
PalmV
3Com
Relatório Técnico Final
"Programando para Palm OS®
em CodeWarrior®"
Professor: Gilmar Grossi
Curso: Informática Industrial
Período: 1997-2000
Aluno: Vinícius Claudino Ferraz
abc
123

CEFET-MG – Curso Técnico de Informática Industrial

Vinícius Claudino Ferraz

Ada Tecnologia e Sistemas Ltda.

Supervisor: Frederico Samarane

Professor Orientador: Gilmar Grossi

Apresentação da Empresa

A Ada Tecnologia e Sistemas Ltda. surgiu a partir do crescimento do setor de programação da Samarane, uma empresa fundada a mais de cinqüenta anos, consolidada na área de automação comercial, representante da Sweda, prestadora de serviços e de assistência técnica relacionados a máquinas registradoras.

Numa associação com a Palm Soluctions Ltda., empresa de Ataliba Faria Jr., voltada exclusivamente para desenvolvimento de sistemas para a linha Palm® de computadores de mão, criou-se a PalmSys®. Hoje estamos com grandes clientes: Itambé, Universo On Line, Telemar e outros.

Percebendo que o mercado Palm era mais promissor, essa parte ficou em proeminência, restando da parte de automação comercial os antigos clientes, com direito a manutenção, e a disponibilidade dos sistemas à venda. O Pocket PC foi posto de parte por sua minoria no mercado (cerca de 20%), mas a PalmSys se coloca na posição de poder vir a aceitar pedidos para essa plataforma.

E eu, estagiário da Ada, fui o primeiro programador a trabalhar exclusivamente para Palm, no design e no desenvolvimento de aplicações.

Este relatório tem o objetivo de mostrar o poder do Palm, ainda comparado preconceituosamente com uma “agendinha”, e tornar pública a grande problemática de se programar para dispositivos pequenos.

Introdução

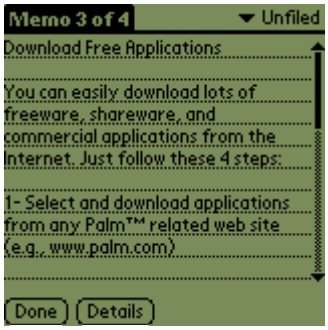
O Palm

O Palm não é uma simples agendinha, como pode parecer à primeira vista; é um pequeno computador de mão (*handheld computer*), microprocessado (“DragonBall” é o apelido do processador, derivado do 68000 da Apple), com seu próprio sistema operacional: o Palm OS. Normalmente ao comprá-lo as seguintes aplicações já vêm em sua EPROM (isso mesmo: programável!):

- Memo Pad, para guardar "notas mentais" a serem lidas posteriormente;
- Address Book, para gerenciar lista de endereços;
- Date Book, para agendar previamente nossas tarefas;
- To-do List, para o gerenciamento de tarefas que não dependem de data;
- uma Calculadora Simples;
- Expense, para gerenciamento de gastos;
- Mail, para leitura e escrita de mensagens, a serem enviadas pelo PC.

Seu sucesso deve-se tanto a essas aplicações que já vêm embutidas quanto pela possibilidade de estender a utilização, comunicando com o PC, acoplando dispositivos em seu conector serial (RS-232) e instalando (ou programando) novos jogos ou utilitários (existem vários sites com programas grátis), nos 2 a 8 MB livres que ele oferece.

Calculadora Padrão



Um memorando no MemoPad

Calculadora Científica, instalada

A filosofia do Palm é ser uma extensão do PC (seu histórico é muito interessante, vale a pena ler na página da PalmSys). O Palm se comunica com outros dispositivos pela tecnologia do *HotSync*®, que permite o backup dos dados do Palm, a alteração de dados fora do Palm e a instalação de novas aplicações ou bancos de dados.

A entrada de dados no Palm é feita via toques de caneta ou *Graffiti*®, o sistema de reconhecimento de escrita mais utilizado.

Tendo o Palm muitos dos recursos de um computador pessoal (eu só mostrei alguns exemplos...), a sua programação é no mínimo tão sofisticada quanto a de um PC.

Máquina Fotográfica para Visor®, da Handspring®

Dispositivo que permite ouvir MP3 em um Visor

O ambiente de programação

O CodeWarrior, utilizado no PC, é o compilador C/C++ criado pela Metrowerks para rodar em Windows 9x, Windows NT ou ambientes Macintosh. Ele vem com o *Constructor for Palm OS*®, todo um conjunto de ferramentas (SDKs – *Software Development Kits*), os códigos-fonte dos programas padrão e uma vasta documentação sobre seu IDE (*Interface Development Environment*), seus SDKs e o Palm OS com suas funções, em arquivos de formato HTML e PDF.

Existem outras linguagens de programação para Palm – por exemplo, a Satellite Forms, da Puma Technology, e a Pendragon Forms, da Pendragon Software Corporation, das quais eu apenas ouvi falar –, mas o CodeWarrior é a opção da empresa, pelas características do C++: relativo alto nível, com acesso a todo o dispositivo em baixo nível.

No Constructor nós criamos os resources de formulário, menus, ícones, bitmaps, etc., que serão linkados pelo CodeWarrior aos códigos-objeto gerados por ele mesmo. O resultado final é um arquivo de extensão .PRC, que deve depois ser transferido para o Palm por HotSync.

