Programação de shell no Windows

CAPÍTULO

16

NESTE CAPÍTULO

- Um componente de ícone da bandeja de notificação
- Barras de ferramentas do desktop da aplicação
- Vínculos do shell
- Extensões do shell

Lançada no Windows 95, o shell do Windows também tem suporte em todas as versões seguintes do Windows (NT 3.51 e superior, 98, 2000, Me e XP). Parente distante do Program Manager (gerenciador de programas), o shell do Windows inclui alguns grandes recursos para estender o shell de modo a atender suas necessidades. O problema é que muitos desses poderosos recursos extensíveis são temas de desenvolvimento do Win32 mal documentados. Este capítulo tem como objetivo dar informações e exemplos de que você precisa para ter acesso aos recursos do shell, como os ícones da bandeja de notificação, barras de ferramentas do desktop da aplicação, vínculos do shell e extensões do shell.

Um componente de ícone da bandeja de notificação

Esta seção ilustra uma técnica para encapsular de modo tranqüilo um ícone da bandeja de notificação do shell do Windows em um componente do Delphi. À medida que constrói o componente – chamado `TTrayNotifyIcon` –, você aprenderá os requisitos da API para a criação de um ícone da bandeja de notificação, bem como lidar com os problemas mais difíceis inerentes ao trabalho de incorporar toda a funcionalidade do ícone dentro do componente. Se você não tem a menor idéia do que seja um ícone da bandeja de notificação, trata-se de um daqueles pequenos ícones que aparecem no canto inferior direito da barra de tarefas do sistema Windows (e aqui estou presumindo que sua barra de tarefas esteja alinhada à parte inferior da tela), mostrada na Figura 16.1.

Figura 16.1 Ícones da bandeja de notificação.

A API

Acredite se quiser, mas somente uma chamada da API está envolvida na criação, modificação e remoção dos ícones da bandeja de notificação. A função se chama `Shell_NotifyIcon( )`. Esta e outras funções relacionadas ao shell do Windows estão contidas na unidade `ShellAPI`. `Shell_NotifyIcon( )` é definida da seguinte maneira:

```
function Shell_NotifyIcon(dwMessage: DWORD; lpData:
    PNotifyIconData): BOOL; stdcall;
```

O parâmetro `dwMessage` descreve a ação a ser executada pelo ícone, que pode ser qualquer um dos valores mostrados na Tabela 16.1.

Tabela 16.1 Valores do parâmetro `dwMessage`

Constante	*Valor*	*Significado*
`NIM_ADD`	`0`	Adiciona um ícone à bandeja de notificação.
`NIM_MODIFY`	`1`	Modifica as propriedades de um ícone existente.
`NIM_DELETE`	`2`	Remove um ícone da bandeja de notificação.

O parâmetro `lpData` é um ponteiro para um registro `TNotifyIconData`. Esse registro é definido da seguinte maneira:

```
type
  TNotifyIconData = record
```

```
    cbSize: DWORD;
    Wnd: HWND;
    uID: UINT;
    uFlags: UINT;
    uCallbackMessage: UINT;
    hIcon: HICON;
    szTip: array [0..63] of AnsiChar;
  end;
```

O campo `cbSize` armazena o tamanho do registro e deve ser inicializado como `SizeOf(TNotifyIconData)`.

`Wnd` é a alça da janela à qual as mensagens de "callback" da bandeja de notificação devem ser enviadas. (*Callback* está entre aspas, pois não se trata de uma callback no sentido estrito da palavra; no entanto, a documentação do Win32 usa essa terminologia para mensagens enviadas para uma janela, não para um ícone da bandeja de notificação.)

`uID` é um número de ID exclusivo definido pelo programador. Se você tiver uma aplicação com diversos ícones, precisará identificar cada um deles colocando um número diferente nesse campo.

`uFlags` descreve qual dos campos do registro `TNotifyIconData` deve ser considerado ativo pela função `Shell_NotifyIcon( )` e, por essa razão, qual das propriedades do ícone será afetada pela ação especificada pelo parâmetro `dwMessage`. Esse parâmetro pode ser qualquer combinação de flags mostrada na Tabela 16.2.

Tabela 16.2 Possíveis flags a serem incluídos em `uFlags`

Constante	*Valor*	*Significado*
`NIF_MESSAGE`	0	O campo `uCallbackMessage` está ativo.
`NIF_ICON`	2	O campo `hIcon` está ativo.
`NIF_TIP`	4	O campo `szTip` está ativo.

`uCallbackMessage` contém o valor da mensagem do Windows a ser enviada para a janela identificada pelo campo `Wnd`. Geralmente, o valor desse campo é obtido pela chamada de `RegisterWindowMessage( )` ou pelo uso de um deslocamento de `WM_USER`. O `lParam` dessa mensagem terá o mesmo valor que o campo `uID`, e `wParam` armazenará a mensagem de mouse gerada pelo ícone de notificação.

`hIcon` identifica a alça para o ícone que será colocado na bandeja de notificação.

`szTip` armazena uma string terminada em nulo, que aparecerá na janela de dica exibida quando o ponteiro do mouse for mantido sobre o ícone de notificação.

O componente `TTrayNotifyIcon` encapsula `Shell_NotifyIcon( )` em um método chamado `SendTrayMessage( )`, que é mostrado a seguir:

```
procedure TTrayNotifyIcon.SendTrayMessage(Msg: DWORD; Flags: UINT);
{ Este método envolve a chamada para o Shell_NotifyIcon da API }
begin
  { Preenche o registro com os valores apropriados }
  with Tnd do
  begin
    cbSize := SizeOf(Tnd);
    StrPLCopy(szTip, PChar(FHint), SizeOf(szTip));
    uFlags := Flags;
    uID := UINT(Self);
    Wnd := IconMgr.HWindow;
    uCallbackMessage := Tray_Callback;
    hIcon  := ActiveIconHandle;
  end;
```

```
  Shell_NotifyIcon(Msg, @Tnd);
end;
```

Nesse método, `szTip` é copiado de um campo de string privado, chamado `FHint`.

`uID` é usado para manter uma referência a `Self`. Como esses dados serão incluídos em mensagens subseqüentes da bandeja de notificação, será fácil correlacionar as mensagens da bandeja de notificação para vários ícones aos componentes individuais.

`Wnd` recebe o valor de `IconMgr.Hwindow`. `IconMgr` é uma variável global do tipo `TIconMgr`. Você verá a implementação desse objeto daqui a pouco, mas, por enquanto, você só precisa saber que é através desse componente que todas as mensagens da bandeja de notificação serão enviadas.

`uCallbackMessage` é atribuído com base em `DDGM_TRAYICON`. `DDGM_TRAYICON` obtém seu valor da função `RegisterWindowMessage( )` da API. Isso garante que `DDGM_TRAYICON` seja um ID de mensagem exclusivo, reconhecido por todo o sistema. O código mostrado a seguir executa essa tarefa:

```
const
{ String para identificar a mensagem da janela registrada }
  TrayMsgStr = 'DDG.TrayNotifyIconMsg';

initialization
  { Obtém um ID de mensagem exclusivo para a callback da bandeja }
  DDGM_TRAYICON := RegisterWindowMessage(TrayMsgStr);
```

`hIcon` assume o valor de retorno fornecido pelo método `ActiveIconHandle( )`. Esse método retorna a alça do ícone atualmente selecionado na propriedade `Icon`.

Manipulando mensagens

Já dissemos que todas as mensagens da bandeja de notificação são enviadas para uma janela mantida pelo objeto `IconMgr` global. Este objeto é construído e liberado nas seções `initialization` e `finalization` da unidade do componente, conforme mostrado a seguir:

```
initialization
  { Obtém o ID de mensagem exclusivo para a callback da bandeja }
  DDGM_TRAYICON := RegisterWindowMessage(TrayMsgStr);
  IconMgr := TIconManager.Create;
finalization
  IconMgr.Free;
```

Esse é um objeto muito pequeno. Veja sua definição a seguir:

```
type
  TIconManager = class
  private
    FHWindow: HWnd;
    procedure TrayWndProc(var Message: TMessage);
  public
    constructor Create;
    destructor Destroy; override;
    property HWindow: HWnd read FHWindow write FHWindow;
  end;
```

A janela para a qual as mensagens da bandeja de notificação serão enviadas é criada no construtor desse objeto usando a função `AllocateHWnd( )`:

```
constructor TIconManager.Create;
begin
  FHWindow := AllocateHWnd(TrayWndProc);
end;
```

O método TrayWndProc( ) serve como procedimento de janela para a janela criada no construtor. Voltaremos a falar sobre esse método mais adiante.

Ícones e dicas

O método mais objetivo para expor ícones e dicas para o usuário final do componente é através das propriedades. Além disso, a criação de uma propriedade Icon do tipo TIcon significa que ela pode automaticamente tirar proveito do editor de propriedades do Delphi para os ícones, o que é um recurso muito interessante. Como o ícone da bandeja é visível inclusive durante o projeto, você precisa se certificar de que o ícone e a dica podem mudar dinamicamente. Fazer isso não implica, como se pode pensar a princípio, muito trabalho extra; basta se certificar de que o método SendTrayMessage( ) é chamado (usando a mensagem NIM_MODIFY) no método write das propriedades Hint e Icon.

Veja, a seguir, os métodos write para essas propriedades:

```
procedure TTrayNotifyIcon.SetIcon(Value: TIcon);
{ Método write para a propriedade Icon. }
begin
  FIcon.Assign(Value);  // define novo ícone
  if FIconVisible then
    { Muda o ícone na bandeja de notificação }
    SendTrayMessage(NIM_MODIFY, NIF_ICON);
end;

procedure TTrayNotifyIcon.SetHint(Value: String);
{ Define o método para a propriedade Hint }
begin
  if FHint < > Value then
  begin
    FHint := Value;
    if FIconVisible then
      { Muda dica no ícone da bandeja de notificação }
      SendTrayMessage(NIM_MODIFY, NIF_TIP);
  end;
end;
```

Cliques do mouse

Uma das partes mais desafiadoras desse componente é garantir que os cliques do mouse sejam manipulados de modo apropriado. Você pode ter percebido que muitos ícones da bandeja de notificação executam três ações diferentes devido a cliques do mouse:

- Abre uma janela com um clique único
- Abre uma janela diferente (geralmente uma folha de propriedades) com um clique duplo
- Chama um menu local com um clique no botão direito

O desafio está na criação de um evento que represente o clique duplo sem acionar também o evento de clique único.

Em termos de mensagem do Windows, quando o usuário dá um clique duplo no botão esquerdo do mouse, a janela selecionada recebe tanto a mensagem WM_LBUTTONDOWN como a mensagem WM_LBUTTONDBLCLK. Para permitir que uma mensagem de clique duplo seja processada independentemente de um clique único, é preciso um mecanismo para retardar a manipulação da mensagem de clique único o tempo necessário para garantir que uma mensagem de clique duplo não esteja acessível.

O intervalo de tempo a esperar antes que você possa ter certeza de que uma mensagem WM_LBUTTONDBLCLK não esteja vindo depois de uma mensagem M_LBUTTONDOWN é bastante fácil de determinar. A função GetDoubleClickTime( ) da API, que não utiliza parâmetros, retorna o maior inter-

valo de tempo possível (em milissegundos) que o Control Panel (painel de controle) permitirá entre os dois cliques de um clique duplo. A escolha óbvia para um mecanismo permitir que você aguarde o número de milissegundos especificado por `GetDoubleClickTime( )`, garantindo que um clique duplo não venha depois de um clique, é o componente `TTimer`. Por essa razão, um componente `TTimer` é criado e inicializado no construtor do componente `TTrayNotifyIcon`, com o seguinte código:

```
FTimer := TTimer.Create(Self);
with FTimer do
begin
  Enabled := False;
  Interval := GetDoubleClickTime;
  OnTimer := OnButtonTimer;
end;
```

`OnButtonTimer( )` é um método que será chamado quando o intervalo do timer expirar. Vamos mostrar esse método daqui a pouco.

Já mencionamos que as mensagens da bandeja de notificação são filtradas através do método `TrayWndProc( )` de `IconMgr`. Agora chegou a hora de você conhecer esse método, e aqui está ele:

```
procedure TIconManager.TrayWndProc(var Message: TMessage);
{ Isto nos permite manipular as mensagens de callback da bandeja }
{ de dentro do contexto do componente. }
var
  Pt: TPoint;
  TheIcon: TTrayNotifyIcon;
begin
  with Message do
  begin
    { caso seja a mensagem de callback da bandeja }
    if (Msg = DDGM_TRAYICON) then
    begin
      TheIcon := TTrayNotifyIcon(WParam);
      case lParam of
        { Ativa o timer no primeiro pressionamento do mouse. }
        { OnClick será acionado pelo método OnTimer, desde que }
        { o clique duplo não tenha ocorrido. }
        WM_LBUTTONDOWN: TheIcon.FTimer.Enabled := True;
        { Não define um flag de clique no clique duplo. Isso eliminará }
        { o clique único. }
        WM_LBUTTONDBLCLK:
          begin
            TheIcon.FNoShowClick := True;
            if Assigned(TheIcon.FOnDblClick) then TheIcon.FOnDblClick(Self);
          end;
        WM_RBUTTONDOWN:
          begin
            if Assigned(TheIcon.FPopupMenu) then
            begin
              { Chamada para SetForegroundWindow é exigida pela API }
              SetForegroundWindow(IconMgr.HWindow);
              { Abre o menu local na posição do cursor. }
              GetCursorPos(Pt);
              TheIcon.FPopupMenu.Popup(Pt.X, Pt.Y);
              { Postagem da mensagem exigida pela API para forçar troca de tarefa }
              PostMessage(IconMgr.HWindow, WM_USER, 0, 0);
            end;
          end;
      end;
```

```
    end
    else
      { Se não for uma mensagem de callback da bandeja, chama DefWindowProc }
      Result := DefWindowProc(FHWindow, Msg, wParam, lParam);
  end;
end;
```

O que faz isso tudo funcionar é que a mensagem de clique único se limita a ativar o timer, enquanto a mensagem de clique duplo define um flag para indicar que o clique duplo ocorreu antes do acionamento do seu evento `OnDblClick`. O clique com o botão direito, a propósito, chama o menu instantâneo dado pela propriedade `PopupMenu` do componente. Agora dê uma olhada no método `OnButtonTimer( )`:

```
procedure TTrayNotifyIcon.OnButtonTimer(Sender: TObject);
begin
  { Desativa o timer, pois só queremos que ele seja acionado uma vez. }
  FTimer.Enabled := False;
  { Se um clique duplo não tiver ocorrido, o clique único é acionado. }
  if (not FNoShowClick) and Assigned(FOnClick) then
    FOnClick(Self);
  FNoShowClick := False;   // reinicia flag
end;
```

Esse método primeiro desativa o timer, para garantir que o evento só será acionado uma vez a cada clique no mouse. Em seguida, o método verifica o status do flag `FNoShowClick`. Lembre-se de que esse flag será definido pela mensagem de clique duplo no método `OwnerWndProc( )`. Por essa razão, o evento `OnClick` só será acionado quando `OnDblClk` não o for.

Ocultando a aplicação

Outro aspecto das aplicações da bandeja de notificação é que elas não aparecem como botões na barra de tarefas do sistema. Para fornecer essa funcionalidade, o componente `TTrayNotifyIcon` expõe uma propriedade `HideTask`, que permite que o usuário decida se a aplicação deve ser visível na barra de tarefas. O método `write` para essa propriedade é mostrado no código a seguir. A linha de código que o trabalho executa é a chamada para o procedimento `ShowWindow( )` da API, que passa a propriedade `Handle` de `Application` e uma constante, para indicar se a aplicação tem que ser mostrada normalmente ou ocultada. Veja o código a seguir:

```
procedure TTrayNotifyIcon.SetHideTask(Value: Boolean);
{ Método write da propriedade HideTask }
const
  { Flags para exibir a aplicação normalmente ou ocultá-la }
  ShowArray: array[Boolean] of integer = (sw_ShowNormal, sw_Hide);
begin
  if FHideTask < > Value then begin
    FHideTask := Value;
    { Não faz nada no modo de projeto }
    if not (csDesigning in ComponentState) then
      ShowWindow(Application.Handle, ShowArray[FHideTask]);
  end;
end;
```

A Listagem 16.1 mostra a unidade `TrayIcon.pas` , que contém o código-fonte completo do componente `TTrayNotifyIcon`.

Listagem 16.1 `TrayIcon.pas` – código fonte do componente `TTrayNotifyIcon`

```
unit TrayIcon;

interface

uses Windows, SysUtils, Messages, ShellAPI, Classes, Graphics, Forms, Menus,
  StdCtrls, ExtCtrls;

type
  ENotifyIconError = class(Exception);

  TTrayNotifyIcon = class(TComponent)

  private
    FDefaultIcon: THandle;
    FIcon: TIcon;
    FHideTask: Boolean;
    FHint: string;
    FIconVisible: Boolean;
    FPopupMenu: TPopupMenu;
    FOnClick: TNotifyEvent;
    FOnDblClick: TNotifyEvent;
    FNoShowClick: Boolean;
    FTimer: TTimer;
    Tnd: TNotifyIconData;
    procedure SetIcon(Value: TIcon);
    procedure SetHideTask(Value: Boolean);
    procedure SetHint(Value: string);
    procedure SetIconVisible(Value: Boolean);
    procedure SetPopupMenu(Value: TPopupMenu);
    procedure SendTrayMessage(Msg: DWORD; Flags: UINT);
    function ActiveIconHandle: THandle;
    procedure OnButtonTimer(Sender: TObject);
  protected
    procedure Loaded; override;
    procedure LoadDefaultIcon; virtual;
    procedure Notification(AComponent: TComponent;
      Operation: TOperation); override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property Icon: TIcon read FIcon write SetIcon;
    property HideTask: Boolean read FHideTask write SetHideTask default False;
    property Hint: String read FHint write SetHint;
    property IconVisible: Boolean read FIconVisible write SetIconVisible
      default False;
    property PopupMenu: TPopupMenu read FPopupMenu write SetPopupMenu;
    property OnClick: TNotifyEvent read FOnClick write FOnClick;
    property OnDblClick: TNotifyEvent read FOnDblClick write FOnDblClick;
  end;

implementation

{ TIconManager }
{ Esta classe cria uma janela oculta que manipula e direciona }
{ as mensagens de ícone da bandeja }
type
```

Listagem 16.1 Continuação

```
  TIconManager = class
  private
    FHWindow: HWnd;
    procedure TrayWndProc(var Message: TMessage);
  public
    constructor Create;
    destructor Destroy; override;
    property HWindow: HWnd read FHWindow write FHWindow;
  end;

var
  IconMgr: TIconManager;
  DDGM_TRAYICON: Integer;

constructor TIconManager.Create;
begin
  FHWindow := AllocateHWnd(TrayWndProc);
end;

destructor TIconManager.Destroy;
begin
  if FHWindow < > 0 then DeallocateHWnd(FHWindow);
  inherited Destroy;
end;

procedure TIconManager.TrayWndProc(var Message: TMessage);
{ Isto nos permite manipular todas as mensagens de callback da bandeja }
{ de dentro do contexto do componente. }
var
  Pt: TPoint;
  TheIcon: TTrayNotifyIcon;
begin
  with Message do
  begin
    { se for a mensagem de callback da bandeja }
    if (Msg = DDGM_TRAYICON) then
    begin
      TheIcon := TTrayNotifyIcon(WParam);
      case lParam of
        { Ativa o timer no primeiro pressionamento do mouse. }
        { OnClick será acionado pelo método OnTimer, desde que o }
        { clique duplo não tenha ocorrido. }
        WM_LBUTTONDOWN: TheIcon.FTimer.Enabled := True;
        { Não define o flag de clique no clique duplo. Isso eliminará }
        { o clique único. }
        WM_LBUTTONDBLCLK:
          begin
            TheIcon.FNoShowClick := True;
            if Assigned(TheIcon.FOnDblClick) then TheIcon.FOnDblClick(Self);
          end;
        WM_RBUTTONDOWN:
          begin
            if Assigned(TheIcon.FPopupMenu) then
            begin
              { Chamada para SetForegroundWindow é exigida pela API }
              SetForegroundWindow(IconMgr.HWindow);
              { Abre menu local na posição do cursor. }
```

Listagem 16.1 Continuação

```
              GetCursorPos(Pt);
              TheIcon.FPopupMenu.Popup(Pt.X, Pt.Y);
              { Postagem da mensagem exigida pela API para forçar a troca de tarefa }
              PostMessage(IconMgr.HWindow, WM_USER, 0, 0);
            end;
          end;
      end;
    end
    else
      { Se não for uma mensagem de callback da bandeja, chama DefWindowProc }
      Result := DefWindowProc(FHWindow, Msg, wParam, lParam);
  end;
end;

{ TTrayNotifyIcon }

constructor TTrayNotifyIcon.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FIcon := TIcon.Create;
  FTimer := TTimer.Create(Self);
  with FTimer do
  begin
    Enabled := False;
    Interval := GetDoubleClickTime;
    OnTimer := OnButtonTimer;
  end;
  { Mantém o ícone padrão da janela... }
  LoadDefaultIcon;
end;

destructor TTrayNotifyIcon.Destroy;
begin
  if FIconVisible then SetIconVisible(False);    //ícone de destruição
  FIcon.Free;                                    // dados livres
  FTimer.Free;
  inherited Destroy;
end;

function TTrayNotifyIcon.ActiveIconHandle: THandle;
{ Retorna alça do ícone ativo }
begin
  { Se nenhum ícone for carregado, retorna ícone default }
  if (FIcon.Handle < > 0) then
    Result := FIcon.Handle
  else
    Result := FDefaultIcon;
end;

procedure TTrayNotifyIcon.LoadDefaultIcon;
{ Carrega o ícone de janela padrão para ficar por perto. }
{ Isso permitirá que o componente use o ícone do logotipo }
{ do Windows como o padrão quando nenhum ícone estiver }
{ selecionado na propriedade Icon. }
begin
  FDefaultIcon := LoadIcon(0, IDI_WINLOGO);
end;
```

Listagem 16.1 Continuação

```
procedure TTrayNotifyIcon.Loaded;
{ Chamado depois que o componente é carregado do fluxo }
begin
  inherited Loaded;
  { Se o ícone deve ser visível, crie-o. }
  if FIconVisible then
    SendTrayMessage(NIM_ADD, NIF_MESSAGE or NIF_ICON or NIF_TIP);
end;

procedure TTrayNotifyIcon.Notification(AComponent: TComponent;
  Operation: TOperation);
begin
  inherited Notification(AComponent, Operation);
  if (Operation = opRemove) and (AComponent = PopupMenu) then
    PopupMenu := nil;
end;

procedure TTrayNotifyIcon.OnButtonTimer(Sender: TObject);
{ Timer usado para monitorar o tempo entre dois cliques de um }
{ clique duplo. Isso retarda o primeiro clique o tempo necessário }
{ para garantir que não ocorreu um clique duplo. O objetivo }
{ de toda essa ginástica é permitir que o componente receba }
{ OnClicks e OnDblClicks de modo independente. }
begin
  { Desativa o timer porque só queremos que ele seja acionado uma vez. }
  FTimer.Enabled := False;
  { Se não ocorreu um clique duplo, aciona o clique único. }
  if (not FNoShowClick) and Assigned(FOnClick) then
    FOnClick(Self);
  FNoShowClick := False;   // reinicia flag
end;

procedure TTrayNotifyIcon.SendTrayMessage(Msg: DWORD; Flags: UINT);
{ Este método envolve a chamada para o Shell_NotifyIcon da API}
begin
  { Preenche o registro com os valores apropriados }
  with Tnd do
  begin
    cbSize := SizeOf(Tnd);
    StrPLCopy(szTip, PChar(FHint), SizeOf(szTip));
    uFlags := Flags;
    uID := UINT(Self);
    Wnd := IconMgr.HWindow;
    uCallbackMessage := DDGM_TRAYICON;
    hIcon  := ActiveIconHandle;
  end;
  Shell_NotifyIcon(Msg, @Tnd);
end;

procedure TTrayNotifyIcon.SetHideTask(Value: Boolean);
{ Escreve métodos da propriedade HideTask }
const
  { Apresenta flags para mostrar a aplicação normalmente ou ocultá-la }
  ShowArray: array[Boolean] of integer = (sw_ShowNormal, sw_Hide);
begin
  if FHideTask < > Value then
  begin
```

Listagem 16.1 Continuação

```
    FHideTask := Value;
    { Não faz nada no modo de projeto }
    if not (csDesigning in ComponentState) then
      ShowWindow(Application.Handle, ShowArray[FHideTask]);
  end;
end;

procedure TTrayNotifyIcon.SetHint(Value: string);
{ Define o método da propriedade Hint }
begin
  if FHint < > Value then
  begin
    FHint := Value;
    if FIconVisible then
      { Muda a dica no ícone na bandeja de notificação }
      SendTrayMessage(NIM_MODIFY, NIF_TIP);
  end;
end;

procedure TTrayNotifyIcon.SetIcon(Value: TIcon);
{ Escreve o método da propriedade Icon. }
begin
  FIcon.Assign(Value);  // define novo ícone
  { Muda ícone na bandeja de notificação }
  if FIconVisible then SendTrayMessage(NIM_MODIFY, NIF_ICON);
end;

procedure TTrayNotifyIcon.SetIconVisible(Value: Boolean);
{ Escreve método da propriedade IconVisible }
const
  { Flags para incluir ou excluir um ícone na bandeja de notificação }
  MsgArray: array[Boolean] of DWORD = (NIM_DELETE, NIM_ADD);
begin
  if FIconVisible < > Value then
  begin
    FIconVisible := Value;
    { Define ícone de modo apropriado }
    SendTrayMessage(MsgArray[Value], NIF_MESSAGE or NIF_ICON or NIF_TIP);
  end;
end;

procedure TTrayNotifyIcon.SetPopupMenu(Value: TPopupMenu);
{ Método write da propriedade PopupMenu }
begin
  FPopupMenu := Value;
  if Value < > nil then Value.FreeNotification(Self);
end;

const
  { String para identificar mensagem registrada do Windows }
  TrayMsgStr = 'DDG.TrayNotifyIconMsg';

initialization
  { Obtém ID exclusivo da mensagem do Windows para a callback da bandeja }
  DDGM_TRAYICON := RegisterWindowMessage(TrayMsgStr);
  IconMgr := TIconManager.Create;
finalization
  IconMgr.Free;
end.
```

A Figura 16.2 mostra uma imagem do ícone gerada por `TTrayNotifyIcon` na bandeja de notificação.

Figura 16.2 O componente TTrayNotifyIcon em ação.

A propósito, como o ícone da bandeja é inicializado dentro do construtor do componente e como os construtores são executados durante o projeto, esse componente exibe o ícone da bandeja de notificação inclusive durante o projeto!

Exemplo de aplicação de bandeja

Para que você tenha uma compreensão mais ampla de como `TTrayNotifyIcon` funciona dentro do contexto de uma aplicação, a Figura 16.3 mostra a janela principal dessa aplicação e a Listagem 16.2 mostra o pequeno código da unidade principal dessa aplicação.

Figura 16.3 Aplicação do ícone de notificação.

Listagem 16.2 `Main.pas` – a unidade principal do exemplo de aplicação de um ícone de notificação

```
unit main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ShellAPI, TrayIcon, Menus, ComCtrls;

type
  TMainForm = class(TForm)
    pmiPopup: TPopupMenu;
    pgclPageCtl: TPageControl;
    TabSheet1: TTabSheet;
    btnClose: TButton;
    btnTerm: TButton;
    Terminate1: TMenuItem;
    Label1: TLabel;
    N1: TMenuItem;
    Propeties1: TMenuItem;
    TrayNotifyIcon1: TTrayNotifyIcon;
    procedure NotifyIcon1Click(Sender: TObject);
    procedure NotifyIcon1DblClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure btnTermClick(Sender: TObject);
    procedure btnCloseClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  end;
```

Listagem 16.2 Continuação

```
var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.NotifyIcon1Click(Sender: TObject);
begin
  ShowMessage('Single click');
end;

procedure TMainForm.NotifyIcon1DblClick(Sender: TObject);
begin
  Show;
end;

procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := caNone;
  Hide;
end;

procedure TMainForm.btnTermClick(Sender: TObject);
begin
  Application.Terminate;
end;

procedure TMainForm.btnCloseClick(Sender: TObject);
begin
  Hide;
end;

procedure TMainForm.FormCreate(Sender: TObject);
begin
  TrayNotifyIcon1.IconVisible := True;
end;

end.
```

Barras de ferramentas do desktop da aplicação

A barra de ferramentas do desktop da aplicação, também conhecida como *AppBars*, são janelas que podem ser encaixadas a uma das extremidades da sua tela. Você já conhece as AppBars, muito embora talvez não saiba disso; a barra de tarefas do shell, com a qual você provavelmente trabalha todos os dias, é um exemplo de uma AppBar. Como mostra a Figura 16.4, a barra de tarefas é apenas um pouco mais do que uma janela AppBar contendo um botão Start, uma bandeja de notificação e outros controles.

Figura 16.4 A barra de tarefas do shell.

Além de ser encaixada nas bordas da tela, AppBars podem empregar recursos do tipo barra de tarefas, como a funcionalidade de auto-ocultar e arrastar-e-soltar. No entanto, você pode se surpre-

ender com o pequeno tamanho da API (ela tem apenas uma função). Como se pode deduzir pelo seu pequeno tamanho, a API não tem muita coisa a oferecer. A função da API é mais consultiva do que funcional. Ou seja, em vez de controlar a AppBar com comandos do tipo "faça isto, faça aquilo", você interroga a AppBar com comandos do tipo "posso fazer isso, posso fazer aquilo?".

A API

Como os ícones da bandeja de notificação, AppBars só tem uma função da API com a qual você vai trabalhar – `SHAppBarMessage( )`, nesse caso. Veja a seguir como `SHAppBarMessage( )` é definida na unidade `ShellAPI`:

```
function SHAppBarMessage(dwMessage: DWORD; var pData: TAppBarData): UINT
  stdcall;
```

O primeiro parâmetro dessa função, `dwMessage`, pode conter um dos valores descritos na Tabela 16.3.

Tabela 16.3 Mensagens da AppBar

Constante	*Valor*	*Significado*
`ABM_NEW`	`$0`	Registra uma nova AppBar e especifica uma nova mensagem de callback
`ABM_REMOVE`	`$1`	Elimina o registro de uma AppBar existente
`ABM_QUERYPOS`	`$2`	Solicita uma posição e um tamanho novos de uma AppBar
`ABM_SETPOS`	`$3`	Define uma nova posição e um novo tamanho de uma AppBar
`ABM_GETSTATE`	`$4`	Obtém os estados de 'Auto-Ocultar' e 'Sempre visível' da barra de tarefas do shell
`ABM_GETTASKBARPOS`	`$5`	Obtém a posição da barra de tarefas do shell
`ABM_ACTIVATE`	`$6`	Avisa ao shell que uma nova AppBar foi criada
`ABM_GETAUTOHIDEBAR`	`$7`	Obtém a alça de uma AppBar de Auto-Ocultar encaixada em uma determinada borda da tela
`ABM_SETAUTOHIDEBAR`	`$8`	Registra uma AppBar de Auto-Ocultar em uma determinada borda da tela
`ABM_WINDOWPOSCHANGED`	`$9`	Avisa ao shell que a posição de uma AppBar foi alterada

O parâmetro `pData` de `SHAppBarMessage( )` é um registro do tipo `TAppBarData`, que é definido na `ShellAPI` da seguinte maneira:

```
type
  PAppBarData = ^TAppBarData;
  TAppBarData = record
    cbSize: DWORD;
    hWnd: HWND;
    uCallbackMessage: UINT;
    uEdge: UINT;
    rc: TRect;
    lParam: LPARAM; { específico da mensagem }
  end;
```

Nesse registro, o campo `cbSize` armazena o tamanho do registro, o campo `hWnd` armazena a alça da janela da AppBar especificada, `uCallbackMessage` armazena o valor da mensagem que será

enviada para a janela AppBar juntamente com as mensagens de notificação, rc armazena o retângulo que envolve a AppBar em questão e lParam armazena algumas informações adicionais específicas da mensagem.

> **DICA**
>
> Para obter maiores informações sobre a função SHAppBarMessage( ) da API e sobre o tipo TAppBarData, consulte a ajuda on-line do Win32.

TAppBar: o formulário da AppBar

Devido ao tamanho mínimo da API, não há nada do outro mundo no processo de encapsular uma AppBar em um formulário VCL. Esta seção explicará as técnicas usadas para envolver a API AppBar em um controle descendente de TCustomForm. Como TCustomForm é um formulário, você vai interagir com o controle como um formulário de nível superior no Form Designer, não como um componente em um formulário.

A maior parte do trabalho em uma AppBar é feita enviando um registro TAppBarData para o shell usando a função SHAppBarMessage( ) da API. O componente TAppBar mantém um registro TAppBarData interno, chamado FABD. FABD é configurado para a chamada de SendAppBarMsg( ) no construtor e nos métodos CreateWnd( ) para criar a AppBar. Em particular, o campo cbSize é inicializado, o campo uCallbackMessage é definido como um valor obtido da função RegisterWindowMessage( ) da API, e o campo hWnd é definido como a alça de janela atual do formulário. SendAppBarMessage( ) é um simples wrapper para SHAppBarMessage( ), sendo definido da seguinte maneira:

```
function TAppBar.SendAppBarMsg(Msg: DWORD): UINT;
begin
  Result := SHAppBarMessage(Msg, FABD);
end;
```

Se a AppBar for criada com sucesso, o método SetAppBarEdge( ) será chamado para definir a AppBar como sua posição inicial. Este método, por sua vez, chama o método SetAppBarPos( ), passando o flag apropriado, definido pela API, que indica a borda de tela solicitada. Como era de se imaginar, os flags ABE_TOP, ABE_BOTTOM, ABE_LEFT e ABE_RIGHT representam cada uma das bordas da tela. Isso é mostrado no trecho de código a seguir:

```
procedure TAppBar.SetAppBarPos(Edge: UINT);
begin
  if csDesigning in ComponentState then Exit;
  FABD.uEdge := Edge;       // define borda
  with FABD.rc do
  begin
    // define coordenada como tela inteira
    Top := 0;
    Left := 0;
    Right := Screen.Width;
    Bottom := Screen.Height;
    // Envia ABM_QUERYPOS para obter retângulo apropriado na margem
    SendAppBarMsg(ABM_QUERYPOS);
    // reajusta retângulo com base naquele modificado por ABM_QUERYPOS
    case Edge of
      ABE_LEFT: Right := Left + FDockedWidth;
      ABE_RIGHT: Left := Right – FDockedWidth;
      ABE_TOP: Bottom := Top + FDockedHeight;
      ABE_BOTTOM: Top := Bottom – FDockedHeight;
    end;
```

```
    // Define a posição da barra da aplicação.
    SendAppBarMsg(ABM_SETPOS);
  end;
  // Define a propriedade BoundsRect de modo que ela se adapte
  // ao retângulo passado para o sistema.
  BoundsRect := FABD.rc;
end;
```

Esse método primeiro define o campo `uEdge` de `FABD` como o valor passado através do parâmetro `Edge`. Em seguida, ele define o campo `rc` como as coordenadas da tela inteira e envia a mensagem `ABM_QUERYPOS`. Essa mensagem redefine o campo `rc` de modo que ele contenha o retângulo apropriado para a borda indicada por `uEdge`. Uma vez obtido o retângulo apropriado, `rc` é ajustado novamente de modo que tenha altura e largura razoáveis. Nesse ponto, `rc` armazena o retângulo final para a AppBar. A mensagem `ABM_SETPOS` é em seguida enviada para informar ao shell quanto ao novo retângulo, e o retângulo é definido por meio da propriedade `BoundsRect` do controle.

Já dissemos que as mensagens de notificação da AppBar serão enviadas para a janela indicada por `FABD.hWnd`, usando o identificador de mensagem armazenado em `FABD.uCallbackMessage`. Essas mensagens de notificação são manipuladas no método `WndProc( )` mostrado a seguir:

```
procedure TAppBar.WndProc(var M: TMessage);
var
  State: UINT;
  WndPos: HWnd;
begin
  if M.Msg = AppBarMsg then
  begin
    case M.WParam of
      // Enviada quando o estado 'Sempre visível' ou 'Auto-Ocultar' é alterado.
      ABN_STATECHANGE:
        begin
          // Verifica se a barra de acesso ainda é ABS_ALWAYSONTOP.
          State := SendAppBarMsg(ABM_GETSTATE);
          if ABS_ALWAYSONTOP and State = 0 then
            SetTopMost(False)
          else
            SetTopMost(True);
        end;
      // Uma aplicação de tela cheia foi iniciada ou a última
      // aplicação de tela inteira foi fechada.
      ABN_FULLSCREENAPP:
        begin
          // Define a ordem z da barra de acesso de modo apropriado.
          State := SendAppBarMsg(ABM_GETSTATE);
          if M.lParam < > 0 then begin
            if ABS_ALWAYSONTOP and State = 0 then
              SetTopMost(False)
            else
              SetTopMost(True);
          end
          else
            if State and ABS_ALWAYSONTOP < > 0 then
              SetTopMost(True);
        end;
      // Enviado quando houver algo que possa afetar a posição da AppBar.
      ABN_POSCHANGED:
        begin
          // A barra de tarefas ou outra barra de acesso
          // mudou seu tamanho ou sua posição.
```

```
          SetAppBarPos(FABD.uEdge);
        end;
    end;
  end
  else
    inherited WndProc(M);
end;
```

Este método manipula algumas mensagens de notificação que permitem que a AppBar responda às mudanças que podem ocorrer no shell enquanto a aplicação estiver sendo executada. O restante do código do componente AppBar é mostrado na Listagem 16.3.

Listagem 16.3 `AppBars.pas` – a unidade que contém a classe básica para dar suporte à AppBar

```
unit AppBars;

interface

uses Windows, Messages, SysUtils, Forms, ShellAPI, Classes, Controls;

type
  TAppBarEdge = (abeTop, abeBottom, abeLeft, abeRight);

  EAppBarError = class(Exception);

  TAppBar = class(TCustomForm)
  private
    FABD: TAppBarData;
    FDockedHeight: Integer;
    FDockedWidth: Integer;
    FEdge: TAppBarEdge;
    FOnEdgeChanged: TNotifyEvent;
    FTopMost: Boolean;
    procedure WMActivate(var M: TMessage); message WM_ACTIVATE;
    procedure WMWindowPosChanged(var M: TMessage); message WM_WINDOWPOSCHANGED;
    function SendAppBarMsg(Msg: DWORD): UINT;
    procedure SetAppBarEdge(Value: TAppBarEdge);
    procedure SetAppBarPos(Edge: UINT);
    procedure SetTopMost(Value: Boolean);
    procedure SetDockedHeight(const Value: Integer);
    procedure SetDockedWidth(const Value: Integer);
  protected
    procedure CreateParams(var Params: TCreateParams); override;
    procedure CreateWnd; override;
    procedure DestroyWnd; override;
    procedure WndProc(var M: TMessage); override;
  public
    constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
    property DockManager;
  published
    property Action;
    property ActiveControl;
    property AutoScroll;
    property AutoSize;
    property BiDiMode;
    property BorderWidth;
    property Color;
    property Ctl3D;
```

Listagem 16.3 Continuação

```
property DockedHeight: Integer read FDockedHeight write SetDockedHeight
  default 35;
property DockedWidth: Integer read FDockedWidth write SetDockedWidth
  default 40;
property UseDockManager;
property DockSite;
property DragKind;
property DragMode;
property Edge: TAppBarEdge read FEdge write SetAppBarEdge default abeTop;
property Enabled;
property ParentFont default False;
property Font;
property HelpFile;
property HorzScrollBar;
property Icon;
property KeyPreview;
property ObjectMenuItem;
property ParentBiDiMode;
property PixelsPerInch;
property PopupMenu;
property PrintScale;
property Scaled;
property ShowHint;
property TopMost: Boolean read FTopMost write SetTopMost default False;
property VertScrollBar;
property Visible;
property OnActivate;
property OnCanResize;
property OnClick;
property OnClose;
property OnCloseQuery;
property OnConstrainedResize;
property OnCreate;
property OnDblClick;
property OnDestroy;
property OnDeactivate;
property OnDockDrop;
property OnDockOver;
property OnDragDrop;
property OnDragOver;
property OnEdgeChanged: TNotifyEvent read FOnEdgeChanged
  write FOnEdgeChanged;
property OnEndDock;
property OnGetSiteInfo;
property OnHide;
property OnHelp;
property OnKeyDown;
property OnKeyPress;
property OnKeyUp;
property OnMouseDown;
property OnMouseMove;
property OnMouseUp;
property OnMouseWheel;
property OnMouseWheelDown;
property OnMouseWheelUp;
property OnPaint;
property OnResize;
```

Listagem 16.3 Continuação

```
    property OnShortCut;
    property OnShow;
    property OnStartDock;
    property OnUnDock;
  end;

implementation

var
  AppBarMsg: UINT;

constructor TAppBar.CreateNew(AOwner: TComponent; Dummy: Integer);
begin
  FDockedHeight := 35;
  FDockedWidth := 40;
  inherited CreateNew(AOwner, Dummy);
  ClientHeight := 35;
  Width := 100;
  BorderStyle := bsNone;
  BorderIcons := [ ];
  // configura o registro TAppBarData
  FABD.cbSize := SizeOf(FABD);
  FABD.uCallbackMessage := AppBarMsg;
end;

procedure TAppBar.WMWindowPosChanged(var M: TMessage);
begin
  inherited;
  // Deve informar ao shell que a posição da AppBar foi alterada
  SendAppBarMsg(ABM_WINDOWPOSCHANGED);
end;

procedure TAppBar.WMActivate(var M: TMessage);
begin
  inherited;
  // Deve informar ao shell que a janela AppBar foi ativada
  SendAppBarMsg(ABM_ACTIVATE);
end;

procedure TAppBar.WndProc(var M: TMessage);
var
  State: UINT;
begin
  if M.Msg = AppBarMsg then
  begin
    case M.WParam of
      // Enviada quando mudar o estado 'Sempre visível' ou 'Auto-Ocultar'.
      ABN_STATECHANGE:
        begin
          // Verifica se a barra de acesso ainda é ABS_ALWAYSONTOP.
          State := SendAppBarMsg(ABM_GETSTATE);
          if ABS_ALWAYSONTOP and State = 0 then
            SetTopMost(False)
          else
            SetTopMost(True);
        end;
      // Uma aplicação de tela inteira foi iniciada ou a última
```

Listagem 16.3 Continuação

```
      // aplicação de tela inteira foi fechada.
      ABN_FULLSCREENAPP:
        begin
          // Define a ordem z da barra de acesso de modo apropriado.
          State := SendAppBarMsg(ABM_GETSTATE);
          if M.lParam < > 0 then begin
            if ABS_ALWAYSONTOP and State = 0 then
              SetTopMost(False)
            else
              SetTopMost(True);
          end
          else
            if State and ABS_ALWAYSONTOP < > 0 then
              SetTopMost(True);
        end;
      // Enviado quando houver algo que possa afetar a posição de AppBar.
      ABN_POSCHANGED:
        // A barra de tarefas ou outra barra de acesso
        // mudou seu tamanho ou sua posição.
        SetAppBarPos(FABD.uEdge);
    end;
  end
  else
    inherited WndProc(M);
end;

function TAppBar.SendAppBarMsg(Msg: DWORD): UINT;
begin
  // Não faz nada em AppBar durante o projeto... muito medroso
  if csDesigning in ComponentState then Result := 0
  else Result := SHAppBarMessage(Msg, FABD);
end;

procedure TAppBar.SetAppBarPos(Edge: UINT);
begin
  if csDesigning in ComponentState then Exit;
  FABD.uEdge := Edge;       // define borda
  with FABD.rc do
  begin
    // define coordenadas para tela inteira
    Top := 0;
    Left := 0;
    Right := Screen.Width;
    Bottom := Screen.Height;
    // Envia ABM_QUERYPOS para obter retângulo apropriado na margem
    SendAppBarMsg(ABM_QUERYPOS);
    // reajusta retângulo com base naquele modificado por ABM_QUERYPOS
    case Edge of
      ABE_LEFT: Right := Left + FDockedWidth;
      ABE_RIGHT: Left := Right – FDockedWidth;
      ABE_TOP: Bottom := Top + FDockedHeight;
      ABE_BOTTOM: Top := Bottom – FDockedHeight;
    end;
    // Define posicão da barra da aplicação.
    SendAppBarMsg(ABM_SETPOS);
  end;
  // Define propriedade BoundsRect de modo que ela se ajuste ao
```

Listagem 16.3 Continuação

```
  // retângulo passado para o sistema.
  BoundsRect := FABD.rc;
end;

procedure TAppBar.SetTopMost(Value: Boolean);
const
  WndPosArray: array[Boolean] of HWND = (HWND_BOTTOM, HWND_TOPMOST);
begin
  if FTopMost < > Value then
  begin
    FTopMost := Value;
    if not (csDesigning in ComponentState) then
      SetWindowPos(Handle, WndPosArray[Value], 0, 0, 0, 0, SWP_NOMOVE or
        SWP_NOSIZE or SWP_NOACTIVATE);
  end;
end;

procedure TAppBar.CreateParams(var Params: TCreateParams);
begin
  inherited CreateParams(Params);
  if not (csDesigning in ComponentState) then
  begin
    Params.ExStyle := Params.ExStyle or WS_EX_TOPMOST or WS_EX_WINDOWEDGE;
    Params.Style := Params.Style or WS_DLGFRAME;
  end;
end;

procedure TAppBar.CreateWnd;
begin
  inherited CreateWnd;
  FABD.hWnd := Handle;
  if not (csDesigning in ComponentState) then
begin
    if SendAppBarMsg(ABM_NEW) = 0 then
      raise EAppBarError.Create('Failed to create AppBar');
    // Inicializa a posição
    SetAppBarEdge(FEdge);
  end;
end;

procedure TAppBar.DestroyWnd;
begin
  // Deve informar ao shell que a AppBar está sendo fechada
  SendAppBarMsg(ABM_REMOVE);
  inherited DestroyWnd;
end;

procedure TAppBar.SetAppBarEdge(Value: TAppBarEdge);
const
  EdgeArray: array[TAppBarEdge] of UINT =
    (ABE_TOP, ABE_BOTTOM, ABE_LEFT, ABE_RIGHT);
begin
  SetAppBarPos(EdgeArray[Value]);
  FEdge := Value;
  if Assigned(FOnEdgeChanged) then FOnEdgeChanged(Self);
end;
```

Listagem 16.3 Continuação

```
procedure TAppBar.SetDockedHeight(const Value: Integer);
begin
  if FDockedHeight < > Value then
  begin
    FDockedHeight := Value;
    SetAppBarEdge(FEdge);
  end;
end;

procedure TAppBar.SetDockedWidth(const Value: Integer);
begin
  if FDockedWidth < > Value then
  begin
    FDockedWidth := Value;
    SetAppBarEdge(FEdge);
  end;
end;

initialization
  AppBarMsg := RegisterWindowMessage('DDG AppBar Message');
end.
```

Usando `TAppBar`

Se você instalou o software encontrado no CD-ROM que acompanha este livro, o uso de `TAppBar` será muito fácil: basta selecionar a opção AppBar da página DDG da caixa de diálogo File, New. Isso chama um assistente que gerará uma unidade contendo um componente `TAppBar`.

> NOTA
> O Capítulo 17 demonstra como criar um assistente que gera automaticamente uma TAppBar. Neste capítulo, no entanto, você pode ignorar a implementação do assistente. Basta entender que algum trabalho está sendo feito nos bastidores para gerar a unidade e o formulário da AppBar para você.

Nesta pequena aplicação de exemplo, `TAppBar` é usada para criar uma barra de ferramentas de aplicação que contenha botões para diversos comandos de edição: Open (abrir), Save (salvar), Cut (recortar), Copy (copiar) e Paste (colar). Os botões manipularão um componente `TMemo` encontrado no formulário principal. O código-fonte dessa unidade aparece na Listagem 16.4, e a Figure 16.5 mostra a aplicação em ação com o controle da AppBar encaixado na parte inferior da tela.

Listagem 16.4 `ApBarFrm.pas` – a unidade principal da aplicação de demonstração AppBar

```
unit ApBarFrm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  AppBars, Menus, Buttons;

type
  TAppBarForm = class(TAppBar)
    sbOpen: TSpeedButton;
```

Listagem 16.4 Continuação

```
    sbSave: TSpeedButton;
    sbCut: TSpeedButton;
    sbCopy: TSpeedButton;
    sbPaste: TSpeedButton;
    OpenDialog: TOpenDialog;
    pmPopup: TPopupMenu;
    Top1: TMenuItem;
    Bottom1: TMenuItem;
    Left1: TMenuItem;
    Right1: TMenuItem;
    N1: TMenuItem;
    Exit1: TMenuItem;
    procedure Right1Click(Sender: TObject);
    procedure sbOpenClick(Sender: TObject);
    procedure sbSaveClick(Sender: TObject);
    procedure sbCutClick(Sender: TObject);
    procedure sbCopyClick(Sender: TObject);
    procedure sbPasteClick(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormEdgeChanged(Sender: TObject);
  private
    FLastChecked: TMenuItem;
    procedure MoveButtons;
  end;

var
  AppBarForm: TAppBarForm;

implementation

uses Main;

{$R *.DFM}

{ TAppBarForm }

procedure TAppBarForm.MoveButtons;
// Este método parece complicado, mas ele apenas organiza os botões de
// modo apropriado, dependendo do lado em que a AppBar foi encaixada.
var
  DeltaCenter, NewPos: Integer;
begin
  if Edge in [abeTop, abeBottom] then
  begin
    DeltaCenter := (ClientHeight – sbOpen.Height) div 2;
    sbOpen.SetBounds(10, DeltaCenter, sbOpen.Width, sbOpen.Height);
    NewPos := sbOpen.Width + 20;
    sbSave.SetBounds(NewPos, DeltaCenter, sbOpen.Width, sbOpen.Height);
    NewPos := NewPos + sbOpen.Width + 10;
    sbCut.SetBounds(NewPos, DeltaCenter, sbOpen.Width, sbOpen.Height);
    NewPos := NewPos + sbOpen.Width + 10;
    sbCopy.SetBounds(NewPos, DeltaCenter, sbOpen.Width, sbOpen.Height);
    NewPos := NewPos + sbOpen.Width + 10;
    sbPaste.SetBounds(NewPos, DeltaCenter, sbOpen.Width, sbOpen.Height);
  end
  else
```

Listagem 16.4 Continuação

```
  begin
    DeltaCenter := (ClientWidth – sbOpen.Width) div 2;
    sbOpen.SetBounds(DeltaCenter, 10, sbOpen.Width, sbOpen.Height);
    NewPos := sbOpen.Height + 20;
    sbSave.SetBounds(DeltaCenter, NewPos, sbOpen.Width, sbOpen.Height);
    NewPos := NewPos + sbOpen.Height + 10;
    sbCut.SetBounds(DeltaCenter, NewPos, sbOpen.Width, sbOpen.Height);
    NewPos := NewPos + sbOpen.Height + 10;
    sbCopy.SetBounds(DeltaCenter, NewPos, sbOpen.Width, sbOpen.Height);
    NewPos := NewPos + sbOpen.Height + 10;
    sbPaste.SetBounds(DeltaCenter, NewPos, sbOpen.Width, sbOpen.Height);
  end;
end;

procedure TAppBarForm.Right1Click(Sender: TObject);
begin
  FLastChecked.Checked := False;
  (Sender as TMenuItem).Checked := True;
  case TMenuItem(Sender).Caption[2] of
    'T': Edge := abeTop;
    'B': Edge := abeBottom;
    'L': Edge := abeLeft;
    'R': Edge := abeRight;
  end;
  FLastChecked := TMenuItem(Sender);
end;

procedure TAppBarForm.sbOpenClick(Sender: TObject);
begin
  if OpenDialog.Execute then
    MainForm.FileName := OpenDialog.FileName;
end;

procedure TAppBarForm.sbSaveClick(Sender: TObject);
begin
  MainForm.memEditor.Lines.SaveToFile(MainForm.FileName);
end;

procedure TAppBarForm.sbCutClick(Sender: TObject);
begin
  MainForm.memEditor.CutToClipboard;
end;

procedure TAppBarForm.sbCopyClick(Sender: TObject);
begin
  MainForm.memEditor.CopyToClipboard;
end;

procedure TAppBarForm.sbPasteClick(Sender: TObject);
begin
  MainForm.memEditor.PasteFromClipboard;
end;

procedure TAppBarForm.Exit1Click(Sender: TObject);
begin
  Application.Terminate;
end;
```

Listagem 16.4 Continuação

```
procedure TAppBarForm.FormCreate(Sender: TObject);
begin
  FLastChecked := Top1;
end;

procedure TAppBarForm.FormEdgeChanged(Sender: TObject);
begin
  MoveButtons;
end;

end.
```

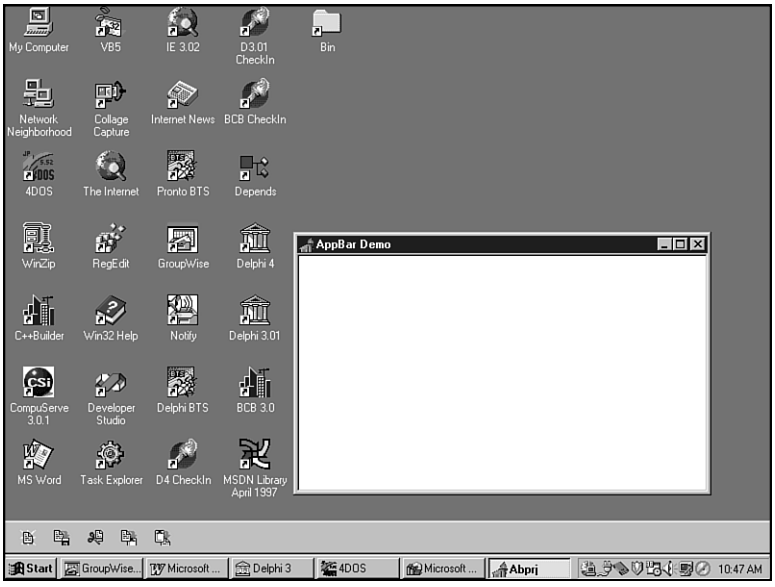

Figura 16.5 **TAppBar** em ação.

Vínculos do shell

O shell do Windows expõe uma série de interfaces que podem ser utilizadas para manipular diferentes aspectos do shell. Essas interfaces são definidas na unidade `ShlObj`. Seria preciso um novo livro para discutir em profundidade todos os objetos dessa unidade e, portanto, vamos concentrar nossos esforços em uma das interfaces mais úteis (e mais usadas): `IShellLink`.

`IShellLink` é uma interface que permite a criação e manipulação dos vínculos do shell nas suas aplicações. Caso esteja em dúvida, a maioria dos ícones do seu desktop provavelmente é composta de vínculos do shell. Além disso, cada item no menu Send To (enviar para) local do shell ou no menu Documents (documentos), ao qual você tem acesso pelo menu Start (iniciar) é um vínculo do shell. A interface `IShellLink` é definida da seguinte maneira:

```
const

type
  IShellLink = interface(IUnknown)
    ['{000214EE-0000-0000-C000-000000000046}']
    function GetPath(pszFile: PAnsiChar; cchMaxPath: Integer;
      var pfd: TWin32FindData; fFlags: DWORD): HResult; stdcall;
    function GetIDList(var ppidl: PItemIDList): HResult; stdcall;
```

```
    function SetIDList(pidl: PItemIDList): HResult; stdcall;
    function GetDescription(pszName: PAnsiChar; cchMaxName: Integer): HResult;
      stdcall;
    function SetDescription(pszName: PAnsiChar): HResult; stdcall;
    function GetWorkingDirectory(pszDir: PAnsiChar; cchMaxPath: Integer):
      HResult;
      stdcall;
    function SetWorkingDirectory(pszDir: PAnsiChar): HResult; stdcall;
    function GetArguments(pszArgs: PAnsiChar; cchMaxPath: Integer): HResult;
      stdcall;
    function SetArguments(pszArgs: PAnsiChar): HResult; stdcall;
    function GetHotkey(var pwHotkey: Word): HResult; stdcall;
    function SetHotkey(wHotkey: Word): HResult; stdcall;
    function GetShowCmd(out piShowCmd: Integer): HResult; stdcall;
    function SetShowCmd(iShowCmd: Integer): HResult; stdcall;
    function GetIconLocation(pszIconPath: PAnsiChar; cchIconPath: Integer;
      out piIcon: Integer): HResult; stdcall;
    function SetIconLocation(pszIconPath: PAnsiChar; iIcon: Integer): HResult;
      stdcall;
    function SetRelativePath(pszPathRel: PAnsiChar; dwReserved: DWORD):
      HResult;
      stdcall;
    function Resolve(Wnd: HWND; fFlags: DWORD): HResult; stdcall;
    function SetPath(pszFile: PAnsiChar): HResult; stdcall;
  end;
```

NOTA

Como `IshellLink` e seus métodos são descritos com detalhes na ajuda on-line do Win32, não vamos discuti-los aqui.

Obtendo uma instância de `IShellLink`

Ao contrário do trabalho com extensões do shell, sobre as quais falaremos ainda neste capítulo, você não implementa a interface `IShellLink` . Em vez disso, essa interface é implementada pelo shell do Windows e você usa a função `CoCreateInstance( )` da COM para criar uma instância. Veja o exemplo a seguir:

```
var
  SL: IShellLink;
begin
  OleCheck(CoCreateInstance(CLSID_ShellLink, nil, CLSCTX_INPROC_SERVER,
    IShellLink, SL));
  // use SL aqui
end;
```

NOTA

Não se esqueça de que, antes de poder usar qualquer uma das funções OLE, você deve inicializar a biblioteca COM usando a função `CoInitialize( )`. Quando você tiver acabado de usar a COM, deverá excluí-la chamando `CoUninitialize( )`. Essas funções serão chamadas pelo Delphi em uma aplicação que use `ComObj` e contenha uma chamada para `Application.Initialize( )`. Caso contrário, você mesmo terá que chamar essas funções.

