo de construcción, no para futuros proyectos que se creen con CodeWarrior. Si se quiere que CodeWarrior reconozca un nuevo tipo de archivo en el futuro, se tienen que modificar los proyectos estacionarios o crear uno propio con el nuevo tipo de archivo incluido.

Navegación dentro del codigo fuente

La habilidad para moverse instantáneamente entre el archivo fuente y su archivo cabecera es invaluable. El usuario puede usar el comando de palanca (toggle) para realizar esta acción mediante el uso de las teclas Control + "acento".

El comando Archivo → Abrir funciona para cualquier archivo, no solamente para archivos de cabecera. Simplemente con dar doble click en alguna palabra dentro de una línea del código fuente se ejecuta el comando. El Editor automáticamente extiende la selección para que se convierta en un archivo.h y el Administrador de Proyectos buscará por dicho archivo en las rutas de acceso; si el archivo es encontrado, el Editor lo abrirá.

El usuario también puede brincar a un archivo en particular en un grupo dentro de la vista de Archivos en la ventana de Proyecto. Simplemente dar click con el botón derecho del mouse sobre el Grupo para mostrar la lista de archivos, después se escoge alguno y el Editor lo abrirá.

Apertura de un archivo después de la compilación.

Después de que el código ha sido compilado, el IDE tiene la información sobre las dependencias entre los archivos de fuente y cabecera. Ahora el usuario puede utilizar el menú de archivo de cabecera dentro de la ventana del Editor o en la ventana de Proyecto para abrir cualquier archivo cabecera incluido en el archivo fuente.

Utilización del menú de función.

El menú de función permite al usuario moverse rápidamente entre dos funciones dentro del código. Las funciones son mostradas en el orden que aparecen en el archivo fuente. Si el usuario quiere ver esta lista en orden alfabético, debe mantener apretada la tecla Control mientras da click en el menú. Si se quiere siempre tener las funciones en orden alfabético se debe habilitar la función Orden dentro de la configuración en el panel de preferencias.

Utilización del navegador de contenidos.

Cuando el usuario compila un proyecto, el IDE recolecta todo tipo de información acerca del programa, como son las variables globales, constantes, macros y nombres de función. Esta información es administrada por el navegador de código fuente del IDE. El navegador muestra la información organizada por categorías con la ventana del Navegador tal como se muestra en la siguiente figura. Se escoge una categoría de este menú para ver los elementos que la componen; si se le da doble click a cualquiera de estos se muestra su definición.

Utilización del menú contextual.

La activación del navegador de código fuente provee capacidades adicionales que el usuario puede utilizar mientras edita un archivo fuente. Cuando se le da click con el botón derecho dentro del navegador se muestra un menú contextual, el cual enlista los lugares donde la selección aparezca dentro del código. Cuando se escoge el destino que se quiere en este menú el Editor se posiciona ahí.

El Editor automáticamente colorea los símbolos como las variables globales, macros, funciones, clases y plantillas haciéndolas reconocibles. El usuario activa esta característica y selecciona los colores que desee que se utilicen con estos símbolos.

Desplazamiento dentro del historial del navegador.

Mediante el uso del navegador el usuario puede moverse a cualquier lugar dentro de los archivos fuente. Se utilizan los comandos "Retroceder" (Go Back) o "Avanzar" (Go Forward) para desplazarse a través del historial de movimientos que se han realizado. Estos botones operan de la misma forma que los de un navegador Web.

Notar que estos botones no son parte de la barra de herramientas predeterminada dentro del IDE y deben ser añadidos manualmente.

Autocompletado de símbolos.

El navegador incluye funciones para localización de símbolos; el autocompletado de símbolos minimiza el tecleado cuando se utiliza el mismo símbolo una y otra vez. Simplemente se escriben las primeras letras del nombre del símbolo, se da click con el botón derecho y se escoge el símbolo apropiado del menú.

ACRÓNIMOS

ACL	Asyncronous Connection Link
ADC	Analog To Digital Converter
AHB	Advanced High Bus
AIPI	AHB to IP Interface
AMBA	Advanced Microcontroller Bus Architecture
AMD	Advanced Micro Devices
ARQ	Automatic Repeated Query
ASP	Analog Signal Processing
BTA	Bluetooth Accelerator
CDK	Conduit Development Kit
CGM	Clock Generation Module
CMOS	Complementary Metal-Oxide Semiconductor
CODEC	Coder Decoder
CPU	Central Processing Unit
CRC	Cyclic Redundancy Code
CSTN	Color Super-Twist Nematic
CVSD	Continously Variable Slope Delta Modulation
CW	CodeWarrior
DAC	Digital To Analog Converter
DECT	Digital Enhanced Cordless Telephone
DLL	Dynamic Link Library
DMA	Direct Memory Access
DMAC	DMA Controller
DSP	Digital Signal Processor
E/S	Entrada y Salida
EIM	External Interface Module
EPROM	Erasable Programmable Read-Only Memory
FCC	Federal Communications Commission
FH/TDD	Frequency Hopping / Time Division Duplex
FIFO	First In First Out