Para fazermos os testes e a depuração dos programas, utilizamos o *Palm OS Emulator*®, que interage com o CodeWarrior. Entretanto, existem certos dispositivos que não são “emuláveis”, isto é, não conseguimos simular sua operação com o Emulator. Existe uma versão light do CodeWarrior na página oficial da Metrowerks e o Emulator é público.

A arte de desenvolver aplicações Palm

Programar para Palm pode ser quase uma revolução para quem está acostumado à plataforma fixa. Vejamos as grandes diferenças:

Ambiente Palm	Computador Tradicional
Poder de processamento limitado	Grande poder de processamento
RAM e capacidade de armazenagem muito limitados	RAM e capacidade de armazenagem quase ilimitados
Tamanho de tela limitado	Janelas muito largas
Uma única janela utilizável de cada vez	Muitas janelas sobrepostas
Acesso às janelas dos programas e à área de trabalho	Acesso a todos os dados
Proposta específica	Proposta geral
O usuário pode estar em qualquer lugar	O usuário está tipicamente numa mesa de trabalho
Elementos simples de interface com o usuário	Complexos elementos de interface
Entrada de dados limitada a toques e Graffiti	Teclado, mouse e outros para entrada de dados
Alimentado por bateria	Alimentado pela rede elétrica

Vale lembrar que essa comparação é ilusória, é como se o Palm fosse visto de dentro para fora; o usuário vê o Palm executando suas tarefas tão bem quanto o PC. Levando-se em consideração os aspectos de engenharia como tamanho e consumo de energia, tem-se uma melhor idéia do potencial do Palm.

Etapas da implementação

Na PalmSys, tudo começa no pedido de uma empresa. Aqui deve-se estar atento às requisições dos futuros usuários e analisar como e se é possível atendê-los. Define-se aqui “o que fazer” e esboça-se o “como fazer”, fazendo um fluxo de dados, começando com de onde eles são coletados, toda sua utilização até a sua remoção do sistema. Uma boa visão geral do problema não causará mudanças drásticas durante o seu desenvolvimento.

Em seguida, analisam-se as operações que o software terá que fazer, em termos de performance, segurança e qualidade. Define-se aqui o “como fazer”, em quais versões do Palm OS vamos trabalhar, o “peso” da aplicação e se é melhor que o processamento de parte dos dados seja feito em outra plataforma.

Depois vamos nos preocupar com o design, a entrada e a exibição dos dados. Normalmente mostra-se um resumo da informação e um toque abre uma janela de detalhes. Nada de muitos formulários e janelas ou preenchimento excessivo da tela (160 x 160 pixels).

Tendo em vista que as aplicações Palm são orientadas a evento e monotarefa (apesar de internamente o Palm OS ser multitarefa), passamos à codificação. Se uma parte do processamento será feita no PC, ela também é codificada nessa parte (por exemplo um programa Windows que permita ordenar, editar, inserir e remover registros de um grande banco de dados). Além disso, talvez a aplicação precise gerenciar a sincronização dos dados. Esse componente gerenciador chama-se *conduit*, roda nas plataformas Windows ou Macintosh e pode ser um terceiro elemento da funcionalidade do projeto.

Finalmente, vamos aos testes. Deve-se testar cada interface e cada módulo por onde passam os dados, certificando de que eles estão sendo cuidados devidamente. Não se pode esquecer também da performance e do stress, pois sabe-se que usuários testando sempre criam situações que os desenvolvedores não esperavam. Sugere-se ignorar todos os eventos de usuário acontecidos durante um longo processamento. Esta é a análise da condição do usuário quando utilizando o programa.

Por último, a aplicação pode precisar de alguma manutenção e novas requisições de usuário. Na prática, as fases do desenvolvimento vão estar em intensa comunicação e não vão ser executadas apenas uma vez.

Exemplo de projeto simples

Para exibir os problemas que surgem durante a implementação, vou agora seguir as etapas de um projeto prático.

A. Requisições dos usuários

Trata-se de um software simples, para gerenciar pequenos textos de no máximo uma linha. Os usuários vão querer ler, modificar e salvar com agilidade. Se possível, extraí-las para o PC.

Como analista, vejo que inicialmente não tenho dados, estes serão criados, salvos e modificados apenas no Palm e lidos por outra aplicação do PC. É possível prever que basta um banco com uma string por registro.

B. Requisições de software/hardware

O melhor alvo do programa é o Palm Professional: existem muitos no Brasil, tem um bom conjunto de funções auxiliares, possui tela gráfica monocromática e memória suficiente (2,5KB para pilha de funções e variáveis locais, mais 12KB para alocações dinâmicas e variáveis globais ou estáticas). Assim, o software rodará em qualquer Palm OS de versão 2.0 ou superior (Palm III, Palm V, etc.).

C. Design

Basicamente, uma tela com a lista de textos existentes e um campo para alteração (quanto menos telas, melhor), opções de Adicionar, Alterar e Excluir.