Usando IShellLink

Aparentemente, os vínculos do shell podem ser considerados algo mágico: você dá um clique com o botão direito do mouse no desktop, cria um novo atalho e *alguma coisa* acontece, que faz com que um ícone apareça no desktop. Na verdade, essa *alguma coisa* não tem mistério algum quando você sabe o que está ocorrendo. Um *vínculo de shell* não passa de um arquivo com uma extensão LNK, que reside em algum diretório. Quando o Windows é inicializado, ele procura arquivos LNK em determinados diretórios, que representam vínculos que residem em diferentes *pastas do shell*. Essas pastas do shell, ou *pastas especiais*, incluem itens como Network Neighborhood (ambiente de rede), Send To (enviar para), Startup (iniciar) e Desktop (área de trabalho), entre outras coisas. O shell armazena as correspondências vínculo/pasta no Registro do sistema – encontradas, em sua maioria, abaixo da seguinte chave, caso você queira examinar:

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer
➥\Shell Folders
```

A criação de um vínculo do shell em uma pasta especial, portanto, é apenas uma questão de colocar um arquivo de vínculo em um determinado diretório. Em vez de mexer com o Registro do sistema, você pode usar `SHGetSpecialFolderPath( )` para obter o caminho de diretório para as diversas pastas especiais. Esse método é definido da seguinte maneira:

```
function SHGetSpecialFolderPath(hwndOwner: HWND; lpszPath: PChar;
  nFolder: Integer; fCreate: BOOL): BOOL; stdcall;
```

`hwndOwner` contém a alça de uma janela que servirá como o proprietária para qualquer caixa de diálogo que a função possa chamar.

`lpszPath` é um ponteiro para um buffer que receberá o caminho. Esse buffer deve ter, pelo menos, o número de caracteres registrado em `MAX_PATH`.

`nFolder` identifica a pasta especial cujo caminho você deseja obter. A Tabela 16.4 mostra os possíveis valores para esse parâmetro e uma descrição de cada um deles.

`fCreate` indica se uma pasta deve ser criada, caso não exista.

Tabela 16.4 Possíveis valores para `NFolder`

Flag	*Descrição*
`CSIDL_ALTSTARTUP`	O diretório que corresponde ao grupo de programas Startup não-localizado do usuário.
`CSIDL_APPDATA`	O diretório que serve como um repositório comum para os dados específicos da aplicação.
`CSIDL_BITBUCKET`	O diretório que contém o objeto de arquivo na Recycle Bin (lixeira) do usuário. Esse diretório não está localizado no Registro; ele é marcado com os atributos ocultos e de sistema para impedir que o usuário o mova ou o exclua.
`CSIDL_COMMON_ALTSTARTUP`	O diretório que corresponde ao grupo de programas Startup não-localizado de todos os usuários.
`CSIDL_COMMON_DESKTOPDIRECTORY`	O diretório que contém arquivos e pastas que aparecem na desktop de todos os usuários.
`CSIDL_COMMON_FAVORITES`	O diretório que serve como um repositório comum dos itens favoritos de todos os usuários.
`CSIDL_COMMON_PROGRAMS`	O diretório que contém os diretórios dos grupos de programas comuns que aparecem no menu Iniciar de todos os usuários.
`CSIDL_COMMON_STARTMENU`	O diretório que contém os programas e pastas que aparecem no menu Iniciar de todos os usuários.

Tabela 16.4 Continuação

Flag	*Descrição*
CSIDL_COMMON_STARTUP	O diretório que contém os programas que aparecem na pasta Iniciar de todos os usuários.
CSIDL_CONTROLS	Uma pasta virtual contendo ícones de todas as aplicações no Control Panel (painel de controle).
CSIDL_COOKIES	O diretório que serve como um repositório comum para todos os cookies da Internet.
CSIDL_DESKTOP	A pasta virtual da Desktop do Windows na raiz do namespace.
CSIDL_DESKTOPDIRECTORY	O diretório usado para armazenar fisicamente objetos de arquivo no desktop (não confunda isso com a pasta Desktop propriamente dita).
CSIDL_DRIVES	A pasta virtual do My Computer (meu computador) contendo tudo o que está armazenado no computador local: dispositivos de armazenamento, impressoras e o Control Panel. A pasta também pode conter unidades de rede mapeadas.
CSIDL_FAVORITES	O diretório que serve como um repositório comum para os itens favoritos do usuário.
CSIDL_FONTS	Uma pasta virtual contendo fontes.
CSIDL_HISTORY	O diretório que serve como um repositório comum para itens do histórico na Internet.
CSIDL_INTERNET	Uma pasta virtual representando a Internet.
CSIDL_INTERNET_CACHE	O diretório que serve como um repositório comum para todos os arquivos temporários da Internet.
CSIDL_NETHOOD	O diretório que contém objetos que aparecem no Network Neighborhood (ambiente de rede).
CSIDL_NETWORK	A pasta virtual do Network Neighborhood representando o nível mais alto na hierarquia da rede.
CSIDL_PERSONAL	O diretório que serve como um repositório comum para os documentos.
CSIDL_PRINTERS	Uma pasta virtual contendo as impressoras instaladas.
CSIDL_PRINTHOOD	O diretório que serve como um repositório comum para os vínculos da impressora.
CSIDL_PROGRAMS	O diretório que contém os grupos de programa do usuário (que também são diretórios).
CSIDL_RECENT	O diretório que contém os últimos documentos usados pelo usuário.
CSIDL_SENDTO	O diretório que contém os itens do menu Send To.
CSIDL_STARTMENU	O diretório que contém os itens do menu Start.
CSIDL_STARTUP	O diretório que corresponde ao grupo de programas Startup do usuário. O sistema inicia esses programas sempre que o usuário é autenticado no Windows NT ou inicia o Windows 95 ou 98.
CSIDL_TEMPLATES	O diretório que serve como um repositório comum para modelos de documento.

Criando um vínculo do shell

A interface `IShellLink` é um encapsulamento de um objeto de vínculo com o shell, mas não possui um conceito quanto ao modo como é lida ou escrita em um arquivo no disco. No entanto, os implementadores da interface `IShellLink` também têm a obrigação de oferecer suporte à interface `IPersistFile` para fornecer o acesso ao arquivo. `IPersistFile` é uma interface que fornece métodos de leitura e gravação de/para disco e é definida da seguinte maneira:

```
type
  IPersistFile = interface(IPersist)
    ['{0000010B-0000-0000-C000-000000000046}']
    function IsDirty: HResult; stdcall;
    function Load(pszFileName: POleStr; dwMode: Longint): HResult;
      stdcall;
    function Save(pszFileName: POleStr; fRemember: BOOL): HResult;
      stdcall;
    function SaveCompleted(pszFileName: POleStr): HResult;
      stdcall;
    function GetCurFile(out pszFileName: POleStr): HResult;
      stdcall;
  end;
```

> NOTA
>
> Para obter uma descrição completa de IPersistFile e seus métodos, consulte a ajuda on-line do Win32.

Como a classe que implementa `IShellLink` também é obrigatória para implementar `IPeristFile`, você pode usar a `QueryInterface` para consultar a instância de `IShellLink` de uma instância de `IPersistFile` usando o operador `as`, como mostramos a seguir:

```
var
  SL: IShellLink;
  PF: IPersistFile;
begin
  OleCheck(CoCreateInstance(CLSID_ShellLink, nil, CLSCTX_INPROC_SERVER,
    IShellLink, SL));
  PF := SL as IPersistFile;
  // use PF e SL
end;
```

Como já dissemos, o uso de objetos da interface COM corresponde a usar os objetos normais da Object Pascal. O código mostrado a seguir, por exemplo, cria um vínculo com o shell do desktop com a aplicação Notepad (bloco de notas):

```
procedure MakeNotepad;
const
  // NOTA: Posição presumida do Notepad:
  AppName = 'c:\windows\notepad.exe';
var
  SL: IShellLink;
  PF: IPersistFile;
  LnkName: WideString;
begin
  OleCheck(CoCreateInstance(CLSID_ShellLink, nil, CLSCTX_INPROC_SERVER,
    IShellLink, SL));
```

```
  { Os implementadores de IShellLink têm que implementar IPersistFile }
  PF := SL as IPersistFile;
  OleCheck(SL.SetPath(PChar(AppName)));  // define caminho do vínculo para o arquivo apropriado
  { cria um caminho e um nome de arquivo para o arquivo de vínculo }
  LnkName := GetFolderLocation('Desktop') + '\' +
    ChangeFileExt(ExtractFileName(AppName), '.lnk');
  PF.Save(PWideChar(LnkName), True);      // salva arquivo de vínculo
end;
```

Nesse procedimento, o método `SetPath( )` de `IShellLink` é usado para apontar o vínculo para um documento ou arquivo executável (nesse caso, o Notepad). Posteriormente, um caminho e nome de arquivo para o vínculo é criado usando o caminho retornado por `GetFolderLocation('Desktop')` (já descrita nesta seção) e pelo uso da função `ChangeFileExt( )` para alterar a extensão do Notepad de `.EXE` para `.LNK`. Esse novo nome de arquivo é armazenado em `LnkName`. Depois disso, o método `Save( )` salva o vínculo em um arquivo no disco. Como você já aprendeu, quando o procedimento terminar e as instâncias de interface `SL` e `PF` saírem do escopo, suas respectivas referências serão liberadas.

Obtendo e definindo informações de vínculo

Como você pode ver na definição da interface `IShellLink`, ela contém um série de métodos `Get`*`XXX`*`( )` e `Set`*`XXX`*`( )` que permitem que você obtenha e defina diferentes aspectos do vínculo com o shell. Considere a declaração de registro a seguir, que contém campos para cada um dos possíveis valores que podem ser definidos ou apanhados:

```
type
  TShellLinkInfo = record
    PathName: string;
    Arguments: string;
    Description: string;
    WorkingDirectory: string;
    IconLocation: string;
    IconIndex: Integer;
    ShowCmd: Integer;
    HotKey: Word;
  end;
```

Dado esse registro, você pode criar funções que recuperam as definições de um determinado vínculo do shell com o registro ou que definem os valores do vínculo como aqueles indicados pelo conteúdo do registro. Essas funções aparecem na Listagem 16.5; `WinShell.pas` é uma unidade que contém o código-fonte completo dessas funções.

Listagem 16.5 `WinShell.pas` – a unidade que contém as funções que operam nos vínculos do shell

```
unit WinShell;

interface

uses SysUtils, Windows, Registry, ActiveX, ShlObj;

type
  EShellOleError = class(Exception);

  TShellLinkInfo = record
    PathName: string;
```

Listagem 16.5 Continuação

```
    Arguments: string;
    Description: string;
    WorkingDirectory: string;
    IconLocation: string;
    IconIndex: integer;
    ShowCmd: integer;
    HotKey: word;
  end;

  TSpecialFolderInfo = record
    Name: string;
    ID: Integer;
  end;

const
  SpecialFolders: array[0..29] of TSpecialFolderInfo = (
    (Name: 'Alt Startup'; ID: CSIDL_ALTSTARTUP),
    (Name: 'Application Data'; ID: CSIDL_APPDATA),
    (Name: 'Recycle Bin'; ID: CSIDL_BITBUCKET),
    (Name: 'Common Alt Startup'; ID: CSIDL_COMMON_ALTSTARTUP),
    (Name: 'Common Desktop'; ID: CSIDL_COMMON_DESKTOPDIRECTORY),
    (Name: 'Common Favorites'; ID: CSIDL_COMMON_FAVORITES),
    (Name: 'Common Programs'; ID: CSIDL_COMMON_PROGRAMS),
    (Name: 'Common Start Menu'; ID: CSIDL_COMMON_STARTMENU),
    (Name: 'Common Startup'; ID: CSIDL_COMMON_STARTUP),
    (Name: 'Controls'; ID: CSIDL_CONTROLS),
    (Name: 'Cookies'; ID: CSIDL_COOKIES),
    (Name: 'Desktop'; ID: CSIDL_DESKTOP),
    (Name: 'Desktop Directory'; ID: CSIDL_DESKTOPDIRECTORY),
    (Name: 'Drives'; ID: CSIDL_DRIVES),
    (Name: 'Favorites'; ID: CSIDL_FAVORITES),
    (Name: 'Fonts'; ID: CSIDL_FONTS),
    (Name: 'History'; ID: CSIDL_HISTORY),
    (Name: 'Internet'; ID: CSIDL_INTERNET),
    (Name: 'Internet Cache'; ID: CSIDL_INTERNET_CACHE),
    (Name: 'Network Neighborhood'; ID: CSIDL_NETHOOD),
    (Name: 'Network Top'; ID: CSIDL_NETWORK),
    (Name: 'Personal'; ID: CSIDL_PERSONAL),
    (Name: 'Printers'; ID: CSIDL_PRINTERS),
    (Name: 'Printer Links'; ID: CSIDL_PRINTHOOD),
    (Name: 'Programs'; ID: CSIDL_PROGRAMS),
    (Name: 'Recent Documents'; ID: CSIDL_RECENT),
    (Name: 'Send To'; ID: CSIDL_SENDTO),
    (Name: 'Start Menu'; ID: CSIDL_STARTMENU),
    (Name: 'Startup'; ID: CSIDL_STARTUP),
    (Name: 'Templates'; ID: CSIDL_TEMPLATES));

function CreateShellLink(const AppName, Desc: string; Dest: Integer): string;
function GetSpecialFolderPath(Folder: Integer; CanCreate: Boolean): string;
procedure GetShellLinkInfo(const LinkFile: WideString;
  var SLI: TShellLinkInfo);
procedure SetShellLinkInfo(const LinkFile: WideString;
  const SLI: TShellLinkInfo);

implementation

uses ComObj;
```

Listagem 16.5 Continuação

```
function GetSpecialFolderPath(Folder: Integer; CanCreate: Boolean): string;
var
  FilePath: array[0..MAX_PATH] of char;
begin
  { Obtém caminho do local selecionado }
  SHGetSpecialFolderPathW(0, FilePath, Folder, CanCreate);
  Result := FilePath;
end;

function CreateShellLink(const AppName, Desc: string; Dest: Integer): string;
{ Cria um vínculo do shell para a aplicação ou documento especificado em  }
{ AppName com a descrição Desc. O vínculo será localizado em uma pasta    }
{ especificada por Dest, que é uma das constantes de string mostradas na  }
{ parte superior desta unidade. Retorna o nome completo do caminho do     }
{ arquivo de vínculo. }
var
  SL: IShellLink;
  PF: IPersistFile;
  LnkName: WideString;
begin
  OleCheck(CoCreateInstance(CLSID_ShellLink, nil, CLSCTX_INPROC_SERVER,
    IShellLink, SL));
  { O implementador de IShellLink também deve aceitar a interface  }
  { IPersistFile. Obtém um ponteiro de interface para ela.         }
  PF := SL as IPersistFile;
  { define caminho do vínculo para o arquivo apropriado }
  OleCheck(SL.SetPath(PChar(AppName)));
  if Desc < > '' then
    OleCheck(SL.SetDescription(PChar(Desc))); // define descrição
  { cria um local de caminho e um nome de arquivo para o arquivo de vínculo }
  LnkName := GetSpecialFolderPath(Dest, True) + '\' +
             ChangeFileExt(AppName, 'lnk');
  PF.Save(PWideChar(LnkName), True);          // salva arquivo de vínculo
  Result := LnkName;
end;

procedure GetShellLinkInfo(const LinkFile: WideString;
  var SLI: TShellLinkInfo);
{ Recupera informações em um vínculo de shell existente }
var
  SL: IShellLink;
  PF: IPersistFile;
  FindData: TWin32FindData;
  AStr: array[0..MAX_PATH] of char;
begin
  OleCheck(CoCreateInstance(CLSID_ShellLink, nil, CLSCTX_INPROC_SERVER,
    IShellLink, SL));
  { O implementador de IShellLink também deve aceitar a interface }
  { IPersistFile. Obtém um ponteiro de interface para ela.        }
  PF := SL as IPersistFile;
  { Carrega arquivo em objeto IPersistFile }
  OleCheck(PF.Load(PWideChar(LinkFile), STGM_READ));
  { Resolve o vínculo chamando a função de interface Resolve. }
  OleCheck(SL.Resolve(0, SLR_ANY_MATCH or SLR_NO_UI));
  { Obtém todas as informações! }
  with SLI do
  begin
```

Listagem 16.5 Continuação

```
    OleCheck(SL.GetPath(AStr, MAX_PATH, FindData, SLGP_SHORTPATH));
    PathName := AStr;
    OleCheck(SL.GetArguments(AStr, MAX_PATH));
    Arguments := AStr;
    OleCheck(SL.GetDescription(AStr, MAX_PATH));
    Description := AStr;
    OleCheck(SL.GetWorkingDirectory(AStr, MAX_PATH));
    WorkingDirectory := AStr;
    OleCheck(SL.GetIconLocation(AStr, MAX_PATH, IconIndex));
    IconLocation := AStr;
    OleCheck(SL.GetShowCmd(ShowCmd));
    OleCheck(SL.GetHotKey(HotKey));
  end;
end;

procedure SetShellLinkInfo(const LinkFile: WideString;
  const SLI: TShellLinkInfo);
{ Define informações para um vínculo de shell existente }
var
  SL: IShellLink;
  PF: IPersistFile;
begin
  OleCheck(CoCreateInstance(CLSID_ShellLink, nil, CLSCTX_INPROC_SERVER,
    IShellLink, SL));
  { O implementador de IShellLink também deve aceitar a interface }
  { IpersistFile. Obtém um ponteiro de interface para ela. }
  PF := SL as IPersistFile;
  { Carrega arquivo em objeto IPersistFile }
  OleCheck(PF.Load(PWideChar(LinkFile), STGM_SHARE_DENY_WRITE));
  { Resolve o vínculo chamando a função de interface Resolve. }
  OleCheck(SL.Resolve(0, SLR_ANY_MATCH or SLR_UPDATE or SLR_NO_UI));
  { Define todas as informações! }
  with SLI, SL do
  begin
    OleCheck(SetPath(PChar(PathName)));
    OleCheck(SetArguments(PChar(Arguments)));
    OleCheck(SetDescription(PChar(Description)));
    OleCheck(SetWorkingDirectory(PChar(WorkingDirectory)));
    OleCheck(SetIconLocation(PChar(IconLocation), IconIndex));
    OleCheck(SetShowCmd(ShowCmd));
    OleCheck(SetHotKey(HotKey));
  end;
  PF.Save(PWideChar(LinkFile), True);   // save file
end;

end.
```

Um método de `IShellLink` que ainda tem que ser explicado é o método `Resolve( )`. `Resolve( )` deve ser chamado depois que a interface `IPersistFile` de `IShellLink` for usada para carregar um arquivo de vínculo. Isso pesquisa o arquivo de vínculo especificado e preenche o objeto `IShellLink` com os valores especificados no arquivo.

DICA

Na função `GetShellLinkInfo( )`, mostrada na Listagem 16.5, observe o uso do array local `AStr`, no qual os valores são recuperados. Essa técnica é usada no lugar de `SetLength( )` para alocar espaço para as strings – o uso de SetLength( ) em tantas strings acarretaria na fragmentação do heap da aplicação. O uso de `Astr` como um intermediário impede que isso ocorra. Além disso, como o tamanho das strings só precisa ser definido uma vez, o uso de `AStr` acaba sendo ligeiramente mais rápido.

Uma aplicação de exemplo

Essas funções e interfaces podem ser divertidas e tudo o mais, mas elas não são nada sem uma aplicação na qual possam ser exibidas. O projeto Shell Link permite que você faça isso. O formulário principal desse projeto aparece na Figura 16.6.

A Listagem 16.6 mostra a unidade principal desse projeto, `Main.pas`. As Listagens 16.7 e 16.8 mostram `NewLinkU.pas` e `PickU.pas`, duas unidades de suporte para o projeto.

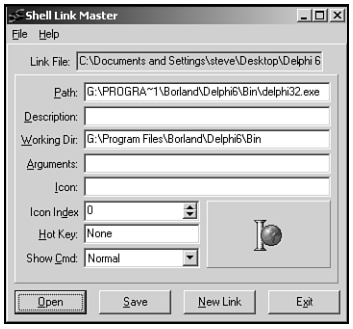

Figura 16.6 O formulário principal de Shell Link, mostrando um dos vínculos com o desktop.

Listagem 16.6 `Main.pas` – o código principal do projeto Shell Link

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ComCtrls, ExtCtrls, Spin, WinShell, Menus;

type
  TMainForm = class(TForm)
    Panel1: TPanel;
    btnOpen: TButton;
    edLink: TEdit;
    btnNew: TButton;
    btnSave: TButton;
    Label3: TLabel;
    Panel2: TPanel;
    Label1: TLabel;
    Label2: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
```

Listagem 16.6 Continuação

```
    Label9: TLabel;
    edIcon: TEdit;
    edDesc: TEdit;
    edWorkDir: TEdit;
    edArg: TEdit;
    cbShowCmd: TComboBox;
    hkHotKey: THotKey;
    speIcnIdx: TSpinEdit;
    pnlIconPanel: TPanel;
    imgIconImage: TImage;
    btnExit: TButton;
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    Open1: TMenuItem;
    Save1: TMenuItem;
    NewLInk1: TMenuItem;
    N1: TMenuItem;
    Exit1: TMenuItem;
    Help1: TMenuItem;
    About1: TMenuItem;
    edPath: TEdit;
    procedure btnOpenClick(Sender: TObject);
    procedure btnNewClick(Sender: TObject);
    procedure edIconChange(Sender: TObject);
    procedure btnSaveClick(Sender: TObject);
    procedure btnExitClick(Sender: TObject);
    procedure About1Click(Sender: TObject);
  private
    procedure GetControls(var SLI: TShellLinkInfo);
    procedure SetControls(const SLI: TShellLinkInfo);
    procedure ShowIcon;
    procedure OpenLinkFile(const LinkFileName: String);
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

uses PickU, NewLinkU, AboutU, CommCtrl, ShellAPI;

type
  THotKeyRec = record
    Char, ModCode: Byte;
  end;

procedure TMainForm.SetControls(const SLI: TShellLinkInfo);
{ Define valores dos controles da IU com base no conteúdo de SLI }
var
  Mods: THKModifiers;
begin
  with SLI do
  begin
    edPath.Text := PathName;
    edIcon.Text := IconLocation;
```

Listagem 16.6 Continuação

```
    { Se nome do ícone estiver em branco e o vínculo for para exe, usa }
    { nome exe para caminho do ícone. Isso é feito porque o índice do  }
    { ícone é ignorado se o caminho do ícone estiver em branco, mas um }
    { exe pode conter mais de um ícone. }
    if (IconLocation = '') and
      (CompareText(ExtractFileExt(PathName), 'EXE') = 0) then
      edIcon.Text := PathName;
    edWorkDir.Text := WorkingDirectory;
    edArg.Text := Arguments;
    speIcnIdx.Value := IconIndex;
    edDesc.Text := Description;
    { Constantes SW_* começam com 1 }
    cbShowCmd.ItemIndex := ShowCmd – 1;
    { Caractere de tecla de atalho no byte baixo }
    hkHotKey.HotKey := Lo(HotKey);
    { Descobre quais flags modificadores estão no byte alto }
    Mods := [ ];
    if (HOTKEYF_ALT and Hi(HotKey)) < > 0 then include(Mods, hkAlt);
    if (HOTKEYF_CONTROL and Hi(HotKey)) < > 0 then include(Mods, hkCtrl);
    if (HOTKEYF_EXT and Hi(HotKey)) < > 0 then include(Mods, hkExt);
    if (HOTKEYF_SHIFT and Hi(HotKey)) < > 0 then include(Mods, hkShift);
    { Define conjunto de modificadores }
    hkHotKey.Modifiers := Mods;
  end;
  ShowIcon;
end;

procedure TMainForm.GetControls(var SLI: TShellLinkInfo);
{ Obtém valores de controles da IU e usa-os para definir valores de SLI }
var
  CtlMods: THKModifiers;
  HR: THotKeyRec;
begin
  with SLI do
  begin
    PathName := edPath.Text;
    IconLocation := edIcon.Text;
    WorkingDirectory := edWorkDir.Text;
    Arguments := edArg.Text;
    IconIndex := speIcnIdx.Value;
    Description := edDesc.Text;
    { Constantes SW_* começam com 1 }
    ShowCmd := cbShowCmd.ItemIndex + 1;
    { Apanha o caractere da tecla de atalho }
    word(HR) := hkHotKey.HotKey;
    { Descobre quais teclas modificadores estão sendo usadas }
    CtlMods := hkHotKey.Modifiers;
    with HR do begin
      ModCode := 0;
      if (hkAlt in CtlMods) then ModCode := ModCode or HOTKEYF_ALT;
      if (hkCtrl in CtlMods) then ModCode := ModCode or HOTKEYF_CONTROL;
      if (hkExt in CtlMods) then ModCode := ModCode or HOTKEYF_EXT;
      if (hkShift in CtlMods) then ModCode := ModCode or HOTKEYF_SHIFT;
    end;
    HotKey := word(HR);
  end;
end;
```

Listagem 16.6 Continuação

```
procedure TMainForm.ShowIcon;
{ Apanha ícone do arquivo apropriado e mostra em IconImage }
var
  HI: THandle;
  IcnFile: string;
  IconIndex: word;
begin
  { Apanha nome do arquivo de ícone }
  IcnFile := edIcon.Text;
  { Se estiver em branco, usa o nome exe }
  if IcnFile = '' then
    IcnFile := edPath.Text;
  { Certifica-se de que o arquivo existe }
  if FileExists(IcnFile) then
  begin
    IconIndex := speIcnIdx.Value;
    { Extrai ícone do arquivo }
    HI := ExtractAssociatedIcon(hInstance, PChar(IcnFile), IconIndex);
    { Atribui alça do ícone a IconImage }
    imgIconImage.Picture.Icon.Handle := HI;
  end;
end;

procedure TMainForm.OpenLinkFile(const LinkFileName: string);
{ Abre um arquivo de vínculo, apanha informações e as exibe na IU }
var
  SLI: TShellLinkInfo;
begin
  edLink.Text := LinkFileName;
  try
    GetShellLinkInfo(LinkFileName, SLI);
  except
    on EShellOleError do
      MessageDlg('Error occurred while opening link', mtError, [mbOk], 0);
  end;
  SetControls(SLI);
end;

procedure TMainForm.btnOpenClick(Sender: TObject);
{ Tratador OnClick para OpenBtn }
var
  LinkFile: String;
begin
  if GetLinkFile(LinkFile) then
    OpenLinkFile(LinkFile);
end;

procedure TMainForm.btnNewClick(Sender: TObject);
{ Tratador OnClick para NewBtn }
var
  FileName: string;
  Dest: Integer;
begin
  if GetNewLinkName(FileName, Dest) then
    OpenLinkFile(CreateShellLink(FileName, '', Dest));
end;
```

Listagem 16.6 Continuação

```
procedure TMainForm.edIconChange(Sender: TObject);
{ Tratador OnChange para IconEd e IcnIdxEd }
begin
  ShowIcon;
end;

procedure TMainForm.btnSaveClick(Sender: TObject);
{ Tratador OnClick para SaveBtn }
var
  SLI: TShellLinkInfo;
begin
  GetControls(SLI);
  try
    SetShellLinkInfo(edLink.Text, SLI);
  except
    on EShellOleError do
      MessageDlg('Error occurred while setting info', mtError, [mbOk], 0);
  end;
end;

procedure TMainForm.btnExitClick(Sender: TObject);
{ Tratador OnClick para ExitBtn }
begin
  Close;
end;

procedure TMainForm.About1Click(Sender: TObject);
{ Tratador OnClick para item de menu Help|About }
begin
  AboutBox;
end;

end.
```

Listagem 16.7 `NewLinkU.pas` – a unidade com formulário que ajuda a criar novo vínculo

```
unit NewLinkU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Buttons, StdCtrls;

type
  TNewLinkForm = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    edLinkTo: TEdit;
    btnOk: TButton;
    btnCancel: TButton;
    cbLocation: TComboBox;
    sbOpen: TSpeedButton;
    OpenDialog: TOpenDialog;
    procedure sbOpenClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
```

Listagem 16.7 Continuação

```
  end;

function GetNewLinkName(var LinkTo: string; var Dest: Integer): Boolean;

implementation

uses WinShell;

{$R *.DFM}

function GetNewLinkName(var LinkTo: string; var Dest: Integer): Boolean;
{ Obtém nome de arquivo e pasta de destino para um novo vínculo do shell. }
{ Só modifica parâmetros se Result = True. }
begin
  with TNewLinkForm.Create(Application) do
  try
    cbLocation.ItemIndex := 0;
    Result := ShowModal = mrOk;
    if Result then
    begin
      LinkTo := edLinkTo.Text;
      Dest := cbLocation.ItemIndex;
    end;
  finally
    Free;
  end;
end;

procedure TNewLinkForm.sbOpenClick(Sender: TObject);
begin
  if OpenDialog.Execute then
    edLinkTo.Text := OpenDialog.FileName;
end;

procedure TNewLinkForm.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := Low(SpecialFolders) to High(SpecialFolders) do
    cbLocation.Items.Add(SpecialFolders[I].Name);
end;

end.
```

Listagem 16.8 `PickU.pas` – a unidade com formulário que permite que o usuário escolha o local do vínculo

```
unit PickU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, FileCtrl;

type
```

Listagem 16.8 Continuação

```
  TLinkForm = class(TForm)
    lbLinkFiles: TFileListBox;
    btnOk: TButton;
    btnCancel: TButton;
    cbLocation: TComboBox;
    Label1: TLabel;
    procedure lbLinkFilesDblClick(Sender: TObject);
    procedure cbLocationChange(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  end;

function GetLinkFile(var S: String): Boolean;

implementation

{$R *.DFM}

uses WinShell, ShlObj;

function GetLinkFile(var S: String): Boolean;
{ Retorna nome de arquivo de vínculo em S. }
{ Só modifica S quando Result é True. }
begin
  with TLinkForm.Create(Application) do
    try
      { Certifica-se de que o local está selecionado }
      cbLocation.ItemIndex := 0;
      { Obtém caminho do local selecionado }
      cbLocationChange(nil);
      Result := ShowModal = mrOk;
      { Retorna nome de caminho completo do arquivo de vínculo }
      if Result then
        S := lbLinkFiles.Directory + '\' +
          lbLinkFiles.Items[lbLinkFiles.ItemIndex];
    finally
      Free;
    end;
end;

procedure TLinkForm.lbLinkFilesDblClick(Sender: TObject);
begin
  ModalResult := mrOk;
end;

procedure TLinkForm.cbLocationChange(Sender: TObject);
var
  Folder: Integer;
begin
  { Obtém caminho do local selecionado }
  Folder := SpecialFolders[cbLocation.ItemIndex].ID;
  lbLinkFiles.Directory := GetSpecialFolderPath(Folder, False);
end;

procedure TLinkForm.FormCreate(Sender: TObject);
var
  I: Integer;
begin
```

Listagem 16.8 Continuação

```
  for I := Low(SpecialFolders) to High(SpecialFolders) do
    cbLocation.Items.Add(SpecialFolders[I].Name);
end;

end.
```

Extensões do shell

Última palavra em termos de extensibilidade, o shell do Windows fornece um meio para você desenvolver um código capaz de ser executado dentro do próprio processo e namespace do shell. As *extensões do shell* são implementadas como servidores COM in-process, que são criados e usados pelo shell.

> NOTA
> Como as extensões do shell no fundo não passam de servidores COM, você só conseguirá entender o que são se conhecer pelo menos um pouco do COM. Se você não tem a menor idéia do que seja COM, o Capítulo 15 explica os fundamentos.

Diversos tipos de extensões do shell estão disponíveis para lidar com uma série de aspectos do shell. Também conhecida como *tratador* (ou *handler*), uma extensão do shell deve implementar uma ou mais interfaces do COM. O shell tem suporte para os seguintes tipos de extensões do shell:

- *Tratadores copy hook* implementam a interface `ICopyHook`. Essas extensões do shell permitem que você receba notificações sempre que uma pasta é copiada, excluída, movida ou renomeada e para opcionalmente impedir que a operação ocorra.
- *Tratadores de menu de contexto* implementam as interfaces `IContextMenu` e `IShellExtInit`. Essas extensões do shell permitem que você adicione itens ao menu de contexto de um determinado objeto de arquivo no shell.
- *Tratadores de arrastar e soltar* também implementam as interface `IContextMenu` e `IShellExtInit`. A implementação dessas extensões do shell é quase idêntica à dos tratadores de menu de contexto, exceto pelo fato de serem chamadas quando um usuário arrasta um objeto e o solta em um novo local.
- *Tratadores de ícones* implementam as interfaces `IExtractIcon` e `IPersistFile`. Os tratadores de ícones permitem que você forneça diferentes ícones para instâncias múltiplas do mesmo tipo de objeto de arquivo.
- *Tratadores de folha de propriedades* implementam as interfaces `IShellPropSheetExt` e `IShellExtInit` e permitem que você adicione páginas à caixa de diálogo de propriedades associada a um tipo de arquivo.
- *Tratadores de soltar no destino* implementam as interfaces `IDropTarget` e `IPersistFile`. Essas extensões do shell permitem que você controle o que acontece quando arrasta um objeto de shell sobre outro.
- *Tratadores de objeto de dados* implementam as interfaces `IDataObject` e `IPersistFile` e fornecem o objeto de dados quando arquivos estão sendo arrastados e soltos ou copiados e colados.

Depurando as extensões do shell

Antes de começarmos a discutir sobre a escrita de extensões do shell, considere a questão da depuração das extensões do shell. Como as extensões do shell são executadas de dentro do próprio processo do shell, como é possível criar um "gancho" para o shell para depurar as extensões do shell?

A solução para o problema é baseada no fato de que o shell é um executável (não muito diferente do que qualquer outra aplicação), chamado `explorer.exe`. No entanto, `explorer.exe` possui uma propriedade exclusiva: a primeira instância de `explorer.exe` chamará o shell. As instâncias subseqüentes simplesmente chamarão as janelas "Explorer" no shell.

Usando um macete pouco conhecido no shell, é possível fechar o shell sem fechar o Windows. Siga estas etapas para depurar suas extensões do shell no Delphi:

1. Torne `explorer.exe` a aplicação host para a sua extensão do shell na caixa de diálogo que aparece após a seleção de Run, Parameters. Certifique-se de incluir o caminho completo (ou seja, `c:\windows\explorer.exe`).
2. No menu Start (iniciar) do shell, selecione Shut Down (desligar). Isso chamará a caixa de diálogo Shut Down Windows (desligar Windows).
3. Na caixa de diálogo Shut Down Windows, mantenha pressionadas as teclas Ctrl+Alt+Shift e dê um clique no botão No (não). Isso fechará o shell sem fechar o Windows.
4. Usando Alt+Tab, volte para o Delphi e execute a extensão do shell. Isso chamará uma nova cópia do shell, sendo executada no depurador do Delphi. Agora você pode definir pontos de interrupção no seu código e depurar como sempre.
5. Quando você estiver pronto para fechar o Windows, poderá fazê-lo de modo apropriado sem o uso do shell: use Ctrl+Esc para chamar a janela Tasks (tarefas) e em seguida selecione Windows, Shutdown Windows; o Windows será desligado.

O restante deste capítulo é dedicado a mostrar uma descrição mais detalhada das extensões do shell que acabamos de descrever. Você vai aprender sobre tratadores de copy hook, tratadores de menu de contexto e tratadores de ícones.

O COM Object Wizard

Antes de discutirmos sobre cada uma das DLLs de extensão do shell, temos que falar um pouco sobre o modo como são criadas. Como as extensões do shell são servidores COM in-process, você pode deixar a IDE fazer a maior parte do trabalho pesado na criação do código-fonte. Em todas as extensões do shell, o trabalho começa com as mesmas duas etapas:

1. Selecione ActiveX Library (biblioteca ActiveX) na página ActiveX da caixa de diálogo New Items (novos itens). Isso criará uma nova DLL de servidor COM, na qual você pode inserir objetos COM.
2. Selecione COM Object (objeto COM) na página ActiveX da caixa de diálogo New Items. Isso chamará o COM Server Wizard (assistente de servidor COM). Na caixa de diálogo do assistente, digite um nome e uma descrição para a extensão do shell e selecione o modelo de threading Apartment. Quando você der um clique em OK, será gerada uma nova unidade, contendo o código do objeto COM.

Tratadores de `copy hook`

Como já dissemos, as extensões do shell de `copy hook` permitem que você instale um tratador que recebe notificações sempre que uma pasta é copiada, excluída, movida ou renomeada. Depois de receber essa notificação, o tratador tem a opção de impedir que a operação ocorra. Observe que o tratador só é chamado para objetos de pasta e impressora; ele não é chamado para arquivos e outros objetos.

A primeira etapa na criação de um tratador de copy hook é criar um objeto que descende de TComObject e implementa a interface ICopyHook. Essa interface é definida na unidade ShlObj da seguinte maneira:

```
type
  ICopyHook = interface(IUnknown)
    ['{000214EF-0000-0000-C000-000000000046}']
    function CopyCallback(Wnd: HWND; wFunc, wFlags: UINT;
      pszSrcFile: PAnsiChar; dwSrcAttribs: DWORD; pszDestFile: PAnsiChar;
      dwDestAttribs: DWORD): UINT; stdcall;
  end;
```

O método CopyCallback( )

Como você pode ver, ICopyHook é uma interface bastante simples e implementa apenas uma função: CopyCallback( ). Essa função será chamada sempre que uma pasta do shell for manipulada. Os próximos parágrafos descrevem os parâmetros dessa função.

Wnd é a alça da janela que o tratador de copy hook deve usar como o pai de qualquer janela que ele apresente. wFunc indica a operação que está sendo executada. Isso pode ser qualquer um dos valores mostrados na Tabela 16.5.

Tabela 16.5 Os valores de wFunc para CopyCallback( )

Constante	*Valor*	*Significado*
FO_COPY	$2	Copia o arquivo especificado por pszSrcFile para o local especificado por pszDestFile.
FO_DELETE	$3	Exclui o arquivo especificado por pszSrcFile.
FO_MOVE	$1	Move o arquivo especificado por pszSrcFile para o local especificado por pszDestFile.
FO_RENAME	$4	Troca o nome do arquivo especificado por pszSrcFile.
PO_DELETE	$13	Exclui a impressora especificada por pszSrcFile.
PO_PORTCHANGE	$20	Muda a porta da impressora. Os parâmetros pszSrcFile e pszDestFile contêm listas de strings terminadas em duplo nulo que são repetidas. Cada lista contém o nome da impressora seguido pelo nome da porta. O nome da porta em pszSrcFile é a porta atual da impressora e o nome da porta em pszDestFile é a nova porta da impressora.
PO_RENAME	$14	Troca o nome da impressora especificada por pszSrcFile.
PO_REN_PORT	$34	Uma combinação de PO_RENAME e PO_PORTCHANGE.

wFlags mantém os flags que controlam a operação. Esse parâmetro pode ser uma combinação dos valores mostrados na Tabela 16.6.

Tabela 16.6 Os valores de wFlags para CopyCallback( )

Constante	*Valor*	*Significado*
FOF_ALLOWUNDO	$40	Preserva informações de undo (quando possível).
FOF_MULTIDESTFILES	$1	A função SHFileOperation( ) especifica múltiplos arquivos de destino (um para cada arquivo de origem), e não um diretório onde todos os arquivos de origem devem ser depositados. Um tratador de copy hook geralmente ignora esse valor.

Tabela 16.6 Continuação

Constante	*Valor*	*Significado*
FOF_NOCONFIRMATION	$10	Responde com "Yes to All" (sim para todos) para qualquer caixa de diálogo exibida.
FOF_NOCONFIRMMKDIR	$200	Não confirma a criação dos diretórios necessários no caso de a operação exigir que um novo diretório seja criado.
FOF_RENAMEONCOLLISION	$8	Atribui um nome novo ao arquivo que está sendo processado (como "Cópia 1 de...") em uma operação de cópia, movimentação ou renomeação quando já houver um arquivo com o nome do destino.
FOF_SILENT	$4	Não exibe uma caixa de diálogo de progresso.
FOF_SIMPLEPROGRESS	$100	Exibe uma caixa de diálogo de progresso, mas a caixa de diálogo não mostra os nomes dos arquivos.

`pszSourceFile` é o nome da pasta de origem, `dwSrcAttribs` contém os atributos da pasta de origem, `pszDestFile` é o nome da pasta de destino e `dwDestAttribs` contém os atributos da pasta de destino.

Ao contrário da maioria dos métodos, essa interface não retorna um código de resultado OLE. Em vez disso, ela deve retornar um dos valores listados na Tabela 16.7, conforme definidos na unidade `Windows`.

Tabela 16.7 Os valores de `wFlags` de `CopyCallback( )`

Constante	*Valor*	*Significado*
IDYES	6	Permite a operação
IDNO	7	Impede a operação neste arquivo, mas continua com as outras operações (por exemplo, uma operação de cópia em lote).
IDCANCEL	2	Impede a operação atual e cancela quaisquer operações pendentes.

Implementação de `TCopyHook`

Sendo um objeto que implementa uma interface com um método, não há muita coisa em `TCopyHook`:

```
type
  TCopyHook = class(TComObject, ICopyHook)
  protected
    function CopyCallback(Wnd: HWND; wFunc, wFlags: UINT;
      pszSrcFile: PAnsiChar;
      dwSrcAttribs: DWORD; pszDestFile: PAnsiChar; dwDestAttribs: DWORD): UINT;
      stdcall;
  end;
```

A implementação do método `CopyCallback( )` também é pequena. A função `MessageBox( )` da API é chamada para confirmar qualquer que seja a operação que está sendo tentada. Convenientemente, o valor de retorno de `MessageBox( )` será igual ao valor de retorno desse método:

```
function TCopyHook.CopyCallback(Wnd: HWND; wFunc, wFlags: UINT;
  pszSrcFile: PAnsiChar; dwSrcAttribs: DWORD; pszDestFile: PAnsiChar;
```

```
  dwDestAttribs: DWORD): UINT;
const
  MyMessage: string = 'Are you sure you want to mess with "%s"?';
begin
  // confirma operação
  Result := MessageBox(Wnd,  PChar(Format(MyMessage, [pszSrcFile])),
    'DDG Shell Extension', MB_YESNO);
end;
```

> **DICA**
>
> Você pode estar se perguntando por que a função `MessageBox( )` da API é usada para exibir uma mensagem em vez de se usar uma função do Delphi, como `MessageDlg( )` ou `ShowMessage( )`. A razão é simples: tamanho e eficiência. A chamada de qualquer função fora da unidade Dialogs ou Forms faria com que uma grande parte da VCL fosse vinculada à DLL. Mantendo essas unidades fora da cláusula `uses`, a DLL da extensão do shell ocupa irrisórios 70KB.

Acredite se quiser, mas isso é tudo que há para se falar sobre o objeto `TCopyHook` propriamente dito. No entanto, ainda há um importante trabalho a ser feito antes de algum dia ele poder ser chamado: a extensão do shell deve ser registrada com o System Registry antes de ele poder funcionar.

Registro

Além do registro normal exigido de qualquer servidor COM, um tratador de `copy hook` deve ter uma entrada adicional no Registro, sob

```
HKEY_CLASSES_ROOT\directory\shellex\CopyHookHandlers
```

Além disso, o Windows NT exige que todas as extensões do shell sejam registradas, conforme as extensões do shell aprovadas, sob

```
HKEY_LOCAL_MACHINE\ SOFTWARE\Microsoft\Windows\CurrentVersion
➥\Shell Extensions\Approved
```

Você pode utilizar várias técnicas para registrar as extensões do shell: elas podem ser registradas através de um arquivo REG ou através de um programa de instalação. A DLL da extensão do shell propriamente dita pode ser auto-registrável. Embora isso signifique um pouco mais de trabalho, a melhor solução é tornar cada DLL de extensão do shell auto-registrável. Isso é mais legível, pois cria sua extensão do shell em um pacote de um arquivo independente.

Conforme você aprendeu no Capítulo 15, os objetos COM são sempre criados a partir de fábricas de classes. Dentro da estrutura da VCL, os objetos de fábrica de classes também são responsáveis pelo registro do objeto COM que criarão. Se um objeto COM requer entradas de Registro personalizadas (como é o caso com uma extensão do shell), a definição dessas entradas é só uma questão de modificar o método `UpdateRegistry( )` da fábrica de classes. A Listagem 16.9 mostra a unidade `CopyMain` completa, que inclui uma fábrica de classes especializada para executar um registro personalizado.

Listagem 16.9 `CopyMain` – unidade principal da implementação de copy hook

```
unit CopyMain;

interface

uses Windows, ComObj, ShlObj;
```

Listagem 16.9 Continuação

```
type
  TCopyHook = class(TComObject, ICopyHook)
  protected
    function CopyCallback(Wnd: HWND; wFunc, wFlags: UINT; pszSrcFile: PAnsiChar; dwSrcAttribs:
      DWORD; pszDestFile: PAnsiChar; dwDestAttribs: DWORD): UINT; stdcall;
  end;

  TCopyHookFactory = class(TComObjectFactory)
  protected
    function GetProgID: string; override;
    procedure ApproveShellExtension(Register: Boolean; const ClsID: string);
      virtual;
  public
    procedure UpdateRegistry(Register: Boolean); override;
  end;

implementation

uses ComServ, SysUtils, Registry;

{ TCopyHook }

// Este é o método que é chamado pelo shell para operações de pasta
function TCopyHook.CopyCallback(Wnd: HWND; wFunc, wFlags: UINT;
  pszSrcFile: PAnsiChar; dwSrcAttribs: DWORD; pszDestFile: PAnsiChar;
  dwDestAttribs: DWORD): UINT;
const
  MyMessage: string = 'Are you sure you want to mess with "%s"?';
begin
  // confirma operação
  Result := MessageBox(Wnd,  PChar(Format(MyMessage, [pszSrcFile])),
    'DDG Shell Extension', MB_YESNO);
end;

{ TCopyHookFactory }

function TCopyHookFactory.GetProgID: string;
begin
  // ProgID não necessário para extensão do shell
  Result := '';
end;

procedure TCopyHookFactory.UpdateRegistry(Register: Boolean);
var
  ClsID: string;
begin
  ClsID := GUIDToString(ClassID);
  inherited UpdateRegistry(Register);
  ApproveShellExtension(Register, ClsID);
  if Register then
    // adicionar clsid da extensão do shell à entrada de CopyHookHandlers
    CreateRegKey('directory\shellex\CopyHookHandlers\' + ClassName, '',
      ClsID)
  else
    DeleteRegKey('directory\shellex\CopyHookHandlers\' + ClassName);
end;

procedure TCopyHookFactory.ApproveShellExtension(Register: Boolean;
  const ClsID: string);
// Esta entrada do registro é necessária para que a extensão opere
```

Listagem 16.9 Continuação

```
// corretamente no Windows NT.
const

  SApproveKey = 'SOFTWARE\Microsoft\Windows\CurrentVersion\Shell
➥Extensions\Approved';

begin
  with TRegistry.Create do
    try
      RootKey := HKEY_LOCAL_MACHINE;
      if not OpenKey(SApproveKey, True) then Exit;
      if Register then WriteString(ClsID, Description)
      else DeleteValue(ClsID);
    finally
      Free;
    end;
end;

const
  CLSID_CopyHook: TGUID = '{66CD5F60-A044-11D0-A9BF-00A024E3867F}';

initialization
  TCopyHookFactory.Create(ComServer, TCopyHook, CLSID_CopyHook,
    'DDG_CopyHook', 'DDG Copy Hook Shell Extension Example',
    ciMultiInstance, tmApartment);
end.
```

O que faz a fábrica de classes `TCopyHookFactory` funcionar é o fato de uma instância dela, não o `TComObjectFactory` normal, estar sendo criada na parte `initialization` da unidade. A Figura 16.7 mostra o que acontece quando você tenta trocar o nome de uma pasta no shell depois que a DLL da extensão do shell `copy hook` é instalada.

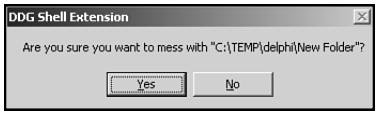

Figura 16.7 O tratador de copy hook em ação.

Tratadores de menu de contexto

Os tratadores de menu de contexto permitem que você adicione itens ao menu local, que estão associados a objetos de arquivo no shell. Um exemplo de menu local para um arquivo EXE aparece na Figura 16.8.

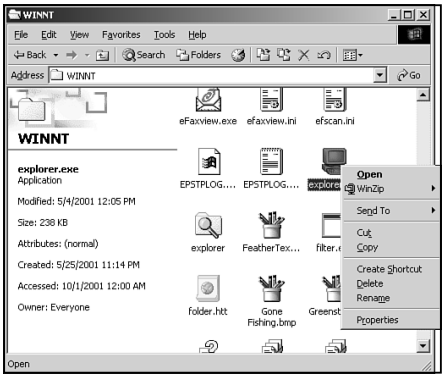

Figura 16.8 O menu local do shell para um arquivo EXE.

As extensões do shell para o menu de contexto funcionam implementando as interfaces `IShellExtInit` e `IContextMenu`. Nesse caso, implementaremos essas interfaces para criar um tratador de menu de contexto para os arquivos BPL (Borland Package Library); o menu local para arquivos de pacote no shell fornecerá uma opção para a obtenção de informações sobre o pacote. Esse objeto tratador de menu de contexto será chamado `TContextMenu` e, como o tratador de copy hook, `TContextMenu` descenderá de `TComObject`.

IShellExtInit

A interface `IShellExtInit` é usada para inicializar uma extensão do shell. Essa interface é definida na unidade `ShlObj` da seguinte maneira:

```
type
  IShellExtInit = interface(IUnknown)
    ['{000214E8-0000-0000-C000-000000000046}']
    function Initialize(pidlFolder: PItemIDList; lpdobj: IDataObject;
      hKeyProgID: HKEY): HResult; stdcall;
  end;
```

`Initialize( )`, sendo o único método dessa interface, é chamado para inicializar o tratador do menu de contexto. Os próximos parágrafos descrevem os parâmetros desse método.

`pidlFolder` é um ponteiro para uma estrutura `PItemIDList` (lista de identificadores de item) para a pasta que contém o item cujo menu de contexto está sendo exibido. `lpdobj` mantém o objeto de interface `IDataObject`, usado para recuperar o objeto sobre o qual a ação está sendo executada. `hkeyProgID` contém a chave do Registro para o objeto de arquivo ou tipo de pasta.

A implementação desse método é mostrada no código a seguir. À primeira vista, o código pode parecer complexo, mas na realidade ele se reduz a três coisas: uma chamada para `lpobj.GetData( )` para obter dados de `IDataObject` e duas chamadas para `DragQueryFile( )` (uma chamada para obter o número de arquivos e outra para obter o nome do arquivo). O nome do arquivo é armazenado no campo `FFileName` do objeto. Veja o código a seguir:

```
function TContextMenu.Initialize(pidlFolder: PItemIDList; lpdobj: IDataObject;
  hKeyProgID: HKEY): HResult;
var
  Medium: TStgMedium;
  FE: TFormatEtc;
begin
  try
    // Falha na chamada se lpdobj for nulo.
    if lpdobj = nil then
    begin
      Result := E_FAIL;
      Exit;
    end;
    with FE do
    begin
      cfFormat := CF_HDROP;
      ptd := nil;
      dwAspect := DVASPECT_CONTENT;
      lindex := -1;
      tymed := TYMED_HGLOBAL;
    end;
    // Apresenta os dados referenciados pelo ponteiro IDataObject em um meio de
    // armazenamento HGLOBAL no formato CF_HDROP.
    Result := lpdobj.GetData(FE, Medium);
    if Failed(Result) then Exit;
    try
```

```
      // Se apenas um arquivo estiver selecionado, apanha o nome do arquivo
      // e o armazena em szFile. Caso contrário, falha na chamada.
      if DragQueryFile(Medium.hGlobal, $FFFFFFFF, nil, 0) = 1 then
      begin
        DragQueryFile(Medium.hGlobal, 0, FFileName, SizeOf(FFileName));
        Result := NOERROR;
      end
      else
        Result := E_FAIL;
    finally
      ReleaseStgMedium(medium);
    end;
  except
    Result := E_UNEXPECTED;
  end;
end;
```

IContextMenu

A interface `IContextMenu` é usada para manipular o menu instantâneo associado a um arquivo no shell. Essa interface é definida na unidade `ShlObj` da seguinte maneira:

```
type
  IContextMenu = interface(IUnknown)
    ['{000214E4-0000-0000-C000-000000000046}']
    function QueryContextMenu(Menu: HMENU;
      indexMenu, idCmdFirst, idCmdLast, uFlags: UINT): HResult; stdcall;
    function InvokeCommand(var lpici: TCMInvokeCommandInfo): HResult; stdcall;
    function GetCommandString(idCmd, uType: UINT; pwReserved: PUINT;
      pszName: LPSTR; cchMax: UINT): HResult; stdcall;
  end;
```

Depois que o tratador for inicializado através da interface `IShellExtInit`, o próximo método a ser chamado é `IContextMenu.QueryContextMenu( )`. Os parâmetros passados para esse método incluem uma alça de menu, o índice no qual o primeiro item de menu será inserido, os valores mínimo e máximo dos IDs de item de menu e flags que indicam os atributos de menu. A implementação de `TContextMenu` desse método, mostrada a seguir, adiciona um item de menu com o texto "Package Info..." à alça de menu passada no parâmetro `Menu` (observe que o valor de retorno de `QueryContextMenu( )` é o índice do último item de menu inserido mais um):

```
function TContextMenu.QueryContextMenu(Menu: HMENU; indexMenu, idCmdFirst,
  idCmdLast, uFlags: UINT): HResult;
begin
  FMenuIdx := indexMenu;
  // Acrescenta um item ao menu de contexto
  InsertMenu (Menu, FMenuIdx, MF_STRING or MF_BYPOSITION, idCmdFirst,
    'Package Info...');
  // Retorna índice do último item inserirdo + 1
  Result := FMenuIdx + 1;
end;
```

O próximo método chamado pelo shell é `GetCommandString( )`. O objetivo desse método é recuperar a string de comando ou ajuda independente de linguagem de um determinado item de menu. Os parâmetros desse método incluem o offset do item de menu, flags indicando o tipo de informação a ser recebida, um parâmetro reservado e um buffer de string com o tamanho do buffer. A implementação de `TContextMenu` desse método, mostrada a seguir, só precisa lidar com o fornecimento da string de ajuda para o item de menu:

```
function TContextMenu.GetCommandString(idCmd, uType: UINT; pwReserved: PUINT;
  pszName: LPSTR; cchMax: UINT): HRESULT;
begin
  Result := S_OK;
  try
    // certifica-se de que o índice de menu está correto
    // e de que o shell está solicitando a string de ajuda
    if (idCmd = FMenuIdx) and ((uType and GCS_HELPTEXT) < > 0) then
      // retorna string de ajuda para o item de menu
      StrLCopy(pszName, 'Get information for the selected package.', cchMax)
    else
      Result := E_INVALIDARG;
  except
    Result := E_UNEXPECTED;
  end;
end;
```

Quando você dá um clique no novo item no menu de contexto, o shell chamará o método `InvokeCommand( )`. O método aceita um registro `TCMInvokeCommandInfo` como parâmetro. Esse registro é definido na unidade `ShlObj` da seguinte maneira:

```
type
  PCMInvokeCommandInfo = ^TCMInvokeCommandInfo;
  TCMInvokeCommandInfo = packed record
    cbSize: DWORD;        { deve ser SizeOf(TCMInvokeCommandInfo) }
    fMask: DWORD;         { qualquer combinação de CMIC_MASK_* }
    hwnd: HWND;           { pode ser NULL (ausência de janela de proprietário) }
    lpVerb: LPCSTR;       { uma string de MAKEINTRESOURCE(idOffset) }
    lpParameters: LPCSTR; { pode ser NULL (ausência de parâmetro) }
    lpDirectory: LPCSTR;  { pode ser NULL (ausência de diretório específico) }
    nShow: Integer;       { um dos valores SW_ da API ShowWindow( ) API }
    dwHotKey: DWORD;
    hIcon: THandle;
  end;
```

A palavra de baixa ordem ou o campo `lpVerb` conterá o índice do item de menu selecionado. Veja, a seguir, a implementação desse método:

```
function TContextMenu.InvokeCommand(var lpici: TCMInvokeCommandInfo): HResult;
begin
  Result := S_OK;
  try
    // Certifica-se de que não estamos sendo chamados por uma aplicação
    if HiWord(Integer(lpici.lpVerb)) < > 0 then
    begin
      Result := E_FAIL;
      Exit;
    end;
    // Executa o comando especificado por lpici.lpVerb.
    // Retorna E_INVALIDARG se passarmos um número de argumentos inválido.
    if LoWord(lpici.lpVerb) = FMenuIdx then
      ExecutePackInfoApp(FFileName, lpici.hwnd)
    else
      Result := E_INVALIDARG;
  except
    MessageBox(lpici.hwnd, 'Error obtaining package information.', 'Error',
      MB_OK or MB_ICONERROR);
    Result := E_FAIL;
  end;
end;
```

Se tudo correr bem, a função `ExecutePackInfoApp( )` é convocada para chamar a aplicação `PackInfo.exe`, que exibe diversas informações sobre um pacote. Não vamos entrar nas particularidades dessa aplicação agora; no entanto, ela é discutida em detalhes no Capítulo 13 da versão eletrônica de *Delphi 5 Guia do Desenvolvedor*, que se encontra neste CD-ROM.

Registro

Os tratadores de menu de contexto devem ser registrados sob

`HKEY_CLASSES_ROOT\<`*`tipo de arquivo`*`>\shellex\ContextMenuHandlers`

no Registro do sistema. Seguindo o modelo da extensão de `copy hook`, a capacidade de registro é adicionada à DLL criando-se um descendente de `TComObject` especializado. O objeto é mostrado na Listagem 16.10, juntamente com todo o código-fonte completo da unidade que contém `TContextMenu`. A Figura 16.9 mostra o menu local do arquivo BPL com o novo item, e a Figura 16.10 mostra a janela `PackInfo.exe` do modo como é chamada pelo tratador do menu de contexto.

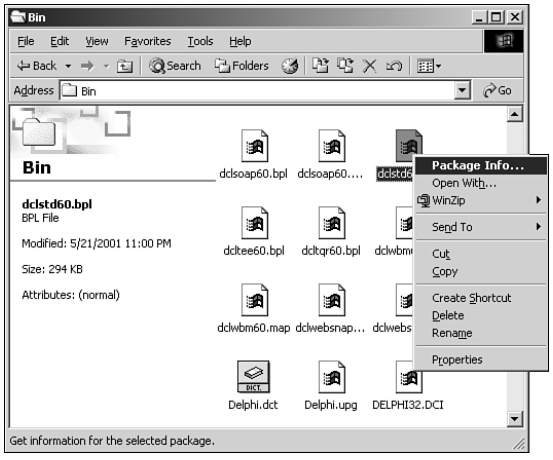

Figura 16.9 O tratador de menu de contexto em ação.

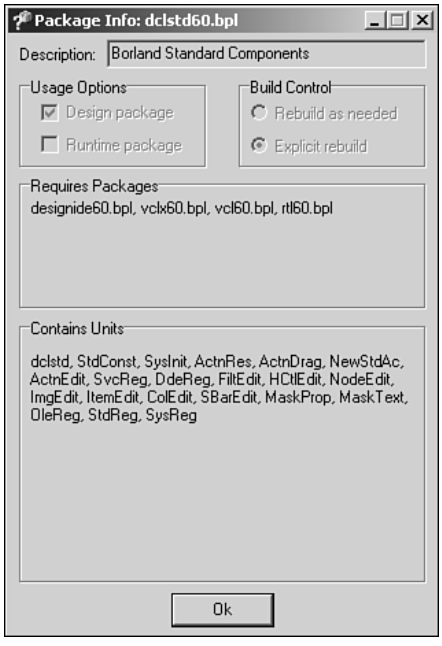

Figura 16.10 Obtendo informações de pacote do tratador de menu de contexto.