GPIO	General Purpose I/O Ports
HH	Hand Held
HTML	Hyper Text Marup Language
I^2C	Inter IC
IBM	International Business Machines
IDE	Integrated Development Enviroment
IP	Internet Protocol (Protocolo de Internet)
IR	Intermediate Representation
IrDA	Infra Red Data Association
ISM	Industrial, Scientific and Medical
LAN	Local Area Network
LCD	Liquid Crystal Display
LCDC	LCD Controller
MAC	Medium Access Control
MB	Megabyte
MHz	Megahertz
MIPS	Million Instructions Per Second
MMC/SD	Multimedia Card & Secure Digital
MODEM	Modulador Demodulador
MP3	Motion Picture Experts Group Layer 3
MPEG4	Motion Picture Experts Group Layer 4
MSHC	Memory Stick Host Controller
OS	Operating System
OSI	Open System Interconnection
PC	Personal Computer
PCS	Personal Communication System
PDA	Personal Digital Assistant
PLL	Phase-Locked Loops
POSE	Palm Operating System Emulator
PRC	Programming Resource Code

PSF	Programming Session File
PWM	Pulse-Width Modulation
RAD	Rapid Application Development
RAM	Random Access Memory
RF	Radio Frecuency
RISC	Reduced Instruction Set Computing
ROM	Read Only Memory
RTC	Real-Time Clock
RTOS	Real-Time Operating System
Rx	Receiver
SCO	Syncing Connection Oriented
SDRAM	Syncronous Dynamic RAM
SDRAMC	SDRAM Controller
SIG	Special Interest Group
SIM	Smartcard Interface Module
SPI	Serial Peripheral Interface
SRAM	Syncronus RAM
SSI	Synchronous Serial Interface
STN	Super-Twist Nematic
TDM	Time Division Multiplex
TFT	Thin Film Transistor
TIC	Telephone International Companies
Tx	Transmiter
UART	Universal Asynchronous Receiver Transmiter
USART	Universal Synchronous Asynchronous Receiver Transmiter
VCS	Version Control System
VMMU	Virtual Memory Management Unit
WAP	Wireless Access Protocol
WTSL	Wireless Transport Secure Layer
XML	Extensible Markup Language

GLOSARIO

Banda	Intervalo de frecuencias en el que opera cierta clase de comunicaciones por radio.
Bus	Vía a la cual pueden estar conectados en paralelo diferentes elementos de una computadora, de tal forma que pueden intercambiarse señales entre sí.
Código	Serie de reglas para expresar la correspondencia existente entre dos sistemas de signos o de términos.
Código Fuente	Forma de redacción de cualquier tipo de lenguaje que, para su procesamiento deberá sufrir el proceso de compilación.
Depurador	Programa de diagnóstico que busca, localiza y elimina errores en un programa.
Emulador	Conjunto de programas adecuados que son capaces de ejecutar programas escritos para otro tipo de computadora.
Encapsulado	Generalmente hecho de polímero, sirve para recubrir circuitos electrónicos.
Estándar	Normas y procedimientos a seguir para fabricar un producto.
Imagen ROM	Código que compone exactamente el funcionamiento de un dispositivo portátil.
Interfaz	Es el medio físico y lógico común y necesario de dos sistemas para intercambiar información.
Montador	Programa que carga o monta otros programas en memoria.
Muestreador	Dispositvo que toma muestras de señales analógicas para ser convertidas en lógicas.
Protocolo	Conjunto organizado de procesos y normas que utilizan los equipos de comunicaciones para transferir bits y bytes (datos).
Puerto	Interfaz por la cual un microprocesador se comunica con otros dispositvos.
Temporizador	Dispositivo electrónico que sirve para diferir o retardar el funcionamiento de un sistema.

BIBLIOGRAFÍA

RHODES, Neil & McKeehan, Julie.

Palm OS Programing: The Developer's Guide. Ed. O'Reilly. Second Edition. U. S. A. 2002.

POGUE, David.

Palm Pilot: The Ultimate Guide. Ed. O'Reilly. Second Edition. U. S. A. 1999.

HOLMES, Matthew et al.

Programing Visual Basic for Palm OS. Ed. O'Reilly. U. S. A. 2002.

MAAS, Brian.

Using Palm OS Emulator. Ed. PalmSource Inc. U. S. A. 2002.

GOSSETT, Brent.

Introduction to Conduit Development. Ed. PalmSource Inc. U. S. A. 2002.

GARZA Mercado, Ario.

Manual de Técnicas de Investigación. Ed. El Colegio de México. 3ª Edición. México. 1981.