O Constructor torna fácil esta tarefa, permitindo-nos criar componentes (formulários, menus, etc.) e arrastar objetos para dentro deles, em seguida mudando suas características. Desta maneira, comparando com o resource criado com projeto padrão C++ CodeWarrior, crio o resource final Starter.rsrc, com os seguintes componentes:

- Formulário Main, com três botões, uma lista e um field (campo de edição).
- Formulário Sobre, com as informações de autor e data (rótulos).
- Uma barra de menu Main Form, um componente de menu "Opções" e um item de menu "Sobre o Linhas..." (para permitir mudança de opções em tempo de execução, existe a seguinte diferenciação: uma barra é um conjunto de colunas, que por sua vez é um conjunto de itens).
- Três *alerts*: um de confirmação sim/não; outro para informação, com botão OK; e um para erro fatal, com botão reset. Um alert é uma caixa de diálogo rápida, contendo apenas texto e botões. O texto de cada um está como "^1^2^3", para podermos formar uma mensagem em tempo de execução concatenando três strings (função `FrmCustomAlert`).
- Um ícone para o programa (apenas três linhas formando um L), de nome Linhas.

Sempre que salvamos a lista de resources, é gerado o arquivo .rsrc e um header com as constantes para utilizarmos no fonte principal. Configurei o nome desse arquivo para StarterRsc.h e sua listagem final está no Anexo I. Utilizando constantes, o código acompanha o resource e a probabilidade de ocorrerem erros ao modificar o resource diminui.

D. Codificação

É aqui que moram o suor e a complexidade. Várias subetapas podem ser notadas.

1. Configurando o novo projeto

Ao criar o novo projeto C++ no CodeWarrior (File => New... => Palm OS 3.1 (English) Stationery => Palm OS C++ App), de nome *Linhas*, ele adiciona os arquivos Starter.cpp, Starter.rsrc e "C++ Readme.txt".

Este último diz apenas que este é um projeto-exemplo da Metrowerks; o Starter.rsrc padrão é substituído pelo já criado; o Starter.cpp é o fonte principal. O padrão é enorme, porque vem preenchido de comentários e preparado para grandes projetos; na primeira compilação já avisa que existe uma variável não utilizada. "Enxugando-o" ao máximo, fica:

```
#include <Pilot.h>
#include <SysEvtMgr.h>
#include "StarterRsc.h"

/***********************************************************************
 *
 *   Constantes
 *
 ***********************************************************************/

#define ROM20   0x02003000   // codigo da ROM 2.0

/***********************************************************************
 *
 *   Variaveis Globais
 *
 ***********************************************************************/

FormPtr pform;

/***********************************************************************
 *
 *   Prototipos e Macros-Funcao
 *
 ***********************************************************************/

Boolean FormLoadHandleEvent(EventPtr eventP);
void    Info(char* m1, char* m2 = "", char* m3 = "");
Boolean MainFormHandleEvent(EventPtr eventP);
Boolean VersaoIncompativel();

/***********************************************************************
 *
 *   Codigo das Rotinas
 *
 ***********************************************************************/

Boolean FormLoadHandleEvent(EventPtr eventP)
{
   if (eventP->eType != frmLoadEvent) return false;

   pform = FrmInitForm(eventP->data.frmLoad.formID);
   FrmSetActiveForm(pform);
   FrmSetEventHandler(pform, MainFormHandleEvent);

   return true;
}
```

```
void Info(char* m1, char* m2 = "", char* m3 = "") {
   FrmCustomAlert(InfoAlert, m1, m2, m3);
}

Boolean MainFormHandleEvent(EventPtr eventP)
{
   switch (eventP->eType) {
      case menuEvent:
      return true;

      case frmOpenEvent:
         FrmDrawForm (pform);
      return true;
   }

   return false;
}

DWord PilotMain(Word cmd, Ptr cmdPBP, Word launchFlags)
{
   if (cmd != sysAppLaunchCmdNormalLaunch || VersaoIncompativel())
      return 0;

   FrmGotoForm(MainForm);

   Word error;
   EventType event;

   do {
      EvtGetEvent(&event, evtWaitForever);

      if (! SysHandleEvent(&event))
         if (! MenuHandleEvent(0, &event, &error))
            if (! FormLoadHandleEvent(&event))
               FrmDispatchEvent(&event);

   } while (event.eType != appStopEvent);

   return 0;
}

Boolean VersaoIncompativel() {
   DWord aux;
   FtrGet(sysFtrCreator, sysFtrNumROMVersion, &aux);
   if (aux >= ROM20) return false;
   Info("Sistema Incompatível. Esta aplicação roda apenas no Sistema
2.0 ou superior.");
   return true;
}
```

Os headers dão acesso às milhares de definições e funções de sistema, formulário e outros objetos, alocação de memória, etc. Por exemplo, Boolean está definido como unsigned char e existem 59 funções documentadas apenas para lidar com janelas, entre elas WinEraseRectangle – `void WinEraseRectangle (RectanglePtr r, Word cornerDiam)`, limpa uma área do vídeo – e WinDrawChars – `void WinDrawChars (CharPtr chars, Word len, SWord x, SWord y)` faz o desenho dos caracteres de uma string.

O ponto de entrada é á função PilotMain. Ali o programa verifica o comando de sistema que o chama e só prossegue se foi chamado “normalmente”, pelo usuário, pois o Palm OS chama as aplicações em outras situações, como durante o HotSync e logo após um reset. Checa também a versão do sistema operacional, gera eventos para iniciar o formulário (`FrmGotoForm`) e fica no loop de eventos até ser gerado o evento de fim de aplicação (`appStopEvent`).