Listagem 16.10 `ContMain` – a unidade principal da implementação do tratador de menu de contexto

```
unit ContMain;

interface

uses Windows, ComObj, ShlObj, ActiveX;

type
  TContextMenu = class(TComObject, IContextMenu, IShellExtInit)
  private
    FFileName: array[0..MAX_PATH] of char;
    FMenuIdx: UINT;
  protected
    // Métodos IContextMenu
    function QueryContextMenu(Menu: HMENU; indexMenu, idCmdFirst, idCmdLast,
      uFlags: UINT): HResult; stdcall;
    function InvokeCommand(var lpici: TCMInvokeCommandInfo): HResult; stdcall;
    function GetCommandString(idCmd, uType: UINT; pwReserved: PUINT;
      pszName: LPSTR; cchMax: UINT): HResult; stdcall;
    // Método IShellExtInit
    function Initialize(pidlFolder: PItemIDList; lpdobj: IDataObject;
      hKeyProgID: HKEY): HResult; reintroduce; stdcall;
  end;

  TContextMenuFactory = class(TComObjectFactory)
  protected
    function GetProgID: string; override;
    procedure ApproveShellExtension(Register: Boolean; const ClsID: string);
      virtual;
  public
    procedure UpdateRegistry(Register: Boolean); override;
  end;

implementation

uses ComServ, SysUtils, ShellAPI, Registry, Math;

procedure ExecutePackInfoApp(const FileName: string; ParentWnd: HWND);
const
  SPackInfoApp = '%sPackInfo.exe';
  SCmdLine = '"%s" %s';
  SErrorStr = 'Failed to execute PackInfo:'#13#10#13#10;
var
  PI: TProcessInformation;
  SI: TStartupInfo;
  ExeName, ExeCmdLine: string;
  Buffer: array[0..MAX_PATH] of char;
begin
  // Obtém diretório dessa DLL. Presume que EXE é executado no mesmo diretório.
  GetModuleFileName(HInstance, Buffer, SizeOf(Buffer));
  ExeName := Format(SPackInfoApp, [ExtractFilePath(Buffer)]);
  ExeCmdLine := Format(SCmdLine, [ExeName, FileName]);
  FillChar(SI, SizeOf(SI), 0);
  SI.cb := SizeOf(SI);
  if not CreateProcess(PChar(ExeName), PChar(ExeCmdLine), nil, nil, False,
    0, nil, nil, SI, PI) then
    MessageBox(ParentWnd, PChar(SErrorStr + SysErrorMessage(GetLastError)),
```

Listagem 16.10 Continuação

```
      'Error', MB_OK or MB_ICONERROR);
end;

{ TContextMenu }

{ TContextMenu.IContextMenu }

function TContextMenu.QueryContextMenu(Menu: HMENU; indexMenu, idCmdFirst,
  idCmdLast, uFlags: UINT): HResult;
begin
  FMenuIdx := indexMenu;
  // Acrescenta um item de menu ao menu de contexto
  InsertMenu (Menu, FMenuIdx, MF_STRING or MF_BYPOSITION, idCmdFirst,
    'Package Info...');
  // Retorna índice do último item inserido + 1
  Result := FMenuIdx + 1;
end;

function TContextMenu.InvokeCommand(var lpici: TCMInvokeCommandInfo): HResult;
begin
  Result := S_OK;
  try
    // Certifica-se de que não estamos sendo chamados por uma aplicação
    if HiWord(Integer(lpici.lpVerb)) <<|>> 0 then
    begin
      Result := E_FAIL;
      Exit;
    end;
    // Executa o comando especificado por lpici.lpVerb.
    // Retorna E_INVALIDARG se recebeu número de argumentos inválido.
    if LoWord(lpici.lpVerb) = FMenuIdx then
      ExecutePackInfoApp(FFileName, lpici.hwnd)
    else
      Result := E_INVALIDARG;
  except
    MessageBox(lpici.hwnd, 'Error obtaining package information.', 'Error',
      MB_OK or MB_ICONERROR);
    Result := E_FAIL;
  end;
end;

function TContextMenu.GetCommandString(idCmd, uType: UINT; pwReserved: PUINT;
  pszName: LPSTR; cchMax: UINT): HRESULT;
const
  SCmdStrA: String = 'Get information for the selected package.';
  SCmdStrW: WideString = 'Get information for the selected package.';
begin
  Result := S_OK;
  try
    // verifica se índice de menu está correto e se o shell
    // está pedindo string de ajuda
    if (idCmd = FMenuIdx) and ((uType and GCS_HELPTEXT) < > 0) then
    begin
      // retorna string de ajuda para item do menu
      if Win32MajorVersion >= 5 then  // trata como Unicode para Win2k em diante
        Move(SCmdStrW[1], pszName^,
          Min(cchMax, Length(SCmdStrW) + 1) * SizeOf(WideChar))
```

Listagem 16.10 Continuação

```
      else                              // se não, trata como ANSI
        StrLCopy(pszName, PChar(SCmdStrA), Min(cchMax, Length(SCmdStrA) + 1));
    end
    else
      Result := E_INVALIDARG;
  except
    Result := E_UNEXPECTED;
  end;
end;

{ TContextMenu.IShellExtInit }

function TContextMenu.Initialize(pidlFolder: PItemIDList; lpdobj: IDataObject;
  hKeyProgID: HKEY): HResult;
var
  Medium: TStgMedium;
  FE: TFormatEtc;
begin
  try
    // Falha na chamada se lpdobj for nulo.
    if lpdobj = nil then
    begin
      Result := E_FAIL;
      Exit;
    end;
    with FE do
    begin
      cfFormat := CF_HDROP;
      ptd := nil;
      dwAspect := DVASPECT_CONTENT;
      lindex := -1;
      tymed := TYMED_HGLOBAL;
    end;
    // Exibe os dados referenciados pelo ponteiro de IDataObject para um meio
    // de armazenamento HGLOBAL no formato CF_HDROP.
    Result := lpdobj.GetData(FE, Medium);
    if Failed(Result) then Exit;
    try
      // Se apenas um arquivo for selecionado, apanha nome do arquivo
      // e o armazena em szFile. Caso contrário, falha na chamada.
      if DragQueryFile(Medium.hGlobal, $FFFFFFFF, nil, 0) = 1 then
      begin
        DragQueryFile(Medium.hGlobal, 0, FFileName, SizeOf(FFileName));
        Result := NOERROR;
      end
      else
        Result := E_FAIL;
    finally
      ReleaseStgMedium(medium);
    end;
  except
    Result := E_UNEXPECTED;
  end;
end;

{ TContextMenuFactory }
```

Listagem 16.10 Continuação

```
function TContextMenuFactory.GetProgID: string;
begin
  // ProgID não-obrigatório para extensão de shell do menu de contexto
  Result := '';
end;

procedure TContextMenuFactory.UpdateRegistry(Register: Boolean);
var
  ClsID: string;
begin
  ClsID := GUIDToString(ClassID);
  inherited UpdateRegistry(Register);
  ApproveShellExtension(Register, ClsID);
  if Register then
  begin
    // precisa registrar .bpl como um tipo de arquivo
    CreateRegKey('.bpl', '', 'BorlandPackageLibrary');
    // registra essa DLL como tratador do menu de contexto
    // para arquivos .bpl
    CreateRegKey('BorlandPackageLibrary\shellex\ContextMenuHandlers\' +
      ClassName, '', ClsID);
  end
  else begin
    DeleteRegKey('.bpl');
    DeleteRegKey('BorlandPackageLibrary\shellex\ContextMenuHandlers\' +
      ClassName);
  end;
end;

procedure TContextMenuFactory.ApproveShellExtension(Register: Boolean;
  const ClsID: string);
// Essa entrada do Registro é obrigatória para que a extensão opere
// corretamente no Windows NT.
const
  SApproveKey = 'SOFTWARE\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved';

begin
  with TRegistry.Create do
    try
      RootKey := HKEY_LOCAL_MACHINE;
      if not OpenKey(SApproveKey, True) then Exit;
      if Register then WriteString(ClsID, Description)
      else DeleteValue(ClsID);
    finally
      Free;
    end;
end;

const
  CLSID_CopyHook: TGUID = '{7C5E74A0-D5E0-11D0-A9BF-E886A83B9BE5}';

initialization
  TContextMenuFactory.Create(ComServer, TContextMenu, CLSID_CopyHook,
    'DDG_ContextMenu', 'DDG Context Menu Shell Extension Example',
    ciMultiInstance, tmApartment);
end.
```

Tratadores de ícones

Os tratadores de ícones permitem que diferentes ícones sejam usados em várias instâncias do mesmo tipo de arquivo. Nesse exemplo, o objeto tratador de ícones `TIconHandler` fornece diferentes ícones para diferentes tipos de arquivos Borland Package (BPL). Dependendo de um pacote ser de runtime, projeto, ambos ou nenhum deles, um ícone diferente será exibido em uma pasta do shell.

Flags de pacote

Antes de obter implementações das interfaces necessárias para essa extensão do shell, reserve um momento para examinar o método que determina o tipo de um determinado arquivo de pacote. O método retorna `TPackType`, que é definida da seguinte maneira:

```
TPackType = (ptDesign, ptDesignRun, ptNone, ptRun);
```

Veja o método a seguir:

```
function TIconHandler.GetPackageType: TPackType;
var
  PackMod: HMODULE;
  PackFlags: Integer;
begin
  // Como só precisamos obter os recursos do pacote,
  // LoadLibraryEx com LOAD_LIBRARY_AS_DATAFILE fornece um meio veloz
  // e eficiente para carregar o pacote.
  PackMod := LoadLibraryEx(PChar(FFileName), 0, LOAD_LIBRARY_AS_DATAFILE);
  if PackMod = 0 then
  begin
    Result := ptNone;
    Exit;
  end;
  try
    GetPackageInfo(PackMod, nil, PackFlags, PackInfoProc);
  finally
    FreeLibrary(PackMod);
  end;
  // mascara todos os flags, exceto de projeto e execução, e retorna resultado
  case PackFlags and (pfDesignOnly or pfRunOnly) of
    pfDesignOnly: Result := ptDesign;
    pfRunOnly: Result := ptRun;
    pfDesignOnly or pfRunOnly: Result := ptDesignRun;
  else
    Result := ptNone;
  end;
end;
```

Esse método funciona chamando o método `GetPackageInfo( )` da unidade `SysUtils` para obter os flags de pacote. Um ponto interessante a ser observado a respeito da otimização do desempenho é que a função `LoadLibraryEx( )` da API é chamada, não o procedimento `LoadPackage( )` do Delphi, para carregar a biblioteca de pacotes. Internamente, o procedimento `LoadPackage( )` chama a API `LoadLibrary( )` para carregar a BPL e em seguida chama `InitializePackage( )` para executar o código de inicialização de cada uma das unidades no pacote. Como tudo o que queremos é obter os flags de pacote e, como os flags residem em um recurso vinculado à BPL, podemos seguramente carregar o pacote com `LoadLibraryEx( )`, usando o flag `LOAD_LIBRARY_AS_DATAFILE`.

Interfaces de tratador de ícones

Como já dissemos, os tratadores de ícones devem oferecer suporte para as interfaces `IExtractIcon` (definida em `ShlObj`) e `IPersistFile` (definida na unidade `ActiveX`). Essas interfaces são listadas a seguir:

```
type
  IExtractIcon = interface(IUnknown)
    ['{000214EB-0000-0000-C000-000000000046}']
    function GetIconLocation(uFlags: UINT; szIconFile: PAnsiChar; cchMax: UINT;
      out piIndex: Integer; out pwFlags: UINT): HResult; stdcall;
    function Extract(pszFile: PAnsiChar; nIconIndex: UINT;
      out phiconLarge, phiconSmall: HICON; nIconSize: UINT): HResult; stdcall;
  end;

  IPersistFile = interface(IPersist)
    ['{0000010B-0000-0000-C000-000000000046}']
    function IsDirty: HResult; stdcall;
    function Load(pszFileName: POleStr; dwMode: Longint): HResult; stdcall;
    function Save(pszFileName: POleStr; fRemember: BOOL): HResult; stdcall;
    function SaveCompleted(pszFileName: POleStr): HResult; stdcall;
    function GetCurFile(out pszFileName: POleStr): HResult; stdcall;
  end;
```

Embora aparentemente dê um bocado de trabalho, isso não acontece realmente; na verdade, apenas dois desses métodos têm que ser implementados. O primeiro arquivo que deve ser implementado é `IPersistFile.Load( )`. Esse é o método que é chamado para inicializar as extensões do shell e nele você deve salvar o nome do arquivo passado pelo parâmetro `pszFileName`. Veja a seguir a implementação `TExtractIcon` desse método:

```
function TIconHandler.Load(pszFileName: POleStr; dwMode: Longint): HResult;
begin
  // Este método é chamado para inicializar a extensão do shell do
  // tratador de ícones. Temos que salvar o nome de arquivo
  // passado em pszFileName.
  FFileName := pszFileName;
  Result := S_OK;
end;
```

O outro método que deve ser implementado é `IExtractIcon.GetIconLocation( )`. Os parâmetros desse método são discutidos nos próximos parágrafos.

`uFlags` indica o tipo de ícone a ser exibido. Esse parâmetro pode ser 0, `GIL_FORSHELL` ou `GIL_OPENICON`. `GIL_FORSHELL` significa que o ícone deve ser exibido em uma pasta do shell. `GIL_OPENICON` significa que o ícone deve estar no estado "aberto" caso as imagens para os estados aberto e fechado estejam disponíveis. Se esse flag não for especificado, o ícone deve estar no estado normal, ou "fechado". Esse flag geralmente é usado para objetos de pasta.

`szIconFile` é o buffer que recebe o local do ícone e `cchMax` é o tamanho do buffer. `piIndex` é um inteiro que recebe o índice de ícone, que dá mais detalhes quanto ao local do ícone. `pwFlags` recebe zero ou mais dos valores mostrados na Tabela 16.8.

A implementação de `GetIconLocation( )` em `TIconHandler` é mostrada a seguir:

```
function TIconHandler.GetIconLocation(uFlags: UINT; szIconFile: PAnsiChar;
  cchMax: UINT; out piIndex: Integer; out pwFlags: UINT): HResult;
begin
  Result := S_OK;
  try
    // retorna a DLL do nome do módulo para localizar ícone
```

```
    GetModuleFileName(HInstance, szIconFile, cchMax);
    // instrui o shell a não fazer cache dos bits de imagem, caso
    // o ícone mude e cada instância possa ter seu próprio ícone
    pwFlags := GIL_DONTCACHE or GIL_PERINSTANCE;
    // índice do ícone coincide com TPackType
    piIndex := Ord(GetPackageType);
  except
    // se houver um erro, usa o ícone de pacote padrão
    piIndex := Ord(ptNone);
  end;
end;
```

Os ícones são vinculados à DLL da extensão do shell como um arquivo de recurso e, portanto, o nome do arquivo atual, retornado por `GetModuleFileName( )`, é escrito no buffer `szIconFile`. Além disso, os ícones são arranjados de um modo que o índice de um tipo de pacote corresponda ao índice do tipo de pacote na enumeração `TPackType` e, portanto, o valor de retorno de `GetPackageType( )` seja atribuído a `piIndex`.

Tabela 16.8 Os valores de **`pwFlags`** de **`GetIconLocation( )`**

Flag	*Significado*
`GIL_DONTCACHE`	Os bits da imagem física desse ícone não devem ser colocados em cache pelo responsável pela chamada. Essa diferença deve ser levada em consideração, pois um flag `GIL_DONTCACHELOCATION` pode ser introduzido em futuras versões do shell.
`GIL_NOTFILENAME`	A localização não é um par nome de arquivo/índice. Os responsáveis pela chamada que decidem extrair o ícone do local devem chamar o método `IExtractIcon.Extract( )` desse objeto para obter as imagens de ícone desejadas.
`GIL_PERCLASS`	Todos os objetos dessa classe têm o mesmo ícone. Esse flag é usado internamente pelo shell. Geralmente, as implementações de `IExtractIcon` não exigem esse flag, pois ele significa que um tratador de ícones não é necessário para apanhar o ícone de cada objeto. O método recomendado para implementação de ícones por classe é registrar um ícone padrão para a classe.
`GIL_PERINSTANCE`	Cada objeto dessa classe tem seu próprio ícone. Esse flag é usado internamente pelo shell para cuidar de casos como `setup.exe`, onde mais de um objeto com nomes idênticos podem ser conhecidos ao shell e usam diferentes ícones. Implementações típicas de `IExtractIcon` não exigem esse flag.
`GIL_SIMULATEDOC`	O responsável pela chamada deve criar um ícone de documento usando o ícone especificado.

Registro

Os tratadores de ícones devem ser registrados na chave do Registro

`HKEY_CLASSES_ROOT\<`*`tipo de arquivo`*`>\shellex\IconHandler.`

Mais uma vez, um descendente de `TComObjectFactory` é criado para lidar com o registro dessa extensão do shell. Isso é mostrado na Listagem 16.11, juntamente com o resto do código-fonte do tratador de ícones.

A Figura 16.11 mostra uma pasta do shell contendo pacotes de diferentes tipos. Observe os diferentes ícones de pacotes.

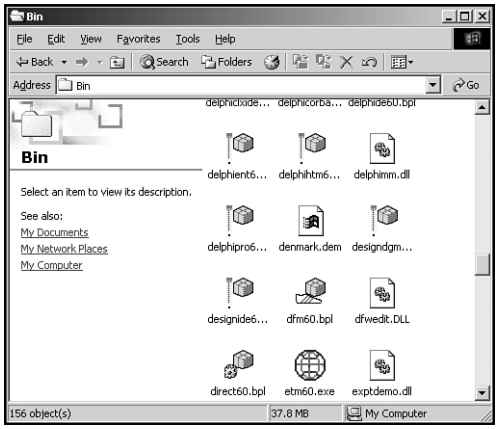

Figura 16.11 O resultado do uso do tratador de ícones.

Listagem 16.11 `IconMain.pas` – a unidade principal da implementação do tratador de ícones

```
unit IconMain;

interface

uses Windows, ActiveX, ComObj, ShlObj;

type
  TPackType = (ptDesign, ptDesignRun, ptNone, ptRun);

  TIconHandler = class(TComObject, IExtractIcon, IPersistFile)
  private
    FFileName: string;
    function GetPackageType: TPackType;
  protected
    // Métodos de IExtractIcon
    function GetIconLocation(uFlags: UINT; szIconFile: PAnsiChar; cchMax: UINT;
      out piIndex: Integer; out pwFlags: UINT): HResult; stdcall;
    function Extract(pszFile: PAnsiChar; nIconIndex: UINT;
      out phiconLarge, phiconSmall: HICON; nIconSize: UINT): HResult; stdcall;
    // Método de IPersist
    function GetClassID(out classID: TCLSID): HResult; stdcall;
    // Métodos de IPersistFile
    function IsDirty: HResult; stdcall;
    function Load(pszFileName: POleStr; dwMode: Longint): HResult; stdcall;
    function Save(pszFileName: POleStr; fRemember: BOOL): HResult; stdcall;
    function SaveCompleted(pszFileName: POleStr): HResult; stdcall;
    function GetCurFile(out pszFileName: POleStr): HResult; stdcall;
  end;

  TIconHandlerFactory = class(TComObjectFactory)
  protected
    function GetProgID: string; override;
    procedure ApproveShellExtension(Register: Boolean; const ClsID: string);
      virtual;
  public
    procedure UpdateRegistry(Register: Boolean); override;
  end;

implementation
```

Listagem 16.11 Continuação

```
uses SysUtils, ComServ, Registry;

{ TIconHandler }

procedure PackInfoProc(const Name: string; NameType: TNameType; Flags: Byte;
  Param: Pointer);
begin
  // não precisamos implementar esse procedimento, pois só estamos interessados
  // em flags de pacote, não em unidades contidas e pacotes obrigatórios.
end;

function TIconHandler.GetPackageType: TPackType;
var
  PackMod: HMODULE;
  PackFlags: Integer;
begin
  // Como só precisamos ter acesso aos recursos do pacote,
  // LoadLibraryEx com LOAD_LIBRARY_AS_DATAFILE fornecem um
  // meio de acesso rápido e eficiente para carregar o pacote.
  PackMod := LoadLibraryEx(PChar(FFileName), 0, LOAD_LIBRARY_AS_DATAFILE);
  if PackMod = 0 then
  begin
    Result := ptNone;
    Exit;
  end;
  try
    GetPackageInfo(PackMod, nil, PackFlags, PackInfoProc);
  finally
    FreeLibrary(PackMod);
  end;
  // elimina a máscara de tudo, exceto os flags de execução
  // e projeto, e retorna resultado
  case PackFlags and (pfDesignOnly or pfRunOnly) of
    pfDesignOnly: Result := ptDesign;
    pfRunOnly: Result := ptRun;
    pfDesignOnly or pfRunOnly: Result := ptDesignRun;
  else
    Result := ptNone;
  end;
end;

{ TIconHandler.IExtractIcon }

function TIconHandler.GetIconLocation(uFlags: UINT; szIconFile: PAnsiChar;
  cchMax: UINT; out piIndex: Integer; out pwFlags: UINT): HResult;
begin
  Result := S_OK;
  try
    // retorna esta DLL para nome do módulo para localizar ícone
    GetModuleFileName(HInstance, szIconFile, cchMax);
    // instrui o shell a não armazenar bits de imagem no cache, caso
    // o ícone mude e cada instância possa ter seu próprio ícone
    pwFlags := GIL_DONTCACHE or GIL_PERINSTANCE;
    // índice do ícone coincide com TPackType
    piIndex := Ord(GetPackageType);
  except
    // se houver um erro, usa o ícone default do pacote
```

Listagem 16.11 Continuação

```
    piIndex := Ord(ptNone);
  end;
end;

function TIconHandler.Extract(pszFile: PAnsiChar; nIconIndex: UINT;
  out phiconLarge, phiconSmall: HICON; nIconSize: UINT): HResult;
begin
  // Este método só precisa ser implementado se o ícone for armazenado
  // em algum formato de dados definido pelo usuário. Como nosso ícone
  // é uma DLL comum, só retornamos S_FALSE.
  Result := S_FALSE;
end;

{ TIconHandler.IPersist }

function TIconHandler.GetClassID(out classID: TCLSID): HResult;
begin
  // este método não é chamado para tratadores de ícones
  Result := E_NOTIMPL;
end;

{ TIconHandler.IPersistFile }

function TIconHandler.IsDirty: HResult;
begin
  // este método não é chamado para tratadores de ícones
  Result := S_FALSE;
end;

function TIconHandler.Load(pszFileName: POleStr; dwMode: Longint): HResult;
begin
  // este método é chamado para inicializar a extensão do shell do
  // tratador de ícones. Devemos salvar o nome do arquivo que
  // é passado em pszFileName
  FFileName := pszFileName;
  Result := S_OK;
end;

function TIconHandler.Save(pszFileName: POleStr; fRemember: BOOL): HResult;
begin
  // este método não é chamado para tratadores de ícones
  Result := E_NOTIMPL;
end;

function TIconHandler.SaveCompleted(pszFileName: POleStr): HResult;
begin
  // este método não é chamado para tratadores de ícones
  Result := E_NOTIMPL;
end;

function TIconHandler.GetCurFile(out pszFileName: POleStr): HResult;
begin
  // este método não é chamado para tratadores de ícones
  Result := E_NOTIMPL;
end;

{ TIconHandlerFactory }
```

Listagem 16.11 Continuação

```
function TIconHandlerFactory.GetProgID: string;
begin
  // ProgID não-obrigatória em extensões do shell para menu de contexto
  Result := '';
end;

procedure TIconHandlerFactory.UpdateRegistry(Register: Boolean);
var
  ClsID: string;
begin
  ClsID := GUIDToString(ClassID);
  inherited UpdateRegistry(Register);
  ApproveShellExtension(Register, ClsID);
  if Register then
  begin
    // precisa registrar .bpl como um tipo de arquivo
    CreateRegKey('.bpl', '', 'BorlandPackageLibrary');
    // registra essa DLL como tratador de ícones para arquivos .bpl
    CreateRegKey('BorlandPackageLibrary\shellex\IconHandler', '', ClsID);
  end
  else begin
    DeleteRegKey('.bpl');
    DeleteRegKey('BorlandPackageLibrary\shellex\IconHandler');
  end;
end;

procedure TIconHandlerFactory.ApproveShellExtension(Register: Boolean;
  const ClsID: string);
// Esta entrada de registro é obrigatória para que a extensão opere
// corretatamente no Windows NT.
const
  SApproveKey = 'SOFTWARE\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved';

begin
  with TRegistry.Create do
    try
      RootKey := HKEY_LOCAL_MACHINE;
      if not OpenKey(SApproveKey, True) then Exit;
      if Register then WriteString(ClsID, Description)
      else DeleteValue(ClsID);
    finally
      Free;
    end;
end;

const
  CLSID_IconHandler: TGUID = '{ED6D2F60-DA7C-11D0-A9BF-90D146FC32B3}';

initialization
  TIconHandlerFactory.Create(ComServer, TIconHandler, CLSID_IconHandler,
    'DDG_IconHandler', 'DDG Icon Handler Shell Extension Example',
    ciMultiInstance, tmApartment);
end.
```

Tratadores de dica de tela

Introduzidos no shell do Windows 2000, os tratadores de dica de tela (InfoTip) oferecem a capacidade de criar *IntoTips* pop-up personalizadas (também chamadas *ToolTips* – dicas de tela – no Delphi) quando o mouse é colocado sobre o ícone que representa um arquivo no shell. A dica de tela default apresentada pelo shell contém o nome do arquivo, o tipo do arquivo (determinado com base na sua extensão) e o tamanho do arquivo. Os tratadores de dica de tela são práticos quando você deseja exibir mais do que essas informações um tanto limitadas e genéricas para o usuário com um passar de olhos.

Para desenvolvedores Delphi, uma ótima oportunidade são os arquivos de pacote. Embora todos nós saibamos que os arquivos de pacote são compostos de uma ou mais unidades, é impossível saber rápido exatamente quais unidades estão contidas em seu interior. Anteriormente neste capítulo, você viu um tratador de menu de contexto que oferece essa informação escolhendo uma opção por um menu local, iniciando uma aplicação externa. Agora você verá como obter essa informação ainda mais facilmente, sem o uso de um programa externo.

Interfaces de tratador de dica de tela

Tratadores de dica de tela precisam implementar as interfaces `IQueryInfo` e `IPersistFile`. Você já aprendeu sobre `IPersistFile` nas discussões sobre links do shell e tratadores de ícones, anteriormente neste capítulo, e essa interface é usada neste caso para obter o nome do arquivo em questão. `IQueryInfo` é uma interface relativamente simples; ela contém dois métodos e é definida na unidade `ShlObj`, como podemos ver aqui:

```
type
  IQueryInfo = interface(IUnknown)
  [SID_IQueryInfo ]
  function GetInfoTip(dwFlags: DWORD; var ppwszTip: PWideChar): HResult;
    stdcall;
  function GetInfoFlags(out pdwFlags: DWORD): HResult; stdcall;
end;
```

O método `GetInfoTip( )` é chamado pelo shell para apanhar a dica de tela para um determinado arquivo. O parâmetro `dwFlags` atualmente não é utilizado. A string da dica de tela é retornada no parâmetro `ppwszTip`.

> **NOTA**
>
> O parâmetro `ppwszTip` aponta para uma string de caracteres larga. A memória para essa string precisa ser alocada dentro do tratador de dica de tela, usando o alocador de memória do shell. O shell é responsável por liberar essa memória.

Implementação

Assim como outras extensões do shell, o tratador de dica de tela é implementado como uma DLL simples de servidor COM. O objeto COM nele contido implementa os métodos `IQueryInfo` e `IPersistFile`. A Listagem 16.12 mostra o conteúdo de `InfoMain.pas`, a unidade principal para o projeto `DDGInfoTip`, que contém a implementação do Delphi para um tratador de dica de tela.

Listagem 16.12 `InfoMain.pas` – a unidade principal para o tratador de dica de tela

```
unit InfoMain;

{$WARN SYMBOL_PLATFORM OFF}
```

Listagem 16.12 Continuação

```
interface

uses
  Windows, ActiveX, Classes, ComObj, ShlObj;

type
  TInfoTipHandler = class(TComObject, IQueryInfo, IPersistFile)
  private
    FFileName: string;
    FMalloc: IMalloc;
  protected
    { IQUeryInfo }
    function GetInfoTip(dwFlags: DWORD; var ppwszTip: PWideChar): HResult; stdcall;
    function GetInfoFlags(out pdwFlags: DWORD): HResult; stdcall;
    { IPersist }
    function GetClassID(out classID: TCLSID): HResult; stdcall;
    { IPersistFile }
    function IsDirty: HResult; stdcall;
    function Load(pszFileName: POleStr; dwMode: Longint): HResult; stdcall;
    function Save(pszFileName: POleStr; fRemember: BOOL): HResult; stdcall;
    function SaveCompleted(pszFileName: POleStr): HResult; stdcall;
    function GetCurFile(out pszFileName: POleStr): HResult; stdcall;
  public
    procedure Initialize; override;
  end;

  TInfoTipFactory = class(TComObjectFactory)
  protected
    function GetProgID: string; override;
    procedure ApproveShellExtension(Register: Boolean; const ClsID: string);
      virtual;
  public
    procedure UpdateRegistry(Register: Boolean); override;
  end;

const
  Class_InfoTipHandler: TGUID = '{5E08F28D-A5B1-4996-BDF1-5D3210BDB5E5}';

implementation

uses ComServ, SysUtils, Registry;

const
  TipBufLen = 1024;

procedure PackageInfoCallback(const Name: string; NameType: TNameType;
  Flags: Byte; Param: Pointer);
var
  S: string;
begin
  // se estamos recebendo o nome de uma unidade contida, então o
  // concatenamos com a lista de unidades, que é passada em Param.
  if NameType = ntContainsUnit then
  begin
    S := Name;
    if PChar(Param)^ < > #0 then
      S := ', ' + S;
    StrLCat(PChar(Param), PChar(S), TipBufLen);
  end;
```

Listagem 16.12 Continuação

```
end;

function TInfoTipHandler.GetClassID(out classID: TCLSID): HResult;
begin
  classID := Class_InfoTipHandler;
  Result := S_OK;
end;

function TInfoTipHandler.GetCurFile(out pszFileName: POleStr): HResult;
begin
  Result := E_NOTIMPL;
end;

function TInfoTipHandler.GetInfoFlags(out pdwFlags: DWORD): HResult;
begin
  Result := E_NOTIMPL;
end;

function TInfoTipHandler.GetInfoTip(dwFlags: DWORD;
  var ppwszTip: PWideChar): HResult;
var
  PackMod: HModule;
  TipStr: PChar;
  Size, Flags, TipStrLen: Integer;
begin
  Result := S_OK;
  if (CompareText(ExtractFileExt(FFileName), '.bpl') = 0) and
    Assigned(FMalloc) then
  begin
    // Como só precisamos entrar nos recursos do pacote, LoadLibraryEx
    // com LOAD_LIBRARY_AS_DATAFILE oferece um meio rápido para se
    // carregar o pacote.
    PackMod := LoadLibraryEx(PChar(FFileName), 0, LOAD_LIBRARY_AS_DATAFILE);
    if PackMod < > 0 then
      try
        TipStr := StrAlloc(TipBufLen);
        try
          FillChar(TipStr^, TipBufLen, 0);  // zera memória da string
          // Preenche TipStr com unidades contidas
          GetPackageInfo(PackMod, TipStr, Flags, PackageInfoCallback);
          TipStrLen := StrLen(TipStr) + 1;
          Size := (TipStrLen + 1) * SizeOf(WideChar);
          ppwszTip := FMalloc.Alloc(Size); // usa alocador do shell
          // copia PAnsiChar para PWideChar
          MultiByteToWideChar(0, 0, TipStr, TipStrLen, ppwszTip, Size);
        finally
          StrDispose(TipStr);
        end;
      finally
        FreeLibrary(PackMod);
      end;
  end;
end;

procedure TInfoTipHandler.Initialize;
begin
  inherited;
  // apanha alocador de memória do shell e o salva
  SHGetMalloc(FMalloc);
```

Listagem 16.12 Continuação

```
end;

function TInfoTipHandler.IsDirty: HResult;
begin
  Result := E_NOTIMPL;
end;

function TInfoTipHandler.Load(pszFileName: POleStr;
  dwMode: Integer): HResult;
begin
  // Este é o único método importande de IPersistFile – temos que
  // salvar o nome do arquivo
  FFileName := pszFileName;
  Result := S_OK;
end;

function TInfoTipHandler.Save(pszFileName: POleStr;
  fRemember: BOOL): HResult;
begin
  Result := E_NOTIMPL;
end;

function TInfoTipHandler.SaveCompleted(pszFileName: POleStr): HResult;
begin
  Result := E_NOTIMPL;
end;

{ TInfoTipFactory }

function TInfoTipFactory.GetProgID: string;
begin
  // ProgID não-exigido para a extensão de shell do tratador de dica de tela
  Result := '';
end;

procedure TInfoTipFactory.UpdateRegistry(Register: Boolean);
var
  ClsID: string;
begin
  ClsID := GUIDToString(ClassID);
  inherited UpdateRegistry(Register);
  ApproveShellExtension(Register, ClsID);
  if Register then
  begin
    // registra essa DLL como tratador de dica de tela para
    // arquivos .bpl
    CreateRegKey('.bpl\shellex\{00021500-0000-0000-C000-000000000046}',
      '', ClsID);
  end
  else begin
    DeleteRegKey('.bpl\shellex\{00021500-0000-0000-C000-000000000046}');
  end;
end;

procedure TInfoTipFactory.ApproveShellExtension(Register: Boolean;
  const ClsID: string);
// Esta entrada do Registro é obrigatória para que a extensão opere
// corretamente no Windows NT.
```

Listagem 16.12 Continuação

```
const
  SApproveKey = 'SOFTWARE\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved';
begin
  with TRegistry.Create do
    try
      RootKey := HKEY_LOCAL_MACHINE;
      if not OpenKey(SApproveKey, True) then Exit;
      if Register then WriteString(ClsID, Description)
      else DeleteValue(ClsID);
    finally
      Free;
    end;
end;

initialization
  TInfoTipFactory.Create(ComServer, TInfoTipHandler, Class_InfoTipHandler,
    'InfoTipHandler', 'DDG sample InfoTip handler', ciMultiInstance, tmApartment);
end.
```

Existem alguns pontos interessantes nessa implementação. Observe que o alocador de memória do shell é apanhado e armazenado no método `Initialize( )`. O alocador mais tarde é usado para alocar memória para a string de dica de tela no método `GetInfoTip( )`. O nome do arquivo em questão é passado para o tratador no método `Load( )`. O trabalho é realizado no método `GetInfoTip( )`, que apanha informações do pacote usando a função `GetPackageInfo( )`, a respeito da qual você aprendeu anteriormente neste capítulo. À medida que a função de callback `PackageInfoCallback( )` é chamada repetidamente de dentro de `GetPackageInfo( )`, a string de dica de tela é concatenada arquivo por arquivo.

Registro

A técnica usada para o registro da DLL do servidor COM é quase idêntica à de outras extensões do shell neste capítulo, como você pode ver na Listagem 16.12. A principal diferença é a chave sob a qual os tratadores de dica de tela são registrados; estes sempre são registrados sob `HKEY_CLASSES_ROOT\<`*`extensão de arquivo`*`>\shellex\{00021500-0000-0000-C000-000000000046}`, onde a <*extensão de arquivo*> é composta do ponto separador e da extensão propriamente dita.

A Figura 16.12 mostra esse tratador de dica de tela em ação.

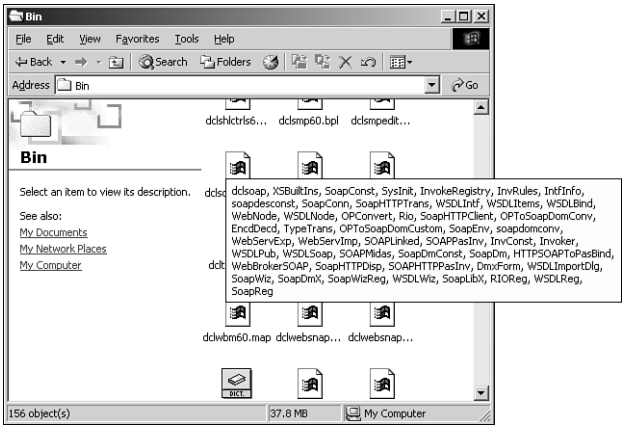

Figura 16.12 A extensão do shell para o tratador de dica de tela.

Resumo

Este capítulo inclui todos os diferentes aspectos de extensão do shell do Windows: ícones da bandeja de notificação, AppBars, vínculos do shell e uma série de extensões do shell. Ele é uma continuação do conhecimento que você obteve no capítulo anterior, trabalhando com COM e ActiveX. No Capítulo 17, você aprenderá mais sobre o desenvolvimento baseado em componentes por meio de interfaces.

Usando a API Open Tools

CAPÍTULO

17

NESTE CAPÍTULO

- Interfaces Open Tools
- Uso da API Open Tools
- Assistentes de formulário

Você já se colocou diante da seguinte questão: "O Delphi é realmente bom, mas por que o IDE não executa esta pequena tarefa que eu gostaria que ele fizesse?" Se esse é o seu caso, a API Open Tools está aí para satisfazer seus desejos. A API Open Tools do Delphi oferece a capacidade de integrar suas próprias ferramentas, que trabalham em conjunto com o IDE do Delphi. Neste capítulo, você aprenderá as diferentes interfaces que compõem a API Open Tools, como usar as interfaces e também como aproveitar sua habilidade recém-adquirida para escrever um assistente repleto de recursos.

Interfaces Open Tools

A API Open Tools é composta de 14 unidades e cada uma delas contém um ou mais objetos, que, por sua vez, fornecem interfaces para uma série de recursos no IDE. O uso dessas interfaces permite que você escreva seus próprios assistentes do Delphi, gerenciadores de controle de versão e editores de componentes e propriedades. Você também ganhará uma janela para o IDE do Delphi e para o editor, através de qualquer um desses suplementos.

Com a exceção das interfaces projetadas para editores de componentes e propriedades, os objetos da interface Open Tools fornecem uma interface totalmente virtual para o mundo exterior – o que significa que o uso desses objetos da interface reduz o trabalho às funções virtuais dos objetos. Você não pode acessar os campos de dados, propriedades e funções estáticas dos objetos. Por causa disso, os objetos da interface Open Tools seguem o padrão COM (veja no Capítulo 15). Com um pouco de trabalho de sua parte, essas interfaces podem ser usadas por qualquer linguagem de programação que tenha suporte para COM. Neste capítulo, você só irá trabalhar com o Delphi, mas é bom saber que a capacidade para usar outras linguagens está disponível (caso você não consiga ficar sem a C++).

> NOTA
>
> A API Open Tools completa só está disponível com o Delphi Professional e Enterprise. O Delphi Personal tem a capacidade de usar suplementos criados com a API Open Tools, mas não pode criar suplementos, pois só contém as unidades para criar editores de propriedades e componentes. Você pode achar o código-fonte para as interfaces Open Tools no subdiretório `\Delphi 6\Source\ToolsAPI`.

A Tabela 17.1 mostra as unidades que compõem a API Open Tools e as interfaces que elas fornecem. A Tabela 17.2 lista as unidades obsoletas da API Open Tools, que permanecem apenas por compatibilidade com os experts escritos em Delphi 4 ou anterior. Como as unidades obsoletas são anteriores ao tipo interface nativo, elas empregam classes regulares do Delphi com métodos abstratos virtuais, como substituto para interfaces verdadeiras. O uso de interfaces verdadeiras foi substituído pela API Open Tools nas últimas versões do Delphi e a edição atual da API Open Tools é baseada principalmente em `interface`.

Tabela 17.1 Unidades na API Open Tools

Nome da unidade	*Finalidade*
`ToolsAPI`	Contém os elementos da API Open Tool baseada na interface mais recente. O conteúdo dessa unidade basicamente aposenta as unidades da API Open Tools anteriores ao Delphi 5, que usam classes abstratas para manipular menus, notificações, o sistema de arquivos, o editor e suplementos do assistente. Ela também contém as novas interfaces para manipular o depurador, os principais mapeamentos do IDE, projetos, grupos de projetos, pacotes e a lista To Do.
`VCSIntf`	Define a classe `TIVCSClient`, que permite que o IDE do Delphi se comunique com o software de controle de versão.

Tabela 17.1 Continuação

Nome da unidade	*Finalidade*
`DesignConst`	Contém strings usadas pela API Open Tools.
`DesignEditors`	Oferece suporte para o editor de propriedades.
`DesignIntf`	Esta unidade substitui a unidade `DsgnIntf` das versões anteriores e oferece suporte básico para as interfaces do IDE durante o projeto. A interface `IProperty` é usada pelo IDE para editar propriedades. `IDesignerSelections` é usada para manipular a lista de objetos selecionados do criador de formulários (substitui `TDesignerSelectionList`, usada nas versões anteriores do Delphi). `IDesigner` é uma das principais interfaces usadas pelos assistentes para serviços gerais do IDE. `IDesignNotification` oferece notificação de eventos de projeto, como itens sendo inseridos, excluídos ou modificados. A interface `IComponentEditor` é implementada por editores de componentes para oferecer edição de componentes durante o projeto, e `ISelectionEditor` oferece a mesma funcionalidade para um grupo de componentes selecionados. A classe `TBaseComponentEditor` é a classe da qual todos os editores de componentes devem ser derivados. `ICustomModule` e `TBaseCustomModule` são fornecidas para instalar módulos que podem ser editados no criador de formulários do IDE.
`DesignMenus`	Contém as interfaces `IMenuItems`, `IMenuItem` e outras relacionadas à manipulação dos menus do IDE durante o projeto.
`DesignWindows`	Declara a classe `TDesignWindow`, que serviria como classe básica para quaisquer novas janelas de projeto que alguém poderia querer incluir no IDE.
`PropertyCategories`	Contém as classes para dar suporte para a categorização de propriedades de componente personalizadas. Usada pela visão da categoria do `Object Inspector`.
`TreeIntf`	Oferece `TSprig`, além de classes e interfaces relacionadas ao suporte de *`sprigs`* personalizadas, ou nós na `Object TreeView` do IDE.
`VCLSprigs`	Implementações de sprigs para componentes VCL.
`VCLEditors`	Declara`ICustomPropertyDrawing` e `ICustomPropertyListDrawing` básicas para lidar com o desenho personalizado de propriedades e listas de propriedades no `Object Inspector` do IDE. Também declara objetos de desenho de propriedade personalizados para propriedades comuns da VCL.
`ClxDesignWindows`	Declara a classe `TClxDesignWindow`, que é o equivalente da CLX para a classe `TDesignWindow`.
`ClxEditors`	Equivalente na CLX da unidade `VCLEditors`, que inclui editores de propriedades para componentes CLX.
`ClxSprigs`	Implementações de sprigs para componentes da CLX.

Tabela 17.2 Unidades obsoletas da API Open Tools

Nome da unidade	*Finalidade*
`FileIntf`	Define a classe `TIVirtualFileSystem`, que o IDE do Delphi usa para arquivamentos. Assistentes, gerenciadores de controle de versão e editores de propriedades e componentes podem usar essa interface para criar um gancho no próprio sistema de arquivos do Delphi, a fim de executar operações especiais com arquivos.

Tabela 17.2 Continuação

Nome da unidade	*Finalidade*
`EditIntf`	Define as classes necessárias para a manipulação do Code Editor e Form Designer. A classe `TIEditReader` fornece acesso de leitura a um buffer do editor. `TIEditWriter` fornece acesso de escrita para o mesmo. `TIEditView` é definido como um modo de exibição individual de um buffer de edição. `TIEditInterface` é a interface básica do editor, que pode ser usada para obter as interfaces do editor mencionadas anteriormente. A classe `TIComponentInterface` é uma interface para um componente individual situado em um formulário durante o projeto. `TIFormInterface` é a interface básica para um módulo de dados ou um formulário durante o projeto. `TIResourceEntry` é uma interface para os dados brutos em um arquivo de recursos (`*.res`) do projeto. `TIResourceFile` é uma interface de nível superior para o arquivo de recursos do projeto.`TIModuleNotifier` é uma classe que fornece notificações quando vários eventos ocorrem para um determinado módulo. Finalmente, `TIModuleInterface` é a interface para qualquer arquivo ou módulo aberto no IDE.
`ExptIntf`	Define a classe `TIExpert` abstrata, da qual todos os experts descendem.
`VirtIntf`	Define a classe `TInterface` básica, da qual as outras interfaces são derivadas. Essa unidade também define a classe `TIStream`, que é um wrapper em torno de uma `TStream` da VCL.
`IStreams`	Define as classes `TIMemoryStream`, `TIFileStream` e `TIVirtualStream`, que são descendentes de `TIStream`. Essas interfaces podem ser usadas para criar um gancho para o próprio mecanismo de streaming do IDE.
`ToolIntf`	Define as classes `TIMenuItemIntf` e `TIMainMenuIntf`, que permite que um desenvolvedor `Open Tools` crie e modifique menus no IDE do Delphi. Essa unidade também define a classe `TIAddInNotifier`, que permite que ferramentas adicionais sejam notificadas quanto a certos eventos dentro do IDE. Mais importante, essa unidade define a classe `TIToolServices`, que fornece uma interface para diversos trechos do IDE do Delphi (como o editor, a biblioteca de componentes, o Code Editor, o Form Designer e o sistema de arquivos).

NOTA

Você pode estar se perguntando onde todo esse material referente a assistentes está documentado no Delphi. Garantimos que está documentado, mas não é nada fácil de se achar. Cada uma dessas unidades contém documentação completa para a interface, classes, métodos e procedimentos declarados em seu interior. Como não vamos repetir as mesmas informações, convidamos para que você dê uma olhada nas unidades, para ter acesso à documentação completa.

Uso da API Open Tools

Agora que você sabe o que é o quê, chegou a hora de meter a mão na lama e encarar um código de verdade. Esta seção é dedicada principalmente à criação de assistentes por meio do uso da API Open Tools. Não vamos discutir a construção de sistemas de controle de versão, pois são poucas as pessoas que se interessam por esse tipo de assunto. Para obter exemplos de editores de componentes e propriedades, você deve consultar os Capítulo 11 e 12.

Um assistente burro

Para começo de conversa, você criará um assistente muito simples que, por isso, é chamado de assistente burro. O requisito mínimo para a criação de um assistente é criar uma classe que implemente a interface `IOTAWizard`. Apenas como referência, `IOTAWizard` é definido na unidade `ToolsAPI` da seguinte maneira:

```
type
  IOTAWizard = interface(IOTANotifier)
    ['{B75C0CE0-EEA6-11D1-9504-00608CCBF153}']
    { Strings de IU do expert }
    function GetIDString: string;
    function GetName: string;
    function GetState: TWizardState;
    { Inicia o AddIn }
    procedure Execute;
  end;
```

Essa interface consiste principalmente em algumas funções `Get`*`XXX`*`( )` que são projetadas para serem modificadas pelas classes descendentes, de modo a fornecer informações específicas para cada assistente. O método `Execute( )` é a finalidade comercial de `IOTAWizard`. `Execute( )` é chamado pelo IDE quando o usuário seleciona o seu assistente no menu principal ou no menu New Items (itens novos), e é nesse método que o assistente deve ser criado e chamado.

Se você tiver um olho astuto, deve ter percebido que `IOTAWizard` descende de outra interface, chamada `IOTANotifier`. `IOTANotifier` é uma interface definida na unidade `ToolsAPI`, que contém métodos que podem ser chamados pelo IDE a fim de notificar um assistente quanto a várias ocorrências. Essa interface é definida da seguinte maneira:

```
type
  IOTANotifier = interface(IUnknown)
    ['{F17A7BCF-E07D-11D1-AB0B-00C04FB16FB3}']
    { Este procedimento é chamado imediatamente depois que o item é salvo
      com sucesso. Ele não é chamado para IOTAWizards }
    procedure AfterSave;
    { Esta função é chamada imediatamente antes de o item ser salvo. Ela
      não é chamada para IOTAWizard }
    procedure BeforeSave;
    { O item associado está sendo destruído de modo que todas as referências
      devem ser liberadas. As exceções são ignoradas. }
    procedure Destroyed;
    { Este item associado foi modificado de alguma forma. Ele não é chamado
      para IOTAWizards }
    procedure Modified;
  end;
```

Como indicam os comentários no código-fonte, a maioria desses métodos não é chamada para assistentes `IOTAWizard` simples. Por causa disso, a `ToolsAPI` fornece uma classe chamada `TNotifierObject`, que fornece implementações vazias para os métodos `IOTANotifier`. Você pode escolher descender seus assistentes dessa classe para tirar proveito da conveniência de ter métodos `IOTANotifier` implementados para você.

Os assistentes não são muito úteis sem um meio para chamá-los, sendo que uma das formas mais simples de se fazer isso é através de uma escolha de menu. Se você quiser colocar o assistente no menu principal do Delphi, só precisa implementar a interface `IOTAMenuWizard`, que é definida em toda a sua complexidade em `ToolsAPI` da seguinte maneira:

```
type
  IOTAMenuWizard = interface(IOTAWizard)
```

```
    ['{B75C0CE2-EEA6-11D1-9504-00608CCBF153}']
    function GetMenuText: string;
  end;
```

Como você pode ver, essa interface descende de `IOTAWizard` e só inclui um método adicional para retornar a string de texto do menu.

Para juntar tudo isso e mostrar para que servem as informações que você aprendeu até agora, a Listagem 17.1 mostra a unidade `DumbWiz.pas`, que contém o código-fonte de `TDumbWizard`.

Listagem 17.1 `DumbWiz.pas` – uma implementação de assistente simples

```
unit DumbWiz;

interface

uses
  ShareMem, SysUtils, Windows, ToolsAPI;

type
  TDumbWizard = class(TNotifierObject, IOTAWizard, IOTAMenuWizard)
    // Métodos de IOTAWizard
    function GetIDString: string;
    function GetName: string;
    function GetState: TWizardState;
    procedure Execute;
    // Método de IOTAMenuWizard
    function GetMenuText: string;
  end;

procedure Register;

implementation

uses Dialogs;

function TDumbWizard.GetName: string;
begin
  Result := 'Dumb Wizard';
end;

function TDumbWizard.GetState: TWizardState;
begin
  Result := [wsEnabled];
end;

function TDumbWizard.GetIDString: String;
begin
  Result := 'DDG.DumbWizard';
end;

procedure TDumbWizard.Execute;
begin
  MessageDlg('This is a dumb wizard.', mtInformation, [mbOk], 0);
end;

function TDumbWizard.GetMenuText: string;
begin
```

Listagem 17.1 Continuação

```
  Result := 'Dumb Wizard';
end;

procedure Register;
begin
  RegisterPackageWizard(TDumbWizard.Create);
end;

end.
```

A função `IOTAWizard.GetName( )` deve retornar um nome exclusivo para esse assistente.

`IOTAWizard.GetState( )` retorna o estado de um assistente `wsStandard` no menu principal. O valor de retorno dessa função é um conjunto que pode conter `wsEnabled` e/ou `wsChecked`, dependendo do modo como você deseja que o item do menu apareça no IDE. Essa função é chamada todas as vezes que o assistente é apresentado, para determinar como o menu deve ser desenhado.

`IOTAWizard.GetIDString( )` deve retornar um identificador de string exclusivo e global para o assistente. A convenção determina que o valor de retorno dessa string deve estar no seguinte formato:

```
NomeEmpresa.NomeAssistente
```

`IOTAWizard.Execute( )` chama o assistente. Como a Listagem 17.1 nos mostra, o método `Execute( )` para `TDumbWizard` não faz muita coisa. Ainda neste capítulo, no entanto, você verá alguns assistentes que realmente realizam algumas tarefas.

`IOTAMenuWizard.GetMenuText( )` retorna o texto que deve aparecer no menu principal. Essa função é chamada todas as vezes que o usuário abre o menu Help e portanto é possível mudar dinamicamente o valor do texto do menu à medida que o assistente é executado.

Dê uma olhada na chamada para `RegisterPackageWizard( )` dentro do procedimento `Register( )`. Você vai perceber que isso é muito parecido com a sintaxe usada para registrar componentes, editores de componentes e editores de propriedades para inclusão na biblioteca de componentes, como descrito nos Capítulos 11 e 12. A razão para essa semelhança é que esse tipo de assistente é armazenado em um pacote que é parte da biblioteca de componentes, juntamente com os componentes e tudo o mais. Você também pode armazenar assistentes em uma DLL independente, como verá no próximo exemplo.

Esse assistente é instalado como qualquer componente: selecione a opção Components, Install Component (instalar componente) no menu principal e adicione a unidade a um pacote novo ou existente. Uma vez instalado, a opção de menu para chamar o assistente aparece no menu Help, como mostra a Figura 17.1. Você pode ver a fantástica saída desse assistente na Figura 17.2.

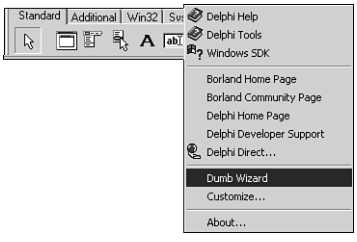

Figura 17.1 O assistente burro no menu principal.

Figura 17.2 O assistente burro em ação.

O assistente Wizard

É preciso um pouco mais de trabalho para criar um assistente baseado em DLL (que é o oposto de um assistente baseado em uma biblioteca de componentes). Além de demonstrar a criação de um assistente baseado em DLL, o assistente Wizard é usado aqui como exemplo por duas razões: ilustrar como os assistentes de DLL se relacionam com o Registro e manter a base de um código-fonte que se destina tanto a um assistente EXE como a um assistente DLL.

NOTA

Se você não está acostumado com os detalhes das DLLs no Windows, dê uma olhada no Capítulo 9, na versão eletrônica de *Delphi 5 Guia do Desenvolvedor*, no CD que acompanha este livro.

DICA

Não há uma regra infalível que determine se um assistente deve residir em um pacote na biblioteca de componentes ou em uma DLL. Do ponto de vista de um usuário, a principal diferença entre as duas é que os assistentes de biblioteca de componentes só precisam da instalação de um pacote para serem reconstruídos, enquanto os assistentes de DLL exigem uma entrada no Registro e o Delphi deve ser fechado e reiniciado para que as mudanças façam efeito. No entanto, como um desenvolvedor, você verá que os assistentes de pacotes são um pouco mais fáceis de se lidar por uma série de razões. Especificamente, as exceções se propagam automaticamente entre o seu assistente e o IDE, você não precisa usar `sharemem.dll` para gerenciamento de memória, não tem que fazer nada especial para inicializar a variável de aplicação da DLL e as mensagens de entrada/saída do mouse e dicas de tela funcionarão a contento.

Com isso em mente, você deve considerar o uso de um assistente de DLL quando quiser que o assistente seja instalado com um mínimo de trabalho por parte do usuário final.

Para que o Delphi reconheça um assistente de DLL, ele deve ter uma entrada no Registro do sistema, sob a seguinte chave:

```
HKEY_CURRENT_USER\Software\Borland\Delphi\6.0\Experts
```

A Figura 17.3 mostra entradas de exemplo usando a aplicação RegEdit do Windows.

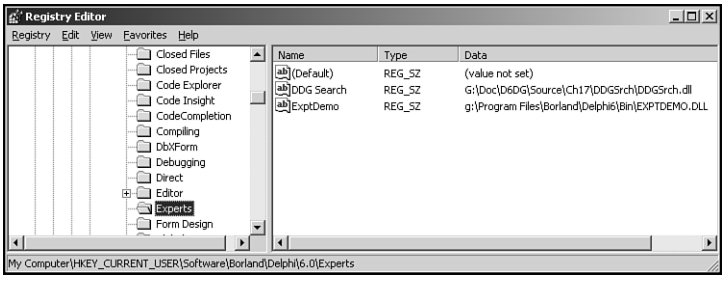

Figura 17.3 Entradas de assistente do Delphi exibidas no RegEdit.

Interface do assistente

O objetivo do assistente Wizard é fornecer uma interface para adicionar, modificar e excluir entradas de assistente de DLL do Registro sem ter que usar a complicada aplicação RegEdit. Primeiro, vamos examinar `InitWiz.pas`, a unidade que contém a classe do assistente (veja a Listagem 17.2).

Listagem 17.2 `InitWiz.pas` – a unidade que contém a classe do assistente de DLL

```
unit InitWiz;

interface

uses Windows, ToolsAPI;

type
  TWizardWizard = class(TNotifierObject, IOTAWizard, IOTAMenuWizard)
    // Métodos de IOTAWizard
    function GetIDString: string;
    function GetName: string;
    function GetState: TWizardState;
    procedure Execute;
    // Método de IOTAMenuWizard
    function GetMenuText: string;
  end;

function InitWizard(const BorlandIDEServices: IBorlandIDEServices;
  RegisterProc: TWizardRegisterProc;
  var Terminate: TWizardTerminateProc): Boolean stdcall;

var
  { Chave do registro onde os assistentes do Delphi 6 são mantidos.
    Versão EXE usa default, enquanto versão DLL apanha chave de
    ToolServices.GetBaseRegistryKey. }
    SDelphiKey: string = '\Software\Borland\Delphi\6.0\Experts';

implementation

uses SysUtils, Forms, Controls, Main;

function TWizardWizard.GetName: string;
{ Retorna nome do expert }
begin
  Result := 'WizardWizard';
end;

function TWizardWizard.GetState: TWizardState;
{ Esse expert sempre está ativado }
begin
  Result := [wsEnabled];
end;

function TWizardWizard.GetIDString: String;
{ String de ID "Vendor.AppName" para expert }
begin
  Result := 'DDG.WizardWizard';
end;

function TWizardWizard.GetMenuText: string;
{ Texto de menu para expert }
begin
  Result := 'Wizard Wizard';
end;

procedure TWizardWizard.Execute;
{ Chamada quando expert é escolhido a partir do menu principal. }
```

Listagem 17.2 Continuação

```
{ Esse procedimento cria, mostra e libera o formulário principal. }
begin
  MainForm := TMainForm.Create(Application);
  try
    MainForm.ShowModal;
  finally
    MainForm.Free;
  end;
end;

function InitWizard(const BorlandIDEServices: IBorlandIDEServices;
  RegisterProc: TWizardRegisterProc;
  var Terminate: TWizardTerminateProc): Boolean stdcall;
var
  Svcs: IOTAServices;
begin
  Result := BorlandIDEServices <<|>> nil;
  if Result then
  begin
    Svcs := BorlandIDEServices as IOTAServices;
    ToolsAPI.BorlandIDEServices := BorlandIDEServices;
    Application.Handle := Svcs.GetParentHandle;
    SDelphiKey := Svcs.GetBaseRegistryKey + '\Experts';
    RegisterProc(TWizardWizard.Create);
  end;
end;

end.
```

Você deve perceber algumas poucas diferenças entre essa unidade e a que é usada para criar o assistente burro. Mais importante, uma função de inicialização do tipo `TWizardInitProc` é exigida como um ponto de entrada para o IDE na DLL do assistente. Nesse caso, essa função é chamada `InitWizard( )`. Essa função executa uma série de tarefas de inicialização de assistente, incluindo as seguintes:

- Obtenção da interface `IOTAServices` do parâmetro `BorlandIDEServices`.
- Salvamento do ponteiro de interface `BorlandIDEServices` para uso posterior.
- Definição da alça da variável `Application` da DLL como o valor retornado por `IOTAServices.GetParentHandle( )` `GetParentHandle( )` retorna a alça da janela que deve servir como o pai para todas as janelas de nível superior criadas pelo assistente.
- Passagem da instância recém-criada do assistente para o procedimento `RegisterProc( )` para registrar o assistente com o IDE. `RegisterProc( )` será chamada uma vez para cada instância de assistente que a DLL registra com o IDE.
- Opcionalmente, `InitWizard( )` também pode atribuir um procedimento do tipo `TWizardTerminateProc` para o parâmetro `Terminate` servir como um procedimento de saída para o assistente. Esse procedimento será chamado imediatamente antes de o assistente ser descarregado pelo IDE e nele você pode executar qualquer limpeza necessária. Inicialmente, esse parâmetro é `nil` e, portanto, se você não precisar de nenhuma limpeza especial, deixe seu valor como `nil`.