Cada evento obtido (`EvtGetEvent`) pode ser de sistema, de menu, de inicialização de formulário ou de ação do usuário no formulário. Essa classificação raramente varia de programa para programa, mas um joguinho, por exemplo, pode precisar interpretar antes do sistema o acionamento das teclas abaixo do display. Em nosso caso, já são tratados automaticamente os eventos de sistema e de desenho do menu. Os de carregar formulário, por praxe aqui na empresa, colocamos em função separada: `FormLoadHandleEvent`. Nela, manualmente temos que iniciar o formulário, torná-lo ativo e definir a função que lida com os eventos referentes a ele. Esta função (`MainFormHandleEvent`) será depois ativada por `FrmDispatchEvent`.

Logo, eventos de formulário chegam em `MainFormHandleEvent`. O primeiro deles que temos que tratar é o `frmOpenEvent`. Nesse instante, e, para variar, manualmente, plotamos os objetos com `FrmDrawForm` e retornamos true porque o evento foi tratado.

Mesmo com todo esse trabalho, o único desses componentes que trabalha sozinho é o field, que já aceita entrada de dados via Graffiti.

2. Interpretando os botões e o menu

Esqueleto pronto, podemos acompanhar o código final no Anexo II.

Três botões, três funções: `void Adicionar()`, `void Alterar()` e `void Excluir()`, que temos que chamar no evento `ctlSelectEvent`, pois, por definição, o botão é um objeto de controle. O formulário Sobre é muito mais simples de tratar, sem necessidade de gerenciador de eventos: com a função `FrmDoDialog`, ele é exibido até um botão ser acionado. Estes dois eventos estão nas linhas 224 a 244.

O membro data, de eventP (linha 225), é uma union com os dados úteis que vêm junto com qualquer evento. No caso, comparei o ID do botão selecionado com as constantes de StarterRsc.h.

3. Tratamento do banco

Bancos de dados são uma complexidade à parte, então preferi criar uma classe, que passou por várias alterações antes de ficar como nas linhas 29-44. Estão aí com bastante clareza as funções que o programa vai precisar: abrir e fechar o banco, ler, adicionar, modificar, excluir, localizar, liberar e obter o número de registros. Os problemas encontrados foram:

- A área de dados do Palm é dividida em cartões de memória, apesar de normalmente utilizar-se apenas um. As funções do gerenciador de bancos de dados possuem campos de cartão, mas ainda é desconhecido como algumas delas funcionarão com mais de um cartão. Por isso utilizei 0 em tudo o que se referia a cartões.

- O tratamento dos registros está intimamente ligado a alocação dinâmica. Cada registro de um banco de dados é um *handle*, que é definido como um ponteiro para ponteiro. Um handle pode ser modificado pelo Palm OS no ato de uma alocação, diferente de um ponteiro comum, estático. Exemplo: se na memória houver três blocos livres de 3KB e o sistema tentar alocar 7KB, ele vai conseguir apenas se puder mover as outras áreas alocadas de modo a unir os blocos livres.

- Handles são melhores que ponteiros, entretanto, mais difíceis de se lidar. Para alterar o conteúdo de um ponteiro comum, utilizamos os operadores padrão C++ de indireção (*) e de seleção de membro (->). Antes de ler ou gravar em um handle, temos que marcá-los como estáticos (`MemHandleLock`). Após alterar seu conteúdo como se altera um ponteiro comum, devemos liberar seu movimento (`MemHandleUnlock`).

- Não existem diretórios no Palm. Os bancos são identificados por um id único e pelo nome, e têm outras propriedades para facilitar o manuseio, como tipo e criador. As propriedades do banco que o programa manipula estão definidas nas constantes das linhas 11 a 13, sendo que suas strings têm no máximo 70 caracteres (linha 16). Ao ser transferido para o PC, ele terá o nome Linhas.PDB.

- Trabalho ao abrir (linhas 70-85): verificar se já existe, senão criar, antes de abrir. Ao criar, marcar atributo de backup, para a aplicação do PC ter acesso aos dados.

  LocalID é um tipo para conter o ID único de um banco, necessário para abri-lo. Da mesma forma, DmOpenRef contém informações necessárias para realizar a maioria das operações de um banco de dados, como ler e gravar registros ou fechar o banco.

- Em caso de erros não contornáveis, a saída é a função `Fatal` (199-204), que força um reset no dispositivo, pois, sem uma função do tipo exit(), seria preciso muito “malabarismo” para finalizar a aplicação. Também, isso não vai causar problemas num sistema monousuário.

- Ao ler uma linha (91-94): “travar” o registro e não esquecer de liberá-lo depois, através de `RegLib` (133-135).

- Verificar se já existe uma dada linha (122-131): ler cada registro e fazer a comparação.

- Tentar adicionar uma nova linha (96-109): verificar se já não existe uma linha igual, criar novo registro no final e mudar seu conteúdo para o que foi passado como parâmetro. Retornar true se houve sucesso, false se já existia.

O programa final abre e fecha o banco em `PilotMain()`, linhas 292 e 309.

4. Tratamento da lista

Para utilizar os objetos do formulário temos que obter seus ponteiros, assim como fizemos com o próprio form. Dado o id de um objeto – que é a constante no header –, a função `ObjectPointer` (linhas 283-285) retorna o ponteiro.

Os ponteiros que vamos precisar são para a lista e para o field (26-27). O melhor a fazer é obtê-los apenas uma vez, assim que o form é iniciado. Daí a função `MainFormInit` (271-281), inicializadora, chamada em `frmOpenEvent` (263-265). Ali, além de obter o número de itens da lista e dar foco ao field, redirecionei a função que plota itens da lista (275). O plano é: ao invés de copiar as linhas do banco para a lista, que uma linha seja lida do banco toda vez que precisar ser exibida, para economizar memória. A função `AtualizaList` (176-181) é a responsável por isso: lê do banco e plota no vídeo.

5. Botão Adicionar

`Adicionar()` (141-155) verifica o texto no field, adiciona na lista e atualiza-a ou exibe por que não pôde adicionar.

O field foi configurado no Constructor para aceitar no máximo 70 caracteres e ter apenas uma linha; não obstante, temos o problema da largura de caracteres. Setenta emes ('m') ultrapassam a largura da tela, ao contrário de setenta is ('i'). Preferi restringir a largura a 145 pixels (constante na linha 14), para caber na largura da lista sem problemas com as setas. Por pura coincidência, essa verificação é feita na linha 145.

6. Botão Alterar

Em `Alterar()` (157-174), há também as mesmas verificações feitas ao adicionar, além de ver se há um item a alterar (158).

Na prática vi que, muitas vezes, o usuário vai querer apenas "colocar um pingo no i", então seria ótimo que ao tocar em um item da lista ele já apareça no field. Este evento é lstSelect (246-261). Para alterar o texto do field, mais problemas de alocação dinâmica: na documentação há uma longa explicação dizendo que devemos modificar o handle para o texto e não o seu ponteiro – este já foi utilizado nas linhas 142 e 163. Então, vale aqui também o que já disse sobre alterar o conteúdo de um handle: travar e liberar.

7. Botão Excluir

Na função `Excluir()`, outra série de verificações. Repare bem que ao excluir realmente, temos que nos preocupar com o item que vai ficar selecionado após a exclusão, bem como o que vai ficar no topo da lista. Como o Constructor não cria constantes para as propriedades dos objetos, eu mesmo tive que criar uma para o número de itens visíveis da lista (17).

8. Configurações finais

Aumento o nível de Warning, ajusto a otimização ao máximo, mudo o nome do arquivo a gerar para Linhas.prc e atributos nome do banco para Linhas e criador para 'VCLF'. Está pronto.

Muitas das características do programa se enquadram no estilo Palm (botões em baixo, apenas uma tela, etc.), mas comparando com o MemoPad, por exemplo, a gente vê que poderia melhorar muito mais, por exemplo integrando ao Find – quando o usuário quer procurar no Palm inteiro alguma coisa – e adicionando os tradicionais copiar-colar. Como isso complica mais ainda a aplicação, considero o programa pronto aqui, pois normalmente isso quer dizer preparado para novas exigências dos usuários...

9. Na outra plataforma

Não é difícil fazer um programa Pascal que lê o LinhasDB.PDB e gera um TXT. Basta ter em mãos o formato do PDB (público) e saber o diretório de backup do Palm. O código é simples e está no Anexo III, comentado.

Conclusões

O Palm é simples e pequeno, mas eficaz, potente e fácil de usar.

A programação Palm tem enfoque diferente da programação para PC.

Mesmo um programa simples possui várias etapas e cuidados próprios para seu desenvolvimento.

No CodeWarrior, trabalha-se muito para ver poucos resultados, mas com o passar do tempo o programador acumula ferramentas para facilitar o trabalho.

Programar para Palm é árduo, mas encontramos bastante auxílio na documentação, nos comentários dos headers do sistema, nos exemplos e nos grupos de discussão na Internet.

Muito já foi descoberto, mas muito está inexplorado, aqui na PalmSys.