ATENÇÃO

O método de inicialização do assistente deve usar a convenção de chamada `stdcall`.

ATENÇÃO

Os assistentes de DLL que chamam funções da API Open Tools que possuem parâmetros de string devem ter a mesma unidade `ShareMem` em sua cláusula `uses`; caso contrário, o Delphi emitirá uma violação de acesso quando a instância do assistente for liberada.

A interface com o usuário do assistente

O método `Execute( )` é um pouco mais complexo dessa vez. Ele cria uma instância de `MainForm` do assistente, mostra-a como um formulário modal e em seguida libera as instâncias. A Figura 17.4 mostra esse formulário e a Listagem 17.3 mostra a unidade `Main.pas`, na qual existe `MainForm`.

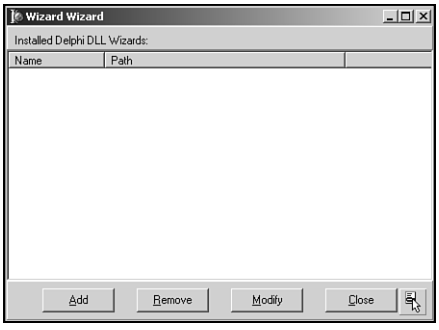

Figura 17.4 `MainForm` no assistente Wizard.

Listagem 17.3 `Main.pas` – a unidade principal do assistente Wizard

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, Registry, AddModU, ComCtrls, Menus;

type
  TMainForm = class(TForm)
    TopPanel: TPanel;
    Label1: TLabel;
    BottomPanel: TPanel;
    WizList: TListView;
    PopupMenu1: TPopupMenu;
    Add1: TMenuItem;
    Remove1: TMenuItem;
    Modify1: TMenuItem;
    AddBtn: TButton;
    RemoveBtn: TButton;
    ModifyBtn: TButton;
    CloseBtn: TButton;
    procedure RemoveBtnClick(Sender: TObject);
    procedure CloseBtnClick(Sender: TObject);
    procedure AddBtnClick(Sender: TObject);
```

Listagem 17.3 Continuação

```
    procedure ModifyBtnClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    procedure DoAddMod(Action: TAddModAction);
    procedure RefreshReg;
  end;

var
  MainForm: TMainForm;
  { Chave do registro onde os assistentes do Delphi 6 são mantidos.
    Versão EXE usa default, enquanto versão DLL apanha chave de
    ToolServices.GetBaseRegistryKey. }
  SDelphiKey: string = '\Software\Borland\Delphi\6.0\Experts';

implementation

uses InitWiz;

{$R *.DFM}

var
  DelReg: TRegistry;

procedure TMainForm.RemoveBtnClick(Sender: TObject);
{ Tratador para clique no botão Remove. Remove item selecionado
  do Registro. }
var
  Item: TListItem;
begin
  Item := WizList.Selected;
  if Item < > nil then
  begin
    if MessageDlg(Format('Remove item "%s"', [Item.Caption]), mtConfirmation,
      [mbYes, mbNo], 0) = mrYes then
      DelReg.DeleteValue(Item.Caption);
    RefreshReg;
  end;
end;

procedure TMainForm.CloseBtnClick(Sender: TObject);
{ Tratador para clique no botão Close. Fecha aplicação. }
begin
  Close;
end;

procedure TMainForm.DoAddMod(Action: TAddModAction);
{ Inclui um novo item de expert no Registro ou modifica existente. }
var
  OrigName, ExpName, ExpPath: String;
  Item: TListItem;
begin
  if Action = amaModify then           // se modifica...
  begin
    Item := WizList.Selected;
    if Item = nil then Exit;           // verifica se item está selecionado
    ExpName := Item.Caption;           // inicializa variáveis
    if Item.SubItems.Count > 0 then
```

Listagem 17.3 Continuação

```
      ExpPath := Item.SubItems[0];
    OrigName := ExpName;              // salva nome original
  end;
  { Chama caixa de diálogo que permite ao usuário incluir ou modificar entrada }
  if AddModWiz(Action, ExpName, ExpPath) then
  begin
    { se ação é Modify e o nome foi mudado, cuida disso }
    if (Action = amaModify) and (OrigName < > ExpName) then
      DelReg.RenameValue(OrigName, ExpName);
    DelReg.WriteString(ExpName, ExpPath);  // escreve novo valor
  end;
  RefreshReg;                             // atualiza caixa de listagem
end;

procedure TMainForm.AddBtnClick(Sender: TObject);
{ Tratador para clique do botão Add }
begin
  DoAddMod(amaAdd);
end;

procedure TMainForm.ModifyBtnClick(Sender: TObject);
{ Tratador para clique do botão Modify }
begin
  DoAddMod(amaModify);
end;

procedure TMainForm.RefreshReg;
{ Atualiza caixa de listagem com conteúdo do Registro }
var
  i: integer;
  TempList: TStringList;
  Item: TListItem;
begin
  WizList.Items.Clear;
  TempList := TStringList.Create;
  try
    { Apanha nomes de expert do Registro }
    DelReg.GetValueNames(TempList);
    { Apanha strings de caminho para cada nome de expert }
    for i := 0 to TempList.Count – 1 do
    begin
      Item := WizList.Items.Add;
      Item.Caption := TempList[i];
      Item.SubItems.Add(DelReg.ReadString(TempList[i]));
    end;
  finally
    TempList.Free;
  end;
end;

procedure TMainForm.FormCreate(Sender: TObject);
begin
  RefreshReg;
end;

initialization
  DelReg := TRegistry.Create;          // cria objeto do Registro
```

Listagem 17.3 Continuação

```
  DelReg.RootKey := HKEY_CURRENT_USER; // define chave raiz
  DelReg.OpenKey(SDelphiKey, True);    // abre/cria chave de expert do Delphi
finalization
  Delreg.Free;                         // libera objeto do Registro
end.
```

Essa é a unidade responsável pelo fornecimento da interface com o usuário para adicionar, remover e modificar entradas do assistente de DLL no Registro. Na seção `initialization` desta unidade, é criado um objeto `TRegistry`, chamado `DelReg`. A propriedade `RootKey` de `DelReg` é definida como `HKEY_CURRENT_USER` e abre a chave `\Software\Borland\Delphi\6.0\Experts` – a chave usada para monitorar assistentes de DLL – usando seu método `OpenKey( )`.

Quando o assistente é acionado, um componente `TListView` chamado `ExptList` é preenchido com os itens e os valores da chave de Registro mencionada anteriormente. Isso é feito pela chamada de `DelReg.GetValueNames( )` para recuperar os nomes dos itens em `TStringList`. Um componente `TListItem` é adicionado a `ExptList` para cada elemento na lista de strings e o método `DelReg.ReadString( )` é usado para ler o valor de cada item, que é colocado na lista `SubItems` de `TListItem`.

O trabalho do Registro é feito nos métodos `RemoveBtnClick( )` e `DoAddMod( )`. `RemoveBtnClick( )` é responsável pela remoção do item de assistente atualmente selecionado do Registro. Ele primeiro verifica para se certificar de que um item está destacado; em seguida, ele abre uma caixa de diálogo de confirmação. Finalmente, ele faz o trabalho chamando o método `DelReg.DeleteValue( )` e passando `CurrentItem` como parâmetro.

`DoAddMod( )` aceita um parâmetro do tipo `TAddModAction`. Esse tipo é definido da seguinte maneira:

```
type
  TAddModAction = (amaAdd, amaModify);
```

Como se pode deduzir pelos valores do tipo, essa variável indica se um novo item será adicionado ou um item existente será modificado. Essa função primeiro verifica se há um item atualmente selecionado ou, se não houver, se o parâmetro `Action` armazena o valor `amaAdd`. Depois disso, se `Action` for `amaModify`, o item e o valor existentes para o assistente são copiados nas variáveis locais `ExpName` e `ExpPath`. Posteriormente, esses valores são passados para uma função chamada `AddModExpert( )`, que é definida na unidade `AddModU`, mostrada na Listagem 17.4. Essa função chama uma caixa de diálogo na qual o usuário pode inserir informações novas ou modificadas sobre nome ou caminho para um assistente (veja a Figura 17.5). Ela retorna `True` quando o usuário fecha a caixa de diálogo com o botão OK. Nesse ponto, um item existente é modificado usando `DelReg.RenameValue( )` e um valor novo ou modificado é escrito com `DelReg.WriteString( )`.

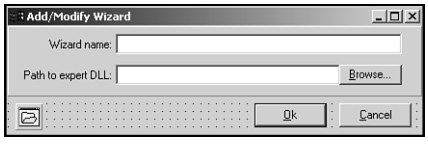

Figura 17.5 `AddModForm` no assistente Wizard.

Listagem 17.4 `AddModU.pas` – a unidade que adiciona e modifica entradas de assistente no Registro

```
unit AddModU;

interface

uses
```

Listagem 17.4 Continuação

```
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls;

type
  TAddModAction = (amaAdd, amaModify);

  TAddModForm = class(TForm)
    OkBtn: TButton;
    CancelBtn: TButton;
    OpenDialog: TOpenDialog;
    Panel1: TPanel;
    Label1: TLabel;
    Label2: TLabel;
    PathEd: TEdit;
    NameEd: TEdit;
    BrowseBtn: TButton;
    procedure BrowseBtnClick(Sender: TObject);
  private
    { Declarações privadas }
  public
    { Declarações públicas }
  end;

function AddModWiz(AAction: TAddModAction; var WizName,
  WizPath: String): Boolean;
implementation

{$R *.DFM}

function AddModWiz(AAction: TAddModAction; var WizName,
  WizPath: String): Boolean;
{ chamada para invocar caixa de diálogo que inclui e modifica entradas do Registro }
const
  CaptionArray: array[TAddModAction] of string[31] =
    ('Add new expert', 'Modify expert');
begin
  with TAddModForm.Create(Application) do       // cria caixa de diálogo
  begin
    Caption := CaptionArray[AAction];            // define texto
    if AAction = amaModify then                  // se modificar...
    begin
      NameEd.Text := WizName;                   // inicializa nome
      PathEd.Text := WizPath;                   // e caminho
    end;
    Result := ShowModal = mrOk;                 // mostra caixa de diálogo
    if Result then                              // se Ok...
    begin
      WizName := NameEd.Text;                   // define nome
      WizPath := PathEd.Text;                   // e caminho
    end;
    Free;
  end;
end;

procedure TAddModForm.BrowseBtnClick(Sender: TObject);
begin
  if OpenDialog.Execute then
    PathEd.Text := OpenDialog.FileName;
```

Listagem 17.4 Continuação

```
end;


end.
```

Destinos duplos: EXE e DLL

Como dissemos, é possível manter um conjunto de módulos de código-fonte que se destinem tanto a um assistente de DLL como a um executável independente. Isso é possível através do uso de diretivas de compilador no arquivo de projeto. A Listagem 17.5 mostra `WizWiz.dpr`, o código-fonte do arquivo de projeto para este projeto.

Listagem 17.5 `WizWiz.dpr` – arquivo de projeto principal do projeto WizWiz

```
{$ifdef BUILD_EXE}
program WizWiz;     // Constrói como EXE
{$else}
library WizWiz;     // Constrói como DLL
{$endif}

uses
{$ifndef BUILD_EXE}
  ShareMem,                  // ShareMem exigida pela DLL
  InitWiz in 'InitWiz.pas',  // Material do assistente
{$endif}
  ToolsAPI,
  Forms,
  Main in 'Main.pas' {MainForm},
  AddModU in 'AddModU.pas' {AddModForm};

{$ifdef BUILD_EXE}
{$R *.RES}                             // obrigatório para EXE
{$else}
exports                                // obrigatório para DLL
  InitWizard name WizardEntryPoint;    // ponto de entrada obrigatório
{$endif}

begin
{$ifdef BUILD_EXE}                     // obrigatório para EXE...
  Application.Initialize;
  Application.CreateForm(TMainForm, MainForm);
  Application.Run;
{$endif}
end.
```

Como podemos ver no código, este projeto construirá um executável se a condicional `BUILD_EXE` foi definida. Caso contrário, ele construirá um assistente baseado em DLL. Você pode definir uma condicional em Conditional Defines (definições condicionais) na página Directories/Conditionals (diretórios/condicionais) da caixa de diálogo Project, Options, que é mostrada na Figura 17.6.

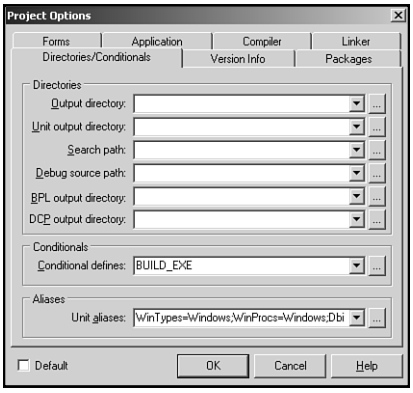

Figura 17.6 A caixa de diálogo Project Options.

Uma última observação sobre este projeto: observe que a função `InitWizard( )` da unidade `InitWiz` está sendo exportada na cláusula exports do arquivo de projeto. Você deve exportar essa função com o nome `WizardEntryPoint`, que é definido na unidade `ToolsAPI`.

> ATENÇÃO
>
> A Borland não fornece um arquivo `ToolsAPI.dcu`, o que significa que EXEs ou DLLs contendo uma referência a `ToolsAPI` em uma cláusula `uses` só podem ser construídos *com pacotes*. Atualmente, não é possível construir assistentes sem pacotes.

DDG Search

Você se lembra do pequeno porém interessante programa que desenvolveu no Capítulo 5? Nesta seção, você aprenderá como pode tornar essa aplicação útil em um assistente do Delphi ainda mais útil, adicionando apenas um pouco de código. Esse assistente é chamado de DDG Search.

Primeiro, a unidade que faz a interface de DDG Search com o IDE, `InitWiz.pas`, é mostrada na Listagem 17.6. Você vai perceber que essa unidade é muito semelhante à unidade homônima no exemplo anterior. Isso foi proposital. Essa unidade é apenas uma cópia da anterior, com algumas mudanças necessárias envolvendo o nome do assistente e o método `Execute( )`. Copiar e colar é o que chamamos de "herança à moda antiga". Afinal de contas, por que digitar mais do que o estritamente necessário?

Listagem 17.6 `InitWiz.pas` – a unidade que contém a lógica de assistente para o assistente DDG Search

```
unit InitWiz;

interface

uses
  Windows, ToolsAPI;

type
  TSearchWizard = class(TNotifierObject, IOTAWizard, IOTAMenuWizard)
    // Métodos de IOTAWizard
    function GetIDString: string;
    function GetName: string;
    function GetState: TWizardState;
    procedure Execute;
    // Método de IOTAMenuWizard
```

Listagem 17.6 Continuação

```
    function GetMenuText: string;
  end;

function InitWizard(const BorlandIDEServices: IBorlandIDEServices;
  RegisterProc: TWizardRegisterProc;
  var Terminate: TWizardTerminateProc): Boolean stdcall;

var
  ActionSvc: IOTAActionServices;

implementation

uses SysUtils, Dialogs, Forms, Controls, Main, PriU;

function TSearchWizard.GetName: string;
{ Retorna nome do expert }
begin
  Result := 'DDG Search';
end;

function TSearchWizard.GetState: TWizardState;
{ Este expert está sempre ativado no menu }
begin
  Result := [wsEnabled];
end;

function TSearchWizard.GetIDString: String;
{ Retorna o nome Vendor.Product exclusivo do expert }
begin
  Result := 'DDG.DDGSearch';
end;

function TSearchWizard.GetMenuText: string;
{ Retorna texto do menu Help }
begin
  Result := 'DDG Search Expert';
end;

procedure TSearchWizard.Execute;
{ Chamado quando o nome do expert for selecionado no menu Help do IDE. }
{ Esta função chama o expert }
begin
  // caso não tenha sido criado, cria e mostra
  if MainForm = nil then
  begin
    MainForm := TMainForm.Create(Application);
    ThreadPriWin := TThreadPriWin.Create(Application);
    MainForm.Show;
  end
  else
  // se criou, restaura a janela e mostra
    with MainForm do
    begin
      if not Visible then Show;
      if WindowState = wsMinimized then WindowState := wsNormal;
      SetFocus;
    end;
```

Listagem 17.6 Continuação

```
end;

function InitWizard(const BorlandIDEServices: IBorlandIDEServices;
  RegisterProc: TWizardRegisterProc;
  var Terminate: TWizardTerminateProc): Boolean stdcall;
var
  Svcs: IOTAServices;
begin
  Result := BorlandIDEServices <<|>> nil;
  if Result then
  begin
    Svcs := BorlandIDEServices as IOTAServices;
    ActionSvc := BorlandIDEServices as IOTAActionServices;
    ToolsAPI.BorlandIDEServices := BorlandIDEServices;
    Application.Handle := Svcs.GetParentHandle;
    RegisterProc(TSearchWizard.Create);
  end;
end;

end.
```

A função `Execute( )` deste assistente mostra uma coisa um pouco diferente do que você viu até agora: o formulário principal do assistente, `MainForm`, está sendo exibido de forma não-modal, ao invés de modal. É claro que isso requer um pouco mais de trabalho, pois você tem que saber quando um formulário é criado e quando a variável do formulário é inválida. Isso pode ser feito certificando-se de que a variável `MainForm` esteja definida como `nil` quando o assistente está inativo. Falaremos mais sobre isso posteriormente.

Outro aspecto desse projeto que foi significativamente alterado desde o Capítulo 5 é que o arquivo de projeto agora é chamado de `DDGSrch.dpr`. Esse arquivo é mostrado na Listagem 17.7.

Listagem 17.7 `DDGSrch.dpr` – arquivo de projeto para `DDGSrch`

```
{$IFDEF BUILD_EXE}
program DDGSrch;
{$ELSE}
library DDGSrch;
{$ENDIF}

uses
{$IFDEF BUILD_EXE}
  Forms,
{$ELSE}
  ShareMem,
  ToolsAPI,
  InitWiz in 'InitWiz.pas',
{$ENDIF}
  Main in 'MAIN.PAS' {MainForm},
  SrchIni in 'SrchIni.pas',
  SrchU in 'SrchU.pas',
  PriU in 'PriU.pas' {ThreadPriWin},
  MemMap in '..\..\Utils\MemMap.pas',
  DDGStrUtils in '..\..\Utils\DDGStrUtils.pas';

{$R *.RES}
```

Listagem 17.7 Continuação

```
{$IFNDEF BUILD_EXE}
exports
  { Ponto de entrada que é chamado pelo IDE do Delphi }
  InitWizard name WizardEntryPoint;
{$ENDIF}

begin
{$IFDEF BUILD_EXE}
  Application.Initialize;
  Application.CreateForm(TMainForm, MainForm);
  Application.Run;
{$ENDIF}
end.
```

Mais uma vez, você pode ver que esse projeto foi criado para ser compilado como um assistente baseado em EXE ou DLL. Quando compilado como um assistente, ele usa o cabeçalho `library` para indicar que é uma DLL e exporta a função `InitWiz( )` para ser inicializado pelo IDE.

Apenas algumas mudanças foram feitas na unidade `Main` nesse projeto. Como já dissemos, a variável `MainForm` precisa ser definida como `nil` quando o assistente não está ativo. Como você aprendeu no Capítulo 2, a variável de instância `MainForm` automaticamente terá o valor `nil` na partida da aplicação. Além disso, no tratador de evento `OnClose` do formulário, a instância do formulário é liberada e a global `MainForm` é redefinida como `nil`. Veja o método a seguir:

```
procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := caFree;
  Application.OnShowHint := FOldShowHint;
  MainForm := nil;
end;
```

O retoque final desse assistente é trazer os arquivos para o Code Editor do IDE quando o usuário der um clique duplo na caixa de listagem do formulário principal. Essa lógica é manipulada por um novo método `FileLBDblClick( )`, mostrado a seguir::

```
procedure TMainForm.FileLBDblClick(Sender: TObject);
{ Chamado quando o usuário dá um clique duplo na caixa de listagem.
  Carrega o arquivo no IDE }
var
  FileName: string;
  Len: Integer;
begin
  { certifica-se de que o usuário deu um clique em um arquivo... }
  if Integer(FileLB.Items.Objects[FileLB.ItemIndex]) > 0) then
  begin
    { Elimina "File " e ":" da string }
    FileName := Copy(FileName, 6, Length(FileName));
    Len := Length(FileName);
    if FileName[Len] = ':' then SetLength(FileName, Len – 1);
    { Abre o projeto ou o arquivo }
{ $IFNDEF BUILD_EXE}
if CompareText(ExtractFileExt(FileName), '.DPR') = 0 then
      ActionSvc.OpenProject(FileName, True)
    else
      ActionSvc.OpenFile(FileName);
{ $ELSE }
```

```
    ShellExecute(0, 'open', PChar(FileName), nil, nil, SW_SHOWNORMAL);
{ $ENDIF}
end;
end;
```

Quando compilado como um assistente, esse método emprega os métodos `OpenFile( )` e `OpenProject( )` de `IOTAActionServices` para abrir um determinado arquivo. Como um EXE independente, esse método chama a função `ShellExecute( )` da API para abrir o arquivo usando a aplicação default associada à extensão do arquivo.

A Listagem 17.8 mostra o código-fonte completo da unidade `Main` no projeto `DDGSrch`, e a Figura 17.7 mostra o assistente DDG Search fazendo seu trabalho dentro do IDE.

Listagem 17.8 `Main.pas` – a unidade principal do projeto DDGSrch

```
unit Main;

interface

{$WARN UNIT_PLATFORM OFF}

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, Menus, SrchIni,
  SrchU, ComCtrls;

type
  TMainForm = class(TForm)
    FileLB: TListBox;
    PopupMenu1: TPopupMenu;
    Font1: TMenuItem;
    N1: TMenuItem;
    Exit1: TMenuItem;
    FontDialog1: TFontDialog;
    StatusBar: TStatusBar;
    AlignPanel: TPanel;
    ControlPanel: TPanel;
    ParamsGB: TGroupBox;
    LFileSpec: TLabel;
    LToken: TLabel;
    lPathName: TLabel;
    EFileSpec: TEdit;
    EToken: TEdit;
    PathButton: TButton;
    OptionsGB: TGroupBox;
    cbCaseSensitive: TCheckBox;
    cbFileNamesOnly: TCheckBox;
    cbRecurse: TCheckBox;
    SearchButton: TBitBtn;
    CloseButton: TBitBtn;
    PrintButton: TBitBtn;
    PriorityButton: TBitBtn;
    View1: TMenuItem;
    EPathName: TEdit;
    procedure SearchButtonClick(Sender: TObject);
    procedure PathButtonClick(Sender: TObject);
    procedure FileLBDrawItem(Control: TWinControl; Index: Integer;
      Rect: TRect; State: TOwnerDrawState);
```

Listagem 17.8 Continuação

```
    procedure Font1Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure PrintButtonClick(Sender: TObject);
    procedure CloseButtonClick(Sender: TObject);
    procedure FileLBDblClick(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure PriorityButtonClick(Sender: TObject);
    procedure ETokenChange(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    FOldShowHint: TShowHintEvent;
    procedure ReadIni;
    procedure WriteIni;
    procedure DoShowHint(var HintStr: string; var CanShow: Boolean;
      var HintInfo: THintInfo);
  protected
    procedure WndProc(var Message: TMessage); override;
  public
    Running: Boolean;
    SearchPri: integer;
    SearchThread: TSearchThread;
    procedure EnableSearchControls(Enable: Boolean);
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

uses Printers, ShellAPI, MemMap, FileCtrl, PriU;

procedure PrintStrings(Strings: TStrings);
{ Este procedimento imprime toda a string do parâmetro Strings }
var
  Prn: TextFile;
  i: word;
begin
  if Strings.Count = 0 then // Existem strings?
  begin
    MessageDlg('No text to print!', mtInformation, [mbOk], 0);
    Exit;
  end;
  AssignPrn(Prn);                          // atribui Prn à impressora
  try
    Rewrite(Prn);                          // abre impressora
    try
      for i := 0 to Strings.Count - 1 do   // percorre todas as strings
        WriteLn(Prn, Strings.Strings[i]);  // escreve na impressora
    finally
      CloseFile(Prn);                      // fecha impressora
    end;
  except
    on EInOutError do
      MessageDlg('Error Printing text.', mtError, [mbOk], 0);
```

Listagem 17.8 Continuação

```
  end;
end;

procedure TMainForm.EnableSearchControls(Enable: Boolean);
{ Ativa ou desativa certos controles de modo que as opções não possam }
{ ser modificadas enquanto a pesquisa está sendo executada. }
begin
  SearchButton.Enabled := Enable;        // ativa/desativa controles
  cbRecurse.Enabled := Enable;
  cbFileNamesOnly.Enabled := Enable;
  cbCaseSensitive.Enabled := Enable;
  PathButton.Enabled := Enable;
  EPathName.Enabled := Enable;
  EFileSpec.Enabled := Enable;
  EToken.Enabled := Enable;
  Running := not Enable;                 // define flag Running
  ETokenChange(nil);
  with CloseButton do
  begin
    if Enable then
    begin             // define propriedades do botão Close/Stop
      Caption := '&Close';
      Hint := 'Close Application';
    end
    else begin
      Caption := '&Stop';
      Hint := 'Stop Searching';
    end;
  end;
end;

procedure TMainForm.SearchButtonClick(Sender: TObject);
{ Chamado quando o usuário dá um clique no botão Search.
  Chama thread de pesquisa. }
begin
  EnableSearchControls(False);          // desativa controles
  FileLB.Clear;                         // apaga caixa de listagem
  { inicia thread }
  SearchThread := TSearchThread.Create(cbCaseSensitive.Checked,
    cbFileNamesOnly.Checked, cbRecurse.Checked, EToken.Text,
    EPathName.Text, EFileSpec.Text, Handle);
end;

procedure TMainForm.ETokenChange(Sender: TObject);
begin
  SearchButton.Enabled := not Running and (EToken.Text <<|>> '');
end;

procedure TMainForm.PathButtonClick(Sender: TObject);
{ Chamado quando botão Path é acionado. Permite que usuário escolha
  novo caminho. }
var
  ShowDir: string;
begin
  ShowDir := EPathName.Text;
  if SelectDirectory(ShowDir, [ ], 0) then
    EPathName.Text := ShowDir;
```

Listagem 17.8 Continuação

```
end;

procedure TMainForm.FileLBDblClick(Sender: TObject);
{ Cahamado quando usuário dá um clique duplo na caixa de listagem.
  Carrega arquivo no IDE. }
var
  FileName: string;
  Len: Integer;
begin
  { verifica se o usuário clicou em um arquivo... }
  if Integer(FileLB.Items.Objects[FileLB.ItemIndex]) > 0 then
  begin
    FileName := FileLB.Items[FileLB.ItemIndex];
    { Remove "File " e ":" da string }
    FileName := Copy(FileName, 6, Length(FileName));
    Len := Length(FileName);
    if FileName[Len] = ':' then SetLength(FileName, Len - 1);
    { Abre o projeto ou arquivo }
{$IFNDEF BUILD_EXE}
    if CompareText(ExtractFileExt(FileName), '.DPR') = 0 then
      ActionSvc.OpenProject(FileName, True)
    else
      ActionSvc.OpenFile(FileName);
{$ELSE}
    ShellExecute(0, 'open', PChar(FileName), nil, nil, SW_SHOWNORMAL);
{$ENDIF}
  end;
end;

procedure TMainForm.FileLBDrawItem(Control: TWinControl;
  Index: Integer; Rect: TRect; State: TOwnerDrawState);
{ Chamado para a propriedade desenhar caixa de listagem. }
var
  CurStr: string;
begin
  with FileLB do
  begin
    CurStr := Items.Strings[Index];
    Canvas.FillRect(Rect);              // apaga retângulo
    if not cbFileNamesOnly.Checked then // se não é apenas nome de arquivo
    begin
      { if current line is file name... }
      if Integer(Items.Objects[Index]) > 0 then
        Canvas.Font.Style := [fsBold]; // fonte em negrito
      end
    else
      Rect.Left := Rect.Left + 15;          // se não, recua
    DrawText(Canvas.Handle, PChar(CurStr), Length(CurStr), Rect, dt_SingleLine);
  end;
end;

procedure TMainForm.Font1Click(Sender: TObject);
{ Permite que o usuário escolha nova fonte para caixa de listagem }
begin
  { Escolhe nova fonte da caixa de listagem }
  if FontDialog1.Execute then
    FileLB.Font := FontDialog1.Font;
```

Listagem 17.8 Continuação

```
end;


procedure TMainForm.FormDestroy(Sender: TObject);
{ Tratador de evento OnDestroy para o formulário }
begin
  WriteIni;
end;

procedure TMainForm.FormCreate(Sender: TObject);
{ Tratador de evento OnCreate para o formulário }
begin
  Application.HintPause := 0;             // não espera para mostrar dicas
  FOldShowHint := Application.OnShowHint; // configura dicas
  Application.OnShowHint := DoShowHint;
  ReadIni;                                // lê arquivo INI
end;

procedure TMainForm.DoShowHint(var HintStr: string; var CanShow: Boolean;
  var HintInfo: THintInfo);
{ Tratador de evento OnHint para Application }
begin
  { Apresenta dicas de aplicação na barra de status }
  StatusBar.Panels[0].Text := HintStr;
  { Não mostra dica de tela se estivermos sobre nossos próprios controles }
  if (HintInfo.HintControl < > nil) and
    (HintInfo.HintControl.Parent < > nil) and
    ((HintInfo.HintControl.Parent = ParamsGB) or
    (HintInfo.HintControl.Parent = OptionsGB) or
    (HintInfo.HintControl.Parent = ControlPanel)) then
    CanShow := False;
  if Assigned(FOldShowHint) then
    FOldShowHint(HintStr, CanSHow, HintInfo);
end;

procedure TMainForm.PrintButtonClick(Sender: TObject);
{ Chamado quando botão Print é acionado. }
begin
  if MessageDlg('Send search results to printer?', mtConfirmation,
    [mbYes, mbNo], 0) = mrYes then
    PrintStrings(FileLB.Items);
end;

procedure TMainForm.CloseButtonClick(Sender: TObject);
{ Chamado para interromper thread ou fechar aplicação }
begin
  // se o thread estiver rodando, então termina thread
  if Running then SearchThread.Terminate
  // caso contrário, fecha aplicação
  else Close;
end;


procedure TMainForm.FormResize(Sender: TObject);
{ Tratador de evento OnResize. Centraliza controles no formulário. }
begin
  { divide barra de status em dois painéis com uma divisão de 1/3 – 2/3 }
```

Listagem 17.8 Continuação

```
  with StatusBar do
  begin
    Panels[0].Width := Width div 3;
    Panels[1].Width := Width * 2 div 3;
  end;
  { centraliza controles no meio do formulário }
  ControlPanel.Left := (AlignPanel.Width div 2) - (ControlPanel.Width div 2);
end;

procedure TMainForm.PriorityButtonClick(Sender: TObject);
{ Mostra formulário de prioridade do thread }
begin
  ThreadPriWin.Show;
end;

procedure TMainForm.ReadIni;
{ Lê valores default do Registro }
begin
  with SrchIniFile do
  begin
    EPathName.Text := ReadString('Defaults', 'LastPath', 'C:\');
    EFileSpec.Text := ReadString('Defaults', 'LastFileSpec', '*.*');
    EToken.Text := ReadString('Defaults', 'LastToken', '');
    cbFileNamesOnly.Checked := ReadBool('Defaults', 'FNamesOnly', False);
    cbCaseSensitive.Checked := ReadBool('Defaults', 'CaseSens', False);
    cbRecurse.Checked := ReadBool('Defaults', 'Recurse', False);
    Left := ReadInteger('Position', 'Left', 100);
    Top := ReadInteger('Position', 'Top', 50);
    Width := ReadInteger('Position', 'Width', 510);
    Height := ReadInteger('Position', 'Height', 370);
  end;
end;

procedure TMainForm.WriteIni;
{ Grava configurações atuais de volta no Registro }
begin
  with SrchIniFile do
  begin
    WriteString('Defaults', 'LastPath', EPathName.Text);
    WriteString('Defaults', 'LastFileSpec', EFileSpec.Text);
    WriteString('Defaults', 'LastToken', EToken.Text);
    WriteBool('Defaults', 'CaseSens', cbCaseSensitive.Checked);
    WriteBool('Defaults', 'FNamesOnly', cbFileNamesOnly.Checked);
    WriteBool('Defaults', 'Recurse', cbRecurse.Checked);
    WriteInteger('Position', 'Left', Left);
    WriteInteger('Position', 'Top', Top);
    WriteInteger('Position', 'Width', Width);
    WriteInteger('Position', 'Height', Height);
  end;
end;

procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := caFree;
  Application.OnShowHint := FOldShowHint;
  MainForm := nil;
end;
```

Listagem 17.8 Continuação

```
procedure TMainForm.WndProc(var Message: TMessage);
begin
  if Message.Msg = DDGM_ADDSTR then
  begin
    FileLB.Items.AddObject(PChar(Message.WParam), TObject(Message.LParam));
    StrDispose(PChar(Message.WParam));
  end
  else
    inherited WndProc(Message);
end;

end.
```

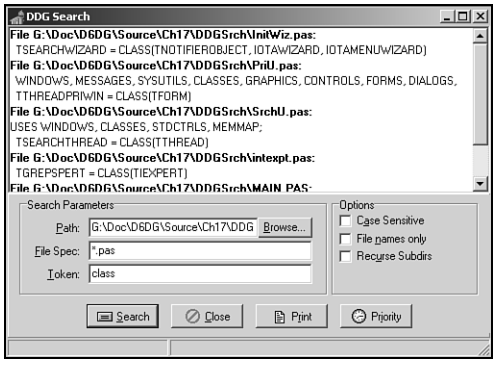

Figura 17.7 O assistente DDG Search em ação.

DICA

Observe a seguinte linha da Listagem 17.8:

```
{ $WARN UNIT_PLATFORM OFF }
```

Essa diretiva do compilador é usada para silenciar o aviso, durante a compilação, que é gerado porque `Main.pas` usa a unidade `FileCtrl`, que é uma unidade específica da plataforma Windows. `FileCtrl` é marcada como tal por meio da diretiva `platform`.

Assistentes de formulário

Um outro tipo de assistente suportado pela API Open Tools é o assistente de formulário. Uma vez instalados, os assistentes de formulário são acessados a partir da caixa de diálogo New Items; eles geram novos formulários e unidades para o usuário. O Capítulo 16 empregou esse tipo de assistente para gerar novos formulários `AppBar`; no entanto, você não conseguiu ver o código que fez o assistente funcionar.

É extremamente simples criar um assistente de formulário, embora você deva implementar uma boa quantidade de métodos de interface. A criação de um assistente de formulário pode ser dividida em cinco etapas básicas:

1. Crie uma classe descendente de `TCustomForm`, `TDataModule` ou qualquer `TWinControl`, que será usada como a classe básica do formulário. Geralmente, essa classe residirá em uma unidade separada do assistente. Nesse caso, `TAppBar` servirá como a classe básica.
2. Crie um descendente de `TNotifierObject` que implemente as seguintes interfaces: `IOTAWizard`, `IOTARepositoryWizard`, `IOTAFormWizard`, `IOTACreator` e `IOTAModuleCreator`.

3. No seu método `IOTAWizard.Execute( )`, você normalmente chamará `IOTAModuleServices.GetNewModuleAndClassName( )` para obter uma nova unidade e nome de classe para seu assistente e `IOTAModuleServices.CreateModule( )` para instruir o IDE a começar a criação do novo módulo.
4. Muitas das implementações de método das interfaces mencionadas acima possuem apenas uma linha. Os não-triviais são os métodos `NewFormFile( )` e `NewImplFile( )` de `IOTAModuleCreator`, que retornam o código para o formulário e a unidade, respectivamente. O método `IOTACreator.GetOwner( )` pode ser um pouco complicado, mas o exemplo a seguir oferece uma boa técnica para adicionar a unidade ao projeto atual (se houver).
5. Complete o procedimento `Register( )` do assistente registrando um tratador para a nova classe de formulário, usando o procedimento `RegisterCustomModule( )` na unidade `DsgnIntf` e criando o assistente com a chamada ao procedimento `RegisterPackageWizard( )` da unidade ToolsAPI.

A Listagem 17.9 mostra o código-fonte de `ABWizard.pas`, que é o assistente `AppBar`.

Listagem 17.9 `ABWizard.pas`, a unidade que contém a implementação do assistente `AppBar`

```
unit ABWizard;

interface

uses Windows, Classes, ToolsAPI;

type
  TAppBarWizard = class(TNotifierObject, IOTAWizard, IOTARepositoryWizard,
    IOTAFormWizard, IOTACreator, IOTAModuleCreator)
  private
    FUnitIdent: string;
    FClassName: string;
    FFileName: string;
  protected
    // Métodos de IOTAWizard
    function GetIDString: string;
    function GetName: string;
    function GetState: TWizardState;
    procedure Execute;
    // Métodos de IOTARepositoryWizard / IOTAFormWizard
    function GetAuthor: string;
    function GetComment: string;
    function GetPage: string;
    function GetGlyph: HICON;
    // Métodos de IOTACreator
    function GetCreatorType: string;
    function GetExisting: Boolean;
    function GetFileSystem: string;
    function GetOwner: IOTAModule;
    function GetUnnamed: Boolean;
    // Métodos de IOTAModuleCreator
    function GetAncestorName: string;
    function GetImplFileName: string;
    function GetIntfFileName: string;
    function GetFormName: string;
    function GetMainForm: Boolean;
    function GetShowForm: Boolean;
    function GetShowSource: Boolean;
```

Listagem 17.9 Continuação

```
    function NewFormFile(const FormIdent, AncestorIdent: string): IOTAFile;
    function NewImplSource(const ModuleIdent, FormIdent,
      AncestorIdent: string): IOTAFile;
    function NewIntfSource(const ModuleIdent, FormIdent,
      AncestorIdent: string): IOTAFile;
    procedure FormCreated(const FormEditor: IOTAFormEditor);
  end;

implementation

uses Forms, AppBars, SysUtils, DsgnIntf;

{$R CodeGen.res}

type
  TBaseFile = class(TInterfacedObject)
  private
    FModuleName: string;
    FFormName: string;
    FAncestorName: string;
  public
    constructor Create(const ModuleName, FormName, AncestorName: string);
  end;

  TUnitFile = class(TBaseFile, IOTAFile)
  protected
    function GetSource: string;
    function GetAge: TDateTime;
  end;

  TFormFile = class(TBaseFile, IOTAFile)
  protected
    function GetSource: string;
    function GetAge: TDateTime;
  end;

{ TBaseFile }

constructor TBaseFile.Create(const ModuleName, FormName,
  AncestorName: string);
begin
  inherited Create;
  FModuleName := ModuleName;
  FFormName := FormName;
  FAncestorName := AncestorName;
end;

{ TUnitFile }

function TUnitFile.GetSource: string;
var
  Text: string;
  ResInstance: THandle;
  HRes: HRSRC;
begin
  ResInstance := FindResourceHInstance(HInstance);
  HRes := FindResource(ResInstance, 'CODEGEN', RT_RCDATA);
```

Listagem 17.9 Continuação

```
  Text := PChar(LockResource(LoadResource(ResInstance, HRes)));
  SetLength(Text, SizeOfResource(ResInstance, HRes));
  Result := Format(Text, [FModuleName, FFormName, FAncestorName]);
end;

function TUnitFile.GetAge: TDateTime;
begin
  Result := -1;
end;

{ TFormFile }

function TFormFile.GetSource: string;
const
  FormText =
    'object %0:s: T%0:s'#13#10'end';
begin
  Result := Format(FormText, [FFormName]);
end;

function TFormFile.GetAge: TDateTime;
begin
  Result := -1;
end;

{ TAppBarWizard }

{ TAppBarWizard.IOTAWizard }

function TAppBarWizard.GetIDString: string;
begin
  Result := 'DDG.AppBarWizard';
end;

function TAppBarWizard.GetName: string;
begin
  Result := 'DDG AppBar Wizard';
end;

function TAppBarWizard.GetState: TWizardState;
begin
  Result := [wsEnabled];
end;

procedure TAppBarWizard.Execute;
begin
  (BorlandIDEServices as IOTAModuleServices).GetNewModuleAndClassName(
    'AppBar', FUnitIdent, FClassName, FFileName);
  (BorlandIDEServices as IOTAModuleServices).CreateModule(Self);
end;

{ TAppBarWizard.IOTARepositoryWizard / TAppBarWizard.IOTAFormWizard }

function TAppBarWizard.GetGlyph: HICON;
begin
  Result := 0;  // usa ícone padrão
end;
```

Listagem 17.9 Continuação

```
function TAppBarWizard.GetPage: string;
begin
  Result := 'DDG';
end;

function TAppBarWizard.GetAuthor: string;
begin
  Result := 'Delphi 5 Developer''s Guide';
end;

function TAppBarWizard.GetComment: string;
begin
  Result := 'Creates a new AppBar form.'
end;

{ TAppBarWizard.IOTACreator }

function TAppBarWizard.GetCreatorType: string;
begin
  Result := '';
end;

function TAppBarWizard.GetExisting: Boolean;
Begin
  Result := False;
end;

function TAppBarWizard.GetFileSystem: string;
begin
  Result := '';
end;

function TAppBarWizard.GetOwner: IOTAModule;
var
  I: Integer;
  ModServ: IOTAModuleServices;
  Module: IOTAModule;
  ProjGrp: IOTAProjectGroup;
begin
  Result := nil;
  ModServ := BorlandIDEServices as IOTAModuleServices;
  for I := 0 to ModServ.ModuleCount – 1 do
  begin
    Module := ModSErv.Modules[I];
    // localiza grupo de projeto atual
    if CompareText(ExtractFileExt(Module.FileName), '.bpg') = 0 then
      if Module.QueryInterface(IOTAProjectGroup, ProjGrp) = S_OK then
      begin
        // retorna projeto ativo do grupo
        Result := ProjGrp.GetActiveProject;
        Exit;
      end;
  end;
end;

function TAppBarWizard.GetUnnamed: Boolean;
begin
```

Listagem 17.9 Continuação

```
  Result := True;
end;

{ TAppBarWizard.IOTAModuleCreator }

function TAppBarWizard.GetAncestorName: string;
begin
  Result := 'TAppBar';
end;

function TAppBarWizard.GetImplFileName: string;
var
  CurrDir: array[0..MAX_PATH] of char;
begin
  // Nota: é obrigatório o nome completo do caminho!
  GetCurrentDirectory(SizeOf(CurrDir), CurrDir);
  Result := Format('%s\%s.pas', [CurrDir, FUnitIdent, '.pas']);
end;

function TAppBarWizard.GetIntfFileName: string;
begin
  Result := '';
end;

function TAppBarWizard.GetFormName: string;
begin
  Result := FClassName;
end;

function TAppBarWizard.GetMainForm: Boolean;
begin
  Result := False;
end;

function TAppBarWizard.GetShowForm: Boolean;
begin
  Result := True;
end;

function TAppBarWizard.GetShowSource: Boolean;
begin
  Result := True;
end;

function TAppBarWizard.NewFormFile(const FormIdent,
  AncestorIdent: string): IOTAFile;
begin
  Result := TFormFile.Create('', FormIdent, AncestorIdent);
end;

function TAppBarWizard.NewImplSource(const ModuleIdent, FormIdent,
  AncestorIdent: string): IOTAFile;
begin
  Result := TUnitFile.Create(ModuleIdent, FormIdent, AncestorIdent);
end;

function TAppBarWizard.NewIntfSource(const ModuleIdent, FormIdent,
```

Listagem 17.9 Continuação

```
  AncestorIdent: string): IOTAFile;
Begin
  Result := nil;
end;

procedure TAppBarWizard.FormCreated(const FormEditor: IOTAFormEditor);
begin
  // não faz nada
end;

end.
```

Essa unidade emprega um truque interessante para a criação de código-fonte: o código-fonte não-formatado é armazenado em um arquivo RES que é vinculado à diretiva $R. Essa é uma forma muito flexível de armazenar o código-fonte de um assistente de modo que possa ser prontamente modificado. O arquivo RES é construído incluindo-se um arquivo de texto e o recurso RCDATA em um arquivo RC e em seguida compilando-se esse arquivo RC com BRCC32. As Listagens 17.10 e 17.11 mostram o conteúdo de CodeGen.txt e CodeGen.rc.

Listagem 17.10 CodeGen.txt – o modelo de recurso do assistente AppBar

```
unit %0:s;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, AppBars;

type
  T%1:s = class(%2:s)
  private
    { Declarações privadas }
  public
    { Declarações públicas }
  end;

var
  %1:s: T%1:s;

implementation

{$R *.DFM}

end.
```

Listagem 17.11 CODEGEN.RC

```
CODEGEN RCDATA CODEGEN.TXT
```

O registro do assistente e do módulo personalizado ocorre dentro de um procedimento Register( ) no pacote de projeto que contém o assistente, usando as duas linhas a seguir:

```
  RegisterCustomModule(TAppBar, TCustomModule);
  RegisterPackageWizard(TAppBarWizard.Create);
```

Resumo

Depois da leitura deste capítulo, você deve ter uma compreensão maior das diversas unidades e interfaces envolvidas na API Open Tools do Delphi. Em particular, você deve saber e entender as questões envolvidas na criação de assistentes plugados no IDE. Este capítulo completa essa parte do livro. Na próxima seção, você aprenderá técnicas para criar aplicações de porte empresarial, começando com aplicações baseadas em COM+ e MTS.

Desenvolvimento BizSnap: escrevendo Web Services baseados em SOAP

CAPÍTULO

20

NESTE CAPÍTULO

- O que são Web Services?
- O que é SOAP?
- Escrevendo um Web Service
- Invocando um Web Service por um cliente

O desenvolvimento de soluções de eBusiness *rapidamente* é a chave para o sucesso de muitas organizações. Felizmente, a Borland possibilitou esse desenvolvimento rápido por meio de um novo recurso no Delphi 6, chamado BizSnap. BizSnap é uma tecnologia que integra XML e Web Services (serviços da Web) usando um protocolo SOAP no Delphi 6.

O que são Web Services?

A Borland descreve os Web Services da seguinte maneira:

> Usando a Internet e a infra-estrutura da Web como plataforma, Web Services conectam, de forma transparente, aplicações, processos comerciais, clientes e fornecedores – em qualquer lugar do mundo – com uma linguagem padronizada e protocolos da Internet independentes de máquina.

As aplicações distribuídas geralmente consistem em servidores e clientes – servidores que oferecem alguma funcionalidade aos clientes. Qualquer aplicação distribuída pode conter muitos servidores, e esses servidores também podem ser clientes. Web Services são um novo tipo de componente servidor para aplicações com arquitetura distribuída. Web Services são aplicações que utilizam protocolos comuns da Internet para oferecer sua funcionalidade.

Como os Web Services se comunicam usando padrões abertos, eles oferecem a oportunidade para a operação integrada entre várias plataformas diferentes. Por exemplo, do ponto de vista de uma aplicação cliente, um Web Service instalado em uma máquina Sun Solaris parecerá (para todas as intenções e propósitos) idêntico ao mesmo servidor instalado em uma máquina Windows NT. Antes dos Web Services, esse tipo de integração era extremamente demorado, dispendioso, e geralmente, usava um padrão próprio.

Essa natureza aberta e a capacidade de usar hardware e software de rede existentes posiciona os Web Services para serem ferramentas poderosas em transações internas e entre empresas.

O que é SOAP?

SOAP é um acrônimo que significa *Simple Object Access Protocol* (protocolo simples de acesso a objeto). SOAP é um protocolo simplificado, usado para a troca de dados em um ambiente distribuído, semelhante a partes da arquitetura CORBA e DCOM, mas com menos funcionalidade e overhead resultante. SOAP troca dados usando documentos XML, por meio de HTTP (ou HTTPS) para suas comunicações. Uma especificação sobre SOAP está disponível para referência na Internet, no endereço `http://www.w3.org/TR/SOAP/`.

Web Services também usam uma forma de XML para instruir os usuários a respeito deles, chamada WSDL. WSDL é uma abreviação de *Web Services Description Languag*e (linguagem de descrição de Web Services). WSDL é usada por aplicações cliente para identificar o que um Web Service pode fazer, onde ele pode ser encontrado e como chamá-lo.

A coisa mais maravilhosa sobre o BizSnap é que você não precisa aprender todos os detalhes de SOAP, XML ou WSDL para criar aplicações de Web Service.

Neste capítulo, vamos mostrar como é simples criar um Web Service, e depois vamos mostrar como acessar esse serviço a partir de uma aplicação cliente.

Escrevendo um Web Service

Para demonstrar a criação de um Web Service, vamos mostrar como criar o popular conversor de temperaturas Fahrenheit/Celsius como um Web Service.

Um Web Service escrito em Delphi consiste em três coisas principais. A primeira é um WebModule com alguns componentes SOAP (descritos mais adiante). O módulo é criado automaticamente para você quando o SOAP Server Wizard é executado. Os dois componentes seguintes precisam ser criados por você mesmo. Um deles é uma implementação de classe, que é simplesmente o código que descreve o que o seu Web Service realmente fará. A segunda coisa a criar é uma interface

para essa classe. A interface irá expor somente as partes da classe que você deseja oferecer ao restante do mundo através do seu Web Service.

O Delphi oferece um Web Service Wizard na guia WebServices do Object Repository. Você verá três itens nessa guia. Neste ponto, vamos nos concentrar apenas no Soap Server Application Wizard. Quando você der um clique nisso, verá a caixa de diálogo New Soap Server Application (veja a Figura 20.1). Essa caixa de diálogo deverá ser conhecida, se você já realizou algum desenvolvimento com servidor Web. De fato, Web Services são, na realidade, servidores Web que tratam da resposta SOAP específica.

Figura 20.1 A caixa de diálogo New Soap Server Application.

Em nosso exemplo, escolhemos um CGI Stand-alone Executable. Dê um clique em OK e o assistente gerará um `TWebModule`, como mostra a Figura 20.2.

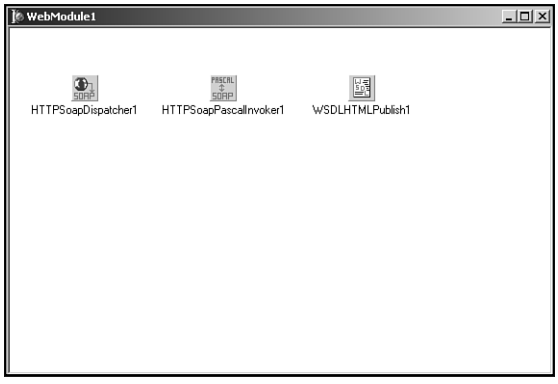

Figura 20.2 O Web Module gerado pelo assistente.

Uma análise de `TWebModule`

Existem três componentes no `TWebModule`. A finalidade desses componentes é a seguinte:

- `THTTPSoapDispatcher` recebe mensagens SOAP e as despacha para o *invocador* apropriado, conforme especificado pela propriedade `Dispatcher`.
- `THTTPSoapPascalInvoker` é o componente referenciado pela propriedade `THTTPSoapDispatcher.Dispatcher`. Esse componente recebe a mensagem SOAP, interpreta-a e depois invoca a interface invocável chamada pela mensagem.
- `TWSDLHTMLPublish` é usado para publicar a lista de documentos WSDL que contêm as informações sobre as interfaces invocáveis disponíveis. Isso permite que clientes fora do Delphi identifiquem e usem os métodos que se tornaram disponíveis por meio de um determinado Web Service.

Não há nada que você tenha que fazer com o Web Module neste ponto. No entanto, é preciso definir e implementar uma interface invocável.

Definindo uma interface invocável

Você precisa criar uma nova unidade na qual colocará sua definição de interface. A Listagem 20.1 mostra o código-fonte para a unidade que criamos para nossa aplicação de demonstração, que você encontrará no CD-ROM que acompanha este livro. Chamamos essa unidade de `TempConverterIntf.pas`.

Listagem 20.1 `TempConverter.pas` – definição da interface invocável

```
unit TempConverterIntf;

interface

type
  ITempConverter = Interface(IInvokable)
    ['{6D239CB5-6E74-445B-B101-F76F5C0F6E42}']
    function FahrenheitToCelsius(AFValue: double): double; stdcall;
    function CelsiusToFahrenheit(ACValue: double): double; stdcall;
    function Purpose: String; stdcall;
  end;

implementation
uses InvokeRegistry;
initialization
  InvRegistry.RegisterInterface(TypeInfo(ITempConverter));

end.
```

Essa pequena unidade contém somente uma interface, que define os métodos que pretendemos publicar como parte do nosso Web Service. Você notará que nossa interface descende de `IInvokable`. `IInvokable` é uma interface simples, compilada com a opção `{M+}` do compilador, para garantir que a RTTI seja compilada em todos os seus descendentes. Isso é necessário para permitir que os serviços e clientes Web traduzam o código e informações simbólicas passadas um para o outro.

Em nosso exemplo, definimos dois métodos para converter temperaturas e um método `Purpose( )`, que retorna uma string. Além disso, observe que fornecemos um GUID para essa interface, para que tenha uma identificação exclusiva (para criar um GUID no seu próprio código, basta pressionar Ctrl+Shift+G no editor).

> ATENÇÃO
>
> Observe que cada método definido na interface invocável é definido usando a convenção de chamada `stdcall`. Essa convenção precisa ser usada ou então a interface invocável não funcionará.

Finalmente, os últimos itens a observar são o usuário da unidade `InvokeRegistry` e a chamada para `InvRegistry.RegisterInterface( )`. O componente `THTTPSoapPascalInvoker` precisa ser capaz de identificar a interface invocável quando receber uma mensagem SOAP. A chamada do método `RegisterInterface( )` registra a interface com o registro de chamadas. Quando discutirmos sobre o código do cliente, mais adiante, você verá que a chamada `RegisterInterface( )` também é feita no cliente. O servidor exige o registro de modo que possa identificar a implementação da interface que será executada em uma chamada de interface. No cliente, o método é usado para permitir que os componentes pesquisem informações sobre interfaces invocáveis e como chamá-las. Colocando a chamada `RegisterInterface( )` no bloco de inicialização, garantimos que o método seja chamado quando o serviço for executado.

Implementando uma interface invocável

A implementação de uma interface invocável não é diferente da implementação de qualquer interface. A Listagem 20.2 mostra o código-fonte para nossa interface de conversão de temperatura.

Listagem 20.2 `TempConverterImpl.pas` – implementação da interface invocável

```
unit TempConverterImpl;

interface
uses InvokeRegistry, TempConverterIntf;
type

  TTempConverter = class(TInvokableClass, ITempConverter)
  public
    function FahrenheitToCelsius(AFValue: double): double; stdcall;
    function CelsiusToFahrenheit(ACValue: double): double; stdcall;
    function Purpose: String; stdcall;
  end;

implementation

{ TTempConverter }

function TTempConverter.CelsiusToFahrenheit(ACValue: double): double;
begin
// Tf = (9/5)*Tc+32
  Result := (9/5)*ACValue+32;
end;

function TTempConverter.FahrenheitToCelsius(AFValue: double): double;
begin
// Tc = (5/9)*(Tf-32)

  Result := (5/9)*(AFValue-32);
end;

function TTempConverter.Purpose: String;
begin
  Result := 'Temperature converstions';
end;

initialization
  InvRegistry.RegisterInvokableClass(TTempConverter);
end.
```

Primeiro, observe que nossa implementação da interface é descendente do objeto `TInvokableClass`. Existem dois motivos principais para fazermos isso. Os motivos a seguir foram apanhados da ajuda on-line do Delphi 6:

- O registro de invocação (`InvRegistry`) sabe como criar instâncias de `TInvokableClass` e (como possui um construtor virtual) seus descendentes. Isso permite que o registro forneça um invocador em uma aplicação de `Web Service` com uma instância da classe `invokable`, que pode lidar com um pedido que chega.
- `TInvokableClass` é um objeto de interface que é liberado quando a contagem de referência em sua interface cai para zero. Os componentes invocadores não sabem quando liberar as classes de implementação das interfaces que chamam. Como `TInvokableClass` sabe quando ser liberada, você não precisa fornecer seu próprio gerenciamento de tempo de vida para esse objeto.

Além do mais, você verá que nossa classe `TTempConverter` implementa a interface `ITempConverter`. Os métodos de implementação para realizar conversões de temperatura são auto-explicativos.

Na seção de inicialização, a chamada para `RegisterInvokableClass( )` registra a classe `TTempConverter` com o registro de invocação. Isso só é necessário no servidor para que o Web Service possa invocar a implementação correta da interface.

Isso realmente é tudo o que é preciso para a criação de um Web Service simples. Nesse ponto, você pode compilar o Web Service e colocá-lo em um diretório executável de um servidor Web, como IIS ou Apache. Normalmente, esse seria um diretório `\Scripts` ou `\cgi-bin`.

Testando o Web Service

O URL `http://127.0.0.1/cgi-bin/TempConvWS.exe/wsdl/ITempConverter` foi usado para exibir o documento WSDL gerado a partir do nosso Web Service. Esse serviço está abrigado em um servidor Apache. Para obter uma lista de todas as interfaces de serviço disponíveis a partir de um Web Service gerado pelo Delphi, o URL pode terminar com `wsdl`. Para ver o documento WSDL específico para um serviço, inclua o nome da interface desejada – nesse caso, `ItempConverter`. O documento WSDL resultante aparece na Listagem 20.3.

Listagem 20.3 Documento WSDL resultante do Web Service

```
<?xml version="1.0" ?>
- <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
➥xmlns:xs="http://www.w3.org/2001/XMLSchema" name="ITempConverterservice"
targetNamespace="http://www.borland.com/soapServices/"
➥xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
- <message name="FahrenheitToCelsiusRequest">
  <part name="AFValue" type="xs:double" />
  </message>
- <message name="FahrenheitToCelsiusResponse">
  <part name="return" type="xs:double" />
  </message>
- <message name="CelsiusToFahrenheitRequest">
  <part name="ACValue" type="xs:double" />
  </message>
- <message name="CelsiusToFahrenheitResponse">
  <part name="return" type="xs:double" />
  </message>
  <message name="PurposeRequest" />
- <message name="PurposeResponse">
  <part name="return" type="xs:string" />
  </message>
- <portType name="ITempConverter">
- <operation name="FahrenheitToCelsius">
  <input message="FahrenheitToCelsiusRequest" />
  <output message="FahrenheitToCelsiusResponse" />
  </operation>
- <operation name="CelsiusToFahrenheit">
  <input message="CelsiusToFahrenheitRequest" />
  <output message="CelsiusToFahrenheitResponse" />
  </operation>
- <operation name="Purpose">
  <input message="PurposeRequest" />
  <output message="PurposeResponse" />
  </operation>
  </portType>
```

Listagem 20.3 Continuação