Anexo I

Listagem final de StarterRsc.h

```
//    Header generated by Constructor for Palm OS® 1.5
//
//    Generated for file: Starter.rsrc
//
//    THIS IS AN AUTOMATICALLY GENERATED HEADER FILE FROM CONSTRUCTOR
FOR PALM OS®;
//    - DO NOT EDIT - CHANGES MADE TO THIS FILE WILL BE LOST
//
//    Palm App Name:          "Linhas"
//
//    Palm App Version:             "1.0"



//    Resource: tFRM 1000

#define MainForm                                    1000     //(Left
Origin = 0, Top Origin = 0, Width = 160, Height = 160, Usable = 1, Mo-
dal = 0, Save Behind = 0, Help ID = 0, Menu Bar ID = 1000, Default
Button ID = 0)

#define MainAdicButton                              1001     //(Left
Origin = 2, Top Origin = 145, Width = 50, Height = 12, Usable = 1, An-
chor Left = 1, Frame = 1, Non-bold Frame = 1, Font = Standard)

#define MainAltButton                               1002     //(Left
Origin = 55, Top Origin = 145, Width = 50, Height = 12, Usable = 1,
Anchor Left = 1, Frame = 1, Non-bold Frame = 1, Font = Standard)

#define MainExcButton                               1003     //(Left
Origin = 108, Top Origin = 145, Width = 50, Height = 12, Usable = 1,
Anchor Left = 1, Frame = 1, Non-bold Frame = 1, Font = Standard)

#define Main_Field                                  1005     //(Left
Origin = 2, Top Origin = 125, Width = 156, Height = 12, Usable = 1,
Editable = 1, Underline = 1, Single Line = 1, Dynamic Size = 0, Left
Justified = 1, Max Characters = 70, Font = Standard, Auto Shift = 0,
Has Scroll Bar = 0, Numeric = 0)

#define Main_List                                   1004     //(Left
Origin = 2, Top Origin = 20, Width = 156, Usable = 1, Font = Standard,
Visible Items = 9)


//    Resource: tFRM 1100
#define SobreForm                                   1100     //(Left
Origin = 2, Top Origin = 88, Width = 156, Height = 70, Usable = 1, Mo-
dal = 1, Save Behind = 1, Help ID = 0, Menu Bar ID = 0, Default Button
ID = 0)
#define SobreUnnamed1103Button                      1103     //(Left
Origin = 62, Top Origin = 52, Width = 36, Height = 12, Usable = 1, An-
chor Left = 1, Frame = 1, Non-bold Frame = 1, Font = Standard)
```

```
#define SobreUnnamed1101Label                       1101     //(Left
Origin = 5, Top Origin = 20, Usable = 1, Font = Standard)

#define SobreUnnamed1102Label                       1102     //(Left
Origin = 5, Top Origin = 35, Usable = 1, Font = Standard)


//    Resource: Talt 1100
#define ErroAlert                                   1100
#define ErroReset                                   0

//    Resource: Talt 1000
#define ConfirmAlert                                1000
#define ConfirmSim                                  0
#define ConfirmNo                                   1

//    Resource: Talt 1200
#define InfoAlert                                   1200
#define InfoOK                                      0

//    Resource: MBAR 1000
#define MainFormMenuBar                             1000

//    Resource: MENU 1000
#define MainOptionsMenu                             1000
#define MainOptionsSobreoLinhas                     1000
```

Anexo II

Listagem final de Starter.cpp

```
1   #include <Pilot.h>
2   #include <SysEvtMgr.h>
3   #include "StarterRsc.h"
4
5   /***********************************************************
6    *
7    *   Constantes
8    *
9    ***********************************************************/
10
11  #define BCREATOR   'VCLF'
12  #define BNOME      "LinhasDB"
13  #define BTIPO      'DATA'
14  #define MAXWIDTH   145
15  #define ROM20      0x02003000   // codigo da ROM 2.0
16  #define TAMSTR     70
17  #define VIS_ITEMS  9
18
19  /***********************************************************
20   *
21   *   Variaveis Globais
22   *
23   ***********************************************************/
24
25  FormPtr pform;
26  ListPtr plist;
27  FieldPtr pfield;
28
29  class TBanco {
30     private:
31        DmOpenRef ref;
32        VoidHand hreg;
33
34     public:
35        void    Abre();
36        void    Fecha();
37        char*   Linha(Word index);
38        Boolean RegAdic(char* nova);
39        void    RegChange(Word index, char* nova);
40        void    RegExclui(Word index);
41        Boolean RegFind(char* s);
42        void    RegLib();
43        Word    RegNro();
44  } Banco;
45
46  /***********************************************************
47   *
48   *   Prototipos e Macros-Funcao
49   *
50   ***********************************************************/
51
52  void    Adicionar();
53  void    Alterar();
54  void    AtualizaList(UInt n, RectanglePtr r, CharPtr *items);
```

```
 55  void    Excluir();
 56  void    Fatal(char* m1 = "", char* m2 = "", char* m3 = "");
 57  Boolean FormLoadHandleEvent(EventPtr eventP);
 58  void    Info(char* m1, char* m2 = "", char* m3 = "");
 59  Boolean MainFormHandleEvent(EventPtr eventP);
 60  void    MainFormInit();
 61  void*   ObjectPointer(Word ID);
 62  Boolean VersaoIncompativel();
 63
 64  /************************************************************
 65   *
 66   *   Codigo das Rotinas
 67   *
 68   ************************************************************/
 69
 70  void TBanco::Abre() {
 71     LocalID id = DmFindDatabase(0, BNOME);
 72     if (!id)
 73        if (! DmCreateDatabase(0, BNOME, BCREATOR, BTIPO, false)) {
 74           id = DmFindDatabase(0, BNOME);
 75        // atributo de backup
 76           UInt attrBackup = dmHdrAttrBackup;
 77           DmSetDatabaseInfo(0, id, NULL, &attrBackup,
 78              NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
 79        }
 80
 81     if (!id) Fatal("Impossível abrir o banco de dados");
 82
 83     if (! (Banco.ref = DmOpenDatabase(0, id, dmModeReadWrite)))
 84        Fatal();
 85  }
 86
 87  void TBanco::Fecha() {
 88     DmCloseDatabase (Banco.ref);
 89  }
 90
 91  char* TBanco::Linha(Word index) {
 92     hreg = DmQueryRecord (ref, index);
 93     return (char*) MemHandleLock(hreg);
 94  }
 95
 96  Boolean TBanco::RegAdic(char* nova) {
 97     Word i;
 98
 99     if (RegFind(nova)) return false;
100
101     i = dmMaxRecordIndex;
102     if (! (hreg = DmNewRecord (ref, &i, TAMSTR+1)))
103        Fatal("Erro ao criar registro");
104     DmReleaseRecord (ref, i, true);
105
106     RegChange(i, nova);
107
108     return true;
109  }
110
111  void TBanco::RegChange(Word index, char* nova) {
112     if (DmWrite(Linha(index), 0, nova, StrLen(nova)+1))
113        Fatal("Erro ao gravar em registro");
114     RegLib();
115  }
```

```
116
117  void TBanco::RegExclui(Word index) {
118     if (DmRemoveRecord (ref, index))
119        Fatal("Erro ao excluir registro");
120  }
121
122  Boolean TBanco::RegFind(char* s) {
123     Boolean existe;
124
125     for (Word i = 0; i < RegNro(); i++) {
126        existe = !StrCompare(Linha(i), s);
127        RegLib();
128        if (existe) return true;
129     }
130     return false;
131  }
132
133  void TBanco::RegLib() {
134     MemHandleUnlock(hreg);
135  }
136
137  Word TBanco::RegNro() {
138     return DmNumRecords(ref);
139  }
140
141  void Adicionar() {
142     char* text = FldGetTextPtr (pfield);
143
144     if (text && StrLen(text))
145        if (FntCharsWidth (text, StrLen(text)) > MAXWIDTH)
146           Info("O texto digitado ultrapassa a largura da lista");
147        else if (Banco.RegAdic(text)) {
148           LstSetSelection(plist, plist->numItems);
149           plist->numItems++;
150           LstSetTopItem(plist, plist->numItems);
151           LstEraseList(plist); LstDrawList(plist);
152        }
153        else Info("Já existe o item \"", text, "\"");
154     else Info("Campo vazio");
155  }
156
157  void Alterar() {
158     if (plist->currentItem == noListSelection) {
159        Info("Você deve primeiro selecionar um item");
160        return;
161     }
162
163     char* text = FldGetTextPtr (pfield);
164
165     if (FntCharsWidth (text, StrLen(text)) > MAXWIDTH)
166        Info("O texto digitado ultrapassa a largura da lista");
167     else if (Banco.RegFind(text))
168        Info("Já existe o item \"", text, "\"");
169     else {
170        Banco.RegChange(plist->currentItem, text);
171        LstEraseList(plist); LstDrawList(plist);
172        FldSetSelection(pfield, 0, StrLen(text));
173     }
174  }
175
176  void AtualizaList(UInt n, RectanglePtr r, CharPtr *items) {
```

```
177      char* p = Banco.Linha(n);
178      WinEraseRectangle(r, 0);
179      WinDrawChars(p, StrLen(p), r->topLeft.x, r->topLeft.y);
180      Banco.RegLib();
181   }
182
183   void Excluir() {
184      if (! plist->numItems)
185         Info("O banco está vazio");
186      else if (plist->currentItem == noListSelection)
187         Info("Você deve primeiro selecionar um item");
188      else {
189         Banco.RegExclui(plist->currentItem);
190         if (plist->currentItem == --plist->numItems)
191            plist->currentItem--;
192         if ((int) (plist->currentItem - VIS_ITEMS + 1) <= 0)
193            plist->topItem = 0;
194         else plist->topItem = plist->currentItem - VIS_ITEMS + 1;
195         LstEraseList(plist); LstDrawList(plist);
196      }
197   }
198
199   void Fatal(char* m1, char* m2, char* m3) {
200      if (StrCompare(m1, ""))
201         FrmCustomAlert(ErroAlert, m1, m2, m3);
202      else FrmCustomAlert(ErroAlert, "Erro desconhecido", "", "");
203      SysReset();
204   }
205
206   Boolean FormLoadHandleEvent(EventPtr eventP)
207   {
208      if (eventP->eType != frmLoadEvent) return false;
209
210      pform = FrmInitForm(eventP->data.frmLoad.formID);
211      FrmSetActiveForm(pform);
212      FrmSetEventHandler(pform, MainFormHandleEvent);
213
214      return true;
215   }
216
217   void Info(char* m1, char* m2, char* m3) {
218      FrmCustomAlert(InfoAlert, m1, m2, m3);
219   }
220
221   Boolean MainFormHandleEvent(EventPtr eventP)
222   {
223      switch (eventP->eType) {
224         case ctlSelectEvent:
225            switch (eventP->data.ctlSelect.controlID) {
226               case MainAdicButton:
227                  Adicionar();
228               break;
229
230               case MainAltButton:
231                  Alterar();
232               break;
233
234               case MainExcButton:
235                  Excluir();
236               break;
237            }
```

```
238         return true;
239
240         case menuEvent:
241            FormPtr pForm = FrmInitForm (SobreForm);
242            FrmDoDialog (pForm);
243            FrmDeleteForm (pForm);
244         return true;
245
246         case lstSelectEvent:
247            char* text = Banco.Linha(plist->currentItem);
248
249            Handle h = FldGetTextHandle(pfield);
250            if (!h)
251               if (! (h = (Handle) MemHandleNew (TAMSTR+1)))
252                  Fatal("Erro de alocação");
253
254            StrCopy((char*) MemHandleLock(h), text);
255            MemHandleUnlock(h);
256            FldSetText(pfield, h, 0, StrLen(text));
257            FldDrawField(pfield);
258
259            FldSetSelection(pfield, 0, StrLen(text));
260            Banco.RegLib();
261         return true;
262
263         case frmOpenEvent:
264            MainFormInit();
265         return true;
266      }
267
268      return false;
269   }
270
271   void MainFormInit() {
272      plist = (ListPtr) ObjectPointer(Main_List);
273      plist->numItems = Banco.RegNro();
274      plist->currentItem = noListSelection;
275      LstSetDrawFunction(plist, AtualizaList);
276
277      pfield = (FieldPtr) ObjectPointer(Main_Field);
278      FrmSetFocus(pform, FrmGetObjectIndex(pform, Main_Field));
279
280      FrmDrawForm (pform);
281   }
282
283   void* ObjectPointer(Word ID) {
284      return FrmGetObjectPtr (pform, FrmGetObjectIndex(pform, ID));
285   }
286
287   DWord PilotMain(Word cmd, Ptr cmdPBP, Word launchFlags)
288   {
289      if (cmd != sysAppLaunchCmdNormalLaunch || VersaoIncompativel())
290         return 0;
291
292      Banco.Abre();
293
294      FrmGotoForm(MainForm);
295
296      Word error;
297      EventType event;
298
```

```
299      do {
300         EvtGetEvent(&event, evtWaitForever);
301
302         if (! SysHandleEvent(&event))
303            if (! MenuHandleEvent(0, &event, &error))
304               if (! FormLoadHandleEvent(&event))
305                  FrmDispatchEvent(&event);
306
307      } while (event.eType != appStopEvent);
308
309      Banco.Fecha();
310
311      return 0;
312   }
313
314   Boolean VersaoIncompativel() {
315      DWord aux;
316      FtrGet(sysFtrCreator, sysFtrNumROMVersion, &aux);
317      if (aux >= ROM20) return false;
318      Info("Sistema Incompatível. Esta aplicação roda apenas no Sistema
      2.0 ou superior.");
319      return true;
320   }
```