```
- <binding name="ITempConverterbinding" type="ITempConverter">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
- <operation name="FahrenheitToCelsius">
  <soap:operation soapAction="urn:TempConverterIntf-
➥ITempConverter#FahrenheitToCelsius" />
- <input>
  <soap:body use="encoded" encodingStyle="http:
➥//schemas.xmlsoap.org/soap/encoding/"
namespace="urn:TempConverterIntf-ITempConverter" />
  </input>
- <output>
  <soap:body use="encoded" encodingStyle=
➥"http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:TempConverterIntf-ITempConverter" />
  </output>
  </operation>
- <operation name="CelsiusToFahrenheit">
  <soap:operation soapAction="urn:TempConverterIntf-
➥ITempConverter#CelsiusToFahrenheit" />
- <input>
  <soap:body use="encoded" encodingStyle="http:
➥// schemas.xmlsoap.org/soap/encoding/"
  namespace="urn:TempConverterIntf-ITempConverter" />
  </input>
- <output>
  <soap:body use="encoded" encodingStyle=
➥"http://schemas.xmlsoap.org/soap/encoding/"
  namespace="urn:TempConverterIntf-ITempConverter" />
  </output>
  </operation>
- <operation name="Purpose">
  <soap:operation soapAction="urn:TempConverterIntf-ITempConverter#Purpose" />
- <input>
  <soap:body use="encoded" encodingStyle=
➥"http://schemas.xmlsoap.org/soap/encoding/"
  namespace="urn:TempConverterIntf-ITempConverter" />
  </input>
- <output>
  <soap:body use="encoded" encodingStyle=
➥"http://schemas.xmlsoap.org/soap/encoding/"
  namespace="urn:TempConverterIntf-ITempConverter" />
  </output>
  </operation>
  </binding>
- <service name="ITempConverterservice">
- <port name="ITempConverterPort" binding="ITempConverterbinding">
  <soap:address location="http://127.0.0.1/cgi-bin/TempConvWS.exe
➥/soap/ITempConverter" />
  </port>
  </service>
  </definitions>
```

Agora vamos mostrar como é simples invocar um Web Service.

Invocando um `Web Service` por um cliente

Para invocar o Web Service, você precisa conhecer o URL usado para apanhar o documento WSDL. Esse é o mesmo URL que usamos anteriormente.

Para demonstrar isso, usamos uma aplicação simples com um único formulário principal (veja a Figura 20.3).

Essa aplicação é simples: o usuário digita uma temperatura no controle de edição, pressiona o botão de conversão desejado e o valor convertido é exibido no label Temperature. O código-fonte para essa aplicação aparece na Listagem 20.4.

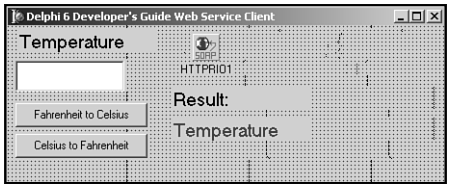

Figura 20.3 O formulário principal para a aplicação cliente do Web Service.

Listagem 20.4 Cliente do Web Service

```
unit MainFrm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Rio, SoapHTTPClient;

type
  TMainForm = class(TForm)
    btnFah2Cel: TButton;
    btnCel2Fah: TButton;
    edtArguement: TEdit;
    lblTemperature: TLabel;
    lblResultValue: TLabel;
    lblResult: TLabel;
    HTTPRIO1: THTTPRIO;
  private
    { Declarações privadas }
  public
    { Declarações públicas }
  end;

var
  MainForm: TMainForm;

implementation

uses TempConvImport;

{$R *.dfm}

end.
```

No formulário principal, acrescentamos um componente THTTPRIO. Um THTTPRIO representa um objeto invocável remotamente e atua como um proxy local para um Web Service, que provavelmente reside em uma máquina de algum lugar remoto. Os dois tratadores de evento TButton realizam o código para chamar o objeto remoto a partir do nosso Web Service. Observe que temos que converter o componente THTTPRIO como ITempConverter para nos referirmos a ele. Depois, podemos invocar sua chamada de método.

Antes que esse código possa ser executado, temos que preparar o componente THTTPRIO, que exige algumas etapas.

Gerando uma unidade de importação para o objeto invocável remoto

Antes que possamos usar o componente THTTPRIO, temos que criar uma unidade de importação para nosso objeto invocável. Felizmente, a Borland facilitou isso oferecendo um assistente para cuidar dessa tarefa. Esse assistente está disponível na página WebServices do Object Repository. Quando iniciado, você verá a caixa de diálogo mostrada na Figura 20.4.

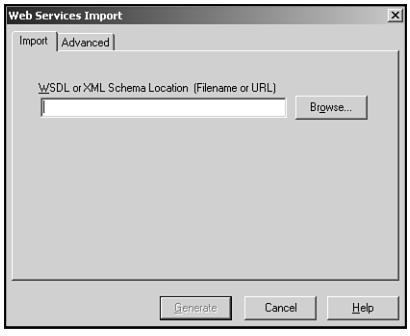

Figura 20.4 O Web Services Import Wizard.

Para importar um Web Service em uma aplicação cliente, você digita o caminho WSDL (o URL especificado anteriormente) no campo WSDL or XML Schema Location e depois pressiona o botão Generate para criar a unidade de importação. A unidade de importação para nosso Web Service aparece na Listagem 20.5 e se parece bastante com nossa unidade de definição de interface original.

Listagem 20.5 Unidade de importação de Web Service

```
Unit TempConvImport;

interface

uses Types, XSBuiltIns;
type

  ITempConverter = interface(IInvokable)
    ['{684379FC-7D4B-4037-8784-B58C63A0280D}']
    function FahrenheitToCelsius(const AFValue: Double): Double;  stdcall;
    function CelsiusToFahrenheit(const ACValue: Double): Double;  stdcall;
    function Purpose: WideString;  stdcall;
  end;

implementation
```

Listagem 20.5 Continuação

```
uses InvokeRegistry;

initialization
  InvRegistry.RegisterInterface(TypeInfo(ITempConverter),
➥ 'urn:TempConverterIntf-ITempConverter', '');

end.
```

Depois que a unidade tiver sido gerada, retorne ao formulário principal da aplicação cliente e use (`uses`) a unidade de importação recém-gerada. Isso fará com que o formulário principal tome conhecimento da nova interface.

Usando o componente `THTTPRIO`

Três propriedades precisam ser definidas para o componente `THTTPRIO`. A primeira, `WSDLLocation`, precisa conter, mais uma vez, o caminho para o documento WSDL. Uma vez definida, você pode soltar a propriedade `Service` para selecionar a única opção disponível. Depois, faça o mesmo para a propriedade `Port`. Nesse ponto, você poderá executar o cliente.

Colocando o `Web Service` em funcionamento

Agora que todas as partes estão prontas, crie um tratador de evento para o evento `OnClick` do botão, por meio de um clique duplo sobre ele. O evento deverá se parecer com o da Listagem 20.6.

Listagem 20.6 Tratador de evento `OnClick`

```
procedure TMainForm.btnFah2CelClick(Sender: TObject);
var
  TempConverter: ITempConverter;
  FloatVal: Double;
begin
  TempConverter := HTTPRIO1 as ITempConverter;
  FloatVal := TempConverter.FahrenheitToCelsius(StrToFloat(edtArguement.Text));
  lblResultValue.Caption := FloatToStr(FloatVal);
end;

procedure TMainForm.btnCel2FahClick(Sender: TObject);
var
  TempConverter: ITempConverter;
  FloatVal: Double;
begin
  TempConverter := HTTPRIO1 as ITempConverter;
  FloatVal := TempConverter.CelsiusToFahrenheit(StrToFloat(edtArguement.Text));
  lblResultValue.Caption := FloatToStr(FloatVal);
end;
```

Enquanto você insere esse código, observe que o CodeInsight do Delphi está disponível para o próprio Web Service. Isso porque o Delphi adaptou o Web Service na sua aplicação como um objeto nativo. As implicações aqui são bastante amplas: qualquer Web Service trazido para uma aplicação Delphi, não importando se esse serviço é instalado em um Solaris, Windows, Linux ou mainframe, e independente da linguagem em que o servidor está escrito, será beneficiado com isso. Além do CodeInsight, a aplicação escrita para usar um Web Service também ganhará uma verificação de tipo do compilador e outros recursos de depuração, devido à forte integração que existe entre estes.

Resumo

Web Services são uma nova e poderosa ferramenta para a computação distribuída, usando padrões abertos e a infra-estrutura existente para permitir a interoperação dentro e entre diferentes plataformas.

Neste capítulo, mostramos como criar um Web Service simples e o cliente para usar esse serviço. Demonstramos as etapas necessárias para se distribuir esse servidor e configurar a propriedade do componente `THTTPRIO` do cliente. Neste ponto, você já deverá estar familiarizado o suficiente com o desenvolvimento de Web Services com maior complexidade. Você pode examinar muito mais exemplos de Web Services no site da comunidade Borland. Um artigo que recomendamos é "Managing Sessions with Delphi 6 Web Services" (gerenciando sessões com Web Services do Delphi 6). Ele foi escrito por Daniel Polistchuck (Article ID: 27575) e pode ser lido em `http://community.borland.com/article/0,1410,27575,00.html`.

Desenvolvimento DataSnap

Por Dan Miser

CAPÍTULO

21

NESTE CAPÍTULO

- **Mecânica da criação de uma aplicação multicamadas**
- **Benefícios da arquitetura multicamadas**
- **Arquitetura típica do DataSnap**
- **Usando o DataSnap para criar uma aplicação**
- **Mais opções para fortalecer sua aplicação**
- **Exemplos do mundo real**
- **Mais recursos de dataset do cliente**
- **Erros clássicos**
- **Distribuindo aplicações DataSnap**

As aplicações multicamadas estão sendo tão comentadas quanto qualquer outro assunto em programação de computador hoje em dia. Isso está acontecendo por um bom motivo. As aplicações multicamadas guardam muitas vantagens em relação às aplicações cliente/servidor mais tradicionais. O DataSnap da Borland é um meio de ajudá-lo a criar e oferecer uma aplicação multicamadas usando Delphi, baseando-se em técnicas e habilidades que você acumulou quando usou o Delphi. Este capítulo o acompanhará por algumas informações gerais sobre o projeto de aplicação multicamadas, mostrando-lhe como aplicar esses princípios para criar aplicações de DataSnap sólidas.

Mecânica da criação de uma aplicação multicamadas

Como estaremos falando sobre uma aplicação multicamadas, pode ser útil oferecer primeiro uma grade de referência para o significado real de uma camada. Uma *camada* (ou *tier*), neste sentido, é uma camada de uma aplicação que oferece algum conjunto específico de funcionalidade. Veja, a seguir, quais são as três camadas básicas usadas em aplicações de banco de dados:

- Dados – A camada de dados é responsável por armazenar seus dados. Normalmente, isso será um RDBMS (ou SGBDR, Sistema de Gerenciamento de Banco de Dados Relacional), como o Microsoft SQL Server, Oracle ou InterBase.
- Comercial – A camada comercial é responsável por apanhar dados da camada de dados em um formato apropriado para a aplicação e realizar a validação final dos dados (também conhecida como *imposição de regras comerciais*). Essa também é a camada do servidor de aplicação.
- Apresentação – Também conhecida como *camada GUI*, essa camada é responsável por exibir os dados em um formato apropriado na aplicação cliente. A camada de apresentação sempre fala com a camada comercial. Ela nunca fala diretamente com a camada de dados.

Nas aplicações cliente/servidor tradicionais, você tem uma arquitetura semelhante à que mostramos na Figura 21.1. Observe que as bibliotecas do cliente para o acesso aos dados precisam estar localizadas em cada máquina cliente isolada. Historicamente, esse tem sido um ponto de muito trabalho na distribuição de aplicações cliente/servidor, devido a versões incompatíveis de DLLs e um dispendioso gerenciamento de configuração. Além disso, como a maior parte da camada comercial está localizada em cada cliente, você precisa atualizar todos os clientes toda vez que precisar atualizar uma regra comercial.

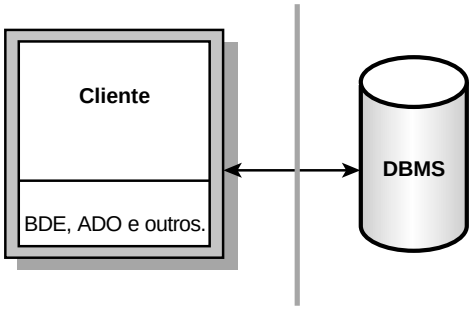

Figura 21.1 A arquivo cliente/servidor tradicional.

Em aplicações multicamadas, a arquitetura se parece mais com a que mostramos na Figura 21.2. Usando essa arquitetura, você encontrará muitos benefícios em relação à aplicação cliente/servidor equivalente.

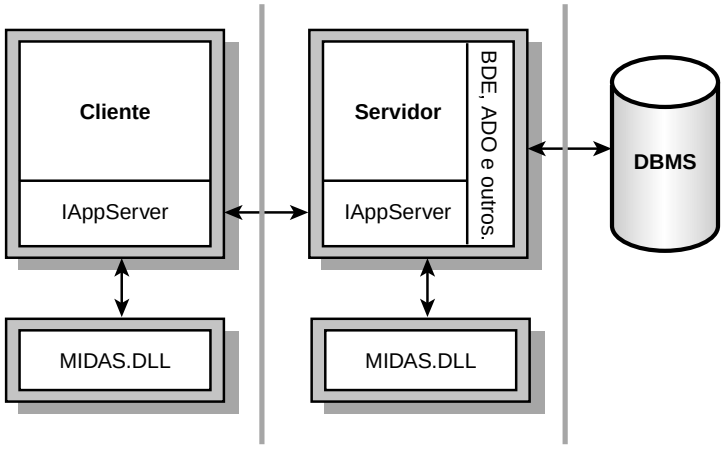

Figura 21.2 Arquitetura multicamadas.

Benefícios da arquitetura multicamadas

Nas próximas seções, listamos os principais benefícios da arquitetura multicamadas.

Lógica comercial centralizada

Na maioria das aplicações cliente/servidor, cada aplicação cliente precisa registrar as regras comerciais individuais para uma solução comercial. Isso não somente aumenta o tamanho do executável, mas também impõe um desafio para o desenvolvedor de software, que deve manter controle estrito sobre a manutenção de versão. Se o usuário A tiver uma versão mais antiga da aplicação do que o usuário B, as regras comerciais podem não ser executadas de modo coerente, resultando assim em erros lógicos nos dados. A colocação das regras comerciais no servidor de aplicação exige apenas uma cópia das regras comerciais para ser criada e mantida. Portanto, todos os que usam esse servidor de aplicação usarão a mesma cópia dessas regras comerciais. Em aplicações cliente/servidor, o RDBMS poderia resolver alguns dos problemas, mas nem todos os sistemas de RDBMS oferecem o mesmo conjunto de recursos. Além disso, a escrita de procedimentos armazenados torna sua aplicação menos transportável. Usando uma técnica de multicamadas, suas regras comerciais são abrigadas de forma independente do seu RDBMS, facilitando assim a independência dos dados, enquanto ainda oferece algum grau de imposição de regras para os seus dados.

Arquitetura de cliente magro

Além das regras comerciais mencionadas, a aplicação cliente/servidor típica também leva o fardo da maior parte da camada de acesso aos dados. Isso produz um executável maior, mais conhecido como *cliente gordo*. Para uma aplicação de banco de dados do Delphi acessando um banco de dados de servidor SQL, você precisaria instalar o BDE, SQL Links e/ou ODBC para acessar o banco de dados, junto com as bibliotecas do cliente necessárias para falar com o servidor SQL. Depois de instalar esses arquivos, você precisará então configurar cada parte de forma apropriada. Isso aumenta bastante o processo de instalação. Usando o DataSnap, o acesso aos dados é controlado pelo servidor de aplicação, enquanto os dados são apresentados ao usuário pela aplicação cliente. Isso significa que você só precisa distribuir a aplicação cliente e uma DLL para ajudar seu cliente a falar com seu servidor. Isso certamente é uma arquitetura de cliente magro.

Reconciliação de erros automática

O Delphi vem com um mecanismo interno para ajudar na reconciliação de erros. A reconciliação de erros é necessária em uma aplicação multicamadas pelos mesmos motivos que seria necessária com atualizações em cache. Os dados são copiados para a máquina cliente, onde as mudanças são feitas. Vários clientes podem estar trabalhando no mesmo registro. A reconciliação de erros ajuda o usuá-

rio a determinar o que fazer com os registros que foram alterados desde que o usuário baixou o registro pela última vez. No verdadeiro espírito do Delphi, se essa caixa de diálogo não atender às suas necessidades, você poderá removê-la e criar uma que atenda.

Modelo de briefcase

O modelo de briefcase é baseado na metáfora de uma maleta física (ou briefcase). Você coloca seus papéis importantes na sua maleta e os transporta de um lugar para outro, desempacotando-os quando for necessário. O Delphi oferece um meio de empacotar todos os seus dados e levá-los consigo para a rua sem exigir uma conexão real com o servidor de aplicação ou servidor de banco de dados.

Tolerância a falhas

Se a máquina servidora estiver indisponível devido a circunstâncias imprevistas, seria bom mudar dinamicamente para um servidor de backup sem recompilar suas aplicações cliente ou servidor. O Delphi já oferece funcionalidade embutida para isso.

Balanceamento de carga

Ao distribuir sua aplicação cliente para mais pessoas, você inevitavelmente começará a saturar a largura de banda do seu servidor. Existem duas maneiras de tentar balancear o tráfego da rede: balanceamento de carga estático e dinâmico. Para o balanceamento de carga estático, você incluiria outra máquina servidora e teria metade dos clientes usando o servidor A e a outra metade acessando o servidor B. No entanto, e se os clientes que usam o servidor A impuserem uma carga muito mais pesada do que aqueles usam o servidor B? Usando o balanceamento de carga dinâmico, você poderia enfrentar essa situação dizendo a cada aplicação cliente qual servidor ela deve acessar. Existem muitos algoritmos diferentes para balanceamento de carga dinâmico, como aleatório, seqüencial, "round robin" e menor tráfego de rede. Do Delphi 4 em diante, isso é feito por meio de um componente para implementar o balanceamento de carga seqüencial.

Arquitetura típica do DataSnap

A Figura 21.3 mostra uma aplicação DataSnap típica depois de criada. No núcleo desse diagrama está um Data Module construído para essa tarefa. Existem diversas variedades. Para simplificar, usaremos uma baseada em COM neste capítulo, chamada Remote Data Module (RDM). RDM é descendente do módulo clássico disponível desde o Delphi 2. Esse método de dados é um container especial que só permite que componentes não-visuais sejam colocados nele. O RDM não é diferente com relação a isso. Além do mais, o RDM é, na realidade, um objeto COM – ou, para ser mais preciso, um *objeto Automation*. Os serviços que você exporta a partir desse RDM estarão disponíveis para uso nas máquinas cliente.

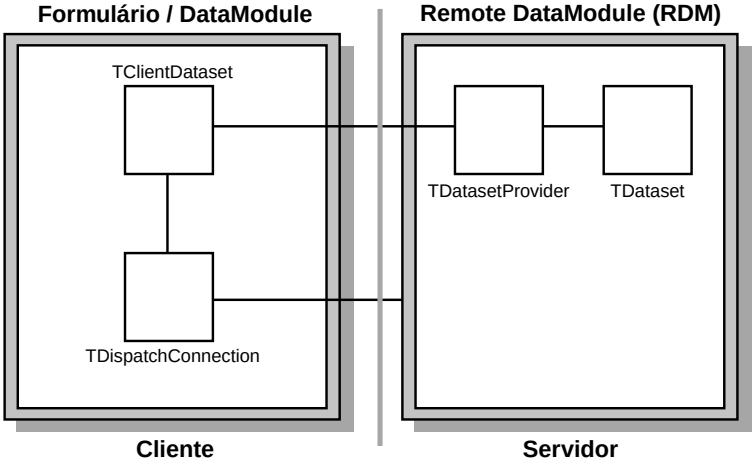

Figura 21.3 Uma aplicação DataSnap típica.

Vejamos algumas das opções que estão disponíveis quando um RDM é criado. A Figura 21.4 mostra a caixa de diálogo que o Delphi apresenta quando você seleciona File, New, Remote Data Module.

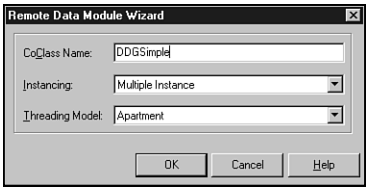

Figura 21.4 A caixa de diálogo New Remote Data Module.

Servidor

Agora que você viu como uma aplicação DataSnap típica é montada, vamos mostrar como fazer com que isso aconteça no Delphi. Vamos começar vendo algumas das opções disponíveis quando se configura o servidor.

Opções de instanciação

A especificação de uma opção de instanciação afeta a quantidade de cópias do processo servidor que serão iniciadas. A Figura 21.5 mostra como as opções feitas aqui controlam como o seu servidor se comporta.

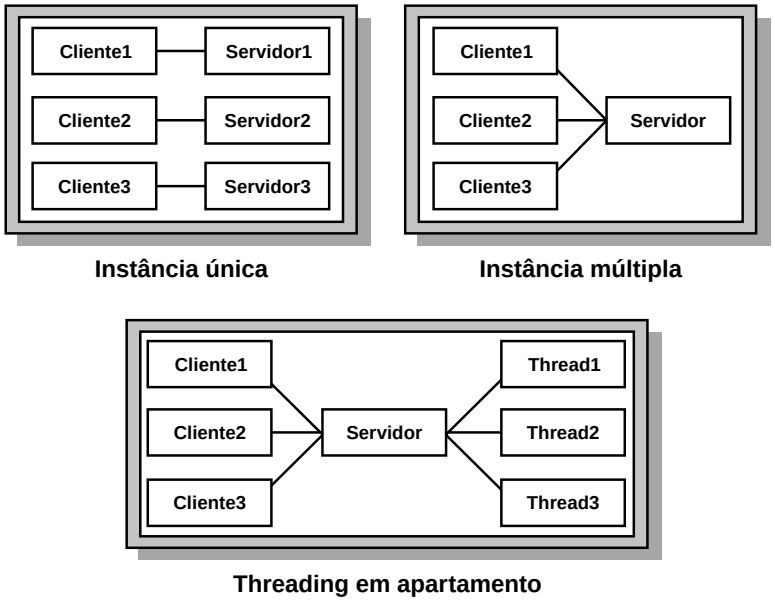

Figura 21.5 Comportamento do servidor com base nas opções de instanciação.

Aqui estão as diferentes opções de instanciação disponíveis a um servidor COM:

- `ciMultiInstance` – Cada cliente que acessa o servidor COM usará a mesma instância do servidor. Por default, isso significa que um cliente precisa esperar por outro antes de ter permissão para operar sobre o servidor COM. Veja na próxima seção, "Opções de threading", informações mais detalhadas sobre como o valor especificado para o modelo de threading também afeta esse comportamento. Isso é equivalente ao acesso serial para os clientes. Todos os clientes precisam compartilhar uma conexão de banco de dados; portanto, a propriedade `TDatabase.HandleShared` precisa ser `True`.

- `ciSingleInstance` – Cada cliente que acessa o servidor COM usará uma instância separada. Isso significa que cada cliente consumirá recursos do servidor para cada instância do servidor a ser carregada. Isso é equivalente ao acesso paralelo para os clientes. Se você decidir seguir com essa opção, saiba que há limites para o BDE que poderiam tornar essa escolha menos atraente. Especificamente, o BDE 5.01 possui um limite de 48 processos por máquina. Como cada cliente gera um novo processo servidor, você só pode ter 48 clientes conectados de cada vez.

- `ciInternal` – O servidor COM não pode ser criado a partir de aplicações externas. Isso é útil quando você deseja controlar o acesso a um objeto COM por meio de uma camada de proxy. Um exemplo de uso dessa opção de instanciação pode ser encontrado no exemplo `<DELPHI>\DEMOS\MIDAS\POOLER`.

Observe também que a configuração do objeto DCOM possui efeito direto sobre o modo de instanciação do objeto. Veja mais informações sobre esse tópico na seção "Distribuindo aplicações DataSnap".

Opções de threading

O suporte para threading no Delphi 5 viu uma mudança drástica para melhor. No Delphi 4, a seleção do modelo de threading para um servidor EXE era insignificativa. O flag simplesmente marcava o Registro para dizer ao COM que uma DLL era capaz de ser executada sob o modelo de threading selecionado. Com o Delphi 5 e 6, a opção de modelo de threading agora se aplica a servidores EXE, permitindo que o COM coloque as conexões em thread sem usar qualquer código externo. A seguir, veja um resumo das opções de threading disponíveis para um RDM:

- Single – A seleção de Single (único) significa que o servidor só é capaz de cuidar de um pedido de cada vez. Ao usar Single, você não precisa se preocupar com as opções de threading, pois o servidor roda em um thread e o COM trata dos detalhes de sincronismo das mensagens para você. No entanto, essa é a pior seleção que você pode fazer se pretende ter um sistema multiusuário, pois o cliente B teria que esperar até que o cliente A terminasse o processamento antes que o cliente B pudesse sequer começar a trabalhar. Isso obviamente não é uma situação boa, pois o cliente A poderia estar fazendo um relatório de resumo do fim do dia ou alguma outra operação semelhante, que exija muito tempo.

- Apartment – A seleção do modelo de threading Apartment (apartamento) oferece o melhor de todos os mundos quando combinada com a instanciação `ciMultiInstance`. Nesse cenário, todos os clientes compartilham um processo servidor, por causa de `ciMultiInstance`, mas o trabalho feito no servidor a partir de um cliente não impede o trabalho do outro cliente, devido à opção de threading Apartment. Ao usar o threading em apartamento, você garante que os dados da instância do seu RDM estão seguros, mas precisa proteger o acesso a variáveis globais usando alguma técnica de sincronismo de thread, como `PostMessage( )`, seções críticas, mutexes, semáforos ou a classe wrapper `TMultiReadExclusiveWriteSynchronizer` do Delphi. Esse é o modelo de threading preferido para os datasets BDE. Observe que, se você usar esse modelo de threading com os datasets BDE, terá que colocar um componente `TSession` no seu RDM e definir a propriedade `AutoSessionName` como `True`, para ajudar o BDE a estar de acordo com os requisitos internos para threading.

- Free – Esse modelo oferece ainda mais flexibilidade no processamento do servidor, permitindo que várias chamadas sejam feitas simultaneamente a partir do cliente para o servidor. No entanto, junto com esse poder vem a responsabilidade. Você precisa ter cuidado para proteger todos os dados contra conflitos de thread – dados de instância e variáveis globais. Esse é o modelo de threading preferido quando se usa ADO.

- Both – Essa opção é efetivamente a mesma que a opção Free, com uma exceção – callbacks são colocados automaticamente em série.

Opções de acesso aos dados

O Delphi 6 Enterprise vem com muitas opções diferentes de acesso aos dados. O BDE continua sendo aceito, permitindo que você use componentes `TDBDataset`, como `TTable`, `TQuery` e `TStoredProc`. No entanto, o DBExpress oferece uma arquitetura mais flexível para o acesso aos dados. Além do mais, você também tem a opção de oferecer suporte a ADO e ter acesso direto ao InterBase, por meio dos novos componentes `TDataset`.

Serviços de anúncio

O RDM é responsável por comunicar quais serviços estarão disponíveis aos clientes. Se o RDM tiver que tornar um `TQuery` disponível para uso no cliente, você precisa colocar o `TQuery` no RDM junto com um `TDatasetProvider`. O componente `TDatasetProvider` é então ligado ao `TQuery` por meio da propriedade `TDatasetProvider.Dataset`. Mais tarde, quando um cliente aparecer e quiser usar os dados do `TQuery`, ele poderá fazer isso vinculando-se ao `TDatasetProvider` que você acabou de criar. Você pode controlar quais provedores estão visíveis ao cliente definindo a propriedade `TDatasetProvider.Exported` como `True` ou `False`.

Se, por outro lado, você não precisar que um dataset inteiro fique exposto a partir do servidor e só precisar que o cliente faça uma chamada de método para o servidor, também poderá fazer isso. Enquanto o RDM tem o foco, selecione a opção de menu `Edit, Add To Interface` (adicionar à interface) e preencha a caixa de diálogo com o protótipo de método padrão (que é simplesmente uma declaração combinando com um método que você criará na sua implementação). Em seguida, você pode especificar a implementação desse método no código, como sempre fez, lembrando-se das implicações do seu modelo de threading.

Cliente

Depois de montar o servidor, você precisa criar um cliente para usar os serviços fornecidos pelo servidor. Vamos dar uma olhada em algumas das opções disponíveis quando estiver montando seu cliente `DataSnap`.

Opções de conexão

A arquitetura do Delphi para a conexão do cliente ao servidor começa com `TDispatchConnection`. Esse objeto básico é o pai de todos os tipos de conexão listados mais adiante. Quando o tipo de conexão é irrelevante para determinada seção, `TDispatchConnection` será usado para indicar esse fato.

`TDCOMConnection` oferece segurança básica e autenticação, usando a implementação padrão do Windows para esses serviços. Esse tipo de conexão é útil especialmente se você estiver usando essa aplicação em uma instalação de intranet/extranet (ou seja, onde as pessoas usando sua aplicação são conhecidas a partir do ponto de vista do domínio). Você pode usar a vinculação inicial quando estiver usando DCOM, e pode usar callbacks e `ConnectionPoints` com facilidade. (Você também pode usar callbacks quando usar soquetes, mas está limitado a usar a vinculação tardia para fazer isso.) As desvantagens do uso dessa conexão são as seguintes:

- Configuração difícil em muitos casos
- Não é um tipo de conexão amiga de firewall
- Exige instalação de DCOM95 para máquinas Windows 95

`TSocketConnection` é a conexão mais fácil de se configurar. Além disso, ela só usa uma porta para o tráfego do DataSnap, de modo que seus administradores de firewall ficarão mais satisfeitos se tiverem que fazer com que o DCOM trabalhe através do firewall. Você precisa estar rodando o ScktSrvr (encontrado no diretório `<DELPHI>\BIN`) para que isso funcione, de modo que há um arquivo extra para distribuir e executar no servidor. O Delphi 4 exigia que você instalasse o WinSock2 para usar esse tipo de conexão, o que significava outra instalação para clientes Windows 9*x*. No en-

tanto, se você não estiver usando callbacks, pode considerar a definição de `TSocketConnection.SupportCallbacks` como `False`. Isso permite que você fique com o WinSock 1 nas máquinas cliente.

Você também pode usar `TCORBAConnection` se quiser usar CORBA como seu protocolo de transporte. CORBA pode ser considerado como o equivalente de padrão aberto do DCOM, e inclui muitos recursos para audodescoberta, recuperação de falha e balanceamento de carga, realizados automaticamente por sua aplicação. Você desejará examinar a arquitetura CORBA ao migrar suas aplicações DataSnap, para permitir conexões entre plataforma e entre linguagem.

O componente `TWebConnection` também está disponível para você. Esse componente da conexão permite que o tráfego seja transportado por HTTP ou HTTPS. Ao usar esse tipo de conexão, algumas limitações são as seguintes:

- Callbacks de qualquer tipo não são aceitas.
- O cliente precisa ter a `WININET.DLL` instalada.
- A máquina servidora precisa estar executando o MS Internet Information Server (IIS) 4.0 ou o Netscape 3.6 em diante.

No entanto, essas limitações parecem compensar quando você precisa oferecer uma aplicação pela Internet ou por um firewall que não esteja sob o seu controle.

O Delphi 6 introduziu um novo tipo de conexão: a `TSOAPConnection`. Essa conexão se comporta de modo semelhante a `WebConnection`, mas conecta-se a um Web Service DataSnap. Ao contrário de quando se usa outros componentes de conexão DataSnap, você não pode usar a propriedade `AppServer` de `TSoapConnection` para chamar métodos da interface do servidor de aplicação que não sejam métodos `IAppServer`. Em vez disso, para se comunicar com um módulo de dados SOAP na interface da aplicação, use um objeto `THTTPRIO` separado.

Observe que todos esses transportes assumem uma instalação válida do TCP/IP. A única exceção a isso é se você estiver usando duas máquinas Windows NT para se comunicarem via DCOM. Nesse caso, você pode especificar qual protocolo o DCOM usará, executando o DCOMCNFG e movendo o protocolo desejado para o topo da lista na guia Default Protocols (protocolos default). O DCOM para Windows 9*x* também tem suporte para TCP/IP.

Conectando os componentes

A partir do diagrama na Figura 21.3, você pode ver como a aplicação `DataSnap` se comunica entre as camadas. Esta seção indica as principais propriedades e componentes que dão ao cliente a capacidade de se comunicar com o servidor.

Para se comunicar do cliente para o servidor, você precisa usar um dos componentes `TDispatchConnection` listados anteriormente. Cada componente possui propriedades específicas apenas a esse tipo de conexão, mas todas elas permitem que você especifique onde encontrar o servidor de aplicação. `TDispatchConnection` é semelhante ao componente `TDatabase` quando usado em aplicações cliente/servidor, pois define o link com o sistema externo e serve como canal para outros componentes quando se comunica com elementos a partir desse sistema.

Quando você tem uma conexão com o servidor, precisa de uma maneira de usar os serviços que expôs no servidor. Isso pode ser feito incluindo-se um `TClientDataset` no seu cliente e conectando-o ao `TDispatchConnection` por meio da propriedade `RemoteServer`. Quando essa conexão estiver feita, você poderá ver uma lista dos provedores exportados no servidor percorrendo a lista na propriedade `ProviderName`. Você verá uma lista de provedores exportados que existem no servidor. Desse modo, o componente `TClientDataset` é semelhante a um `TTable` nas aplicações cliente/servidor.

Você também tem a capacidade de chamar métodos personalizados que existem no servidor usando a propriedade `TDispatchConnection.AppServer`. Por exemplo, a linha de código a seguir chamará a função `Login` no servidor, passando dois parâmetros de string e retornando um valor booleano:

```
LoginSucceeded := DCOMConnection1.AppServer.Login(UserName, Password);
```

Usando o DataSnap para criar uma aplicação

Agora que já vimos muitas das opções disponíveis para a criação de aplicações DataSnap, vamos usar o DataSnap para realmente criar uma aplicação que coloque essa teoria em prática.

Configurando o servidor

Primeiro, vamos focalizar a mecânica da criação do servidor de aplicação. Depois que você tiver criado o servidor, vamos explorar como o cliente é criado.

Remote Data Module

O Remote Data Module (RDM) é fundamental para a criação de um servidor de aplicação. Para criar um RDM para uma nova aplicação, selecione o ícone Remote Data Module a partir da guia Multitier do Object Repository (disponível pela seleção de File, New). Uma caixa de diálogo será exibida para permitir a personalização inicial de algumas opções pertinentes ao RDM.

O nome para o RDM é importante porque o ProgID para esse servidor de aplicação será montado usando o nome do projeto e o nome do RDM. Por exemplo, se o projeto (DPR) se chama AppServer e o nome do RDM é MyRDM, o ProgID será AppServer.MyRDM. Não se esqueça de selecionar as opções de instanciação e threading apropriadas, com base nas explicações anteriores e no comportamento desejado para esse servidor de aplicação.

Tanto TSocketConnection quanto TWebConnection evitam o processamento de autenticação default do Windows, de modo que é imperativo certificar-se de que somente os objetos que são executados no servidor sejam aqueles que você especifica. Isso é feito marcando-se o Registro com certos valores, para permitir que o DataSnap saiba que você pretende permitir que esses objetos sejam executados. Felizmente, tudo o que é preciso para fazer isso é redefinir o método de classe UpdateRegistry. Veja, na Listagem 21.1, a implementação fornecida pelo Delphi automaticamente quando você cria um novo Remote Data Module.

Listagem 21.1 UpdateRegistry – método de classe a partir de um Remote Data Module

```
class procedure TDDGSimple.UpdateRegistry(Register: Boolean;
const ClassID, ProgID: string);
begin
  if Register then
  begin
    inherited UpdateRegistry(Register, ClassID, ProgID);
    EnableSocketTransport(ClassID);
    EnableWebTransport(ClassID);
  end else
  begin
    DisableSocketTransport(ClassID);
    DisableWebTransport(ClassID);
    inherited UpdateRegistry(Register, ClassID, ProgID);
  end;
end;
```

Esse método é chamado sempre que o servidor é registrado ou perde seu registro. Além das entradas do Registro específicas do COM, que são criadas na chamada UpdateRegistry herdada, você pode chamar os métodos EnableXXXTransport( ) e DisableXXXTransport( ) para marcar esse objeto como protegido.

NOTA

`TSocketConnection` só mostrará objetos registrados e protegidos na propriedade `ServerName`. Se você não quiser impor segurança alguma, desmarque a opção de menu Connections, Registered Objects Only (somente objetos registrados) no SCKTSRVR.

Provedores

O servidor de aplicação será responsável por fornecer dados para o cliente, de modo que você precisa encontrar um meio de servir dados a partir do servidor em um formato que seja utilizável no cliente. Felizmente, o `DataSnap` oferece um componente `TDatasetProvider` para facilitar essa etapa.

Comece incluindo um `TQuery` no RDM. Se você estiver usando um RDBMS, inevitavelmente também precisará da configuração de um componente `TDatabase`. Por enquanto, você vinculará `TQuery` a `TDatabase` e especificará uma consulta simples na propriedade SQL, como `select *from customer`. Por último, inclua um componente `TDatasetProvider` no RDM e vincule-o a `TQuery` por meio da propriedade `Dataset`. A propriedade `Exported` no `DatasetProvider` determina se esse provedor será visível aos clientes. Essa propriedade também oferece a capacidade de controlar com facilidade quais provedores são visíveis em runtime.

NOTA

Embora a discussão nesta seção focalize o uso do `TDBDataset` baseado no BDE, os mesmos princípios se aplicam se você quiser usar qualquer outro descendente de `TDataset` para o acesso aos seus dados. Existem várias possibilidades já prontas, como DBExpress, ADO e InterBase Express, e vários componentes de terceiros estão disponíveis para acessar banco de dados específicos.

Registrando o servidor

Quando o servidor de aplicação estiver montado, ele precisa ser registrado com o COM para que esteja disponível para as aplicações cliente que se conectarão a ele. As entradas do Registro discutidas no Capítulo 15 também são usadas para servidores `DataSnap`. Você só precisa executar a aplicação servidora e a configuração do Registro será acrescentada. No entanto, antes de registrar o servidor, não se esqueça de salvar o projeto em primeiro lugar. Isso garante que o ProgID estará correto desse ponto em diante.

Se você preferir não executar a aplicação, poderá passar o parâmetro `/regserver` na linha de comandos ao executar a aplicação. Isso só realizará o processo de registro e terminará a aplicação imediatamente. Para remover as entradas do Registro associadas a essa aplicação, você pode usar o parâmetro `/unregserver`.

Criando o cliente

Agora que você tem um servidor de aplicação funcionando, vejamos como realizar algumas tarefas básicas com o cliente. Vamos discutir como apanhar os dados, como editar os dados, como atualizar o banco de dados com as mudanças feitas no cliente e como lidar com erros durante o processo de atualização do banco de dados.

Apanhando dados

No decorrer de uma aplicação de banco de dados, é preciso trazer dados do servidor para o cliente, a fim de que sejam editados. Trazendo os dados para um cache local, você pode reduzir o tráfego na rede e diminuir os tempos de transação. Em versões anteriores do Delphi, você usaria atualizações em cache para realizar essa tarefa. No entanto, as mesmas etapas gerais ainda servem para aplicações `DataSnap`.

O cliente fala com o servidor por meio de um componente TDispatchConnection. Oferecendo a TDispatchConnection o nome do componente onde o servidor de aplicação reside, essa tarefa é realizada com facilidade. Se você usar TDCOMConnection, poderá especificar o nome de domínio totalmente qualificado (FQDN; por exemplo, nt.dmiser.com), o endereço IP numérico do componente (por exemplo, 192.168.0.2) ou o nome do NetBIOS do componente (por exemplo, nt). Entretanto, devido a um bug no DCOM, você não pode usar o nome localhost ou ainda alguns endereços IP, de modo confiável em todos os casos. Se você usar TSocketConnection, deve especificar endereços IP numéricos na propriedade Address ou o FQDN na propriedade Host. Examinaremos as opções para TWebConnection um pouco mais adiante.

Quando você especificar onde o servidor de aplicação reside, terá que dar a TDispatchConnection um meio de identificar esse servidor de aplicação. Isso é feito por meio da propriedade ServerName. A atribuição da propriedade ServerName preenche a propriedade ServerGUID para você. A propriedade ServerGUID é a parte mais importante. Na realidade, se você quiser distribuir sua aplicação cliente da maneira mais genérica possível, não se esqueça de excluir a propriedade ServerName e simplesmente usar o ServerGUID.

NOTA

Se você usar TDCOMConnection, a lista ServerName só mostrará a lista de servidores que estão registrados na máquina atual. No entanto, TSocketConnection é inteligente o bastante para exibir a lista de servidores de aplicação registrados na máquina remota.

Nesse ponto, a definição de TDispatchConnection.Connected como True o conectará ao servidor de aplicação.

Agora que você já está com o cliente falando com o servidor, precisa de um meio para usar o provedor que criou no servidor. Faça isso usando o componente TClientDataset. Um TClientDataSet é usado para se vincular a um provedor (e, portanto, ao TQuery que está vinculado ao provedor) no servidor.

Primeiro, você precisa ligar TClientDataSet a TDispatchConnection, atribuindo a propriedade RemoteServer de TClientDataSet. Depois de fazer isso, você pode obter uma lista de provedores disponíveis nesse servidor examinando a lista na propriedade ProviderName.

Nesse ponto, tudo está configurado corretamente para abrir um ClientDataset.

Como TClientDataSet é um descendente virtual de TDataset, você pode se basear em muitas das técnicas que já aprendeu usando os componentes TDBDataset em aplicações cliente/servidor. Por exemplo, a definição de Active como True abre TClientDataSet e exibe os dados. A diferença entre isso e configurar TTable.Active como True é que TClientDataSet está realmente obtendo seus dados do servidor de aplicação.

Editando dados no cliente

Todos os registros passados do servidor para TClientDataSet são armazenados na propriedade Data de TClientDataSet. Essa propriedade é uma representação variante do pacote de dados DataSnap. O TClientDataset sabe como decodificar esse pacote de dados em um formato mais útil. O motivo de a propriedade ser definida como uma variante é por causa dos tipos limitados disponíveis ao subsistema COM quando se usa a condução de biblioteca de tipos.

Ao manipular os registros em TClientDataset, uma cópia dos registros inseridos, modificados ou excluídos é colocada na propriedade Delta. Isso permite que o DataSnap seja extremamente eficaz quando se trata da aplicação de atualizações de volta ao servidor de aplicação e, por fim, o banco de dados. Somente os registros alterados precisam ser enviados de volta ao servidor de aplicação.

O formato da propriedade Delta também é muito eficaz. Ele armazena um registro para cada inserção ou exclusão, e armazena dois registros para cada atualização. Os registros atualizados também são armazenados de forma eficiente. O registro não-modificado é fornecido no primeiro registro, enquanto o registro modificado correspondente é armazenado em seguida. No entanto, somente os campos alterados são armazenados no registro modificado para serem salvos no meio de armazenamento.

Um aspecto interessante da propriedade `Delta` é que ela é compatível com a propriedade `Data`. Em outras palavras, ela pode ser atribuída diretamente à propriedade `Data` do componente `ClientDataset`. Isso permitirá investigar o conteúdo atual da propriedade `Delta` em determinado momento.

Vários métodos estão disponíveis para lidar com a edição dos dados no `TClientDataset`. Vamos nos referir a esses métodos como métodos de controle de alteração. Os métodos de *controle de alteração* permitem que você modifique as alterações feitas no `TClientDataset` de várias maneiras.

NOTA

`TClientDataset` provou ser útil de mais maneiras do que a intenção original. Ele também serve como um excelente método para armazenar tabelas na memória, o que não tem nada a ver com `DataSnap` especificamente. Além disso, devido ao modo como ele expõe dados por meio das propriedades `Data` e `Delta`, ele provou ser útil em diversas implementações de padrão de OOP. A discussão a respeito dessas técnicas está fora do escopo deste capítulo. No entanto, você encontrará documentos sobre esses tópicos em `http://www.xapware.com` ou `http://www.xapware.com/ddg`.

Desfazendo mudanças

A maioria dos usuários tem usado uma aplicação de processamento de textos que permita a operação Undo (desfazer). Essa operação apanha sua ação anterior e a retorna ao estado imediatamente antes de ser iniciada. Usando `TClientDataset`, você pode chamar `cdsCustomer.UndoLastChange( )` para simular esse comportamento. A pilha de undo é ilimitada, permitindo que o usuário continue a recuar até o início da sessão de edição, se assim desejar. O parâmetro que você passa para esse método especifica se o cursor está posicionado no registro sendo afetado.

Se o usuário quisesse se livrar de todas as suas atualizações ao mesmo tempo, haveria um modo mais fácil do que chamar `UndoLastChange( )` repetidamente. Basta chamar `cdsCustomer.CancelUpdates( )` para cancelar todas as alterações que foram feitas em uma única sessão de edição.

Revertendo à versão original

Outra possibilidade é permitir que o usuário restaure um registro específico de volta ao estado em que se encontrava quando ele foi apanhado inicialmente. Faça isso chamando `cdsCustomer.RevertRecord( )` enquanto o `TClientDataset` está posicionado no registro que você pretende restaurar.

Transações no cliente: `SavePoint`

A propriedade `ClientDataset.SavePoint` oferece a capacidade de usar transações no cliente. Essa propriedade é ideal para o desenvolvimento de cenários de análise hipotética (what-if) para o usuário. O ato de apanhar o valor da propriedade `SavePoint` armazena um instantâneo dos dados nesse ponto do tempo. O usuário pode continuar a editar enquanto for necessário. Se, em algum ponto, o usuário decidir que o conjunto de dados naquele momento for realmente o que queria, essa variável salva poderá ser atribuída de volta a `SavePoint` e o `TClientDataset` é retornado de volta ao mesmo estado em que se encontrava no momento em que o instantâneo inicial foi tirado. É importante observar que você também pode ter vários níveis aninhados de `SavePoint`, para um cenário complexo.

ATENÇÃO

Uma palavra de aviso sobre `SavePoint` está por ordem: você pode invalidar um `SavePoint` chamando `UndoLastChange( )` após o ponto que atualmente está salvo. Por exemplo, suponha que o usuário edite dois registros e emita um `SavePoint`. Nesse ponto, o usuário edita outro registro. No entanto, ele usa `UndoLastChange( )` para reverter as mudanças duas vezes em seguida. Como o estado do `TClientDataset` agora é o estado anterior ao `SavePoint`, o `SavePoint` está em um estado indefinido.

Reconciliando dados

Depois que você tiver acabado de fazer as mudanças na cópia local dos dados em `TClientDataset`, precisará sinalizar sua intenção de aplicar essas mudanças de volta ao banco de dados. Isso é feito chamando `cdsCustomer.ApplyUpdates( )`. Nesse ponto, `DataSnap` tirará o `Delta` de `cdsCustomer` e o passará ao servidor de aplicação, onde o `DataSnap` aplicará essas mudanças ao servidor de banco de dados usando o mecanismo de reconciliação que você escolheu para esse dataset. Todas as atualizações são realizadas dentro do contexto de uma transação. Logo, explicaremos como os erros são tratados durante esse processo.

O parâmetro que você passa a `ApplyUpdates( )` especifica o número de erros que o processo de atualização permitirá antes de considerar a atualização como ruim e, subseqüentemente, rolará todas as mudanças que foram feitas. A palavra *erros* aqui se refere a erros básicos de violação, erros de integridade referencial ou qualquer outra lógica comercial ou erros de banco de dados. Se você especificar zero para esse parâmetro, estará dizendo ao `DataSnap` que não tolerará quaisquer erros. Portanto, se ocorrer um erro, todas as mudanças que você fez não serão submetidas ao banco de dados. Essa é a configuração que você mais usará, pois combina melhor com as orientações e princípios sólidos de banco de dados.

No entanto, se você quiser, poderá especificar que um certo número de erros poderá ocorrer, enquanto ainda submete todos os registros que tiverem sucesso. A principal extensão desse conceito é passar `-1` como parâmetro para `ApplyUpdates( )`. Isso diz ao `DataSnap` que ele deve submeter cada registro que puder, independente do número de erros encontrados in-process. Em outras palavras, a transação sempre será submetida quando esse parâmetro for usado.

Se você quiser ter o máximo de controle sobre o processo de atualização – incluindo a mudança da SQL que executará para uma inserção, atualização ou exclusão –, poderá fazer isso no evento `TDatasetProvider.BeforeUpdateRecord( )`. Por exemplo, quando um usuário quer excluir um registro, você pode não querer realizar realmente uma operação de exclusão no banco de dados. Em vez disso, um flag é definido para dizer às aplicações que esse registro não está disponível. Mais tarde, um administrador pode rever essas exclusões e submeter a operação física da exclusão. O exemplo a seguir mostra como fazer isso:

```
procedure TDataModule1.Provider1BeforeUpdateRecord(Sender: TObject;
  SourceDS: TDataset; DeltaDS: TClientDataset; UpdateKind: TUpdateKind;
  var Applied: Boolean);
begin
  if UpdateKind=ukDelete then
  begin
    Query1.SQL.Text:='update CUSTOMER set STATUS="DEL" where ID=:ID';
    Query1.Params[0].Value:=DeltaDS.FieldByName('ID').OldValue;
    Query1.ExecSQL;
    Applied:= true;
  end;
end;
```

Você pode criar tantas consultas quantas quiser, controlando o fluxo e o conteúdo do processo de atualização, com base em diferentes fatores, como `UpdateKind` e valores no `Dataset`. Ao inspecionar ou modificar registros do `DeltaDS`, não se esqueça de usar as propriedades `OldValue` e `NewValue` do `TField` apropriado. O uso das propriedades `TField.As`*`XXX`* gerará resultados imprevisíveis.

Além disso, você pode impor regras comerciais aqui ou evitar totalmente a postagem de um registro para o banco de dados. Qualquer exceção que você levanta aqui passará pelo mecanismo de tratamento de erro do `DataSnap`, que explicaremos em seguida.

Depois que a transação terminar, você terá oportunidade para lidar com erros. O erro pára em eventos no servidor e no cliente, dando-lhe uma chance para tomar uma medida corretiva, registrar o erro ou fazer qualquer outra coisa que queira com ele.

A primeira parada para o erro é o evento `DatasetProvider.OnUpdateError`. Esse é um ótimo lugar para se lidar com erros que você esteja esperando ou que possa resolver sem maiores intervenções do cliente.

O destino final para o erro é de volta ao cliente, onde você pode lidar com o erro permitindo que o usuário ajude a determinar o que fazer com o registro. Você faz isso atribuindo um tratador ao evento `TClientDataset.OnReconcileError`.

Isso é útil especialmente porque o DataSnap é baseado em uma estratégica otimista de bloqueio de registro. Essa estratégia permite que vários usuários trabalhem no mesmo registro ao mesmo tempo. Em geral, isso causa conflitos quando o DataSnap tenta reconciliar os dados de volta ao banco de dados, pois o registro foi modificado desde o momento em que foi apanhado.

Usando a caixa de diálogo de reconciliação de erros da Borland

Felizmente, a Borland oferece uma caixa de diálogo de reconciliação de erros, que você pode usar para exibir o erro ao usuário. A Figura 21.6 mostra essa caixa de diálogo. O código-fonte também é fornecido para essa unidade, de modo que você possa modificá-lo, caso não atenda às suas necessidades. Para usar essa caixa de diálogo, selecione File, New no menu principal do Delphi e depois selecione Reconcile Error Dialog (caixa de diálogo reconciliar erro) na página Dialogs. Lembre-se de remover essa unidade da lista Autocreate Forms (criação automática de formulários); caso contrário, você receberá erros de compilação.

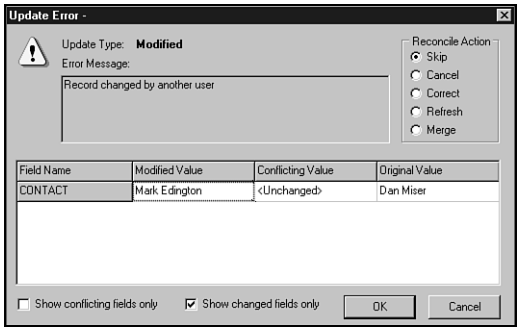

Figura 21.6 Caixa de diálogo Reconcile Error em ação.

A funcionalidade principal dessa unidade está contida na função `HandleReconcileError( )`. Existe um grande relacionamento entre o evento `OnReconcileError` e a função `HandleReconcileError`. Na realidade, a ação típica no evento `OnReconcileError` é chamar a função `HandleReconcileError`. Fazendo isso, a aplicação permite que o usuário final na máquina cliente interaja com o processo de reconciliação de erro na máquina servidora e especifique como esses erros devem ser tratados. O código é o seguinte:

```
procedure TMyForm.CustomerCDSReconcileError(DataSet: TCustomClientDataSet;
  E: EReconcileError; UpdateKind: TUpdateKind;
  var Action: TReconcileAction);
begin
  Action:=HandleReconcileError(Dataset, UpdateKind, E);
end;
```

O valor do parâmetro `Action` determina o que o `DataSnap` fará com esse registro. Vamos focalizar, um pouco mais adiante, alguns outros fatores que afetam as ações que são válidas nesse ponto. A lista a seguir mostra as ações válidas:

- `raSkip` – Não atualiza esse registro de banco de dados específico. Deixe o registro inalterado na cache do cliente.

- raMerge – Mescla os campos desse registro no registro do banco de dados. Esse registro não se aplicará aos registros que foram inseridos.
- raCorrect – Atualiza o registro do banco de dados com os valores que você especifica. Ao selecionar essa ação na caixa de diálogo Reconcile Error, você pode editar os valores na grade. Você não pode usar esse método se outro usuário alterou o registro do banco de dados.
- raCancel – Não atualiza o registro do banco de dados. Remova o registro do cache do cliente.
- raRefresh – Atualiza o registro no cache do cliente com o registro atual no banco de dados.
- raAbort – Aborta a operação de atualização inteira.

Nem todas essas opções fazem sentido (e, portanto, não serão exibidas) em todos os casos. Um requisito para que as ações raMerge e raRefresh estejam disponíveis é que o DataSnap possa identificar o registro por meio da chave primária do banco de dados. Para tanto, defina a propriedade TField.ProviderFlags.pfInKey como True no componente TDataset do RDM para todos os campos na sua chave primária.

Mais opções para fortalecer sua aplicação

Quando você tiver dominado esses fundamentos, a pergunta inevitável é: "O que vem em seguida?" Esta seção dará mais algum conhecimento sobre o DataSnap e como usar esses recursos para fazer com que suas aplicações atuem como você deseja.

Técnicas de otimização do cliente

O modelo de recuperação de dados é bem elegante. No entanto, como o TClientDataset armazena todos os seus registros na memória, você precisa ter muito cuidado com os resultsets que retorna ao TClientDataSet. A técnica mais limpa é garantir que o servidor de aplicação seja bem projetado e só retornar os registros em que o usuário está interessado. Como o mundo real raramente segue a solução utópica, você pode usar a técnica a seguir para ajudar a acelerar o número de registros que apanha em determinado momento para o cliente.

Limitando o pacote de dados

Ao abrir um TClientDataSet, o servidor apanha o número de registros especificado na propriedade TClientDataSet.PacketRecords em determinado momento. No entanto, o DataSnap apanhará registros suficiente para preencher os controles visuais disponíveis com dados. Por exemplo, se você tiver um controle TDBGrid em um formulário, que possa exigir 10 registros de uma só vez, e especificar um valor 5 para PacketRecords, a busca inicial dos dados terá 10 registros. Depois disso, o pacote de dados terá somente 5 registros por busca. Se você especificar -1 nessa propriedade (o default), todos os registros serão transferidos. Se você especificar um valor maior do que zero para PacketRecords, isso introduz estado em sua aplicação. Isso acontece devido ao requisito de que o servidor de aplicação precisa registrar a posição do cursor de cada cliente, de modo que possa retornar o pacote de registros apropriado para o cliente que solicita o pacote. No entanto, você pode registrar o estado no cliente, passando a posição do último registro ao servidor, conforme a necessidade. Como um exemplo simples, examine este código, que faz exatamente isso:

```
Server RDM:
procedure TStateless.DataSetProvider1BeforeGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
begin
  with Sender as TDataSetProvider do
  begin
```

```
    DataSet.Open;
    if VarIsEmpty(OwnerData) then
      DataSet.First
    else
    begin
      while not DataSet.Eof do
      begin
        if DataSet.FieldByName('au_id').Value = OwnerData then
          break;
      end;
    end;
  end;
end;

procedure TStateless.DataSetProvider1AfterGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
begin
  with Sender as TDataSetProvider do
  begin
    OwnerData := Dataset.FieldValues['au_id'];
    DataSet.Close;
  end;
end;