Anexo III

Listagem do leitor das linhas para MS-DOS

```
Program linhas;
uses dos;
var
   f: file of char;   {entrada}
   ch: char;
   g: text;            {saída}
   N: word;            {número de registros}
   numerar,
   gravar: boolean;

{auxiliares}
   s: string;
   i: byte;

type str2 = string[2];

Function Zero(x: Word): str2;
var aux: str2;
begin
   str(x:2, aux);
   if aux[1] = ' ' then
      aux[1] := '0';
   Zero := aux;
end;

Procedure Cabecalho;
var
   dia, mes, ano, semana,
   hora, min, seg, cent: word;
begin
   writeln(g, 'Lista de Linhas');
   writeln(g, '==================');
   writeln(g);
   GetDate(ano, mes, dia, semana);
   writeln(g, 'DATA: ', Zero(dia), '/', Zero(mes), '/', ano);
   GetTime(hora, min, seg, cent);
   writeln(g, 'HORA: ', Zero(hora), ':', Zero(min), ':', Zero(seg));
   writeln(g);
end;

Procedure Msg(s: string);
begin
   writeln(s);
   halt(0)
end;

begin
   {/N para numerar}
   if (paramcount = 1) and (paramstr(1) = '/N') then
      numerar := true;

   FileMode := 0; {somente leitura}
   writeln('Leitor de linhas');
   writeln('==================');
   write('Seu diretório de backup: ');
   readln(s);
```

```
   assign(f, s + '\LinhasDB.PDB');
   {$I-} reset(f); {$I+}
   if ioresult <> 0 then
      Msg('Banco de dados não encontrado');

   {PDB = FileHeader + N * RecordHeader + 2 + N * Record}
   {FILESIZE = 78 + 8N + 2 + 71N}
   {79 * N = FILESIZE - 80}
   N := (filesize(f) - 80) div 79;
   if (filesize(f) - 80) mod 79 <> 0 then
      Msg('Banco corrompido');
   seek(f, 78 + 8*N + 2);  {primeiro registro}

   writeln(N, ' registros.'#10);
   write('Arquivo de saída (SERÁ SOBRESCRITO, SE EXISTIR): ');
   readln(s);
   if s = '' then
      Msg('Cancelado.');
   assign(g, s);  rewrite(g);

   Cabecalho;

   for N := 1 to N do                {processar cada registro}
      begin
         gravar := true;

         if numerar then
            write(g, N:5, #9);       {número com 5 caracteres + TAB}

         for i := 1 to 71 do         {ler sempre 71 bytes}
            begin
               read(f, ch);

               if gravar then
                  if (ch = #00) then
                     begin
                        writeln(g);     {no #00 saltar uma linha}
                        gravar := false {e parar de gravar}
                     end
                  else write(g, ch)
            end
      end;

   close(f);  close(g)
end.
```

Bibliografia

Palm Programming
Glenn Bachmann
SAMS

Palm (TM) Programming in 24 Hours
Gavin Maxwell
SAMS

Palm OS (TM) Proggraming for Dummies
Liz O'Hara & John Schettino
IDG Books Worldwide, Inc.

Na Internet…

www.palmsys.com.br
Página oficial da empresa; histórico do Palm, versão do Emulator.

www.palmbr.com e links
Excelente página em português com conteúdo Palm.

www.palmbrasil.com.br e links
Conteúdo e eventos Palm no Brasil.

http://www.developer.com/directories/pages/roadcoders/pdb.html
Estrutura interna completa dos arquivos PDB, utilizada na empresa.

www.nicholson.com/rhn/pilot/pdb.txt
Formato PDB. Só há um erro: attributes é Word.

www.metrowerks.com
Página oficial da Metrowerks