Client:
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
begin
  // KeyValue é uma variável OleVariant privada
  if not (Sender as TClientDataSet).Active then
    KeyValue := Unassigned;
  OwnerData := KeyValue;
end;

procedure TForm1.ClientDataSet1AfterGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
begin
  KeyValue := OwnerData;
end;
```

Um último ponto quando se usa a busca parcial é que a execução de `TClientDataSet.Last( )` apanha o restante dos registros que ficou no resultset. Isso pode ser feito inocentemente pressionando-se Ctrl+End no controle `TDBGrid`. Para contornar esse problema, você precisa definir `TClientDataSet.FetchOnDemand` como `False`. Essa propriedade controla a opção que indica se um pacote de dados será apanhado automaticamente quando o usuário tiver lido todos os registros existentes no cliente. Para simular esse comportamento no código, você pode usar o método `GetNextPacket( )`, que retornará o próximo pacote de dados para você.

> **NOTA**
>
> Observe que o exemplo de código anterior percorre o dataset até que encontre o registro apropriado. Isso é feito de modo que os datasets unidirecionais, como o dbExpress, possam usar o mesmo código sem modificação. Naturalmente, existem muitas maneiras de se encontrar o registro apropriado, como a modificação da instrução SQL ou o uso de parâmetros em uma consulta, mas este exemplo se concentra na mecânica de passagem da chave entre cliente e servidor.

Usando o modelo de briefcase

Outra otimização para reduzir o tráfego da rede é usar o suporte para o modelo de briefcase oferecido com o DataSnap. Faça isso atribuindo um nome de arquivo à propriedade TClientDataset.Filename. Se o arquivo especificado nessa propriedade existir, o TClientDataSet abrirá a cópia local do arquivo, em vez de ler os dados diretamente do servidor de aplicação. Além de permitir que os usuários trabalhem com arquivos enquanto estão desconectados da rede, isso é tremendamente útil para itens que raramente mudam, como tabelas de pesquisa.

> DICA
>
> Se você especificar um TClientDataset.Filename que tenha uma extensão .XML, o pacote de dados será armazenado no formato XML, permitindo-lhe usar qualquer uma das inúmeras ferramentas XML disponíveis para atuar sobre o arquivo de briefcase.

Enviando SQL dinâmica ao servidor

Algumas aplicações exigem a modificação das propriedades básicas do TDataset, como a propriedade SQL de TQuery, a partir do cliente. Desde que os princípios sólidos de multicamadas sejam seguidos, esta pode realmente ser uma solução bastante eficaz e elegante. O Delphi torna trivial a realização dessa tarefa.

Duas etapas são necessárias para se permitir consultas ocasionais. Primeiro, você precisa atribuir a instrução de consulta à propriedade TClientDataset.CommandText. Segundo, você também precisa incluir a opção poAllowCommandText na propriedade DatasetProvider.Options. Quando você abre o TClientDataSet ou chama TClientDataSet.Execute( ), o CommandText é passado para o servidor. Essa mesma técnica também funciona se você quiser alterar o nome da tabela ou do procedimento armazenado no servidor.

Técnicas do servidor de aplicação

O DataSnap agora possui muitos eventos diferentes para você personalizar o comportamento da sua aplicação. Existem eventos Before*XXX* e After*XXX* para praticamente qualquer método na interface IAppServer. Esses dois eventos em particular serão úteis quando você migrar seu servidor de aplicação para que fique completamente sem estado.

Resolvendo a disputa por registro

A discussão anterior, sobre o mecanismo de resolução, incluiu uma rápida menção de que dois usuários trabalhando no mesmo registro causariam um erro quando o segundo usuário tentasse aplicar o registro de volta ao banco de dados. Felizmente, você tem controle completo sobre a detecção dessa colisão.

A propriedade TDatasetProvider.UpdateMode é usada para gerar a instrução SQL que será usada para verificar se o registro mudou desde que foi apanhado por último. Considere o cenário em que dois usuários editam o mesmo registro. Veja como DatasetProvider.UpdateMode afeta o que acontece com o registro para cada usuário:

- upWhereAll – Essa configuração é a mais restritiva, mas oferece a maior garantia de que o registro é o mesmo que o usuário apanhou inicialmente. Se dois usuários editam o mesmo registro, o primeiro usuário poderá atualizar o registro, enquanto o segundo receberá a infame mensagem de erro Another user changed the record (outro usuário alterou o registro). Se você quiser escolher ainda mais quais os campos são usados para realizar essa verificação, poderá remover o elemento pfInWhere da propriedade TField.ProviderFlags correspondente.

- upWhereChanged – Essa configuração permite que dois usuários realmente editem o mesmo registro ao mesmo tempo; desde que os dois usuários editem campos diferentes no mesmo registro, não haverá detecção de colisão. Por exemplo, se o usuário A modifica o campo Endereço e atualiza o registro, o usuário B ainda pode modificar o campo DataNascimento e atualizar o registro com sucesso.
- upWhereKeyOnly – Essa configuração é a mais liberal de todas. Desde que o registro exista no banco de dados, a mudança de cada usuário será aceita. Isso sempre modificará o registro existente no banco de dados, de modo que pode ser visto como um meio de oferecer a funcionalidade do tipo "o último ganha".

Opções diversas para o servidor

Muito mais opções estão disponíveis na propriedade TDatasetProvider.Options para controlar como o pacote de dados DataSnap se comporta. Por exemplo, a inclusão de poReadOnly tornará o dataset somente para leitura no cliente. A especificação de poDisableInserts, poDisableDeletes ou poDisableEdits impede que o cliente realize essa operação e dispare o evento OnEditError ou OnDeleteError correspondente a ser disparado no cliente.

Ao usar datasets aninhados, você propaga atualizações ou exclusões a partir do registro mestre para os registros de detalhe se incluir poCascadeUpdates ou poCascadeDeletes na propriedade DatasetProvider.Options. O uso dessa propriedade requer que o seu banco de dados de back-end aceite a propagação de integridade referencial.

Uma limitação nas versões anteriores do DataSnap era a impossibilidade de mesclar com facilidade as alterações feitas no servidor ao seu TClientDataset no cliente. Para conseguir isso, o usuário tinha que lançar mão do uso de RefreshRecord (ou, em alguns casos, possivelmente Refresh, para preencher novamente o dataset inteiro).

Definindo DatasetProvider.Options para incluir poPropogateChanges, todas as mudanças feitas aos seus dados no servidor de aplicação (por exemplo, no evento DatasetProvider.BeforeUpdateRecord para impor uma regra comercial) agora são trazidas automaticamente de volta para o TClientDataSet. Além do mais, a definição de TDatasetProvider.Options para incluir poAutoRefresh mesclará automaticamente AutoIncrement e os valores default de volta ao TClientDataSet.

> ATENÇÃO
>
> A opção poAutoRefresh não funciona no Delphi 5 e 6. poAutoRefresh só funcionará com uma versão futura do Delphi que inclua um reparo para esse bug. A alternativa, nesse meio tempo, é chamar Refresh( ) para os seus TClientDatasets ou tomar o controle do processo inteiro da aplicação de atualizações.

A discussão inteira sobre o processo de reconciliação até aqui tem girado em torno da reconciliação default, baseada em SQL. Isso significa que todos os eventos no TDataset básico não serão usados durante o processo de reconciliação. A propriedade TDatasetProvider.ResolveToDataset foi criada para usar esses eventos durante a reconciliação. Por exemplo, se TDatasetProvider.ResolveToDataset for verdadeiro, a maioria dos eventos no TDataset será disparada. Saiba que os eventos usados são chamados apenas quando da aplicação de atualizações de volta ao servidor. Em outras palavras, se você tiver um evento TQuery.BeforeInsert definido no servidor, ele só será disparado no servidor quando você chamar TClientDataSet.ApplyUpdates. Os eventos não se integram aos eventos correspondentes do TClientDataSet.

Lidando com relacionamentos mestre/detalhe

Nenhuma discussão das aplicações de banco de dados seria completa sem, pelo menos, uma menção dos relacionamentos mestre/detalhe. Com o DataSnap, você tem duas opções para lidar com mestre/detalhe.

Datasets aninhados

A opção para relacionamentos mestre/detalhe são datasets aninhados. Os datasets aninhados permitem que uma tabela mestre realmente contenha datasets de detalhe. Além da atualização de registros mestre e detalhe em uma transação, eles permitem que o armazenamento de todos os registros mestre e detalhe seja feito em um único arquivo de briefcase, e você pode usar as melhorias em `DBGrid` para fazer exibir datasets de detalhe em suas próprias janelas. Um aviso se você decidir usar datasets aninhados: todos os registros de detalhe serão apanhados e trazidos para o cliente ao selecionar um registro mestre. Isso se tornará um engarrafamento possível ao desempenho, se você aninhar vários níveis de datasets de detalhe. Por exemplo, se você apanhar apenas um registro mestre que tenha 10 registros de detalhe, e cada registro de detalhe tiver três registros de detalhes ligados ao detalhe de primeiro nível, você apanharia 41 registros inicialmente. Ao usar o vínculo no cliente, você só apanharia 14 registros inicialmente e obteria os outros registros netos ao rolar pelo `TClientDataSet` do detalhe.

Para configurar um relacionamento de dataset aninhado, você precisa definir o relacionamento mestre/detalhe no servidor de aplicação. Isso é feito usando a mesma técnica que você usou nas aplicações cliente/servidor, ou seja, definindo a instrução SQL para a `TQuery` do detalhe, incluindo o parâmetro de vínculo. Veja um exemplo:

```
"select * orders where custno=:custno"
```

Depois você atribui o `TQuery.Datasource` para o `TQuery` do detalhe apontar para um componente `TDatasource` que está ligado ao `TDataset` mestre. Quando esse relacionamento for estabelecido, você só precisa exportar o `TDatasetProvider` que está ligado ao dataset mestre. O `DataSnap` é inteligente o bastante para entender que o dataset mestre possui datasets de detalhe vinculados a ele e, portanto, enviará os datasets de detalhe para o cliente como um `TDatasetField`.

No cliente, você atribui a propriedade `TClientDataset.ProviderName` do mestre para o provedor do mestre. Depois, você inclui campos persistentes ao `TClientDataset`. Observe o último campo no `Fields Editor`. Ele contém um campo com o mesmo nome do dataset de detalhe no servidor, sendo declarado como um tipo `TDatasetField`. Nesse ponto, você tem informações suficientes para usar o dataset aninhado no código. No entanto, para tornar as coisas realmente fáceis, você pode incluir um `TClientDataset` de detalhe e atribuir sua propriedade `DatasetField` ao `TDatasetField` apropriado a partir do mestre. É importante observar aqui que você não configurou quaisquer outras propriedades no `TClientDataset` de detalhe, como `RemoteServer`, `ProviderName`, `MasterSource`, `MasterFields` ou `PacketRecords`. A única propriedade que você definiu foi a propriedade `DatasetField`. Nesse ponto, você também pode vincular controles cientes de dados ao `TClientDataset` de detalhe.

Depois que você acabar de trabalhar com os dados no dataset aninhado, terá que aplicar as atualizações de volta ao banco de dados. Isso é feito chamando o método `ApplyUpdates( )` do `TClientDataset` mestre. O `DataSnap` aplicará todas as mudanças no `TClientDataset` mestre, incluindo os datasets de detalhe, de volta ao servidor dentro do contexto da transação.

Você verá um exemplo no CD-ROM que acompanha este livro, no diretório para este capítulo, sob `\NestCDS`.

Vínculo no cliente

Lembre-se de que fizemos alguns avisos anteriormente com relação ao uso de datasets aninhados. A alternativa ao uso de datasets aninhados é a criação de um relacionamento mestre/detalhe no cliente. Para criar um vínculo mestre/detalhe usando esse método, você simplesmente cria um `TDataset` e `TDatasetProvider` para o mestre e o detalhe no servidor.

No cliente, você vincula dois componentes `TClientDataset` aos datasets que exportou no servidor. Depois, crie o relacionamento mestre/detalhe atribuindo a propriedade `TClientDataset.MasterSource` do detalhe ao componente `TDatasource` que aponta para o `TClientDataset` mestre.

A definição de `MasterSource` em um `TClientDataset` define a propriedade `PacketRecords` como zero. Quando `PacketRecords` é igual a zero, isso significa que o `DataSnap` deve apenas retornar as informações de metadados para esse `TClientDataset`. No entanto, quando `PacketRecords` é igual a

zero no contexto de um relacionamento mestre/detalhe, o significado muda. O DataSnap agora apanhará os registros para o dataset de detalhe para cada registro mestre. Resumindo, deixe a propriedade PacketRecords definida com o valor default.

Para reconciliar os dados mestre/detalhe no banco de dados em uma transação, você precisa escrever sua própria lógica de ApplyUpdates. Isso não é tão simples quanto a maioria das tarefas em Delphi, mas oferece controle flexível sobre o processo de atualização.

A aplicação de atualizações a uma única tabela normalmente é disparada por uma chamada a TClientDataset.ApplyUpdates. Esse método envia os registros alterados do ClientDataset para seu provedor na camada intermediária, onde o provedor gravará então as alterações no banco de dados. Tudo isso é feito dentro do escopo de uma transação e é feito sem qualquer intervenção do programador. Para fazer a mesma coisa para as tabelas mestre/detalhe, você precisa entender o que o Delphi está fazendo por você quando é feita a chamada a TClientDataset.ApplyUpdates.

Quaisquer alterações que você faça a um TClientDataset são armazenadas na propriedade Delta. A propriedade Delta contém todas as informações que por fim serão gravadas no banco de dados. O código a seguir ilustra o processo de atualização para a aplicação de propriedades Delta de volta ao banco de dados. As Listagens 21.2 e 21.3 mostram as seções relevantes do cliente e do servidor para a aplicação de atualizações em uma configuração mestre/detalhe.

Listagem 21.2 Atualizações do cliente no relacionamento mestre/detalhe

```
procedure TClientDM.ApplyUpdates;
var
  MasterVar, DetailVar: OleVariant;
begin
  Master.CheckBrowseMode;
  Detail_Proj.CheckBrowseMode;
  if Master.ChangeCount > 0 then
    MasterVar := Master.Delta else
    MasterVar := NULL;
  if Detail.ChangeCount > 0 then
    DetailVar := Detail.Delta else
    DetailVar := NULL;
  RemoteServer.AppServer.ApplyUpdates(DetailVar, MasterVar);
  { Reconcia os pacotes de dados com erro. Como permitimos 0 erros, somente um
    pacote de erro pode conter erros. Se nenhum pacote tiver erros, então
    atualizamos os dados. }
  if not VarIsNull(DetailVar) then
    Detail.Reconcile(DetailVar) else
  if not VarIsNull(MasterVar) then
    Master.Reconcile(MasterVar) else
  begin
    Detail.Reconcile(DetailVar);
    Master.Reconcile(MasterVar);
    Detail.Refresh;
    Master.Refresh;
  end;
end;
```

Listagem 21.3 Atualizações do servidor no relacionamento mestre/detalhe

```
procedure TServerRDM.ApplyUpdates(var DetailVar, MasterVar: OleVariant);
var
  ErrCount: Integer;
begin
Database.StartTransaction;
```

Listagem 21.3 Continuação

```
  try
    if not VarIsNull(MasterVar) then
    begin
      MasterVar := cdsMaster.Provider.ApplyUpdates(MasterVar, 0, ErrCount);
      if ErrCount >0 then
        SysUtils.Abort;    // Isso causará o Rollback
    end;
    if not VarIsNull(DetailVar) then
    begin
      DetailVar := cdsDetail.Provider.ApplyUpdates(DetailVar, 0, ErrCount);
      if ErrCount > 0 then
        SysUtils.Abort;    // Isso causará o Rollback
    end;
    Database.Commit;
  except
    Database.Rollback
  end;
end;
```

Embora esse método funcione muito bem, ele realmente não oferece oportunidade para reutilização de código. Essa seria uma boa hora para estender o Delphi e oferecer uma reutilização fácil. Aqui estão as principais etapas exigidas para resumir o processo de atualização:

1. Coloque os deltas para cada CDS em um array de variantes.
2. Coloque os provedores para cada CDS em um array de variantes.
3. Aplique todos os deltas em uma transação.
4. Reconcilie os pacotes de dados de erro retornados na etapa anterior e atualize os dados.

O resultado dessa abstração é fornecido na unidade utilitária mostrada na Listagem 21.4.

Listagem 21.4 Uma unidade oferecendo rotinas utilitárias e abstração

```
unit CDSUtil;

interface

uses
  DbClient, DbTables;

function RetrieveDeltas(const cdsArray : array of TClientDataset): Variant;
function RetrieveProviders(const cdsArray : array of TClientDataset): Variant;
procedure ReconcileDeltas(const cdsArray : array of TClientDataset;
                          vDeltaArray: OleVariant);

procedure CDSApplyUpdates(ADatabase : TDatabase; var vDeltaArray: OleVariant;
                          const vProviderArray: OleVariant);

implementation

uses
  SysUtils, Provider, Midas, Variants;

type
  PArrayData = ^TArrayData;
  TArrayData = array[0..1000] of Olevariant;
```

Listagem 21.4 Continuação

```
{ Delta é o CDS.Delta na entrada. No retorno, Delta terá um pacote
{ de dados contendo todos os registros que não poderiam ser aplicados
{ ao banco de dados. Lembre-se de que o Delphi precisa do nome do
{ provedor, e esse é passado no elemento 1 da variante AProvider. }
procedure ApplyDelta(AProvider: OleVariant; var Delta : OleVariant);
var
  ErrCount : integer;
  OwnerData: OleVariant;
begin
  if not VarIsNull(Delta) then
  begin
    // ScktSrvr não aceita vinculação inicial
    Delta := (IDispatch(AProvider[0]) as IAppServer).AS_ApplyUpdates(
                AProvider[1], Delta, 0, ErrCount, OwnerData);
    if ErrCount > 0 then
      SysUtils.Abort;  // Isso causará o Rollback no procedimento que chama
  end;
end;

{ Chamada do servidor }
procedure CDSApplyUpdates(ADatabase : TDatabase; var vDeltaArray: OleVariant;
  const vProviderArray: OleVariant);
var
  i : integer;
  LowArr, HighArr: integer;
  P: PArrayData;
begin
  { Envolve as atualizações em uma transação. Se qualquer etapa resultar
  { em um erro, levanta uma exceção, que causará o Rollback da transação. }
  ADatabase.Connected:=true;
  ADatabase.StartTransaction;
  try
    LowArr:=VarArrayLowBound(vDeltaArray,1);
    HighArr:=VarArrayHighBound(vDeltaArray,1);
    P:=VarArrayLock(vDeltaArray);
    try
      for i:=LowArr to HighArr do
        ApplyDelta(vProviderArray[i], P^[i]);
    finally
      VarArrayUnlock(vDeltaArray);
    end;
    ADatabase.Commit;
  except
    ADatabase.Rollback;
  end;
end;

{ Chamadas no cliente }
function RetrieveDeltas(const cdsArray : array of TClientDataset): Variant;
var
  i : integer;
  LowCDS, HighCDS : integer;
begin
  Result:=NULL;
  LowCDS:=Low(cdsArray);
  HighCDS:=High(cdsArray);
  for i:=LowCDS to HighCDS do
```

Listagem 21.4 Continuação

```
   cdsArray[i].CheckBrowseMode;

  Result:=VarArrayCreate([LowCDS, HighCDS], varVariant);
  { Configura a variante com as mudanças (ou NULL, se não houver) }
  for i:=LowCDS to HighCDS do
  begin
    if cdsArray[i].ChangeCount>0 then
      Result[i]:=cdsArray[i].Delta else
      Result[i]:=NULL;
  end;
end;

{ Se estivermos usando o Delphi 5 ou mais recente, temos que retornar
  o nome do provedor E o AppServer por meio dessa função. Usaremos
  ProviderName para chamar AS_ApplyUpdates na funcao CDSApplyUpdates
  mais adiante. }
function RetrieveProviders(const cdsArray : array of TClientDataset): Variant;
var
  i: integer;
  LowCDS, HighCDS: integer;
begin
  Result:=NULL;
  LowCDS:=Low(cdsArray);
  HighCDS:=High(cdsArray);

  Result:=VarArrayCreate([LowCDS, HighCDS], varVariant);
  for i:=LowCDS to HighCDS do
    Result[i]:=VarArrayOf([cdsArray[i].AppServer, cdsArray[i].ProviderName]);
end;

procedure ReconcileDeltas(const cdsArray : array of TClientDataset;
  vDeltaArray: OleVariant);
var
  bReconcile : boolean;
  i: integer;
  LowCDS, HighCDS : integer;
begin
  LowCDS:=Low(cdsArray);
  HighCDS:=High(cdsArray);

  { Se a etapa anterior resultou em erros, reconcilia os pacotes
    de dados de erro. }
  bReconcile:=false;
  for i:=LowCDS to HighCDS do
    if not VarIsNull(vDeltaArray[i]) then begin
      cdsArray[i].Reconcile(vDeltaArray[i]);
      bReconcile:=true;
      break;
    end;

  { Atualiza os datasets, se for preciso }
  if not bReconcile then
    for i:=HighCDS downto LowCDS do begin
      cdsArray[i].Reconcile(vDeltaArray[i]);
      cdsArray[i].Refresh;
    end;
end;

end.
```

A Listagem 21.5 mostra a modificação do exemplo anterior usando a unidade CDSUtil.

Listagem 21.5 Modificação do exemplo anterior usando **CDSUtil.pas**

```
procedure TForm1.btnApplyClick(Sender: TObject);
var
  vDelta: OleVariant;
  vProvider: OleVariant;
  arrCDS: array[0..1] of TClientDataset;
begin
  arrCDS[0]:= cdsMaster;  // Configura array ClientDataset
  arrCDS[1]:= cdsDetail;

  vDelta:= RetrieveDeltas(arrCDS);                 // Etapa 1
  vProvider:= RetrieveProviders(arrCDS);           // Etapa 2
  DCOMConnection1.ApplyUpdates(vDelta, vProvider); // Etapa 3
  ReconcileDeltas(arrCDS, vDelta);                 // Etapa 4
end;

procedure TServerRDM.ApplyUpdates(var vDelta, vProvider: OleVariant);
begin
  CDSApplyUpdates(Database1, vDelta, vProvider);   // Etapa 3
end;
```

Você pode usar essa unidade em aplicações de duas camadas ou três camadas. Para passar de um modelo de duas camadas para três, você exportaria uma função no servidor que chame CDSApplyUpdates em vez de chamar CDSApplyUpdates no cliente. Tudo o mais no cliente permanece inalterado.

Você encontrará um exemplo neste CD-ROM no diretório para este capítulo, sob \MDCDS.

Exemplos do mundo real

Agora que já vimos os fundamentos, vejamos como o DataSnap pode ajudá-lo explorando vários exemplos do mundo real.

Associações

A escrita de uma aplicação de banco de dados relacional depende bastante de percorrer os relacionamentos entre as tabelas. Normalmente, você achará conveniente representar seus dados altamente normalizados em uma visão mais achatada do que a estrutura de dados básica. No entanto, a atualização dos dados a partir dessas associações exige mais cuidado da sua parte.

Atualização de uma tabela

A aplicação de atualizações em uma consulta associada é um caso especial na programação de banco de dados, e o DataSnap não é exceção. O problema está na própria consulta com associação. Embora algumas consultas com associação produzirão dados que poderiam ser atualizados automaticamente, outras nunca serão feitas conforme as regras que permitem a recuperação, edição e atualização automática dos dados básicos. Para isso, o Delphi atualmente o força a resolver atualizações para associar consultas por si só.

Para associações que exigem apenas uma tabela para serem atualizadas, o Delphi pode lidar com a maior parte dos detalhes de atualização para você. Aqui estão as etapas necessárias para se gravar uma tabela de volta ao banco de dados:

1. Inclua campos persistentes na TQuery associada.
2. Defina TQuery.ProviderFlags=[ ] para cada campo da tabela que você estará atualizando.
3. Escreva o código a seguir no evento DatasetProvider.OnGetTableName, para informar ao DataSnap qual tabela você deseja atualizar. Lembre-se de que esse evento facilita a especificação do nome da tabela, embora você pudesse fazer a mesma coisa nas versões anteriores do Delphi, usando o evento DatasetProvider.OnGetDatasetProperties:

```
procedure TJoin1Server.prvJoinGetTableName(Sender: TObject;
  DataSet: TDataSet; var TableName: String);
begin
  TableName := 'Emp';
end;
```

Fazendo isso, você está dizendo a ClientDataset para registrar o nome da tabela para você. Agora, quando você chamar ClientDataset1.ApplyUpdates( ), o DataSnap saberá experimentar e traduzir o nome de tabela que você especificou, ao contrário de permitir que o DataSnap tente descobrir qual seria o nome da tabela.

Uma técnica alternativa seria usar um componente TUpdateSQL que só atualiza a tabela em que você está interessado. Isso permite que TQuery.UpdateObject seja usado durante o processo de reconciliação e combina mais de perto com o processo usado nas aplicações cliente/servidor tradicionais.

NOTA

Nem todos os TDatasets possuem uma propriedade UpdateObject. No entanto, você ainda pode usar a mesma técnica, devido à modificação feita em TUpdateSQL. Basta definir sua SQL para cada ação (exclusão, inserção, modificação) e usar um código semelhante a este:

```
procedure TForm1.DataSetProvider1BeforeUpdateRecord(Sender: TObject;
  SourceDS: TDataSet; DeltaDS: TCustomClientDataSet;
  UpdateKind: TUpdateKind; var Applied: Boolean);
begin
  UpdateSQL1.DataSet := DeltaDS;
  UpdateSQL1.SetParams(UpdateKind);
  ADOCommand1.CommandText := UpdateSQL1.SQL[UpdateKind].Text;
  ADOCommand1.Parameters.Assign(UpdateSQL1.Query[UpdateKind].Params);
  ADOCommand1.Execute;
  Applied := true;
end;
```

Você encontrará um exemplo neste CD-ROM, no diretório para este capítulo, sob \Join1.

Atualização de múltiplas tabelas

Para cenários mais complexos, como permitir a edição e a atualização de várias tabelas, você precisa escrever algum código por conta própria. Existem duas técnicas para resolver esse problema:

- O método mais antigo, que é usar DatasetProvider.BeforeUpdateRecord( ) para desmembrar o pacote de dados e aplicar as atualização às tabelas básicas
- O método mais recente, que é aplicar atualizações usando a propriedade UpdateObject

Ao usar atualizações em cache com uma associação de múltiplas tabelas, você precisa configurar um componente TUpdateSQL para cada tabela que será atualizada. Como a propriedade UpdateObject só pode ser atribuída a um componente TUpdateSQL, você precisa vincular programaticamente todas as propriedades TUpdateSQL.Dataset ao dataset associado em TQuery.OnUpdateRecord e chamar TUpdateSQL.Apply para associar os parâmetros e executar a instrução SQL básica. Nesse caso, o dataset em que você está interessado é o dataset Delta. Esse dataset é passado como parâmetro para o evento TQuery.OnUpdateRecord.

Tudo o que você precisa fazer é atribuir as propriedades SessionName e DatabaseName para permitir que a atualização ocorra no mesmo contexto das outras transações e ligar a propriedade Dataset ao Delta que é passado ao evento. O código resultante para o evento TQuery.OnUpdateRecord aparece na Listagem 21.6.

Listagem 21.6 Associação usando TUpdateSQL

```
procedure TJoin2Server.JoinQueryUpdateRecord(DataSet: TDataSet;
  UpdateKind: TUpdateKind; var UpdateAction: TUpdateAction);
begin
  usqlEmp.SessionName := JoinQuery.SessionName;
  usqlEmp.DatabaseName := JoinQuery.DatabaseName;
  usqlEmp.Dataset := Dataset;
  usqlEmp.Apply(UpdateKind);

  usqlFTEmp.SessionName := JoinQuery.SessionName;
  usqlFTEmp.DatabaseName := JoinQuery.DatabaseName;
  usqlFTEmp.Dataset := Dataset;
  usqlFTEmp.Apply(UpdateKind);

  UpdateAction := uaApplied;
end;
```

Como você cumpriu as regras de atualização de dados dentro da arquitetura DataSnap, o processo de atualização inteiro é disparado de forma transparente, como sempre acontece no DataSnap, com uma chamada a ClientDataset1.ApplyUpdates(0).

Você encontrará um exemplo neste CD-ROM, no diretório para este capítulo, sob \Join2.

DataSnap na Web

Até mesmo com a introdução do Kylix, o Delphi está ligado à plataforma Windows (ou Linux); portanto, quaisquer clientes que você escreva precisam ser executados nesse tipo de máquina. Isso nem sempre é desejável. Por exemplo, você poderia oferecer acesso fácil aos dados que existem no seu banco de dados a qualquer um que tenha uma conexão com a Internet. Como você já escreveu um servidor de aplicação que atua como agente para seus dados – além de abrir regras comerciais para esses dados –, seria desejável reutilizar o servidor de aplicação em vez de reescrever, em outro ambiente, toda a camada de acesso aos dados e regra comercial.

HTML pura

Esta seção explica como aproveitar seu servidor de aplicação e, ao mesmo tempo, oferecer uma nova camada de apresentação que usará HTML pura. Esta seção considera que você está acostumado com o material abordado no Capítulo 31 de *Delphi 5 Guia do Desenvolvedor*, que está no CD-ROM deste livro. Usando esse método, você está introduzindo outra camada na sua arquitetura. O WebBroker atua como cliente para o servidor de aplicação e empacota esses dados na HTML que será exibida no browser. Você também perde alguns dos benefícios do trabalho com o IDE do Delphi, como o uso de controles cientes dos dados. No entanto, essa é uma opção bastante viável para permitir o acesso aos seus dados em um formato HTML simples.

Depois de criar uma aplicação WebBroker e um `WebModule`, você simplesmente inclui `TDispatchConnection` e `TClientDataset` no `WebModule`. Quando as propriedades estiverem preenchidas, você poderá usar diversos métodos diferentes para traduzir esses dados em HTML que, por fim, será vista pelo cliente.

Uma técnica válida seria incluir um `TDatasetTableProducer` vinculado ao `TClientDataset` de seu interesse. A partir daí, o usuário poderá dar um clique em um vínculo e passar para uma página de edição, onde poderá editar os dados e aplicar as atualizações. Veja as Listagens 21.7 e 21.8 para uma implementação de exemplo dessa técnica.

Listagem 21.7 HTML para editar e aplicar atualizações

```
<form action="<#SCRIPTNAME>/updaterecord" method="post">
<b>EmpNo: <#EMPNO></b>
<input type="hidden" name="EmpNo" value=<#EMPNO>>
<table cellspacing="2" cellpadding="2" border="0">
<tr>
  <td>Last Name:</td>
  <td><input type="text" name="LastName" value=<#LASTNAME>></td>
</tr>
<tr>
  <td>First Name:</td>
  <td><input type="text" name="FirstName" value=<#FIRSTNAME>></td>
</tr>
<tr>
  <td>Hire Date:</td>
  <td><input type="text" name="HireDate" size="8" value=<#HIREDATE>></td>
</tr>
<tr>
  <td>Salary:</td>
  <td><input type="text" name="Salary" size="8" value=<#SALARY>></td>
</tr>
<tr>
  <td>Vacation:</td>
  <td><input type="text" name="Vacation" size="4" value=<#VACATION>></td>
</tr>
</table>
<input type="submit" name="Submit" value="Apply Updates">
<input type="Reset">
</form>
```

Listagem 21.8 Código para editar e aplicar atualizações

```
unit WebMain;

interface

uses
  Windows, Messages, SysUtils, Classes, HTTPApp, DBWeb, Db, DBClient,
  MConnect, DSProd, HTTPProd;

type
  TWebModule1 = class(TWebModule)
    dcJoin: TDCOMConnection;
    cdsJoin: TClientDataSet;
    dstpJoin: TDataSetTableProducer;
    dsppJoin: TDataSetPageProducer;
```

Listagem 21.8 Continuação

```
    ppSuccess: TPageProducer;
    ppError: TPageProducer;
    procedure WebModuleBeforeDispatch(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
    procedure WebModule1waListAction(Sender: TObject; Request: TWebRequest;
      Response: TWebResponse; var Handled: Boolean);
    procedure dstpJoinFormatCell(Sender: TObject; CellRow,
      CellColumn: Integer; var BgColor: THTMLBgColor;
      var Align: THTMLAlign; var VAlign: THTMLVAlign; var CustomAttrs,
      CellData: String);
    procedure WebModule1waEditAction(Sender: TObject; Request: TWebRequest;
      Response: TWebResponse; var Handled: Boolean);
    procedure dsppJoinHTMLTag(Sender: TObject; Tag: TTag;
      const TagString: String; TagParams: TStrings;
      var ReplaceText: String);
    procedure WebModule1waUpdateAction(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  private
    { Declarações privadas }
    DataFields : TStrings;
  public
    { Declarações públicas }
  end;

var
  WebModule1: TWebModule1;

implementation

{$R *.DFM}

procedure TWebModule1.WebModuleBeforeDispatch(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  with Request do
    case MethodType of
      mtPost: DataFields:=ContentFields;
      mtGet: DataFields:=QueryFields;
    end;
end;

function LocalServerPath(sFile : string = '') : string;
var
  FN: array[0..MAX_PATH- 1] of char;
  sPath : shortstring;
begin
  SetString(sPath, FN, GetModuleFileName(hInstance, FN, SizeOf(FN)));
  Result := ExtractFilePath( sPath ) + ExtractFileName( sFile );
end;

procedure TWebModule1.WebModule1waListAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  cdsJoin.Open;
  Response.Content := dstpJoin.Content;
end;
```

Listagem 21.8 Continuação

```
procedure TWebModule1.dstpJoinFormatCell(Sender: TObject; CellRow,
  CellColumn: Integer; var BgColor: THTMLBgColor; var Align: THTMLAlign;
  var VAlign: THTMLVAlign; var CustomAttrs, CellData: String);
begin
  if (CellRow > 0) and (CellColumn = 0) then
    CellData := Format('<a href="%s/getrecord?empno=%s">%s</a>',
      [Request.ScriptName, CellData, CellData]);
end;

procedure TWebModule1.WebModule1waEditAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  dsppJoin.HTMLFile := LocalServerPath('join.htm');
  cdsJoin.Filter := 'EmpNo = ' + DataFields.Values['empno'];
  cdsJoin.Filtered := true;
  Response.Content := dsppJoin.Content;
end;

procedure TWebModule1.dsppJoinHTMLTag(Sender: TObject; Tag: TTag;
  const TagString: String; TagParams: TStrings; var ReplaceText: String);
begin
  if CompareText(TagString, 'SCRIPTNAME')=0 then
   ReplaceText:=Request.ScriptName;
end;

procedure TWebModule1.WebModule1waUpdateAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var
  EmpNo, LastName, FirstName, HireDate, Salary, Vacation: string;
begin
  EmpNo:=DataFields.Values['EmpNo'];
  LastName:=DataFields.Values['LastName'];
  FirstName:=DataFields.Values['FirstName'];
  HireDate:=DataFields.Values['HireDate'];
  Salary:=DataFields.Values['Salary'];
  Vacation:=DataFields.Values['Vacation'];

  cdsJoin.Open;
  if cdsJoin.Locate('EMPNO', EmpNo, [ ]) then
  begin
    cdsJoin.Edit;
    cdsJoin.FieldByName('LastName').AsString:=LastName;
    cdsJoin.FieldByName('FirstName').AsString:=FirstName;
    cdsJoin.FieldByName('HireDate').AsString:=HireDate;
    cdsJoin.FieldByName('Salary').AsString:=Salary;
    cdsJoin.FieldByName('Vacation').AsString:=Vacation;
    if cdsJoin.ApplyUpdates(0)=0 then
      Response.Content:=ppSuccess.Content else
      Response.Content:=pPError.Content;
  end;
end;

end.
```

Observe que esse método exige que muito código personalizado seja escrito, e o conjunto de recursos completo do DataSnap não é implementado nesse exemplo – especificamente, a reconciliação de erro. Você pode continuar a melhorar esse exemplo para ser mais robusto se usar essa técnica extensivamente.

ATENÇÃO

É imperativo que você considere o conceito de estado ao escrever seu WebModule e servidor de aplicação. Como o HTTP é um protocolo sem estado, você não pode contar com os valores das propriedades como sendo iguais aos que você deixou quando a chamada terminou.

DICA

O WebBroker é um meio de apanhar seus dados para browsers Web. Usando o WebSnap, você pode estender as capacidades da sua aplicação ainda mais, usando os novos recursos que o WebSnap oferece, como scripting e suporte para sessão.

Para executar este exemplo, não se esqueça de compilar e registrar a aplicação de exemplo Join2. Em seguida, compile a aplicação da Web (uma versão CGI ou ISAPI) e coloque o executável em um diretório capaz de trabalhar com script para o seu servidor da Web. O código também espera encontrar o arquivo `join.htm` no diretório de scripts; portanto, copie-o também. Depois, basta apontar seu browser para `http://localhost/scripts/WebJoin.exe` e ver os resultados desse exemplo.

Você encontrará um exemplo neste CD-ROM, no diretório para este capítulo, sob `\WebBrok`.

InternetExpress

Com o InternetExpress, você pode melhorar a funcionalidade de uma técnica `WebModule` pura para permitir uma experiência mais rica no cliente. Isso é possível devido ao uso de padrões abertos, como XML e JavaScript, no InternetExpress. Usando o InternetExpress, você pode criar um front-end apenas com browser para o seu servidor de aplicação DataSnap: nada de controles ActiveX para baixar; nada de instalação no cliente e requisitos de configuração; nada além de um browser Web alcançando um servidor Web.

Para usar o InternetExpress, você deverá ter algum código rodando em um servidor Web. Para este exemplo, usaremos uma aplicação ISAPI, mas você poderia usar CGI ou ASP. A finalidade do agente Web é apanhar pedidos do browser e passá-los para o servidor de aplicação. A colocação de componentes InternetExpress na aplicação agente da Web torna essa tarefa muito fácil.

Este exemplo usará um servidor de aplicação DataSnap padrão, que possui clientes, pedidos e funcionários. Os clientes e os pedidos estão associados em um relacionamento de dataset aninhado (para obter mais informações sobre datasets aninhados, leia a próxima seção), enquanto o dataset de funcionários servirá como tabela de pesquisa. Depois que o servidor de aplicação tiver sido criado e registrado, você poderá focalizar a criação da aplicação agente Web, que se comunicará com o servidor de aplicação.

Crie uma nova aplicação ISAPI selecionando File, New, Web Server Application no Object Repository. Acrescente um componente `TDCOMConnection` no WebModule. Isso atuará como vínculo com o servidor de aplicação; portanto, preencha a propriedade `ServerName` com o ProgID do servidor de aplicação.

Em seguida, você acrescentará no WebModule um componente `TXMLBroker` a partir da página InternetExpress da Component Palette e definirá as propriedades `RemoteServer` e `ProviderName` como o CustomerProvider. O componente `TXMLBroker` atua de modo semelhante ao `TClientDataset`. Ele é responsável por apanhar pacotes de dados do servidor de aplicação e passar esses pacotes de dados para o browser. A principal diferença entre o pacote de dados em um `TXMLBroker` e um `TClientDataset` é que o `TXMLBroker` traduz os pacotes de dados do `DataSnap` para XML. Você também

incluirá um `TClientDataset` ao `WebModule` e o vinculará ao provedor de funcionários no servidor de aplicação. Mais adiante, você usará isso como origem de dados de pesquisa.

O componente `TXMLBroker` é responsável pela comunicação com o servidor de aplicação e também pela navegação das páginas HTML. Existem muitas propriedades disponíveis para se personalizar o modo como sua aplicação InternetExpress se comportará. Por exemplo, você pode limitar o número de registros que serão transmitidos ao cliente ou especificar o número de erros permitidos durante uma atualização.

Você agora precisa de um meio para mover esses dados para o browser. Usando o componente `TInetXPageProducer`, você pode usar a tecnologia WebBroker no Delphi para levar uma página HTML até o browser. No entanto, o `TInetXPageProducer` também permite a criação visual da página Web por meio do Web Page Editor.

Dê um clique duplo no `TInetXPageProducer` para fazer surgir o Web Page Editor. Esse editor visual o ajuda a personalizar os elementos presentes em determinada página Web. Uma das coisas mais interessantes sobre o InternetExpress é que ele é completamente extensível. Você pode criar seus próprios componentes, que podem ser usados no Web Page Editor seguindo algumas regras bem definidas. Para obter exemplos de componentes InternetExpress personalizados, consulte o diretório `<DELPHI>\DEMOS\MIDAS\INTERNETEXPRESS\INETXCUSTOM`.

ATENÇÃO

`TInetXPageProducer` possui um diretório chamado `IncludePathURL`. É essencial definir essa propriedade corretamente ou então sua aplicação InternetExpress não funcionará. Defina esse valor como o diretório virtual que contém os arquivos JavaScript do InternetExpress. Por exemplo, se você incluir os arquivos em `c:\inetpub\wwwroot\jscript`, o valor para essa propriedade será `/jscript/`.

Com o Web Page Editor ativo, dê um clique no botão Insert (inserir) para exibir a caixa de diálogo Add Web Component (adicionar componente Web) (veja a Figura 21.7). Essa caixa de diálogo contém uma lista de componentes Web que podem ser acrescentados na página HTML. Essa lista é baseada no componente pai (a seção no canto superior esquerdo) que está atualmente selecionado. Por exemplo, inclua um componente Web DataForm ao nó raiz para permitir que os usuários finais apresentem e editem informações de banco de dados em um layout tipo formulário.

Figura 21.7 A caixa de diálogo Add Web Component trazida a partir do Web Page Editor.

Se, depois disso, você selecionar o nó DataForm no Web Page Editor, poderá dar um clique no botão Insert (inserir) novamente. Observe que a lista de componentes disponíveis nesse ponto é diferente da lista exibida na etapa anterior. Depois de selecionar o componente FieldGroup, você verá um aviso no painel de visualização, informando que a propriedade `TXMLBroker` para o FieldGroup não está atribuída. Atribuindo o XMLBroker no Object Inspector, você notará imediata-

mente o layout da HTML no painel de visualização do Web Page Editor. Ao continuar a modificar propriedades ou incluir componentes, o estado da página HTML será constantemente atualizado (veja a Figura 21.8).

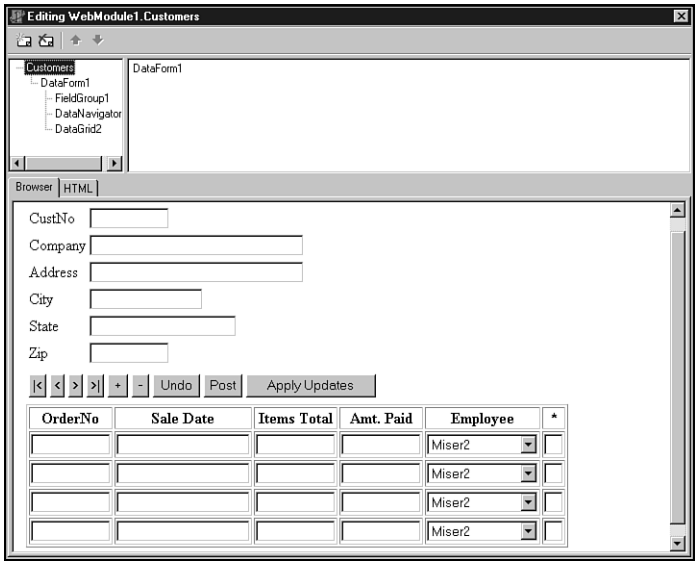

Figura 21.8 O Web Page Editor após a criação de uma página HTML.

O nível de personalização disponível com os componentes Web padrão é praticamente ilimitado. As propriedades facilitam a mudança das legendas de campo, do alinhamento e das cores; a inclusão de código HTML diretamente personalizado; e até mesmo o uso de folhas de estilo. Além do mais, se o componente não se ajustar exatamente às suas necessidades, você sempre poderá criar um componente descendente e usar isso em seu lugar. A estrutura é realmente tão extensível quanto a sua imaginação permita.

Para poder chamar a DLL ISAPI, você terá que colocá-la em um diretório virtual capaz de executar o script. Você também precisa mover os arquivos JavaScript encontrados em `<DELPHI>\SOURCE\WEBMIDAS` para um local válido no seu servidor Web e modificar a propriedade `TInetXPageProducer.IncludePathURL` para que aponte para o URI dos arquivos JavaScript. Depois disso, a página está pronta para ser vista.

Para acessar a página, tudo o que você precisa é de um browser capaz de executar JavaScript. Basta apontar o browser para `http://localhost/inetx/inetxisapi.dll` e os dados aparecerão no browser. A Figura 21.9 mostra uma tela capturada durante o uso da aplicação.

Você pode detectar erros de reconciliação durante o processo `ApplyUpdates`, como já está acostumado a fazer em uma aplicação DataSnap independente. Essa capacidade torna-se possível quando você atribui a propriedade `TXMLBroker.ReconcileProducer` a um `TPageProducer`. Sempre que ocorre um erro, o `Content` do `TPageProducer` atribuído a essa propriedade será retornado ao usuário final.

Um `TPageProducer` especializado, `TReconcilePageProducer`, está disponível por meio da instalação do pacote `InetXCustom.dpk`, encontrado em `<DELPHI>\DEMOS\MIDAS\INTERNETEXPRESS\INETXCUSTOM`. Esse PageProducer gera HTML que atua de modo semelhante à caixa de diálogo DataSnap Reconciliation Error padrão (veja a Figura 21.10).

Você encontrará um exemplo neste CD-ROM, no diretório para este capítulo, sob `\InetX`.

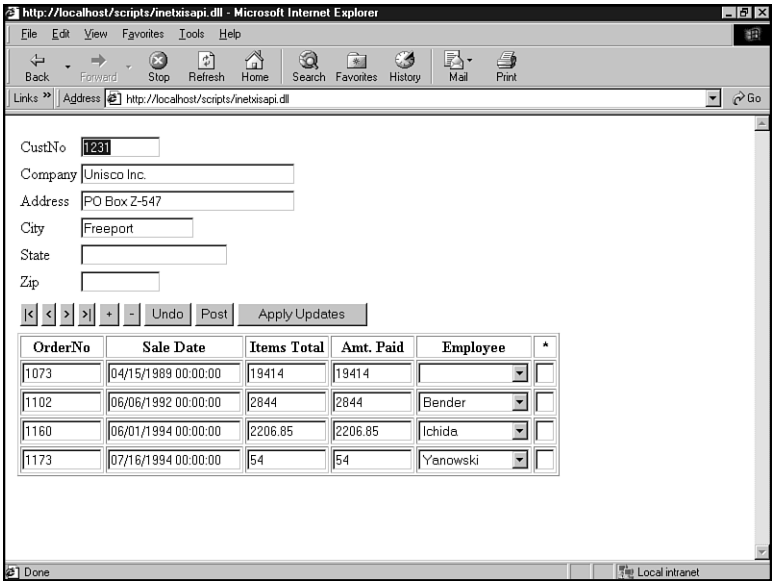

Figura 21.9 Internet Explorer acessando a página Web do InternetExpress.

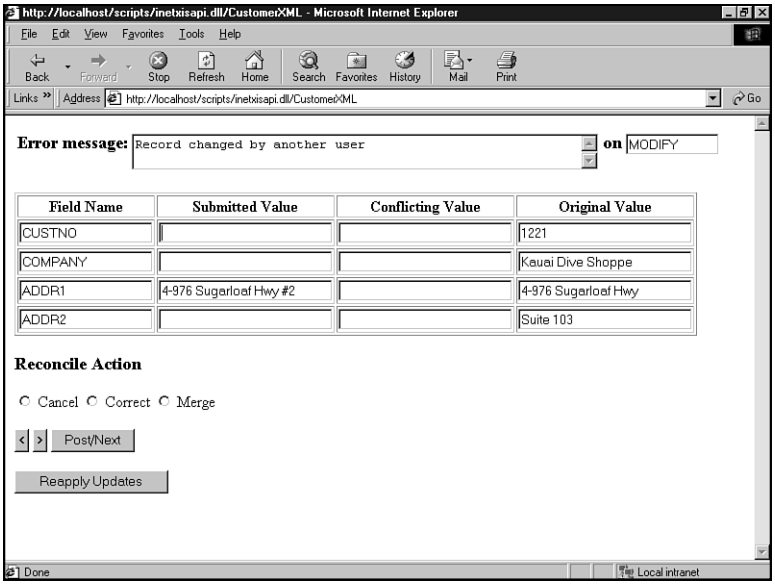

Figura 21.10 Visão da página HTML gerada por **`TReconcilePageProducer`**.

Mais recursos de dataset do ciente

Há muitas opções disponíveis para se controlar o componente `TClientDataset`. Nesta seção, veremos as maneiras de usar o `TClientDataset` para facilitar a codificação em aplicações complexas.

Aplicações de duas camadas

Você viu como atribuir o provedor – e, portanto, os dados – ao `ClientDataset` em uma aplicação de três camadas. No entanto, muitas vezes uma aplicação simples de duas camadas é tudo do que é necessário. Assim, como você usa o DataSnap em uma aplicação de duas camadas? Existem quatro possibilidades:

- Atribuição de dados em runtime
- Atribuição de dados durante o projeto

- Atribuição de um provedor em runtime
- Atribuição de um provedor durante o projeto

As duas opções básicas quando se usa `ClientDataset` são atribuir a propriedade `AppServer` e atribuir os dados. Se você decidir atribuir o `AppServer`, terá um vínculo entre o `TDatasetProvider` e o `ClientDataset`, que permitirá ter comunicação entre o `ClientDataset` e o `TDatasetProvider`, conforme a necessidade. Se, por outro lado, você decidir atribuir os dados, terá efetivamente criado um mecanismo de armazenamento local para os seus dados e o `ClientDataset` não se comunicará com o componente `TDatasetProvider` para obter mais informações ou dados.

Para atribuir os dados diretamente a partir de um `TDataset` para um `TClientDataset` em runtime, use o código da Listagem 21.9.

Listagem 21.9 Código para atribuir dados diretamente de um `TDataSet`

```
function GetData(ADataset: TDataset): OleVariant;
begin
  with TDatasetProvider.Create(nil) do
  try
    Dataset:=ADataset;
    Result:=Data;
  finally
    Free;
  end;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  ClientDataset1.Data:=GetData(ADOTable1);
end;
```

Esse método exige mais código e esforço do que nas versões anteriores do Delphi, onde você simplesmente atribuiria a propriedade `Table1.Provider.Data` à propriedade `ClientDataset1.Data`. No entanto, essa função ajudará a tornar o código adicional menos observável.

Você também pode usar o componente `TClientDataset` para apanhar os dados de um `TDataset` durante o projeto, selecionando o comando `Assign Local Data` (atribuir dados locais) no menu de contexto do componente `TClientDataset`. Depois, você especifica o cmp `TDataset` que contém os dados que você deseja, e os dados são trazidos para o `TClientDataset` e armazenados na propriedade `Data`.

> ATENÇÃO
>
> Se você tivesse que salvar o arquivo nesse estado e comparar o tamanho do arquivo DFM para o tamanho antes de executar esse comando, notaria um aumento no tamanho do DFM. Isso porque o Delphi armazenou todos os metadados e registros associados ao `TDataset` no DFM. O Delphi só colocará esses dados em stream no DFM se o `TClientDataset` estiver `Active`. Você também pode remover esse espaço executando o comando Clear Data (apagar dados) no menu de contexto de `TClientDataset`.

Se você quiser ter a flexibilidade total que uma atribuição de provedor oferece, terá que atribuir a propriedade `AppServer`. Em runtime, você pode atribuir a propriedade `AppServer` no código. Isso pode ser tão simples quanto a seguinte instrução, encontrada em `FormCreate`:

```
ClientDataset1.AppServer:=TLocalAppServer.Create(Table1);
ClientDataset1.Open;
```

Você pode atribuir a propriedade AppServer durante o projeto. Se deixar a propriedade RemoteServer em branco em um TClientDataset, você poderá atribuir um componente TDatasetProvider à propriedade TClientDataset.ProviderName.

Uma desvantagem importante no uso da propriedade TClientDataset.ProviderName é que ela não pode ser atribuída a provedores que residem em outro formulário ou DataModule durante o projeto. Isso porque o Delphi 6 introduziu o componente TLocalConnection. TLocalConnection descobrirá e exporá automaticamente quaisquer TDatasetProviders que encontrar com o mesmo proprietário. Para usar esse método de atribuição de provedores, atribua a propriedade ClientDataset.RemoteServer para ser o componente LocalConnection no formulário ou DataModule externo. Depois disso, você terá a lista de provedores para essa LocalConnection na propriedade ClientDataset.ProviderName.

A diferença principal entre o uso de componentes TDataset e ClientDataset é que, quando você está usando ClientDataset, está usando a interface IAppServer para agenciar seus pedidos de dados ao componente TDataset básico. Isso significa que você estará manipulando as propriedades, os métodos, os eventos e os campos do componente TClientDataset, e não do componente TDataset. Pense no componente TDataset como se estivesse em uma aplicação separada e, portanto, não pudesse ser manipulado diretamente por você no código. Coloque todos os componentes do seu servidor em um DataModule separado. A colocação dos componentes TDatabase, TDataset e TLocalConnection em um DataModule separado efetivamente prepara sua aplicação para uma transição mais fácil, mais adiante, para uma distribuição multicamadas. Outro benefício de se fazer isso é que pode ajudá-lo a pensar em DataModule como algo em que o cliente não pode tocar com facilidade. Novamente, essa é uma boa preparação para a sua aplicação (e para o seu próprio raciocínio) quando for a hora de transportar essa aplicação para uma distribuição em multicamadas.

Erros clássicos

O erro mais comum na criação de uma aplicação multicamadas é a introdução de conhecimento desnecessário da camada de dados na camada de apresentação. Alguma validação é mais adequada na camada de apresentação, mas é o modo como essa validação é realizada que determina sua adequação em uma aplicação multicamadas.

Por exemplo, se você estiver passando instruções da SQL dinâmica do cliente para o servidor, isso gera uma dependência, em que a aplicação cliente sempre deve estar sincronizada com a camada de dados. Fazer as coisas dessa maneira introduz mais partes em movimento que precisam ser coordenadas na aplicação multicamadas geral. Se você mudar uma das estruturas das tabelas na camada de dados, terá que atualizar todas as aplicações cliente que enviam SQL dinâmica, de modo que agora possam enviar a instrução SQL correta. Isso certamente limita o benefício mantido por uma aplicação de cliente magro desenvolvida de forma adequada.

Outro exemplo de um erro clássico é quando a aplicação cliente tenta controlar o tempo de vida da transação, em vez de permitir que a camada comercial cuide disso em favor do cliente. Quase sempre, isso é implementado pela exposição de três métodos da instância de TDataBase no servidor – BeginTransaction( ), Commit( ) e Rollback( ) – e pela chamada desses métodos a partir do cliente. Quando as coisas são feitas dessa forma, o código do cliente se torna muito mais complicado para se manter e infringe o princípio de que a camada de apresentação deve ser a única camada responsável pela comunicação com a camada de dados. A camada de apresentação nunca deverá se basear nessa tática. Em vez disso, você deve enviar suas atualizações para a camada comercial e permitir que essa camada trate da atualização dos dados em uma transação.

Distribuindo aplicações DataSnap

Depois de ter criado uma aplicação DataSnap completa, o último obstáculo que ainda temos que ultrapassar é a distribuição dessa aplicação. Esta seção esboça o que precisa ser feito para facilitar a distribuição da sua aplicação DataSnap.

Questões de licenciamento

O licenciamento tem sido um assunto difícil para muitos pessoas desde que o DataSnap foi introduzido inicialmente no Delphi 3. As inúmeras opções para distribuição dessa tecnologia contribuíram para essa confusão. Esta seção explica os requisitos gerais de quando você precisa adquirir uma licença do DataSnap. No entanto, o único documento legalmente relacionado ao licenciamento está em `DEPLOY.TXT`, localizado no diretório do Delphi 6. Finalmente, para obter a resposta definitiva para uma situação específica, você precisa entrar em contato com o representante de vendas local da Borland. Outras orientações e exemplos estão disponíveis em

```
http://www.borland.com/midas/papers/licensing/
```

ou em nosso site Web, em

```
http://www.xapware.com/ddg
```

As informações desse documento foram preparadas para responder a alguns dos cenários mais comuns em que o DataSnap é utilizado. As informações sobre preços e opções também estão incluídas no documento.

O critério principal para se determinar a necessidade de uma licença do DataSnap para a sua aplicação é se o pacote de dados DataSnap atravessa um limite de máquina. Se isso acontecer e você utilizar os componentes DataSnap em ambas as máquinas, então terá que adquirir uma licença. Se não acontecer (como nos exemplos de uma e duas camadas, apresentados anteriormente), você estará usando a tecnologia DataSnap, mas não será preciso adquirir uma licença para usar o DataSnap dessa maneira.

Configuração do DCOM

A configuração do DCOM se parece mais com uma arte do que uma ciência. Existem muitos aspectos envolvidos em uma configuração DCOM completa e segura, mas esta seção o ajudará a entender alguns dos fundamentos dessa arte oculta.

Depois de registrar seu servidor de aplicação, seu objeto servidor estará disponível para personalização no utilitário DCOMCNFG da Microsoft. Esse utilitário está incluído nos sistemas NT automaticamente, mas é um download separado para máquinas Win9*x*. Como um lembrete, existem muitos bugs no DCOMCNFG; o principal é que o DCOMCNFG só pode ser executado em máquinas Wi9*x* que estejam com o compartilhamento em nível de usuário ativado. Isso, naturalmente, exige um domínio. Isso nem sempre é possível ou desejável em uma rede ponto a ponto, como em duas máquinas Windows 9*x*. Isso tem feito com que muitas pessoas pensem incorretamente que é necessário o uso de uma máquina NT para se trabalhar com DCOM.

Se você puder executar o DCOMCNFG, poderá selecionar o servidor de aplicação registrado e dar um clique no botão Properties para revelar informações sobre o seu servidor. A página Identity (identidade) é um bom lugar para iniciarmos nosso passeio pelo DCOMCNFG. O valor default para um objeto servidor registrado é Launching User (usuário que inicia). A Microsoft não poderia ter tomado uma decisão pior para o default, se tentasse.

Quando o DCOM cria o servidor, ele usa o contexto de segurança do usuário especificado na página Identity. O usuário que inicia gerará um novo processo do objeto servidor para todo e qualquer login de usuário distinto. Muitas pessoas vêem o fato de que selecionaram o modo de instanciação `ciMultiple` e se perguntam por que várias cópias de seu servidor estão sendo criadas. Por exemplo, se o usuário A se conecta ao servidor e depois o usuário B se conecta, o DCOM gerará um processo inteiramente novo para o usuário B. Além disso, você não verá a parte GUI do servidor para usuários que se conectam sob uma conta diferente da que está atualmente em uso na máquina servidora. Isso acontece por causa de um conceito do NT conhecido como *estações Windows*. A única estação Windows capaz de escrever na tela é a Interactive User (usuário interativo), que é o usuário atualmente conectado na máquina servidora. Além do mais, estações Windows são um re-

curso escasso, e você pode não conseguir executar muitos processos do servidor se usar essa configuração. Resumindo, nunca use a opção Launching User como sua identidade para o seu servidor.

A próxima opção interessante nessa página é Interactive User, que significa que cada cliente que cria um servidor fará isso sob o contexto do usuário que está conectado ao servidor naquele momento. Isso também permitirá que você tenha interação visual com o seu servidor de aplicação. Infelizmente, a maior parte dos administradores de sistemas não permite que um login aberto fique ocioso em uma máquina NT. Além disso, se o usuário conectado decidir sair, o servidor de aplicação não funcionará mais conforme desejado.

Para esta discussão, isso nos deixa com a última opção na página Identity: This User (este usuário). Usando essa opção, todos os clientes criarão um servidor de aplicação e usarão as credenciais de login e o contexto do usuário especificado na página Identity. Isso também significa que a máquina NT não exige que um usuário esteja conectado para usar o servidor de aplicação. A única desvantagem dessa técnica é que não haverá uma tela GUI do servidor quando a opção é utilizada. No entanto, de longe essa é a melhor de todas as opções disponíveis para colocar seu servidor de aplicação em produção.

Depois que o objeto servidor for configurado corretamente com a identidade certa, você precisará voltar sua atenção para a guia Security (segurança). Certifique-se de que o usuário que estará usando esse objeto possui privilégios apropriados. Não se esqueça também de conceder o acesso de usuário SYSTEM ao servidor; caso contrário, você encontrará erros durante o processo.

Muitas nuances sutis estão espalhadas pelo processo de configuração do DCOM. Para ver o que há de mais recente sobre problemas de configuração do DCOM, especialmente com relação ao Windows 9*x*, Delphi e DataSnap, visite a página DCOM em nosso site Web, em

`http://www.DistribuCon.com/dcom95.htm`

Arquivos a distribuir

Os requisitos para a distribuição de uma aplicação DataSnap mudaram a cada nova versão do Delphi. O Delphi 6 facilita a distribuição, mais do que qualquer outra versão.

Com o Delphi 6, os arquivos mínimos necessários para distribuição da sua aplicação DataSnap aparecem nas listas a seguir.

Aqui estão as etapas para o servidor (essas etapas consideram um servidor COM; elas diferirão ligeiramente para outras variedades):

1. Copie o servidor de aplicação para um diretório com privilégios suficientes no NTFS ou privilégios em nível de compartilhamento definidos corretamente, se estiver em uma máquina Win9*x*.
2. Instale sua camada de acesso a dados para permitir que o servidor de aplicação atue como cliente para o RDBMS (por exemplo, BDE, MDAC, bibliotecas de banco de dados específicas no cliente e assim por diante).
3. Copie `MIDAS.DLL` para o diretório `%SYSTEM%`. Por default, esse seria `C:\Winnt\System32` para máquinas NT e `C:\Windows\System` para máquinas 9*x*.
4. Execute o servidor de aplicação uma vez para registrá-lo no COM.

Aqui estão as etapas para o cliente:

1. Copie o cliente para um diretório, junto com quaisquer outros arquivos de dependência usados pelo seu cliente (por exemplo, pacotes de runtime, DLLs, controles ActiveX e assim por diante).
2. Copie `MIDAS.DLL` para o diretório `%SYSTEM%`. Observe que o Delphi 6 pode vincular estaticamente o `MIDAS.DLL` à sua aplicação, tornando essa etapa desnecessária. Para fazer isso, basta incluir a unidade MidasLib em sua cláusula `uses` e montar sua aplicação novamente. Você verá um aumento no tamanho do arquivo EXE, devido ao vínculo estático.

3. Opcional: se você especificar a propriedade `ServerName` na sua `TDispatchConnection` ou se empregar a vinculação inicial no seu cliente, terá que registrar o arquivo da biblioteca de tipos do servidor (TLB). Isso pode ser feito usando um utilitário como `<DELPHI>\BIN\TREGSVR.EXE` (ou programaticamente, se você optar por isso).

Considerações de distribuição para Internet (firewalls)

Ao distribuir sua aplicação por uma rede local (LAN), não há nada para atrapalhá-lo. Você pode escolher qualquer tipo de conexão que melhor atenda às necessidades da sua aplicação. No entanto, se você precisar contar com a Internet como seu backbone, muitas coisas poderão sair erradas – principalmente com firewalls.

DCOM não é o protocolo mais amigo do firewall. Ele exige a abertura de várias portas em um firewall. A maioria dos administradores de sistema teme a abertura de um intervalo inteiro de portas (particularmente, as portas DCOM mais reconhecidas) porque isso convida hackers a baterem na porta. Usando `TSocketConnection`, a história melhora um pouco. O firewall só precisa de uma porta aberta. No entanto, o administrador de sistemas ocasional ainda assim se recusará a fazer isso por haver uma brecha na segurança.

`TWebConnection` é semelhante a `TSocketConnection` porque permite que o tráfego do DataSnap seja reunido no tráfego HTTP válido, e depois usa a porta mais aberta no mundo – a porta HTTP (porta default 80). Na realidade, o componente oferece suporte até mesmo para SSL, de modo que você possa ter comunicações seguras. Fazendo isso, todos os problemas com firewall são completamente eliminados. Afinal, se uma empresa não permitir a entrada ou saída de tráfego HTTP, nada poderá ser feito para a comunicação com ela.

Essa mágica é realizada com a extensão ISAPI fornecida pela Borland, que traduz o tráfego HTTP em tráfego DataSnap e vice-versa. Com relação a isso, a DLL ISAPI faz o mesmo trabalho que o ScktSrvr faz para as conexões com soquete. A extensão ISAPI `httpsrvr.dll` precisa ser colocada em um diretório capaz de executar código. Por exemplo, com o IIS, o local default para esse arquivo seria em `C:\Inetpub\Scripts`.

Outro benefício do uso de `TWebConnection` é o suporte para o pooling de objetos. O pooling de objetos é usado para poupar o servidor do overhead da criação de objetos todas as vezes em que um cliente se conecta ao servidor. Além do mais, o mecanismo de pooling no DataSnap permite a criação de um número máximo de objetos. Depois que esse máximo tiver sido alcançado, um erro será enviado ao cliente, dizendo que o servidor está muito ocupado para processar esse pedido. Isso é mais flexível e escalável do que simplesmente criar um número arbitrário de threads para cada cliente individual que queira se conectar ao servidor.

Para levar isso um passo adiante, a criação do seu RDM como um Web Service usando um módulo de dados SOAP não apenas oferece os benefícios de uma `TWebConnection`, mas também permite que sejam construídos clientes usando protocolos SOAP padrão do setor. Essa plataforma habilita seu servidor de aplicação para uso por .Net, Sun ONE e outros sistemas SOAP compatíveis.

Para dizer ao DataSnap que esse RDM será mantido em um pool, você precisa chamar `RegisterPooled` e `UnregisterPooled` no método `UpdateRegistry` do RDM. (Veja, na Listagem 21.1, um exemplo de implementação de `UpdateRegistry`.) A seguir, apresentamos um exemplo de chamada ao método `RegisterPooled`:

```
RegisterPooled(ClassID, 16, 30);
```

Essa chamada diz ao DataSnap que 16 objetos estarão disponíveis no pool, e que o DataSnap pode liberar quaisquer instâncias de objetos que tiverem sido criadas se não houver atividade por 30 minutos. Se você nunca quiser liberar os objetos, então poderá passar `0` como parâmetro de tempo limite.

O cliente não é muito diferente disso. Basta usar uma `TWebConnection` como `TDispatchConnection` para o cliente e preencher as propriedades de forma adequada, e o cliente estará se comunicando com o servidor de aplicação através do HTTP. A única grande diferença quando se usa `TWebConnection` é a necessidade de especificar o URL completo para o arquivo `httpsrvr.dll`, em vez de simplesmente identificar o componente servidor por nome ou por endereço. A Figura 21.11 mostra uma captura de tela de uma configuração típica, usando `TWebConnection`.

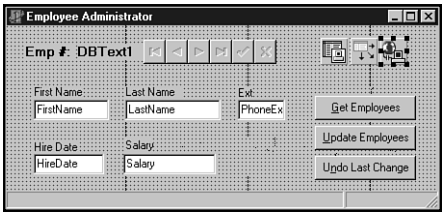

Figura 21.11 Configuração de **`TWebConnection`** durante o projeto.

Outro benefício do uso do HTTP para o seu transporte é que um sistema operacional como o NT Enterprise permite o uso de clusters de servidores. Isso oferece balanceamento de carga e tolerância a falhas automatizados para o seu servidor de aplicação. Para obter mais informações sobre o clustering, consulte `http://www.microsoft.com/ntserver/ntserverenterprise/exec/overview/clustering`.

As limitações do uso de `TWebConnection` são bastante triviais, e compensam qualquer concessão para que haja mais clientes capazes de atingir seu servidor de aplicação. As limitações são que você precisa instalar `wininet.dll` no cliente e nenhuma callback está disponível quando se usa `TWebConnection`.

Resumo

Este capítulo fornece muitas informações sobre DataSnap. Mesmo assim, ele é apenas uma introdução ao que pode ser feito com essa tecnologia – algo muito além do escopo de um único capítulo. Mesmo depois que você explorar todos os detalhes internos do DataSnap, ainda poderá aumentar seu conhecimento e suas capacidades usando DataSnap com C++Builder e JBuilder. Usando JBuilder, você pode atingir o máximo em acesso de múltiplas plataformas a um servidor de aplicação, enquanto usa a mesma tecnologia e conceitos aprendidos aqui.

O DataSnap é uma tecnologia em rápida evolução, que traz a promessa de aplicações multicamadas a cada programador. Quando você experimentar o verdadeiro poder da criação de uma aplicação com DataSnap, pode nunca retornar ao desenvolvimento de aplicações de banco de dados conforme o conhece atualmente.

Criando aplicações WebSnap

Por Nick Hodges

CAPÍTULO

23

NESTE CAPÍTULO

- **Características do WebSnap**
- **Criando uma aplicação WebSnap**
- **Tópicos avançados**

O Delphi 6 introduz uma nova estrutura de aplicação Windows, chamada WebSnap, que traz os pontos fortes do *Rapid Application Development* (*RAD*) para o desenvolvimento na Web. Baseado no WebBroker e no InternetExpress, WebSnap é um grande salto adiante para os desenvolvedores Delphi que desejam usar sua ferramenta favorita para criar aplicações Web. Ele oferece todos os apetrechos padrão para as aplicações Web, incluindo gerenciamento de sessão, login de usuário, acompanhamento de preferências do usuário e scripting. Naturalmente, o Delphi 6 leva o RAD para o desenvolvimento de servidor Web, tornando fácil e rápida a criação de aplicações Web robustas, dinâmicas e baseadas em banco de dados.

Características do WebSnap

WebSnap não é uma tecnologia totalmente nova, e não despreza suas aplicações WebBroker e InternetExpress. WebSnap é compatível com essas duas tecnologias mais antigas, e a integração do seu código existente a uma nova aplicação WebSnap é um processo relativamente simples. O WebSnap oferece várias características, listadas nas próximas seções.

Múltiplos módulos Web

Nas versões anteriores do Delphi, aplicações WebBroker e InternetExpress tinham que fazer todo o seu trabalho em um único módulo Web. Múltiplos módulos Web não eram permitidos. Para incluir módulos de dados, eles tinham que ser criados manualmente em runtime, e não automaticamente. O WebSnap elimina essa restrição e permite que diversos módulos Web e módulos de dados façam parte de uma aplicação. WebSnap é baseado em módulos múltiplos, e cada módulo representa uma única página Web. Isso permite que diferentes desenvolvedores trabalhem em diferentes partes da aplicação sem ter que se preocupar com a modificação do código um do outro.

Scripting no servidor

O WebSnap integra de forma transparente o script no servidor às suas aplicações, permitindo-lhe criar facilmente objetos poderosos que aceitam script, os quais você pode usar para criar e personalizar suas aplicações e a HTML. O componente `TAdapter` e todos os seus descendentes são componentes que aceitam script, o que significa que podem ser chamados por seu script no servidor e produzir HTML e JavaScript no cliente para as suas aplicações.

Componentes `TAdapter`

Os componentes `TAdapter` definem uma interface entre uma aplicação e o scripting no servidor. O script no servidor só tem acesso à sua aplicação por meio de adaptadores, garantindo que o script não mude inadvertidamente os dados em uma aplicação ou exponha funções que não sejam voltadas para consumo público. Você pode criar descendentes personalizados de `TAdapter`, que gerenciam o conteúdo para as suas necessidades específicas, e esse conteúdo pode até mesmo ser visível e configurável durante o projeto. `TAdapter`s podem manter dados e executar ações. Por exemplo, o `TDataSetAdapter` pode exibir registros de um dataset, bem como tomar as ações normais em um dataset, como rolar, adicionar, atualizar e excluir.

Múltiplos métodos de despacho

O WebSnap oferece diversas maneiras de gerenciar pedidos HTTP. Você pode acessar seu conteúdo da Windows por nome de página, por ações de `TAdapter` ou por simples pedidos de ação da Web, como o WebBroker faz. Isso dá o poder e a flexibilidade de exibir suas páginas Windows com base em diversos tipos de entradas diferentes. Você pode querer exibir uma página em resposta a um botão de envio ou então pode querer criar um conjunto de links em um menu com base na coleção de páginas do seu site.

Componentes produtores de páginas

O WebBroker introduziu o `TPageProducer`, um componente para gerenciar HTML e inserir e atualiza o conteúdo com base em tags personalizadas. O InternetExpress aprimorou essa noção com `TMidasPageProducers`. O WebSnap aprimora a noção de produtores de página ainda mais, incluindo diversos controles novos e poderosos, que podem acessar o conteúdo de `TAdapter`, bem como dados XSL/XML. O mais poderoso desses novos descendentes de `TPageProducer` é `TAdapterPageProducer`, que sabe como produzir HTML com base nas ações e campos dos componentes `TAdapter`.

Gerenciamento de sessão

As aplicações WebSnap contêm gerenciamento de sessão interno e automático; agora você pode acompanhar as ações do usuário entre os diversos pedidos HTTP. Como o HTTP é um protocolo sem estado, suas aplicações Web precisam registrar os usuários deixando algo no cliente que identifique cada usuário. Normalmente, isso é feito com cookies, referências de URL ou controles de campo oculto. O WebSnap oferece suporte transparente para sessão, facilitando bastante o acompanhamento de usuários. O WebSnap faz isso por meio do seu componente `SessionsService`. O componente `SessionsService` mantém, de forma transparente, um valor de identificação de sessão para cada usuário, simplificando bastante a tarefa de registrar cada usuário enquanto realiza pedidos individuais. Isso normalmente é um serviço difícil de se controlar, mas o WebSnap cuida de todos os detalhes e disponibiliza a informação de sessão tanto para o script no servidor quanto para o próprio código da aplicação Web.

Serviços de login

Suas aplicações Web provavelmente terão que implementar alguma segurança, exigindo que os usuários se registrem em determinada aplicação. O WebSnap automatiza esse processo, oferecendo um componente adaptador de login especializado. Esse componente contém as funções necessárias para consultar e autenticar os usuários corretamente, de acordo com o modelo de segurança escolhido pela aplicação. Ele reúne informações de login e, em conjunto com o gerenciamento de sessão do WebSnap, oferece as credenciais atuais do login para cada pedido. Os componentes de login também automatizam a validação do login e a expiração do login. Por toda a sua aplicação, os usuários que tentarem acessar páginas não-autorizadas podem ser levados automaticamente para a página de login.

Acompanhamento do usuário

A função mais comum que o rastreamento de sessão oferece é a capacidade de acompanhar seus usuários e suas preferências para a sua aplicação. O WebSnap oferece componentes que permitem acompanhar informações do usuário com facilidade e exibi-las no seu site. Você pode armazenar as informações de login do usuário e depois apanhar as informações do usuário com base nisso. Você pode manter direitos de acesso do usuário e preferências do site, além de outras coisas, como informações do carrinho de compras.

Gerenciamento de HTML

Normalmente, em uma aplicação Web dinâmica, o acompanhamento e o gerenciamento da HTML pode ser difícil. O conteúdo HTML pode residir em diversos lugares, como em arquivos e recursos, ou então pode ser gerado dinamicamente. O WebSnap oferece um meio para que você gerencie esse processo com seus serviços de localização de arquivo.

Serviços de uploading de arquivo

O gerenciamento do uploading de arquivos normalmente requer muito código personalizado. O WebSnap oferece uma solução adaptadora simples, que gerencia os formulários em múltiplas partes necessários para o upload de arquivos. Você pode oferecer a capacidade de uploading de arquivo na sua aplicação WebSnap de modo rápido e fácil, usando a funcionalidade interna do componente `TAdapter`.

Criando uma aplicação WebSnap

Como sempre, a melhor maneira de aprender a respeito da nova tecnologia é experimentá-la. Vamos começar criando uma versão básica de uma aplicação WebSnap.

Projetando a aplicação

Primeiro, você desejará incluir uma barra de ferramentas WebSnap no IDE; portanto, dê um clique com o botão direito na área de botões da barra de título do IDE e selecione a barra de ferramentas Internet (veja a Figura 23.1). Isso acrescenta uma barra de ferramentas à janela principal do IDE, facilitando a criação de aplicações WebSnap e a inclusão de formulários e módulos Web.

Em seguida, dê um clique no botão com a mão segurando o globo, e você verá a caixa de diálogo que aparece na Figura 23.2.

Figura 23.1 Barra de ferramentas Internet.

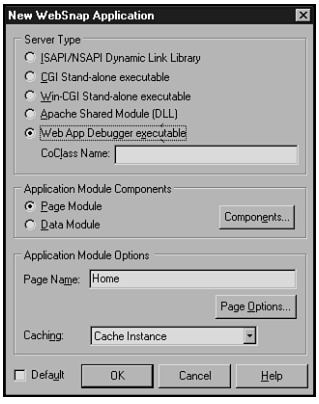

Figura 23.2 A caixa de diálogo New WebSnap Application.

A caixa de diálogo da Figura 23.2 oferece uma série de opções para configurar sua aplicação WebSnap. A primeira é o tipo de servidor em que sua aplicação será executada. Você tem cinco opções:

- ISAPI/NSAPI Dynamic Link Library – Essa opção produz um projeto que roda sob IIS (ou sob servidores Netscape, se o adaptador ISAPI correto for instalado). O projeto produz uma DLL quando compilado e roda no mesmo espaço de memória do servidor Web. O servidor Web mais comum para rodar aplicações ISAPI é o Internet Information Server da Microsoft, embora outros servidores Web possam rodar DLLs ISAPI.
- CGI Standalone executable – Essa opção cria um projeto que produz um executável pelo console, que lê e escreve nas portas de entrada e saída padrão. Ela se ajusta à especificação CGI. Quase todos os servidores Web aceitam CGI.

- Win-CGI Standalone executable – Essa opção produz um projeto Win-CGI que se comunica com um servidor Web por meio de arquivos INI baseados em texto. Win-CGI é muito rara e não é recomendada.
- Apache Shared Module (DLL) – Essa opção produz um projeto que rodará no servidor Web Apache. Para obter mais informações sobre o Apache, consulte `http://www.apache.org`.
- Web App Debugger Executable – Se você selecionar essa opção, obterá uma aplicação que será executada no Web App Debugger do Delphi (veja a Figura 23.3). Sua aplicação Web será um servidor COM out-of-process, e o Web App Debugger controlará a execução da aplicação. Esse tipo de aplicação Web permitirá usar o poder completo do depurador do Delphi para depurá-lo. Isso significa que não há mais disputa com os servidores Web, ligando-os e desligando-os para carregar e descarregar suas aplicações. Em vez disso, a depuração de sua aplicação será rápida e fácil.

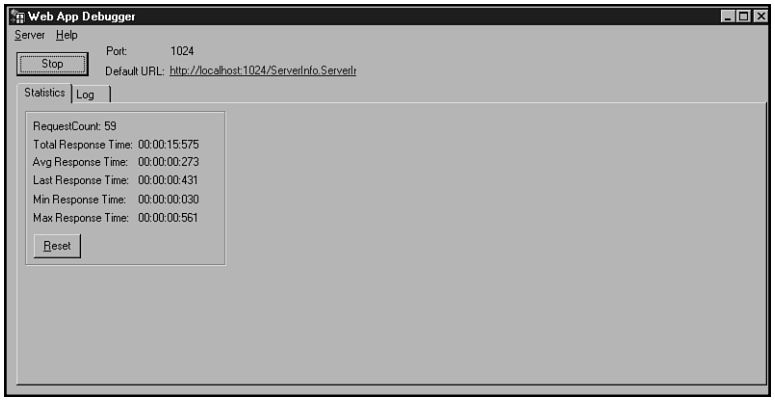

Figura 23.3 A aplicação Web App Debugger simplifica bastante a depuração de suas aplicações Web.

NOTA

O Web App Debugger pode ser acessado por meio do menu Tools no IDE. Para que funcione corretamente, você precisa registrar a aplicação encontrada no diretório `<Delphi Dir>\bin`, chamada `serverinfo.exe`. Tudo o que você precisa fazer para registrá-la é executá-la uma vez, e ela mesmo se registrará. O Web App Debugger é uma aplicação baseada em COM, que atua como um servidor Web para testar suas aplicações. Quando você cria uma aplicação para o Web App Debugger, seu novo projeto terá um formulário e um módulo Web. O formulário atua como um marcador de lugar para o servidor COM, e a execução da aplicação uma vez a registrará. Depois disso, o Web App Debugger a controlará por meio do browser Web, e servirá à sua aplicação no browser. Como a aplicação é um executável do Delphi, e não uma extensão do servidor Web, você pode definir um ponto de interrupção nela e executá-la no IDE do Delphi. Depois, quando você a acessar por meio do browser, o depurador do Delphi assumirá o comando quando seus pontos de interrupção forem alcançados, e você poderá depurar a aplicação normalmente. Para acessar sua aplicação por meio do browser, execute o Web App Debugger e dê um clique no hyperlink intitulado Default URL. Isso trará uma aplicação Web que lista todas as aplicações registradas com o servidor. Você pode, então, selecionar sua aplicação e executá-la. A opção View Details (exibir detalhes) permitirá ver mais informações sobre as diferentes aplicações, removendo-as do Registro quando não forem mais necessárias. No entanto, tenha cuidado para não excluir a aplicação `ServerInfo`; caso contrário, você terá que retornar e registrá-la novamente.

Para a aplicação de exemplo que você criará aqui, selecione a opção Web App Debugger. Isso permitirá depurar a aplicação enquanto você a cria.

A próxima opção no assistente permite selecionar o tipo de módulo que você deseja e os diferentes componentes que serão incluídos. Se você escolher a opção Page Module (módulo de página), receberá um módulo Web que representa uma página na sua aplicação. Se você escolher a opção Data Module, receberá um módulo de dados que pode ser usado em uma aplicação WebSnap. Ele pode realizar a mesma função dos módulos de dados nas aplicações tradicionais cliente/servidor. Para essa aplicação, selecione a opção Page Module.

Em seguida, dê um clique no botão Components, e você verá a caixa de diálogo mostrada na Figura 23.4.

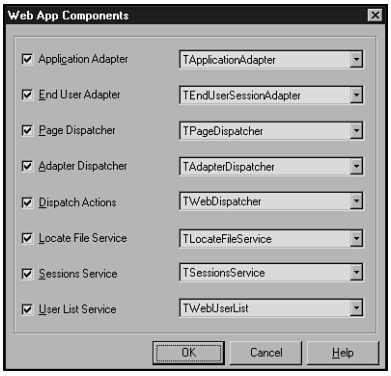

Figura 23.4 A caixa de diálogo Web App Components permite que você selecione os componentes que serão incluídos em seu novo módulo.

Você tem a opção dos seguintes componentes listados:

- Application Adapter – Esse componente gerencia os campos e as ações disponíveis por meio do objeto de scripting no servidor de aplicação. A propriedade mais comum que você usará nesse componente é a propriedade Title.
- End User Adapter – Esse componente gerencia as informações sobre o usuário atual da aplicação, como o ID de sessão, nome de usuário, direitos do usuário e outras informações personalizadas do usuário. Ele também gerenciará as ações de login e logout do usuário.
- Page Dispatcher – Esse componente gerencia e despacha os pedidos HTTP feitos pelo nome da página. Você pode criar links HREF ou ações que chamam páginas específicas, e o Page Dispatcher apanhará a resposta apropriada.
- Adapter Dispatcher – O Adapter Dispatcher cuida de todos os pedidos que chegam como resultado de ações do adaptador. Estes geralmente são o resultado de uma submissão de formulário HTML.
- Dispatcher Actions – Essa opção acrescenta um `TWebDispatcher` às suas aplicações. Os usuários do WebBroker se lembrarão desse componente. Ele cuida dos pedidos da aplicação com base nos URLs, assim como as aplicações WebBroker faziam. Você pode usar esse componente para incluir suas próprias ações personalizadas em sua aplicação, da mesma forma como fazia com o WebBroker.
- Locate File Service – Os eventos desse componente são chamados sempre que um módulo Web exige entrada HTML. Você pode incluir tratadores de evento que permitam evitar o mecanismo de descoberta HTML default e apanhar HTML de praticamente qualquer origem. Esse componente é usado freqüentemente para obtenção de conteúdo de página e modelos para a criação de páginas padrão.

- Sessions Service – Esse componente gerencia sessões para os usuários, permitindo-lhe manter o estado para usuários individuais entre os pedidos HTTP. Sessions Service pode armazenar informações sobre usuários e expirar automaticamente suas sessões depois de um certo período de inatividade. Você pode acrescentar qualquer informação específica da sessão que desejar na propriedade `Session.Values`, um array de variantes indexado por string. Por default, as sessões são gerenciadas com cookies na máquina do usuário, embora você pudesse criar uma classe para lidar com elas de alguma outra maneira, como com URLs gordos ou campos ocultos.
- User List Service – Esse componente mantém uma lista de usuários que estão autorizados a se registrar na aplicação e informações sobre eles.

NOTA

Cada uma dessas opções possui caixas suspensas que permitem escolher o componente que atenderá a cada um dos papéis indicados. Você pode criar seus próprios componentes, que atenderão a esses papéis e os registrarão com o WebSnap. Com isso, eles aparecerão como opções nessa caixa de diálogo. Por exemplo, você poderia criar um componente de sessão que mantenha informações de sessão em um URL gordo, em vez de usar cookies.

Para este exemplo, marque todas as caixas de seleção. Depois, para o componente End User Adapter, desça a caixa de combinação e selecione `TEndUserSessionAdapter`. Esse componente associará automaticamente um ID de sessão a um usuário final. Depois, dê um clique em OK.

A próxima opção no assistente é o nome da página. Chame essa página principal de `Home` e depois dê um clique em Page Options (opções de página). Você verá a caixa de diálogo que aparece na Figura 23.5.

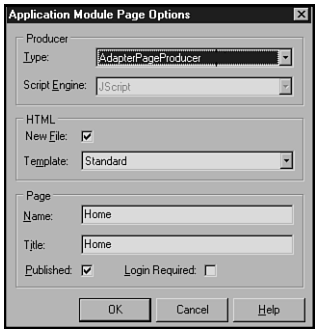

Figura 23.5 A caixa de diálogo Application Module Page Options permite selecionar as opções para a página no seu módulo Web.

Essa caixa de diálogo permite personalizar o componente `PageProducer` do seu módulo Web e a HTML associada a ele. Ela apresenta diversas opções. A primeira é o tipo de produtor de página. O WebSnap inclui uma série de PageProducers padrão, que podem produzir e gerenciar HTML de várias maneiras. Para começar, selecione a opção default, um PageProducer simples. Você também pode selecionar o tipo de scripting no servidor que deseja utilizar. O Delphi já vem com suporte para JScript e VBScript (além de XML/XSL, que serão discutidos mais adiante.) Aqui, deixe o valor default JScript.

Cada módulo possui uma página HTML associada a ele. A próxima opção permite selecionar que tipo de HTML você deseja. Por default, o Delphi oferece uma página Standard com um menu de navegação simples, ativado por script. Você pode criar seus próprios modelos HTML, registrá-los com o WebSnap e depois selecioná-los aqui. Veremos como fazer isso mais adiante, neste capítulo. Por enquanto, mantenha o valor default, Standard.

Chame a página de **Home** (o Title é preenchido automaticamente com o mesmo nome). Certifique-se de que Published (publicado) esteja marcado e mantenha Login Required (exige login) desmarcado. Uma página publicada aparecerá na lisa de páginas da aplicação e poderá ser referenciada pelo objeto de scripting Pages. Isso é necessário para a criação de menus baseados em página no script, usando o objeto de scripting Pages.

Depois que você tiver feito isso, dê um clique em OK e depois novamente em OK no assistente principal. O assistente, em seguida, criará sua aplicação para você, e o novo módulo Web ficará semelhante ao que aparece na Figura 23.6.

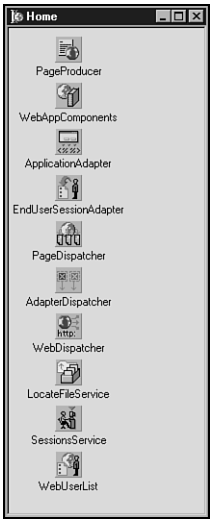

Figura 23.6 O módulo Web para a aplicação de demonstração, conforme criado pelo WebSnap Wizard.

Ainda não discutimos sobre o controle `TWebAppComponents`. Esse controle é a carteira de compensação central para todos os outros componentes. Como muitos dos componentes em uma aplicação WebSnap trabalham em conjunto, o componente `WebAppComponents` é aquele que os une e permite que se comuniquem e façam referência um ao outro. Suas propriedades consistem simplesmente em outros componentes que atendem aos papéis específicos discutidos anteriormente.

Nesse ponto, você deverá salvar o projeto. Para manter coerência com o restante do capítulo, chame o módulo Web (unit2) de wmHome, chame o formulário (Unit1) de ServerForm e chame o projeto em si de DDG6Demo.

Examinando o Code Editor, você deverá ver alguns recursos novos e desconhecidos. Primeiro, observe as guias na parte inferior. Cada módulo Web – por representar uma página em uma aplicação Web – possui um arquivo HTML associado, que pode conter script no servidor. A segunda guia na parte inferior mostra essa página (veja a Figura 23.7). Como você selecionou o modelo de página HTML Standard no assistente, a HTML possui script no servidor, que saudará o usuário se ele for conectado, e oferecerá um menu de navegação básico montado automaticamente com base em todas as páginas publicadas da aplicação. À medida que as páginas são acrescentadas nessa aplicação de demonstração, esse menu crescerá e permitirá que os usuários naveguem para cada uma das páginas. O código HTML default aparece na Listagem 23.1.

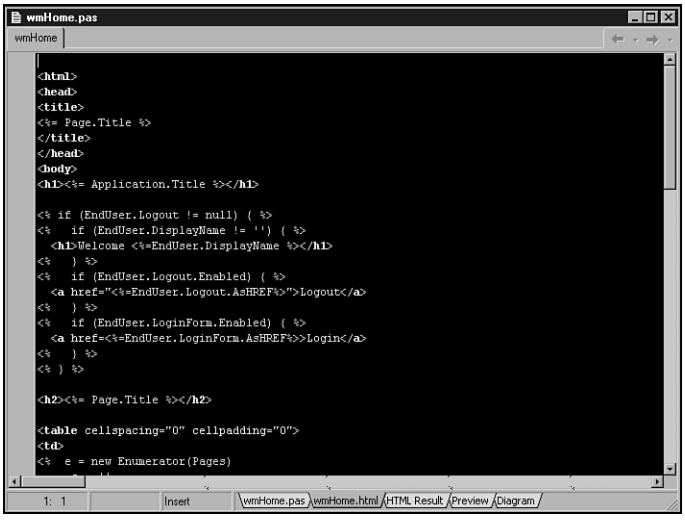

Figura 23.7 A página HTML associada ao módulo Web.

Listagem 23.1 Código HTML default

```
<html>
<head>
<title>
<%= Page.Title %>
</title>
</head>
<body>
<h1><%= Application.Title %></h1>

<% if (EndUser.Logout != null) { %>
<%   if (EndUser.DisplayName != '') { %>
  <h1>Welcome <%=EndUser.DisplayName %></h1>
<%   } %>
<%   if (EndUser.Logout.Enabled) { %>
  <a href="<%=EndUser.Logout.AsHREF%>">Logout</a>
<%   } %>
<%   if (EndUser.LoginForm.Enabled) { %>
  <a href=<%=EndUser.LoginForm.AsHREF%>>Login</a>
<%   } %>
<% } %>

<h2><%= Page.Title %></h2>

<table cellspacing="0" cellpadding="0">
<td>
<%  e = new Enumerator(Pages)
     s = ''
     c = 0
     for (; !e.atEnd( ); e.moveNext( ))
     {
       if (e.item( ).Published)
       {
         if (c>0) s += ' | '
         if (Page.Name != e.item( ).Name)
           s += '<a href="' + e.item( ).HREF + '">' + e.item( ).Title + '</a>'
         else
           s += e.item( ).Title
```

Listagem 23.1 Continuação

```
          c++
        }
      }
      if (c>1) Response.Write(s)
%>
</td>
</table>
</body>
</html>
```

Esse código contém tags HTML normais e também JScript no servidor.

Você também deverá observar que a HTML tem a sintaxe realçada no IDE. (Você pode definir as cores a serem usadas na página de propriedades Tools, Editor Options, Color.) Além disso, você pode definir seu próprio editor de HTML externo, como o HomeSite, e acessá-lo também por meio do IDE. Defina o HTML Editor em Tools, Environment Options, Internet. Selecione HTML no modo de exibição de lista e dê um clique em Edit. A partir daí, selecione a ação de edição apropriada para usar seu editor externo. Depois, quando você der um clique com o botão direito na página HTML do Code Editor, poderá selecionar a opção HTML Editor e chamar seu editor.

Além da guia de exibição da HTML, a próxima guia mostra a HTML que resulta do script sendo executado. A guia seguinte mostra uma visualização da HTML em uma janela do Internet Explorer. Observe que nem todo o script será executado e exibido nesse modo de exibição, pois parte do código se baseia em valores em runtime. No entanto, pelo menos você pode ter uma idéia de como a página ficará sem ter que executá-la no browser.

Acrescentando funcionalidade à aplicação

Agora, vamos incluir um pouco de código e fazer com que a aplicação realize algo. Primeiro, vá para o módulo Web Home e selecione o adaptador Application. Defina a propriedade `ApplicationTitle` como Delphi Developers Guide 6 WebSnap Demo Application. Observe que isso aparecerá imediatamente na guia de visualização, pois a HTML contém o seguinte script no servidor como primeira coisa de sua seção `<BODY>`:

```
<h1><%= Application.Title %></h1>
```

Isso faz com o que o objeto de scripting Application apresente o valor para `ApplicationTitle` na HTML.

Em seguida, vá para o Code Editor e selecione a página HTML para o Home Module. Depois mova o cursor para baixo da tag `</table>`, próximo do final, e inclua uma descrição resumida da página, saudando o usuário. O código neste CD-ROM possui tal entrada, acrescentando o seguinte:

```
<P>
<FONT SIZE="+1" COLOR="Red">Welcome to the Delphi 6 Developers Guide WebSnap
➥Demonstration Application!</FONT>
<P>
This application will demonstrate many of the new features in Delphi 6 and
➥WebSnap. Feel free to browse around and look at the code involved. There is
➥a lot of power, and thus a lot to learn, in WebSnap, so take your time and
➥don't try to absorb it all at once.
<P>
```

Esse novo código, naturalmente, também aparece imediatamente no painel HTML Preview.

Em seguida, só para provar que você está realmente criando uma aplicação para o browser, execute o projeto. A primeira coisa que você verá é um formulário em branco. Esse é o servidor COM. Você pode fechá-lo e depois iniciar o Web App Debugger no menu Tools. Depois de fazer

isso, dê um clique no hyperlink Default URL (ele será chamado `DDG6DemoApp.DDG6TestApp`), localize a aplicação na caixa de listagem no seu browser e dê um clique no botão Go. Seu browser deverá mostrar sua página conforme ilustrada na Figura 23.8.

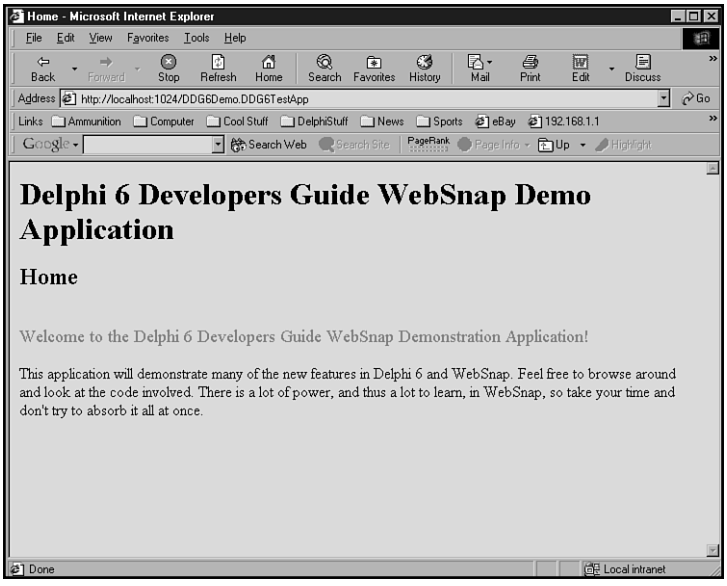

Figura 23.8 Resultados da primeira página incluída na aplicação de demonstração.

Como você pode ver, essa realmente é uma aplicação Web!

Barra do menu de navegação

Agora, você incluirá outra página, que demonstra o menu de navegação. Vá para a barra de menus principal do IDE e selecione o segundo botão no menu Internet, aquele com o pequeno globo e a folha de papel. Isso trará o New WebSnap Page Module Wizard, que é semelhante à caixa de diálogo que você viu como parte do assistente principal. Deixe todas as opções com os valores default, exceto pela caixa de edição Name. Chame a página de Simple. O resultado é um módulo Web com um único PageProducer. Observe que uma página HTML está associada a essa página, e ela possui o mesmo código da primeira página que vimos. Salve a unidade como `wmSimple.pas`.

Definindo opções de threading e caching

O New WebSnap Page Module Wizard possui duas opções na parte inferior, que determinam como as instâncias de cada módulo Web devem ser tratadas. A primeira é a opção Creation (criação). Os módulos Web podem ser criados On Demand (por demanda) ou Always (sempre). Os módulos Web criados por demanda são instanciados apenas quando chega um pedido para eles. Escolha essa opção para as páginas usadas com menos freqüência. Escolha Always para páginas que são criadas imediatamente na partida da aplicação. A segunda opção é Caching Option (opção de caching), que determina o que acontecerá com o módulo Web quando ele tiver terminado de atender seu pedido. Se for escolhido Cache Instance (manter instância em cache), cada módulo Web criado é colocado em cache quando terminar de fornecer um pedido, e permanecerá em um pool de instâncias em cache, pronto para ser usado novamente. É importante observar que, quando ele for usado novamente, os dados do campo estarão no mesmo estado em que se encontravam quando o último pedido foi atendido. Escolha Destroy Instance (destruir instância) se você quiser que cada instância do módulo Web seja destruída ao terminar, em vez de ser mantida em cache.

Em seguida, inclua alguma mensagem simples na página HTML, no mesmo ponto abaixo da tabela, na página padrão. Depois, compile e execute a aplicação por meio do Web App Debugger, como fizemos anteriormente. Se a página estava lá desde a última vez em que foi examinada, tudo o que você precisa fazer é dar um clique no botão Refresh (atualizar) no seu browser.

Dessa vez, quando você executar a aplicação, deverá notar que o menu de navegação agora aparece. Esse menu é resultado do seguinte script no servidor:

```
<%  e = new Enumerator(Pages)
     s = ''
     c = 0
     for (; !e.atEnd( ); e.moveNext( ))
     {
       if (e.item( ).Published)
       {
         if (c>0) s += ' | '
         if (Page.Name != e.item( ).Name)
           s += '<a href="' + e.item( ).HREF + '">' + e.item( ).Title + '</a>'
         else
           s += e.item( ).Title
         c++
       }
     }
     if (c>1) Response.Write(s)
%>
```

Esse código simplesmente percorre o objeto de scripting Pages, montando um menu de nomes de página. O código cria um link se a página encontrada não for a página atual. Assim, a página atual não é um link, e todos os outros nomes de página são, não importa qual seja a página atual. Esse é um menu bastante simples e, naturalmente, você poderia escrever seus próprios menus mais sofisticados para a sua aplicação personalizada.

NOTA

Se você alternar entre as duas páginas, poderá observar que o formulário da aplicação pisca em segundo plano toda vez que um pedido for feito. Isso porque o Web App Debugger está chamando a aplicação como um objeto COM para cada pedido, executando a aplicação, apanhando a resposta HTTP e fechando a aplicação.

Em seguida, você pode tornar parte da aplicação restrita apenas a usuários que efetuaram o login. Primeiro, inclua uma página que exija o login de um usuário para que ele possa vê-la. Dê um clique no botão New WebSnap Page (nova página WebSnap) na barra de ferramentas Internet e chame a página de "LoggedIn". Depois, selecione a caixa de seleção Login Required. Dê um clique em OK para criar uma página Web que só possa ser vista por um usuário que efetuou login. Salve a página como `wmLoggedIn.pas`. Depois, inclua algum código HTML na página, permitindo que o usuário saiba que somente os usuários com login poderão ver a página. A aplicação neste CD-ROM inclui o seguinte:

```
<P>
<FONT COLOR= "Green "><B>Congratulations!</B></FONT>
<BR>
You are successfully logged in!Only logged in users are granted access
➥.to this page.All others are sent back to the Login page.
<P>

<P>
<FONT COLOR="Green"><B>Congratulations! </B></FONT>
```

```
<BR>
You are successfully logged in!  Only logged in users are granted access
➥to this page. All others are sent back to the Login page.
<P>
```

A única diferença entre a página LoggedIn e a página Simple é um parâmetro no código que registra a página com o gerenciador de aplicação. Toda página WebSnap possui uma seção de inicialização que se parece com isto:

```
initialization
  if WebRequestHandler < > nil then
  WebRequestHandler.AddWebModuleFactory(TWebPageModuleFactory.Create(TLoggedIn,
TWebPageInfo.Create([wpPublished, wpLoginRequired], '.html'),
➥crOnDemand, caCache));
```

Esse código registra a página com o objeto `WebRequestHandler`, que gerencia todas as páginas e oferece seu conteúdo HTML quando for preciso. `WebRequestHandler` sabe como criar, colocar em cache e destruir instâncias de módulos Web quando for preciso. O código anterior é o código para a página LoggedIn, e possui o parâmetro `wpLoginRequired`, dizendo à página que somente os usuários que efetuaram login podem ter acesso a ela. Por default, o New Page Wizard acrescenta esse valor, mas o transforma em comentário. Se você quiser que a página fique protegida por senha mais tarde, poderá simplesmente retirar o símbolo de comentário do parâmetro e recompilar a aplicação.

Login

Você precisa criar uma página que permita o login do usuário. Primeiro, no entanto, existe uma tarefa de manutenção a ser feita na página Home.

Primeiro, crie uma nova página e dê-lhe o nome Login. Depois, selecione `TAdapterPageProducer` para o tipo de produtor de página. Dessa vez, no entanto, não a publique, removendo a seleção da caixa Publish, e, obviamente, não exija que um usuário efetue login para ver a página de login! Desmarcando a opção Publish, a página estará disponível para uso, mas não fará parte do objeto de scripting Pages, e por isso não aparecerá no menu de navegação. Salve-a como wmLogin. Dessa vez, vá para a página WebSnap da Component Palette e inclua um componente `TLoginAdapter` no módulo.

O `TAdapterPageProducer` é um PageProducer especializado, que sabe como exibir e tratar dos campos e controles HTML apropriados para um `TAdapter`. No caso da aplicação de demonstração, esse `TAdapterPageProducer` irá exibir as caixas de edição Username (nome de usuário) e Password (senha), que o usuário terá que usar para efetuar o login. Quando você começar a entender melhor o WebSnap, rapidamente desejará usar `TAdapterPageProducers` em todas as suas páginas, pois eles facilitam bastante a exibição de informações de `TAdapter`, a execução de ações de `TAdapter` e a criação de formulários HTML com base em campos `TAdapter`.

Como o `TLoginFormAdapter` possui todos os campos necessários para isso, a criação da página de login será muito fácil, e feita sem qualquer código – é isso mesmo, sem código! Você poderá acrescentar usuários, criar uma página de login e impor o login sobre páginas especificadas, tudo sem uma única linha de código.

Primeiro, para gerenciar os logins, você terá que criar alguns usuários. Vá para o módulo Web Home e dê um clique duplo no componente `WebUserList`. Esse componente gerencia usuários e senhas. Você pode facilmente acrescentar usuários e suas senhas. Dê um clique no botão New e inclua dois usuários diferentes. Acrescente quaisquer senhas que você quiser para cada usuário. Os dois usuários na aplicação de demonstração no CD-ROM são ddg6 e user. Suas senhas são iguais aos seus nomes de usuário, como mostra a Figura 23.9.

Figura 23.9 O editor de componentes para o componente **`WebUserList`**, com dois usuários acrescentados.

Selecione o `EndUserSessionAdapter` e defina a propriedade `LoginPage` como `Login`, ou seja, o nome da página que possui os controles para o login dos usuários. Em seguida, volte ao módulo Web Login e dê um clique duplo no componente `TAdapterPageProducer`. Isso trará o projetista Web Surface Designer, mostrado na Figura 23.10.

Selecione o `AdapterPageProducer` no canto superior direito e dê um clique no botão New Component. Selecione `AdapterForm` e dê um clique em OK. Depois, selecione o `AdapterForm1` e dê um clique no botão New Component novamente. Selecione `AdapterErrorList`. Faça o mesmo para `AdapterFieldGroup` e `AdapterCommandGroup`. Depois, defina a propriedade `Adapter` para esses três componentes como `LoginFormAdapter1`. Depois, selecione o `AdapterFieldGroup` e inclua dois objetos `AdapterDisplayField`. Defina a propriedade `FieldName` no primeiro como `UserName`, e o segundo como `Password`. Selecione o `AdapterCommandGroup` e defina sua propriedade `DisplayComponent` como `AdapterFieldGroup1`. Você deverá então estar com um formulário semelhante ao da Figura 23.10. Se você fechar esse formulário e depois passar para o Code Editor, poderá ver que o formulário agora possui os controles de login.

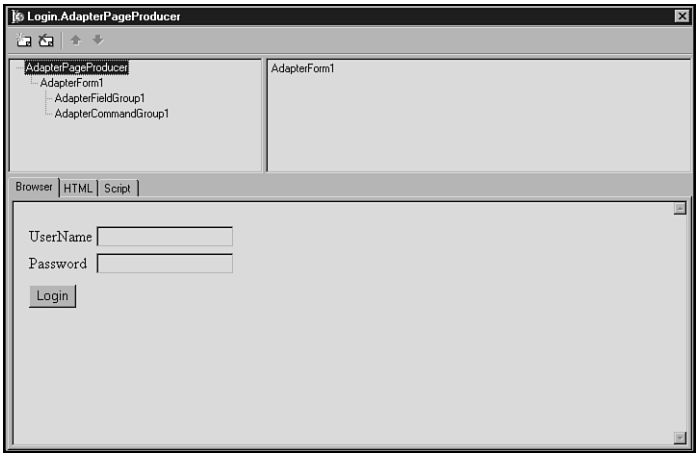

Figura 23.10 O Web Surface Designer **`TAdapterPageProducer`** com os componentes **`LoginFormAdapter`** incluídos.

E isso é tudo o que você precisa fazer. Execute a aplicação e deixe-a em execução. Agora, como você está se baseando nas informações de sessão sobre o usuário, precisa deixar a aplicação na memória para que ela se lembre que você efetuou o login. Execute a aplicação no browser e depois tente navegar até a página que exige o login. Ela deverá levá-lo para a página Login. Digite um nome de usuário e senha válidos, dê um clique no botão Login e você será levado à página que pedirá seu login. A partir daí, qualquer página que você especifique como exigindo um login válido só aparecerá se você tiver efetuado o login corretamente. Caso contrário, ela o levará para a página de login. Tudo isso acontece sem a escrita de uma única linha de código em Pascal.

Experimente isto também: efetue o logout, selecionando o link Logout, e depois tente efetuar o login com um nome de usuário ou senha inválida. Observe que aparecerá uma mensagem de erro. Isso é o componente `AdapterErrorList` em ação. Ele coletará automaticamente os erros de login e os mostrará para você.

Quando você estiver autenticado na aplicação e navegando por suas páginas, notará que ela se lembra de quem você é e apresenta seu nome de login no cabeçalho de cada página. Isso é resultado do seguinte script no servidor, no arquivo HTML para os módulos Web:

```
<% if (EndUser.Logout != null) { %>
<%   if (EndUser.DisplayName != '') { %>
  <h1>Welcome <%=EndUser.DisplayName %></h1>
<%   } %>
```

Gerenciando dados de preferência do usuário

A próxima coisa que você pode querer fazer é manter algumas informações de preferência de usuário. As aplicações mais dinâmicas, baseadas no usuário, desejarão mostrar todos os diferentes tipos de informações do usuário, desde os itens em um carrinho de compras até as preferências de cores de um usuário. Naturalmente, o WebSnap facilita bastante essa tarefa. Mas, dessa vez, você terá realmente que escrever algumas linhas de código.

Primeiro, inclua outra página na aplicação. Dê-lhe um `TAdapterPageProducer` e exija que o usuário efetue o login para exibi-la. (Nesse ponto, você já saberá como fazer isso usando a barra de ferramentas e o assistente resultante.) Salve o arquivo como `wmPreferenceInput`. Inclua um `TAdapter` ao módulo Web. Troque o nome do `Adapter` de `Adapter1` para `PrefAdapter`, como mostra a Figura 23.11.

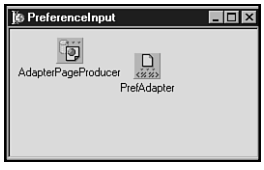

Figura 23.11 O módulo Web **`PreferenceInput`** que colherá as preferências do usuário.

Primeiro, dê um clique duplo no componente `PrefAdapter` e depois inclua dois `AdapterFields` e um `AdapterBooleanField`. Chame os dois `AdapterFields` de FavoriteMovie e PasswordHint. Chame o `AdapterBooleanField` de LikesChocolate. (Observe que, quando você troca o nome desses componentes, os valores de `DisplayLabel` e `FieldName` também mudam.) Você também pode mudar os valores de `DisplayLabel` para outros que façam mais sentido na sua HTML.

O componente `PrefAdapter` manterá os valores para essas preferências, e eles poderão ser acessados a partir de outras páginas. `TAdapters` são componentes que aceitam script, que podem manter, gerenciar e manipular informações para você, mas isso exigirá algum código. Cada um dos três `AdapterFields` que você criou precisam ser capazes de apanhar seus valores quando solicitados no script, de modo que cada um possui um evento `OnGetValue` que faz exatamente isso. Como você deseja que essa informação seja mantida entre os pedidos, ela será armazenada na propriedade `Session.Values`. A variável `Session.Values` é um array de variantes indexado por string, de modo que você pode armazenar quase tudo nela, e ela manterá essa informação enquanto a sessão atual estiver ativa.

A classe `TAdapter` também permite tomar ações sobre seus dados. Normalmente, isso terá a forma de um botão Submit no seu formulário HTML. Selecione o componente `PrefAdapter`, vá para o Object Inspector e dê um clique duplo na propriedade Actions. Inclua uma única ação e chame-a de `SubmitAction`. Mude sua propriedade `DisplayLabel` para Submit Information. Depois, vá para a página Events no Object Inspector e inclua este código no evento `OnExecute` da ação, como mostra a Listagem 23.2.

Listagem 23.2 Tratador `OnExecute`

```
procedure TPreferenceInput.SubmitActionExecute(Sender: TObject;
  Params: TStrings);
var
  Value: IActionFieldValue;
begin

  Value := FavoriteMovieField.ActionValue;
  if Value.ValueCount > 0 then
  begin
    Session.Values[sFavoriteMovie] := Value.Values[0];
  end;

  Value := PasswordHintField.ActionValue;
  if Value.ValueCount > 0 then
  begin
    Session.Values[sPasswordHint] := Value.Values[0];
  end;

  Value := LikesChocolateField.ActionValue;
  if Value < > nil then
  begin
    if Value.ValueCount > 0 then
    begin
      Session.Values[sLikesChocolate] := Value.Values[0];
    end;
  end else
  begin
    Session.Values[sLikesChocolate] := 'false';
  end;;
end;
```

Esse código apanha os valores dos campos de entrada na sua HTML quando o usuário aciona o botão Submit e coloca os valores na variável de sessão, para serem recuperados pelos `AdapterFields`.

Naturalmente, você precisa ser capaz de apanhar esses valores depois que estiverem definidos, de modo que cada campo do adaptador apanhará seu valor de volta do objeto `SessionsService`. Para cada campo no adaptador, torne os tratadores de evento `OnGetValue` semelhantes ao código da Listagem 23.3.

Listagem 23.3 Tratadores de evento `OnGetValue`

```
...
const
  sFavoriteMovie = 'FavoriteMovie';
  sPasswordHint = 'PasswordHint';
  sLikesChocolate = 'LikesChocolate';
  sIniFileName = 'DDG6Demo.ini';
...

procedure TPreferenceInput.LikesChocolateFieldGetValue(Sender: TObject;
  var Value: Boolean);
var
  S: string;
begin
  S := Session.Values[sLikesChocolate];
  Value := S = 'true';
```

Listagem 23.3 Continuação

```
end;

procedure TPreferenceInput.FavoriteMovieFieldGetValue(Sender: TObject;
  var Value: Variant);
begin
  Value := Session.Values[sFavoriteMovie];
end;

procedure TPreferenceInput.PasswordHintFieldGetValue(Sender: TObject;
  var Value: Variant);
begin
  Value := Session.Values[sPasswordHint];
end;
```

Em seguida, você precisa ser capaz de exibir controles que realmente apanham os dados do usuário. Você fará isso por meio de `TAdapterPageProducer`, assim como fez com a página Login. Primeiro, dê um clique duplo em `TAdapterPageProducer`; você voltará ao Web Surface Designer mais uma vez. Crie um novo `AdapterForm` e inclua um `AdapterFieldGroup`, além de um `AdapterCommandGroup`. Defina a propriedade `Adapter` do `AdapterFieldGroup` como `PrefAdaper` e defina o `DisplayComponent` do `AdapterCommandGroup` como `AdapterFieldGroup`. Depois, dê um clique com o botão direito no `AdapterFieldGroup` e selecione Add All Fields (adicionar todos os campos) no menu. Para cada um dos campos resultantes, use o Object Inspector para definir a propriedade `FieldName` com os valores apropriados. Você também pode mudar as propriedades `Caption` para valores mais amigáveis do que o default. Depois selecione o `AdapterCommandGroup`, dê um clique nele com o botão direito e selecione Add All Commands (adicionar todos os comandos) no menu. Defina a propriedade `ActionName` do `AdapterActionButton` resultante como `SubmitAction`. Finalmente, defina a propriedade `AdapterActionButton.PageName` como `PreferencesPage`. (Essa é a página para onde a ação irá quando terminar com o processamento da ação. Você criará essa página em um minuto.)

Se algo não for conectado corretamente no Web Surface Designer, você verá uma mensagem de erro na guia Browser. A mensagem o instruirá sobre as propriedades que precisam ser definidas para que tudo seja conectado corretamente e para que a HTML seja exibida de modo adequado.

Depois que tudo isso for feito e a HTML estiver correta, a página estará pronta. Agora, se você executar a aplicação, verá uma página adicional no menu. Se você efetuar o login, poderá ver os controles de entrada para inserir seus dados de preferência. Não dê um clique no botão Submit, simplesmente porque ainda não há um lugar para ir.

Em seguida, crie uma página para exibir as preferências do usuário usando a barra de ferramentas e depois chame-a de PreferencesPage. Publique a página e exija que os usuários efetuem o login para exibi-la. (Novamente, o assistente fará tudo isso por você, como antes.) Salve a nova unidade como `wmPreferences`.

Depois, vá para a HTML da página e, na área logo abaixo da tabela que mantém o menu de navegação, inclua o seguinte script:

```
<P>
<B>Favorite Movie: <%= Modules.PreferenceInput.PrefAdapter.FavoriteMovieField.
➥Value %>
<BR>
Password Hint: <%= Modules.PreferenceInput.PrefAdapter.PasswordHintField.
➥Value %>

<BR>
<% s = ''
  if (Modules.PreferenceInput.PrefAdapter.LikesChocolateField.Value)
     s = 'You like Chocolate'
   else
```

```
    s = 'You do not like chocolate'
  Response.Write(s);
%>
```

Agora, quando você compilar e executar a aplicação, poderá digitar suas preferências e dar um clique no botão Submit – a aplicação se lembrará e mostrará suas preferências na página Preferences. Você também pode acessar esses valores no script para qualquer outra página, uma vez definidos. Os valores são mantidos entre os pedidos HTTP pelo objeto Session e apanhados do componente `Adapter` por meio do script.

> NOTA
>
> Cada uma das páginas em uma aplicação WebSnap possui um arquivo HTML associado, como você viu. Como esses arquivos existem fora da aplicação, você pode editá-los, salvar as alterações, atualizar a página no seu browser e ver os resultados sem recompilar sua aplicação. Isso significa que você mesmo pode atualizar a página sem ter que parar seu servidor Web. Você também pode experimentar facilmente seu script no servidor durante o desenvolvimento, sem ter que recompilar sua aplicação. Mais adiante, neste capítulo, você verá meios alternativos de armazenar e apanhar sua HTML.

Mantendo dados de preferência entre as sessões

Só há um problema agora – as seleções do usuário não são mantidas entre as sessões. As preferências são perdidas se o usuário efetuar o logout. Você pode fazer com que esses valores sejam mantidos até mesmo entre diferentes sessões, armazenando-os toda vez que a sessão terminar e apanhando-os toda vez que um usuário efetuar o login. A aplicação de demonstração lê quaisquer dados armazenados por meio do evento `LoginFormAdapter.OnLogin`, e depois grava quaisquer dados com o evento `SessionService.OnEndSession`. O código para esses dois eventos aparece na Listagem 23.4.

Listagem 23.4 Eventos `OnLogin` e `OnEndSession`

```
procedure TLogin.LoginFormAdapter1Login(Sender: TObject; UserID: Variant);
var
  IniFile: TIniFile;
  TempName: string;
begin
  // Apanhe dados da sessão aqui
  TempName := Home.WebUserList.UserItems.FindUserID(UserId).UserName;
➥  //WebContext.EndUser.DisplayName;
  Home.CurrentUserName := TempName;

  Lock.BeginRead;
  try
    IniFile := TIniFile.Create(IniFileName);
    try
      Session.Values[sFavoriteMovie] := IniFile.ReadString(TempName,
      ➥sFavoriteMovie, '');
      Session.Values[sPasswordHint] := IniFile.ReadString(TempName,
      ➥sPasswordHint, '');
      Session.Values[sLikesChocolate] := IniFile.ReadString(TempName,
      ➥sLikesChocolate, 'false');
    finally
      IniFile.Free;
    end;
  finally
```

Listagem 23.4 Continuação

```
    Lock.EndRead;
  end;
end;

procedure THome.SessionsServiceEndSession(ASender: TObject;
  ASession: TAbstractWebSession; AReason: TEndSessionReason);
var
  IniFile: TIniFile;
begin
  // Salve as preferências aqui
  Lock.BeginWrite;
  if FCurrentUserName < > '' then
  begin
    try
      IniFile := TIniFile.Create(IniFileName);
      try
        IniFile.WriteString(FCurrentUserName, sFavoriteMovie,
        ➥ASession.Values[sFavoriteMovie]);
        IniFile.WriteString(FCurrentUserName, sPassWordHint,
        ➥ASession.Values[sPasswordHint]);
        IniFile.WriteString(FCurrentUserName, sLikesChocolate,
        ➥ASession.Values[sLikesChocolate]);
      finally
        IniFile.Free;
      end;
    finally
      Lock.EndWrite
    end;
  end;
end;
```

Esses tratadores de evento armazenam os dados em um arquivo INI, mas não há motivo para que você não armazene os dados em um banco de dados ou em qualquer outro método de armazenamento persistente.

A variável `Lock` é uma variável global do tipo `TMultiReadExclusiveWriteSynchronizer`, e é criada na seção de inicialização da página Home. Como várias sessões poderiam estar lendo e gravando no arquivo INI, esse componente torna a leitura e escrita no arquivo INI à prova de threads. Inclua a declaração a seguir na parte de interface da sua unidade `wmHome`:

```
var
    Lock: TMultiReadExclusiveWriteSynchronizer;
```

E depois inclua isto nas seções de inicialização e finalização da mesma unidade:

```
Initialization
  ...
  Lock := TMultiReadExclusiveWriteSynchronizer.Create;
finalization
  Lock.Free;
```

Esse código também utiliza uma função chamada `IniFileName`, que é declarada da seguinte forma:

```
const
      sIniFileName ='DDG6Demo.ini';

...
```

```
function IniFileName: string;
begin
  Result := ExtractFilePath(GetModuleName(HInstance)) + sIniFileName;
end;
```

Inclua isso em sua unidade `wmHome` e você deverá ter uma aplicação Web totalmente funcionando, que efetua login de usuários e mantém suas preferências, até mesmo entre diferentes sessões.

Tratamento de imagens

Praticamente toda aplicação Web apresenta gráficos. Os gráficos podem melhorar a atratividade e a funcionalidade da sua aplicação. Naturalmente, o WebSnap torna a inclusão de imagens e gráficos nas suas aplicações tão fácil quanto, bem, tudo o mais o que o WebSnap faz. Como você poderia esperar, o WebSnap permitirá usar gráficos e imagens de qualquer origem que você preferir – arquivos, recursos, streams de banco de dados e assim por diante. Se os dados da sua imagem puderem ser colocados em um stream, eles podem ser usados em uma aplicação WebSnap.

Use a barra de ferramentas Internet para incluir outra página na sua aplicação. Use um `TAdapterPageProducer`, publique a página e exija que os usuários efetuem login para obter acesso a ele. Em seguida, chame a página de Images e salve a unidade resultante como `wmImages`. Depois que isso for feito, vá para o módulo Web Imagens, inclua um `TAdapter` no módulo e dê-lhe o nome ImageAdapter. Finalmente, dê um clique duplo em ImageAdapter e inclua dois campos do tipo `TAdapterImageField`. Cada um desses mostrará um modo diferente de exibir imagens.

Primeiro, você pode exibir uma imagem com base em um URL. Destaque o primeiro `AdaperImageField` e defina a propriedade `HREF` como um URL totalmente qualificado, que aponta para uma imagem no seu sistema ou em qualquer lugar na Internet, pelo mesmo motivo. Por exemplo, se você quiser ver o histórico de um ano do preço das ações da Borland, defina a propriedade `HREF` como `http://chart.yahoo.com/c/1y/b/borl.gif`.

Dê um clique duplo no `TAdapterPageProducer`, no módulo Web Images, inclua um `AdapterForm` e depois inclua um `AdapterFieldGroup` neste. Defina a propriedade adaptadora desse novo `AdapterFieldGroup` para o `ImageAdapter`. Depois, dê um clique com o botão direito novamente no `AdapterFieldGroup` e selecione Add All Fields. Em seguida, defina o campo `ReadOnly` do `AdapterImageField` como `True`. Se essa propriedade for `True`, ela mostrará a imagem na sua página. Se for definida como `False`, ela dará uma caixa de edição e um botão para pesquisar um nome de arquivo. Obviamente, para ver as imagens, você definiria essa propriedade como `True`. Quando você olhar para a imagem pela primeira vez, notará que a imagem possui uma pequena e incômoda legenda. Normalmente, você não desejará isso, de modo que, para livrar-se dela, defina a propriedade `Caption` como um único espaço. (Observe que uma legenda em branco não é aceita.) Você deverá então ver o gráfico aparecendo no Web Surface Designer, como mostra a Figura 23.12.

Figura 23.12 O Web Surface Designer com um gráfico vindo de ImageAdapterField.

NOTA

Se você quiser que as imagens sejam referenciadas por links relativos, para aparecerem durante o projeto, terá que incluir o diretório onde elas residem em Search Path (caminho de pesquisa) na página Options do Web App Debugger.

Agora você pode exibir imagens com base em um URL. Outras vezes, no entanto, você pode querer apanhar uma imagem de um stream. O componente `AdapterImageField` também oferece suporte para isso. Selecione o segundo `AdapterImageField` no seu `ImageAdapter` e abra o Object Inspector. Vá para a página de eventos e dê um clique duplo em `OnGetImage`. Coloque uma imagem JPG no mesmo diretório da aplicação (a demonstração neste CD-ROM usa `athena.jpg`) e faça com que seu tratador de evento fique semelhante a este:

```
procedure TImages.AdapterImageField2GetImage(Sender: TObject;
  Params: TStrings; var MimeType: String; var Image: TStream;
  var Owned: Boolean);
begin
  MimeType := 'image\jpg';
  Image := TFileStream.Create('athena.jpg', fmOpenRead);
end;
```

Esse código é bastante simples – Image é uma variável de stream que você cria e preenche com uma imagem. Naturalmente, a aplicação precisa saber que tipo de imagem está apanhando, de modo que você pode retornar essa informação no parâmetro `MimeType`. Um `TFileStream` é uma solução simples, mas poderia apanhar a imagem de qualquer origem, como um `BlobStream` de um banco de dados, ou criar a imagem em plena execução e retorná-la em um stream da memória. Agora, quando você executar a aplicação, deverá ver o JPG que escolheu logo abaixo do gráfico de ações.

Exibindo dados

Logicamente, você deseja que sua aplicação faça mais do que as coisas simples que ela faz até aqui. Você certamente deseja ser capaz de exibir dados a partir de um banco de dados, tanto em formato tabular quanto registro por registro. Naturalmente, o WebSnap facilita isso, e você pode criar aplicações de banco de dados poderosas com somente uma pequena quantidade de código. Usando o `TDatasetAdapter` e seus campos e ações internas, você pode facilmente exibir dados, além de fazer acréscimos, atualizações e exclusões em qualquer banco de dados.

Na realidade, exibir um dataset em um formulário é muito fácil. Inclua uma nova unidade na sua aplicação de demonstração – mas, dessa vez, torne-a um WebDataModule, usando o terceiro botão na barra de ferramentas Internet. Esse assistente é simples; portanto, simplesmente aceite as opções default. Depois inclua um `TDatasetAdapter` a partir da guia WebSnap na Component Palette e uma `TTable` a partir da guia BDE. Aponte a `TTable` para o banco de dados DBDemos e depois para a tabela BioLife. Depois defina a propriedade `Dataset` de `DatasetAdapter1` como `Table1`. Finalmente, defina `Table1.Active` como `True` para abrir a tabela. Chame o WebDataModule de `BioLife data` e salve a unidade como `wdmBioLife`.

NOTA

Sua aplicação está usando uma tabela do Paradox simples, baseada no BDE, mas o componente `TDatasetAdapter` exibirá dados a partir de qualquer descendente de `TDataset`. Observe, também, que não é realmente uma boa idéia usar uma `TTable` em uma aplicação Web sem suporte explícito para sessão. A aplicação de demonstração faz isso simplesmente para facilitar o uso e manter sua atenção nos recursos do WebSnap, e não nos dados.

Depois, como uma mudança de rumo, use a Object Treeview para definir as propriedades dos componentes. Se a Object Treeview não estiver visível, selecione View, Object Treeview no menu principal. Selecione o `DatasetAdapter`, dê um clique com o botão direito no nó Actions e selecione Add All Actions (adicionar todas as ações). Depois, conecte a `TTable` ao `TAdapterDataset` por meio de sua propriedade `Dataset`. Selecione o nó Fields e dê um clique com o botão direito, selecionando Add All Fields. Faça o mesmo para a `TTable`, incluindo todos os campos do dataset ao WebDataModule. Depois, como o WebSnap cria servidores sem estado para operações de banco de dados, você precisa indicar uma chave primária para o dataset, para permitir a navegação solicitada pelo cliente e a manipulação de dados. O WebSnap fará tudo isso por você automaticamente, depois que você especificar a chave primária. Faça isso selecionando o campo Species_No Field na Object Treeview e incluindo o valor `pfInKey` em sua propriedade `ProviderFlags`.

Em seguida, inclua uma página normal na aplicação. Torne essa página Login Required, dê-lhe um `TAdapterPageProducer` e chame a página de Biolife. Salve a unidade como `wmBioLife`. Como você deseja exibir os dados nessa nova página, inclua o nome de unidade `wdmBioLife` à cláusula `uses` da sua unidade `wmBioLife`. Depois, dê o foco ao módulo Web Biolife e dê um clique com o botão direito no componente `AdapterPageProducer`. Dê um clique com o botão direito no nó WebPageItems, logo abaixo dele, selecione New Component (novo componente) e selecione um `AdapterForm`. Selecione o `AdapterForm`, dê um clique com o botão direito nele e inclua uma `AdapterErrorList`. Depois inclua uma `AdapterGrid`. Defina a propriedade `Adapter` dos dois componentes como `DatasetAdapter`. Dê um clique com o botão direito na `AdapterGrid` e selecione Add All Columns (adicionar todas as colunas). Depois, selecione o nó Actions sob o `DatasetAdapter`, dê um clique com o botão direito nele e selecione Add All Actions. Em seguida, selecione o nó Fields, dê um clique com o botão direito e inclua todos os campos também. Você agora deverá ter todas as propriedades definidas corretamente para exibir dados.

Vá até o módulo Web BioLife e dê um clique duplo no `AdaperPageProducer`. Você verá o Web Surface Designer com dados vivos. Se não, verifique se você abriu a tabela e conectou todas as propriedades Adapter para os componentes dentro do `DatasetAdapter`. O campo Notes torna a tabela muito longa. Portanto, selecione a `AdapterGrid` no canto superior esquerdo e o componente `ColNotes` no painel superior direito; depois, apague-o. Agora você deverá ter algo semelhante ao que mostramos na Figura 23.13.

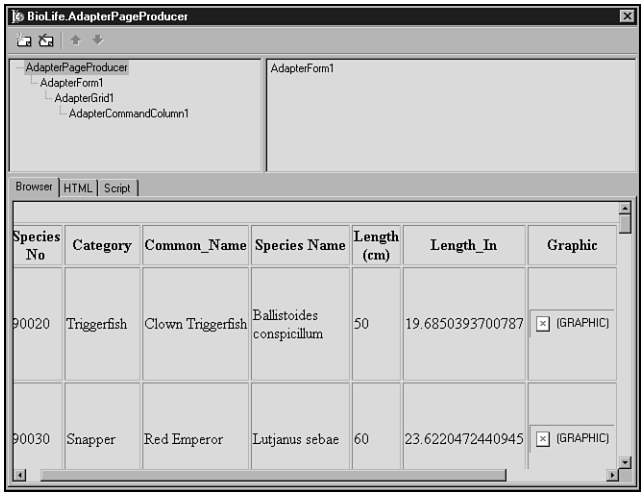

Figura 23.13 A tabela BioLife no Web Surface Designer de um **`TAdapterPageProducer`**; a tabela HTML é produzida pelo componente **`TDatasetAdapter`**.

Os gráficos não aparecem durante o projeto, mas sim em runtime. Na realidade, você agora pode compilar e executar a aplicação, e pode ver todos os dados na página BioLife – tudo sem escrever uma única linha de código.

Naturalmente, olhar apenas para os dados não é muito útil. Provavelmente você desejará manipular os registros individuais. Certamente, isso é fácil de se fazer no WebSnap. Vá para o Web Surface Designer e selecione a `AdapterGrid`. Dê um clique com o botão direito nela e inclua uma `AdapterCommandColumn`. Depois, dê um clique com o botão direito nisso e selecione os comandos DeleteRow, EditRow, BrowseRow e NewRow, como mostra a Figura 23.14.

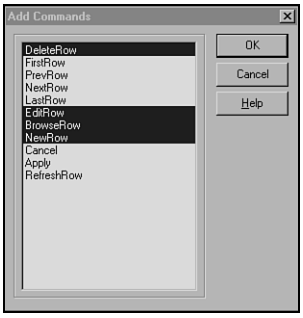

Figura 23.14 Use a caixa de diálogo Add Commands para selecionar as ações que você deseja fazer sobre as linhas individuais do dataset.

Dê um clique em OK. Depois, empilhe verticalmente os botões, definindo a propriedade `DisplayColumns` do componente `AdapterCommandColumn` como 1. Depois de fazer isso, você deverá ver uma coleção de botões de comando no Web Designer (veja a Figura 23.15).

Atualmente, esses botões precisam fazer algo, e também precisarão de uma página para exibir o registro individual. Inclua outra página no projeto, com um `TAdapterPageProducer`, e exija que o usuário efetue login para ver a página. Chame a página de BioLifeEdit e salve a unidade como `wmBioLifeEdit`. Inclua `wdmBioLife` na cláusula `uses`, para que você possa acessar os dados.

Dê um clique duplo no `TAdapterPageProducer`, no novo módulo Web, e inclua um `AdapterForm`. Depois, inclua uma `AdapterErrorList`, um `AdapterFieldGroup` e um `AdapterCommandGroup`. Dê um clique com o botão direito no `AdapterFieldGroup` e inclua todos os campos e depois todos os comandos no `AdapterCommandGroup`. O Web Surface Designer é semelhante ao que aparece na Figura 23.16.

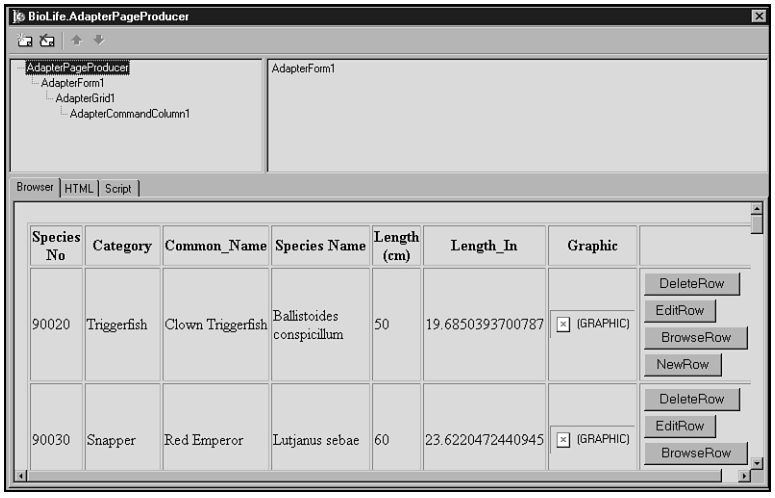

Figura 23.15 A aplicação de demonstração exibindo os botões de ação.

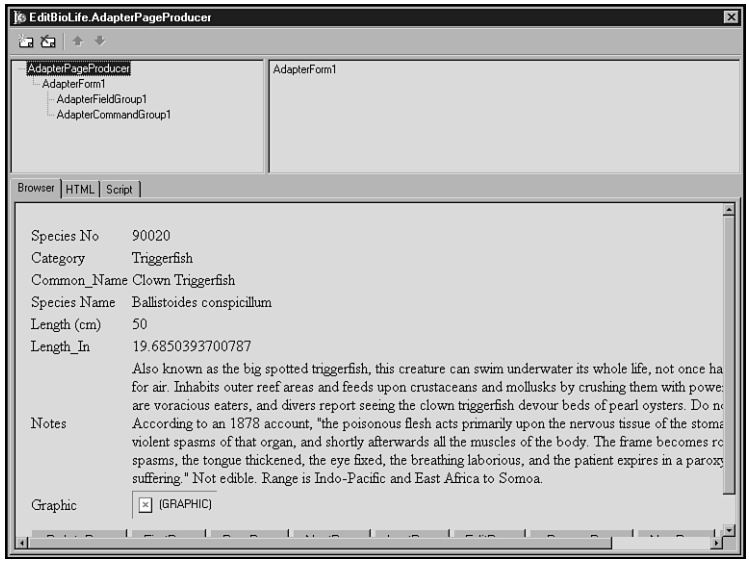

Figura 23.16 A página BioLifeEdit com todos os campos e ações adicionados ao Web Surface Designer.

Agora, para usar essa página a fim de editar um único registro, volte à unidade `wmBioLife` onde a grade se encontra e use a Object TreeView e a tecla Shift para selecionar todos os quatro botões na `AdapterCommandColumn`. Defina sua propriedade de página como EditBioLife – o nome da página que exibirá um único registro. Agora, quando você der um clique no botão na grade, a página EditBioLife será exibida. Se você pedir para navegar pelo registro, os dados serão exibidos como texto simples. Mas, se você pedir para editar o registro, eles aparecerão em caixas de edição. O campo gráfico até mesmo permitirá que você inclua um novo gráfico no banco de dados, procurando um novo arquivo. Você pode navegar pelo dataset usando os botões de comando. Novamente, tudo isso foi conseguido sem a escrita de uma única linha de código – ou script, tanto faz.

NOTA

Você pode querer mexer um pouco na apresentação do campo Notes. Por default, o controle TextArea, usado quando a página está no modo de edição, é muito pequeno e não quebra o texto. Você pode selecionar o componente `FldNotes` e ajustar as propriedades `TextAreaWrap`, `DisplayRows` e `DisplayWidth` para obter melhores resultados.

Convertendo a aplicação em uma DLL ISAPI

Sua aplicação até aqui tem sido executada sob o Web App Debugger, facilitando assim a depuração e o teste. No entanto, você certamente não deseja distribuir a aplicação dessa forma. Em vez disso, você provavelmente desejará que a aplicação seja uma DLL ISAPI, para que sempre resida na memória e mantenha toda a informação de sessão necessária para manter as coisas em ordem.

A conversão da sua aplicação de um servidor baseado no Web App Debugger para um servidor ISAPI é muito simples. Basta criar um novo projeto em branco, baseado no ISAPI, e remover todas as unidades dele. Depois, inclua todas as unidades da versão da aplicação Web, exceto o formulário. Após, compile e execute a aplicação. É realmente simples. De fato, você pode manter dois projetos que usam os mesmos módulos Web – um projeto para teste e outro para distribuição. A maior parte das aplicações de demonstração no diretório WebSnap faz isso, e a aplicação de demonstração neste CD-ROM possui projetos para o Web App Server e ISAPI. Ao distribuir uma nova DLL ISAPI, não se esqueça de incluir, no mesmo diretório da DLL, quaisquer arquivos HTML que serão necessários.

Tópicos avançados

Até aqui, você viu o que pode ser considerado o básico. Você criou uma aplicação WebSnap que gerencia usuários e informações de sessão sobre esses usuários, e que também gerencia e manipula dados. Entretanto, o WebSnap faz muito mais do que isso, dando-lhe mais controle sobre o que a aplicação pode fazer. A próxima seção aborda alguns dos tópicos avançados que permitirão ajustar mais de perto suas aplicações WebSnap.

LocateFileServices

O desenvolvimento de aplicações WebSnap para Windows normalmente exige a coordenação de diferentes recursos. HTML, script no servidor, código Delphi, acesso a banco de dados e gráficos precisam ser unidos corretamente a uma única aplicação. Normalmente, muitos desses recursos encontram-se embutidos e são gerenciados por meio de um arquivo HTML. O WebSnap oferece suporte para separar a HTML da implementação de uma página Web dinâmica, significando que você pode editar os arquivos HTML separadamente do arquivo binário da aplicação Web. No entanto, por default, essa HTML precisa residir em arquivos no mesmo local do arquivo binário. Isso nem sempre é conveniente ou possível, e pode haver ocasiões em que você deseja que a HTML resida em locais longe do seu binário. Ou então você pode querer obter conteúdo HTML de origens que não sejam arquivos, digamos, de um banco de dados.

O WebSnap oferece a capacidade de obter HTML de qualquer origem que você queira. O componente `LocateFileService` permite apanhar HTML de qualquer local de arquivo, arquivos `include` ou qualquer descendente de `TStream`. Ser capaz de acessar a HTML por um `TStream` significa que você pode apanhar a HTML de qualquer origem, desde que ela possa ser colocada em um `TStream`.

Por exemplo, a HTML pode ser colocada em um stream a partir de um arquivo RES embutido no arquivo binário na sua aplicação. A aplicação de demonstração pode mostrar como isso é feito. Naturalmente, você precisará de alguma HTML para embutir. Usando um editor de textos ou o seu editor de HTML favorito, apanhe o arquivo wmLogin.html como modelo e salve-o no diretório da sua aplicação de demonstração como `embed.html`. Depois, inclua algum texto no arquivo para notar que o arquivo está embutido no arquivo RES. Desse modo, você saberá com certeza que possui o arquivo correto quando ele for exibido.

Depois, naturalmente, você precisa embutir essa HTML na sua aplicação. O Delphi gerencia isso com facilidade por meio de arquivos RC, compilando-os automaticamente e acrescentando-os em uma aplicação. Portanto, use o Bloco de Notas ou alguma ferramenta de edição de textos para criar um arquivo de texto e chame-o de `HTML.RC`. Salve-o no mesmo diretório da sua aplicação de demonstração e acrescente-o ao seu projeto. Depois, inclua este testo ao arquivo RC:

```
#define HTML 23  // Identificador de recurso HTML
EMBEDDEDHTML HTML embed.html
```

Quando incluído em um projeto Delphi, este compilará um arquivo RC em um arquivo RES e inclua-o na sua aplicação.

Quando a HTML estiver na sua aplicação, crie uma nova página com um `TPageProducer` e chame-a de Embedded. Salve o arquivo como `wmEmbedded`. Depois, vá para a página Home e selecione o componente `LocateFileServices`. Vá para a página Object Inspector Events e dê um clique duplo no evento `OnFindStream`. Você obterá um tratador de eventos semelhante a este:

```
procedure THome.LocateFileServiceFindStream(ASender: TObject;
  AComponent: TComponent; const AFileName: String;
  var AFoundStream: TStream; var AOwned, AHandled: Boolean);

begin

end;
```

Os principais parâmetros aqui são AFileName e AFoundStream. Você os usará para apanhar a HTML dos recursos embutidos. Faça com que seu tratador de evento se torne semelhante a este:

```
procedure THome.LocateFileServiceFindStream(ASender: TObject;
  AComponent: TComponent; const AFileName: String;
  var AFoundStream: TStream; var AOwned, AHandled: Boolean);
begin
  // estamos procurando o arquivo embutido
  if Pos('EMBEDDED', UpperCase(AFileName)) > 0 then  begin
    AFoundStream := TResourceStream.Create(hInstance, 'EMBEDDED', 'HTML');
    AHandled := True; // não precisa procurar mais
  end;
end;
```

AFileName será o nome não-qualificado do arquivo HTML que o Delphi usaria como default. Você pode usar esse nome para determinar qual recurso irá pesquisar. AFoundStream será nil quando passado para o tratador de evento, de modo que ficará por sua conta a criação de um stream usando a variável. Nesse caso, AFoundStream torna-se um TResourceStream, que apanha a HTML dos recursos no executável. A definição de AHandled como True garante que LocateFileServices não se esforçará mais para localizar o conteúdo HTML.

Execute a aplicação e você verá sua HTML aparecendo quando exibir a página Embedded.

Uploading de arquivo

No passado, uma das tarefas mais desafiadoras para um desenvolvedor de aplicação Web era fazer o uploading de arquivos do cliente para o servidor. Isso normalmente envolvia recursos bastante arcaicos da especificação HTTP e a contagem cuidadosa de cada byte passado. Como você pode imaginar, o WebSnap torna essa tarefa anteriormente difícil em algo fácil. O WebSnap oferece toda a funcionalidade para o uploading de arquivo dentro de um TAdapter, e sua parte não é muito mais difícil do que colocar um arquivo em um stream.

Como sempre, crie outra página na sua aplicação, que fará o upload de arquivos para o servidor a partir do cliente. Chame a página de Upload e dê-lhe um TAdapterPageProducer. Depois, salve o arquivo como wmUpload. Em seguida, inclua um TAdapter no formulário. Dê ao TAdapter um novo AdapterFileField. Esse campo controlará todo o uploading dos arquivos selecionados no cliente. Além disso, dê ao Adapter uma única ação e chame-a de UploadAction.

Em seguida, dê ao AdapterPageProducer um AdapterForm com uma AdapterErrorList, um AdapterFieldGroup e um AdapterCommandGroup. Conecte os dois primeiros a Adapter1 e o AdapterCommandGroup ao AdapterFieldGroup. Depois, acrescente todos os campos ao AdapterFieldGroup e todas as ações ao AdapterCommandGroup. Mude a legenda no botão para Upload File. A Figura 23.17 mostra o que você deverá ver no Surface Designer.

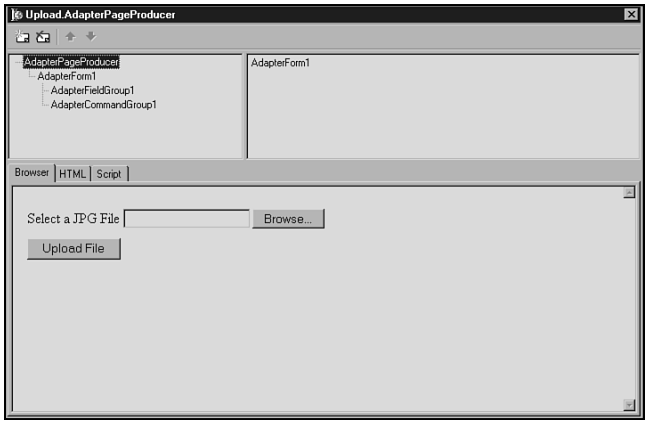

Figura 23.17 O Web Surface Designer para a página Upload, com o botão Browse (procurar) acrescentado automaticamente.

O código pode ser incluído em dois lugares. O primeiro deles é no tratador de evento `Adapter1.AdapterFileFieldOnFileUpload`. Esse código deverá ser semelhante ao da Listagem 23.5.

Listagem 23.5 Tratador de evento `OnFileUpload`

```
procedure TUpload.AdapterFileField1UploadFiles(Sender: TObject;
  Files: TUpdateFileList);
var
  i: integer;
  CurrentDir: string;
  Filename: string;
  FS: TFileStream;
begin
  // Uploading do arquivo aqui
    if Files.Count <= 0 then
    begin
      raise Exception.Create('You have not selected any files to be uploaded');
    end;
    for i := 0 to Files.Count – 1 do
    begin
    // Cuida para que o arquivo seja um .jpg ou .jpeg
    if (CompareText(ExtractFileExt(Files.Files[I].FileName), '.jpg') < > 0)
        and (CompareText(ExtractFileExt(Files.Files[I].FileName), '.jpeg')
        ➥< > 0) then
    begin
      Adapter1.Errors.AddError('You must select a JPG or JPEG file to upload');
    end else
    begin
      CurrentDir := ExtractFilePath(GetModuleName(HInstance)) + 'JPEGFiles';
      ForceDirectories(CurrentDir);
      FileName := CurrentDir + '\' + ExtractFileName(Files.Files[I].FileName);
      FS := TFileStream.Create(Filename, fmCreate or fmShareDenyWrite);
      try
        FS.CopyFrom(Files.Files[I].Stream, 0); // 0 = copia tudo do início
      finally
        FS.Free;
      end;
    end;
  end;
end;
```

Esse código primeiro verifica se você selecionou um arquivo e depois certifica-se de que você selecionou um arquivo JPEG. Depois de determinar que você fez isso, ele apanha o nome do arquivo, garante que o diretório receptor existe e coloca o arquivo em um `TFileStream`. O trabalho real aqui é feito nos bastidores pela classe `TUpdateFileList` que gerencia todo o tratamento de formulário esotérico e em várias partes do HTTP, necessário para o uploading de um arquivo do cliente para o servidor.

O segundo lugar para incluir código é no tratador `OnExecute` para `UploadAction` em `Adapter1`. Ele é o seguinte:

```
procedure TUpload.UploadActionExecute(Sender: TObject; Params: TStrings);
begin
  Adapter1.UpdateRecords;
end;
```

 que simplesmente diz ao `Adapter` para atualizar seus registros e apanhar os arquivos que foram solicitados.

Incluindo modelos personalizados

Uma coisa que você provavelmente notou é que, quando uma nova página é criada com o New Page Wizard, você tem apenas duas opções para a HTML na sua aplicação – o modelo padrão ou um modelo em branco. O modelo padrão é bom para coisas como a aplicação de demonstração neste capítulo. Porém, quando você começa a desenvolver sites mais sofisticados, desejará ser capaz de incluir automaticamente seus próprios modelos HTML quando incluir páginas em suas aplicações. O WebSnap também permite fazer isso.

Você pode incluir novos modelos nas seleções do New Page Wizard criando e registrando um descendente de `TProducerTemplatesList` em um pacote durante o projeto. Existe um pacote de demonstração que faz isso no diretório `<Delphi>\Demos\WebSnap\Producer Template`. Você pode examinar esse pacote e incluir seus próprios modelos HTML/script no arquivo RC incluído no pacote. Observe que, para que esse pacote seja compilado, primeiro é preciso ter compilado o pacote `<Delphi>\Demos\WebSnap\Util\TemplateRes.dpk`. Depois que você compilar e instalar esses pacotes, terá ainda mais modelos para escolher no New Page Wizard.

Componentes personalizados em `TAdapterPageProducer`

Grande parte do trabalho de exibição da HTML no decorrer deste capítulo tem sido feita pelos componentes `TAdapterPageProducer`, e os componentes que estão embutidos dentro dele. No entanto, você certamente desejará personalizar a HTML lá existente além do código padrão que você viu até o momento. O WebSnap permite fazer isso, criando seus próprios componentes, que se encaixam no `TAdapterPageProducer`, para que você possa acrescentar sua própria HTML personalizada na mistura.

Seus componentes `TAdapterPageProducer` personalizados precisam descender de `TWebContainedComponent` e implementar a interface `IWebContent`. Como todos os componentes precisam fazer isso, essa é uma oportunidade perfeita para usar uma classe abstrata, como na Listagem 23.6.

Listagem 23.6 Classe abstrata descendente de `TWebContainedComponent`

```
type

Tddg6BaseWebSnapComponent =class(TWebContainedComponent,IWebContent)
  protected
    { IWebContent }
    function Content(Options: TWebContentOptions; ParentLayout: TLayout):
    ➥string;
    function GetHTML: string; virtual; abstract;
  end;
```

Essa classe é implementada da seguinte forma:

```
function Tddg6BaseWebSnapComponent.Content(Options: TWebContentOptions;
  ParentLayout: TLayout): string;
var
  Intf: ILayoutWebContent;
begin
  if Supports(ParentLayout, ILayoutWebContent, Intf) then
    Result := Intf.LayoutField(GetHTML, nil)
  else
    Result := GetHTML;
end;
```

A classe abstrata implementa a função `Content` somente porque a função `GetHTML` é declarada como abstrata. A função `Content` basicamente verifica se o componente que a contém é um `Layout-`

Group. Se for LayoutGroup, a função Content coloca seu conteúdo dentro do LayoutGroup. Caso contrário, Content simplesmente retorna os resultados de GetHTML. Os componentes descendentes, portanto, só precisam implementar a função GetHTML, retornando o código HTML apropriado, e podem ser registrados para trabalhar dentro de um TAdapterPageProducer.

O código neste CD-ROM implementa dois componentes que permitem incluir conteúdo HTML em um TAdapterPageProducer, seja como uma string ou como um arquivo. O código para o componente Tddg6HTMLCode aparece na Listagem 23.7.

Listagem 23.7 Componente **Tddg6HTMLCode**

```
  Tddg6HTMLCode = class(Tddg6BaseWebSnapComponent)
  private
    FHTML: TStrings;
    procedure SetHTML(const Value: TStrings);
  protected
    function GetHTML: string; override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property HTML: TStrings read FHTML write SetHTML;
  end;

constructor Tddg6HTMLCode.Create(AOwner: TComponent);
begin
  inherited;
  FHTML := TStringList.Create;
end;

destructor Tddg6HTMLCode.Destroy;
begin
  FHTML.Free;
  inherited;
end;

function Tddg6HTMLCode.GetHTML: string;
begin
  Result := FHTML.Text;
end;

procedure Tddg6HTMLCode.SetHTML(const Value: TStrings);
begin
  FHTML.Assign(Value);
end;
```

Essa é uma classe muito simples. Ela simplesmente oferece uma propriedade publicada do tipo TString, que apanhará qualquer código HTML e depois o colocará no TAdapterPageProducer como se encontra. A função GetHTML simplesmente retorna a HTML em formato de string. Você pode criar componentes para retornar qualquer código HTML que queira incluir – imagens, links, arquivos e outro conteúdo. Tudo o que os componentes descendentes precisam fazer é oferecer seu conteúdo HTML em um método GetHTML( ) modificado. Observe que existem funções de registro de suporte na unidade onde os componentes são implementados. Ao criar componentes, certifique-se de registrá-los na sua unidade, semelhante aos que estão neste CD-ROM. Para usar esses componentes, basta instalá-los em um pacote durante o projeto e os componentes aparecerão no Web Surface Designer do TAdapterPageProducer (veja a Figura 23.18).

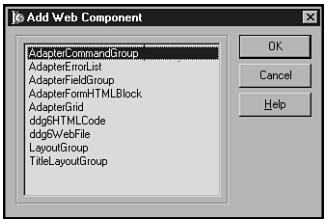

Figura 23.18 Componentes **`TAdapterPageProducer`** no Web Surface Designer.

Resumo

Essa foi uma visão geral rápida do poder do WebSnap. Este capítulo apenas arranhou a superfície do que o WebSnap pode fazer. Não deixe de verificar as diversas aplicações de demonstração no diretório `<Delphi>\Demos\WebSnap`. Muitas delas dão mais funcionalidade ao estado padrão dos componentes WebSnap.

Logicamente, o WebSnap é uma tecnologia poderosa, mas é preciso que você faça algum esforço para entendê-la. Entretanto, depois que você passar pela curva de aprendizado inicial, logo estará criando, com facilidade, sites Web poderosos, dinâmicos e controlados por banco de dados.

Desenvolvimento para dispositivo sem fio

CAPÍTULO

24

NESTE CAPÍTULO

- **Evolução do desenvolvimento – como chegamos até aqui?**
- **Dispositivos portáteis sem fio**
- **Tecnologias de rádio**
- **Tecnologias de dados sem fio baseadas em servidor**
- **Experiência do usuário sem fio**

Sem sombra de dúvida, duas tecnologias nos últimos 10 anos tocaram nossas vidas mais do que quaisquer outras: a Internet e os dispositivos portáteis. Apesar dos altos e baixos dos negócios baseados na Internet, esta – e, em particular, a Web – tem mudado nossas vidas permanentemente. Ela tem afetado o modo como nos comunicamos, compramos, trabalhamos e brincamos. Mais que isso, muitos de nós agora sentem arrepios ao pensar que poderia estar em algum lugar sem nosso confiável celular (ou telefone portátil) ou Personal Digital Assistant (PDA). Devido à sua importância em nossas vidas, parece natural que essas duas tecnologias agora estejam em um estado de convergência, com os celulares se tornando tentáculos sem fio que saem da Internet com fio para nos oferecer os serviços e as informações às quais nos tornamos viciados.

Com dispositivos de informação portáteis tornando-se mais uma necessidade do que uma novidade no clima comercial e social de hoje, nós, os desenvolvedores encaramos o desafio de aproveitar esse hardware e infra-estrutura para atender à demanda cada vez maior para colocar dados e aplicações nos dispositivos portáteis. Com uma gama incrível de dispositivos portáteis, redes e tecnologias no mercado, as principais perguntas para os desenvolvedores se tornam: Qual das plataformas sem fio você deve focalizar? Qual é o modo mais eficaz de focalizá-las? Que tecnologias posso aproveitar para fazer com que meus dados e aplicações se adaptem a dispositivos portáteis? Quais são os prós e os contras de todas essas plataformas e tecnologias?

Este capítulo, de forma alguma, tem como objetivo servir como um guia completo, descrevendo como implementar todas as diversas tecnologias portáteis. Isso exigiria muitos volumes. No entanto, estaríamos com o dever cumprido se você conseguisse duas coisas deste capítulo. Primeiro, você conseguirá usar este capítulo como uma introdução para entender o papel que muitos dos vários tipos de hardware, software e tecnologias desempenham na computação móvel, do ponto de vista de um desenvolvedor. Segundo, você deverá entender como algumas dessas tecnologias portáteis podem ser implementadas com o Delphi.

Evolução do desenvolvimento – como chegamos até aqui?

Antes de discutirmos como você poderia montar as aplicações para aproveitar essas tendências emergentes na tecnologia da informação, é importante voltar e ver o que nos trouxe aqui. Veja uma abordagem um tanto simplificada das tendências recentes em tecnologia de informação.

Antes da década de 1980: aqui havia dragões

Antes que a revolução do PC nos anos 1980 trouxesse a tecnologia da informação às massas, o desenvolvimento desses sistemas era uma mistura de mainframes, terminais e sistemas patenteados. As ferramentas para desenvolvedor geralmente eram rudimentares, tornando o desenvolvimento de aplicações um processo dispendioso e demorado, reservado para verdadeiros feras.

Final da década de 1980: aplicações de banco de dados em desktop

Depois que a revolução do PC pegou, as pessoas começaram a aproveitar o novo poder encontrado, residindo em suas mesas, usando aplicações de banco de dados para desktop, como dBASE, FoxPro e Paradox. As ferramentas de desenvolvimento de aplicação em geral também se tornaram mais maduras, tornando o desenvolvimento de aplicações uma tarefa relativamente simples, usando linguagens de terceira geração, como C, Pascal e BASIC. DOS era o rei do desktop, oferecendo aplicações com uma plataforma de montagem comum. As redes locais estavam começando a se tornar práticas para empresas de todos os tamanhos, fornecendo armazenamento centralizado de dados em servidores de arquivos.

Início da década de 1990: cliente/servidor

As redes corporativas agora já eram aceitas sem questionamento; quase todos no escritório estavam conectados. A questão era como unir o intervalo entre os sistemas de mainframe envelhecendo e os

banco de dados de desktop não-escaláveis, ambos importantes para os negócios. A resposta estava em sistemas cliente/servidor, a noção de banco de dados poderosos de empresas como Oracle, Sybase e Informix, conectados a interfaces com o usuário rodando em PCs. Isso permitiu que os sistemas aproveitassem o poder em cada desktop enquanto permitiram que os servidores de banco de dados realizassem suas tarefas especializadas. Ferramentas de desenvolvimento de quarta geração, como Visual Basic e Delphi, facilitam o desenvolvimento mais do que nunca, e o suporte para o banco de dados estava embutido como um cidadão de primeira classe das ferramentas.

Final da década de 1990: transações baseadas em multicamadas e Internet

O principal problema com o modelo cliente/servidor é a noção do local onde a lógica comercial deveria residir – coloque-a no servidor e você limita a escalabilidade; coloque-a no cliente e você terá um pesadelo de manutenção. Os sistemas multicamadas resolveram esse problema, colocando a lógica comercial em uma ou mais *camadas* adicionais, separadas lógica e/ou fisicamente do cliente e do servidor. Isso permitiu que sistemas escritos corretamente fossem utilizados até uma extensão quase ilimitada, pavimentando a estrada para as transações complexas que atenderiam a milhares ou milhões de clientes por meio da Internet. As ferramentas de desenvolvimento se estenderam para o mundo multicamadas, com tecnologias como CORBA, EJB e COM. As empresas rapidamente aproveitaram a Internet para oferecer informações e serviços a funcionários, clientes e parceiros, e as indústrias cresceram em torno da capacidade de gerenciar, publicar e trocar dados entre máquinas pela Internet.

Início da década de 2000: a infra-estrutura da aplicação se estende para dispositivos portáteis sem fio

Portanto, qual é o resultado final da grande disponibilidade de informações fornecida pela Internet? A resposta pode ser resumida nestas palavras: apego à informação. A disponibilidade de informações e serviços pela Internet nos tornou dependentes de aspectos cada vez maiores de nossas vidas. PDAs e telefones portáteis serviram para aguçar nosso desejo, alimentando nosso apego às informações enquanto estamos longe de nossas mesas. Os servidores de aplicação e as ferramentas de desenvolvimento estão crescendo em escopo para gerenciar a investida de funcionalidade para esses tipos de dispositivos. O mercado em potencial para aplicações em dispositivos portáteis é imenso em tamanho, pois o número projetado desses dispositivos chegando ao mercado nos próximos anos ultrapassa as previsões para PCs.

Dispositivos portáteis sem fio

Entre os telefones portáteis, PDAs e pagers inteligentes, não há escassez de dispositivos para escolher, caso você queira permanecer conectado enquanto está fora de sua mesa. Aqueles que têm problemas para escolher às vezes carregam todos os três no cinto, fazendo com que se sintam cada vez mais como Batman nos dias atuais. Também estamos vendo uma convergência desses dispositivos em dispositivos únicos e multifuncionais. Os exemplos recentes disso incluem o módulo Springboard de telefone portátil para handhelds Handspring, o Smartphone Kyocera rodando PalmOS e o telefone portátil Stinger da Microsoft, controlado pelo Windows CE. Nesta seção, citaremos alguns dos líderes nesse setor.

Telefones portáteis

Os telefones portáteis, sem dúvida, são o tipo mais difundido de dispositivo portátil sem fio. Os telefones portáteis ultrapassaram o âmbito dos sistemas de comunicação puramente para vez e entraram para a comunicação de dados. A grande maioria dos novos telefones que chegam ao mercado aceita mensagens de texto usando o Short Message Service (SMS) e navegação tipo Web, usando o

Wireless Application Protocol (WAP). As taxas de dados atuais ainda são baixas, entre 9,6 e 14,4k, mas novas tecnologias prometem oferecer velocidades de até 2 Mbits dentro de 2 a 3 anos.

Dispositivos PalmOS

Dispositivos rodando o sistema operacional PalmOS, da Palm Computing, têm liderado o mercado no espaço do PDA há vários anos. Alguns dispositivos PalmOS possuem capacidade de sem fio embutida (como a série Palm VII ou o Smartphone da Kyocera), e a capacidade sem fio pode ser acrescentada a outros, por meio de um modem sem fio (como aqueles fabricados pela Novatel) ou por um conector para telefone portátil, disponível na Palm. Diversas empresas licenciaram o PalmOS, da Palm, Inc., para inclusão em seus próprios dispositivos, incluindo Handspring, Sony, Kyocera, Symbol, Nokia, Samsung e TRG. Entre as vantagens do PalmOS está o fato de que eles possuem uma fatia esmagadora do mercado de PDAs e existe uma forte comunidade de desenvolvimento com um ativo mercado de terceiros.

Pocket PC

Compaq, HP, Casio e outros fabricantes produzem PDAs baseados no sistema operacional Pocket PC (anteriormente Windows CE) da Microsoft. Até agora, nenhum desses dispositivos possui capacidade interna para uso sem fio, mas eles oferecem suporte para modems sem fio de modo semelhante aos dispositivos PalmOS. Mais ainda, dispositivos Pocket PC costumam ser um pouco mais poderosos do que seus correspondentes PalmOS, e alguns têm a capacidade de aceitar PC Cards (PCMCIA) padrão. Isso tem o potencial de permitir uma faixa de expansão ainda mais extensa para redes sem fio com maior largura de banda.

RIM BlackBerry

A BlackBerry oferece a funcionalidade tipo PDA no tamanho de um pager. Com um modem sem fio interno e um teclado, o BlackBerry é especialmente adequado para tarefas de e-mail portátil. No entanto, o BlackBerry também oferece suporte para navegação Web por meio de um browser de terceiros. Descobri que o BlackBerry é uma plataforma incrível para e-mail corporativo, graças à integração interna com MS Exchange ou Lotus Domino, mas o dispositivo pretende se tornar um utensílio para Web, por causa do tamanho de sua tela e suas capacidades de navegação.

Tecnologias de rádio

As tecnologias de rádio oferecem a conexão entre os dispositivos portáteis e a Internet ou a rede local da empresa.

GSM, CDMA e TDMA

Essas são as principais tecnologias usadas como transporte para telefones portáteis, e normalmente são referenciadas como *2G*, pois incorporam a segunda geração de redes de comunicações portáteis (*1G* sendo o servidor analógico). A maioria das redes nos Estados Unidos é baseada em CDMA ou TDMA, enquanto a maior parte do restante do mundo utiliza GSM. Os detalhes dessas tecnologias não são tão importantes para um desenvolvedor de software, exceto por saber que a própria existência desses padrões competidores dificulta a criação de aplicações que funcionam entre o espectro de telefones e redes. Em geral, as velocidades de dados nesses tipos de redes atingem de 9,6 a 14,4k.

CDPD

Cellular Digital Packet Data (CDPD) é uma tecnologia que permite a transferência de dados em pacotes através de redes sem fio, oferecendo um aumento na largura de banda e a funcionalidade de

estar "sempre ligada". CDPD é comum com o serviço de PDA sem fio nos Estados Unidos, como aquele oferecido pela GoAmerica ou OmniSky, com velocidades chegando a 19,2k.

3G

Redes portáteis 3G, ou de terceira geração, são criadas desde o princípio para lidar com diversos tipos diferentes de streams de mídia e ostentam uma largura de banda estimada como algo na faixa de 384k-2M. Os candidatos mais prováveis para serem os suportes do padrão 3G são tecnologias conhecidas como EDGE e UMTS. No entanto, embora exista a tecnologia e várias operadoras possuam espectro suficiente para implementar redes 3G, parece que nenhuma deseja ser a primeira a fazer o investimento multibilionário em upgrades de rede a fim de seguir adiante com o 3G.

GPRS

General Packet Radio Service (GPRS) é considerado como o caminho de migração do 2G para o 3G, e normalmente é considerado como 2.5G. GPRS permite que o tráfego baseado em pacote através da infra-estrutura 2G existente, apenas com upgrades relativamente pequenos. O throughput observado nas redes GPRS provavelmente estará na faixa de 20 a 30k.

Bluetooth

Dispositivos incorporando a tecnologia de rádio Bluetooth só agora estão começando a aparecer no mercado. Bluetooth é uma importante tecnologia emergente, pois permite o uso de rede ocasional em onda curta entre diferentes tipos de dispositivos. Como os módulos de rádio Bluetooth são muito pequenos, consomem pouca energia e são relativamente baratos, eles estarão embutidos em todos os tipos de dispositivos portáteis, incluindo telefones, PDAs, laptops e assim por diante. A maioria dos rádios Bluetooth terá um alcance de cerca de 10 metros e desfrutará de aproximadamente 700k de largura de banda. As aplicações em potencial para o Bluetooth incluem o sincronismo de dados entre PDA e componente, quando estiverem nas proximidades um do outro, ou o fornecimento de conectividade com Internet para um laptop, por meio de um telefone portátil no bolso de uma pessoa. Um novo termo, Personal Area Network (PAN), é usado para descrever essa noção de pequena rede sem fio, onde todos os nossos dispositivos portáteis pessoais se comunicam regularmente uns com os outros.

É mais correto pensar no Bluetooth como um substituto para os cabos serial, USB ou IEEE 1394 do que como uma tecnologia de rede tipo Ethernet. A versão atual do Bluetooth oferece suporte apenas para um dispositivo mestre controlando um máximo de sete dispositivos escravos simultâneos.

802.11

Embora a tecnologia Bluetooth seja projetada como uma tecnologia de rede pessoal de curto alcance, o 802.11 foi criado visando o uso em redes locais. A geração atual dessa tecnologia, 802.11b ou *WiFi*, oferece até 11Mb de largura de banda, com uma versão de 45MB conhecida como 802.11a no horizonte. 802.11 possui um alcance de cerca de 30 metros, com alcances maiores sendo possíveis com antenas especiais. Os requisitos de energia do 802.11 são maiores do que os do Bluetooth, e os dispositivos possuem maior tamanho; o dispositivo de rádio pode caber dentro de um PC Card padrão, o que é ótimo para laptops, mas não é conveniente para telefones ou para a maioria dos PDAs.

Uma observação importante que se deve ter em mente é que o Bluetooth e o 802.11 compartilham o mesmo espectro de 2,4 GHz, de modo que é possível que os dois interfiram um com o outro quando ocupam o mesmo espaço. Embora não seja provável que eles congelem completamente um ao outro, visto que ambos usam tecnologia de espectro amplo para pular de freqüências várias vezes por segundo, é provável que o desempenho sofra em uma ou ambas as conexões, devido à interferência mútua.

Tecnologias de dados sem fio baseadas em servidor

As tecnologias de dados sem fio aproveitam a tecnologia de rádio para oferecer dados e serviços a dispositivos portáteis. Essas tecnologias envolvem servidores gerando conteúdo, que é enviado sem fio para os clientes e interpretado pelo software embutido no cliente.

SMS

A tecnologia Short Message Service (SMS) é usada para enviar mensagens de texto curtas (geralmente de 100 a 160 caracteres no máximo) para telefones portáteis. Fora o tamanho limitado para a mensagem, a tecnologia SMS é limitada devido a questões de interoperabilidade entre as operadoras de rede e protocolos SMS variados, empregados pelos operadoras. No entanto, SMS está se tornando bastante popular – principalmente na Europa – devido à sua facilidade de uso e grande disponibilidade.

Como cada operadora pode empregar pequenas variações ao tema SMS, as técnicas para se desenvolver aplicações que oferecem suporte ao SMS podem variar, dependendo da operadora que você esteja visando. Embora o GSM tenha vantagem em relação a outras redes de telefone móvel porque o suporte para SMS está embutido no padrão GSM, do ponto de vista de um desenvolvimento de aplicação, ainda pode ser algo desafiador o envio de mensagens SMS de um servidor conectado à Internet para um cliente portátil. Isso porque você precisa trabalhar com servidores SMS no lado da operadora, o que poderia envolver um suporte para padrões variados e até mesmo taxas de licenciamento.

Recomendamos um dentre dois caminhos para incorporar o suporte a SMS nos servidores. A primeira opção é simplesmente usar e-mail; a maioria das operadoras oferece suporte para o envio de uma mensagem SMS, enviando uma mensagem de e-mail para um endereço de e-mail específico, que contém o número do telefone do destinatário. Embora essa seja uma técnica relativamente simples, do ponto de vista técnico, a desvantagem é que o suporte não é universal e acrescenta outra camada de falha em potencial. A segunda opção é comprar qualquer uma das muitas ferramentas de terceiros que cuidam do envio de mensagens SMS por diversos tipos de redes. Essa é a técnica preferida, embora envolva alguns custos iniciais e/ou taxas de licenciamento.

WAP

Wireless Application Protocol (WAP) foi estabelecido como meio padrão para acessar informações da Internet por meio de um dispositivo portátil. A aceitação geral do WAP no mercado tem sido dividida. Por um lado, WAP tem sido bem recebido por operadores de rede e fabricantes de telefone, pois foi projetado desde o início para trabalhar com qualquer serviço sem fio e padrão de rede em praticamente qualquer dispositivo. No entanto, a experiência com WAP da parte do usuário não tem sido totalmente positiva, devido a limitações no tamanho da tela, em suas capacidades de entrada de dados e na largura de banda sem fio. Além do mais, como os sites WAP geram receita limitada com base no uso, não existe um incentivo comercial forte para o desenvolvimento de sites WAP de alta qualidade. O conteúdo para sistemas WAP foi desenvolvido em uma linguagem baseada em XML, conhecida como Wireless Markup Language (WML) – linguagem de marcação sem fio.

A arquitetura típica da aplicação WAP é ilustrada na Figura 24.1.

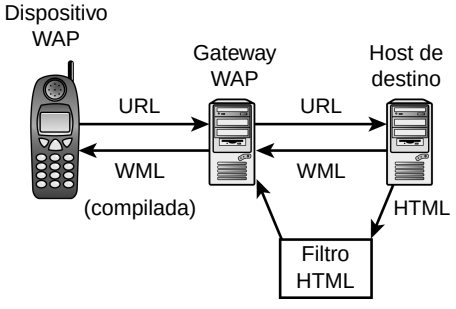

Figura 24.1 Arquitetura da aplicação WAP.

O dispositivo portátil, normalmente um telefone, possui uma parte de software residente, conhecida como *microbrowser*. Como o nome indica, esse software é semelhante a um browser Web, mas específico para dispositivos como telefones portáteis, com memória e processamento limitados. A maioria dos telefones portáteis no mercado atualmente utiliza o microbrowser da OpenWave (anteriormente, Phone.com). Além disso, o microbrowser normalmente é projetado para representar as linguagens WML ou HDML, em vez da HTML, conforme descrevemos na próxima seção.

Como a geração atual de telefones portáteis não sabe inerentemente como se comunicar com recursos na Internet, o gateway WAP atua como intermediário entre o dispositivo móvel e a Internet pública. A maioria dos gateways WAP é gerenciada pelo provedor de serviço sem fio e executa um software criado por empresas como OpenWave, Nokia ou SAS.

O host de destino geralmente é apenas um servidor Web comum, que simplesmente retorna conteúdo formatado corretamente para WAP. Formatação correta significa que o conteúdo é descrito usando WML ou, em último caso, porém não-ideal, empregando um filtro para converter o conteúdo HTML dinamicamente para WML.

O principal benefício do WAP é o amplo suporte para praticamente todos os dispositivos portáteis e sem fio. Mais do que isso, a funcionalidade disponível no WAP essencialmente é o denominador comum dos dispositivos portáteis, ganhando no quesito de compatibilidade e perdendo em funcionalidade. Além do mais, entre o servidor de aplicação, servidor Web, gateway WAP, microbrowser e dispositivo cliente, os desenvolvedores do WAP precisam se preocupar muito em seus esforços para criar aplicações que funcionem corretamente para o maior número de usuários finais.

As principais desvantagens do WAP incluem tamanho de tela limitado e menos capacidade de processamento nos dispositivos, a falta de um teclado completo para entrada de dados, as velocidades de download muito baixas e o fato de que a transmissão sem fio para aplicações WAP ainda pode ser muito cara.

WML: a linguagem do WAP

Como já dissemos, as informações são trocadas em WAP usando a linguagem de marcação sem fio (WML). WML, de certa forma, é modelada a partir da HTML, mas ela possui duas coisas que a distinguem da HTML. Primeiro, ela é composta de um conjunto relativamente pequeno de tags e atributos, tornando-a compacta o suficiente para ser usada com eficiência em máquinas com muita memória e poder de processamento. Segundo, ela é baseada na Extensible Markup Language (XML), de modo que o conteúdo é bem formado e não tão aberto para interpretação do browser quanto a HTML. Este capítulo não pretende ser um manual sobre WAP, mas gostaríamos que você aprendesse alguns dos fundamentos.

Você provavelmente sabe que HTML é baseada em uma metáfora de página, com cada arquivo `.html` levado a um browser geralmente representando uma página de informações. A WML, por outro lado, é baseada em uma metáfora de baralho de cartas, com um arquivo `.wml` representando um baralho que contém alguma quantidade de cartas. Cada carta representa uma tela de informações. Desse modo, a funcionalidade de um baralho WML inteiro pode ser enviada a um cliente com apenas uma viagem de ida de volta do cliente ao servidor, ao contrário do sistema de ida e volta por página, que é normal na Web. Um arquivo `.wml` comum, então, poderia se parecer com isto:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.WAPforum.org/DTD/wml_1.1.xml">
<wml>
  <card>
    <do type="accept">
      <go href="#hello "/>
    </do>
    <p>Punch the Button</p>
  </card>
  <card id="hello">
```

```
    <p>Hello from WAP!</p>
  </card>
</wml>
```

Se você souber apenas um pouco sobre HTML e XML, provavelmente já desvendou esse código com relativa facilidade. O prólogo do documento, composto das três primeiras linhas, é XML padrão e descreve a versão da XML desse documento e o local da DTD usada para descrever as tags e os atributos nele contidos. Depois disso, o código passa a criar um baralho com duas cartas, uma com um botão OK e uma com uma saudação.

A sintaxe da WML também oferece suporte a coisas como eventos, timers, conjuntos de campos, listas e imagens (embora nem todos os dispositivos aceitem imagens). Algumas das versões mais recentes dos browsers WAP aceitam ainda uma linguagem de scripting chamada WMLScript. Não podemos abordar toda a linguagem WML aqui, mas, se você estiver interessado, poderá ver os detalhes da especificação WML em `http://www.WAPforum.org`.

Se você quiser experimentar o desenvolvimento de algum conteúdo WML, o modo mais fácil de começar é obter um simulador. Você pode obter um simulador com o desenvolvedor do microbrowser, em `http://www.openwave.com`, ou dois outros simuladores populares que vêm diretamente dos líderes de comunicações portáteis Nokia e Ericsson; visite `http://forum.nokia.com` ou `http://www.ericsson.com/developerszone`. É sempre bom fazer com que as coisas funcionem nos simuladores primeiro, antes de passar para o hardware real, pois os simuladores oferecem um tempo de escrita-execução-depuração muito mais rápido. Antes da liberação final, também é uma boa idéia testar seu produto final no máximo de dispositivos possível, pois as características exclusivas de cada dispositivo podem fazer com que seu baralho se comporte ou apareça de forma diferente do que você tinha em mente.

Segurança do WAP

A especificação WAP exige que uma pilha de criptografia sem fio, conhecida como Wireless Transport Layer Security (WTLS), seja usada para conexões seguras. Como a SSL exige muitos recursos para ser usada com a geração atual de dispositivos portáteis, a WTLS foi criada para oferecer serviços de criptografia e autenticação entre o dispositivo e o gateway WAP. O gateway, por sua vez, já é capaz de se comunicar com os hosts da Internet por meio do protocolo SSL padrão. Apesar do fato de que tanto WTLS quanto SSL sejam bastante seguros por si sós, existe a possibilidade de brechas de segurança no gateway WAP, no ponto onde o stream de dados WTLS é decodificado e recodificado com SSL. A arquitetura WTLS é ilustrada na Figura 24.2.

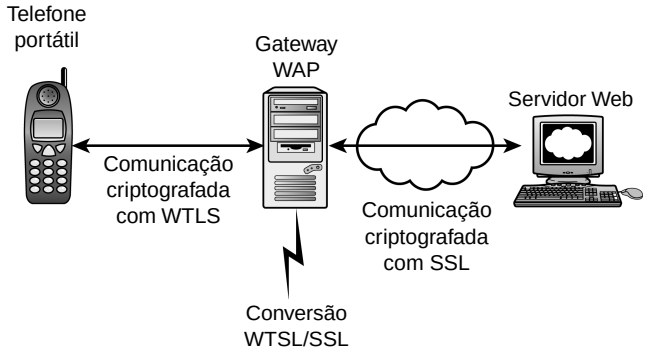

Figura 24.2 Wireless Transport Layer Security (WTLS) do WAP.

Uma aplicação WAP simples

A criação de uma aplicação WAP no Delphi é um pouco diferente da criação de uma aplicação Web regular no Delphi. WAP talvez seja ainda mais fácil de se direcionar, devido às limitações inerentes ao WAP e os dispositivos de destino costumam produzir aplicações mais simples no servidor do que

as aplicações tradicionais, baseadas em browser. No entanto, o lado oposto da moeda é que, para os desenvolvedores, a criação de aplicações que sejam atraentes e úteis para os usuários finais é mais desafiadora, dadas essas limitações.

Para este exemplo, comece criando uma aplicação WebBroker normal, conforme aprendeu no Capítulo 23. Essa aplicação possui um único módulo Web com uma única ação. Essa ação é marcada como default, conforme mostramos na Figura 24.3.

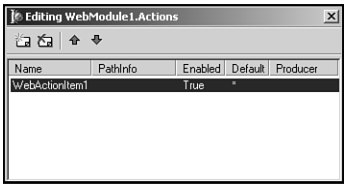

Figura 24.3 Uma aplicação WAP simples para WebBroker.

A Listagem 24.1 mostra o código-fonte para a unidade principal dessa aplicação, incluindo o tratador de evento `OnAction` para a ação default do módulo Web.

Listagem 24.1 `Main.pas` – a unidade principal para o projeto `SimpWap`

```
unit Main;

interface

uses
  SysUtils, Classes, HTTPApp;

type
  TWebModule1 = class(TWebModule)
    procedure WebModule1WebActionItem1Action(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  private
    { declarações privadas }
  public
    { declarações públicas }
  end;

var
  WebModule1: TWebModule1;

implementation

{$R *.DFM}

const
  SWMLContent = 'text/vnd.wap.wml';
  SWBMPContent = 'image/vnd.wap.wbmp';
  SWMLDeck =
    '<?xml version="1.0"?>'#13#10 +
    '<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"'#13#10 +
    '"http://www.WAPforum.org/DTD/wml_1.1.xml">'#13#10 +
    '<wml>'#13#10 +
    '  <card>'#13#10 +
    '    <do type="accept">'#13#10 +
    '      <go href="#hello"/>'#13#10 +
```

Listagem 24.1 Continuação

```
    '    </do>'#13#10 +
    '    <p>Punch the Button</p>'#13#10 +
    '  </card>'#13#10 +
    '  <card id="hello">'#13#10 +
    '    <p>Hello from WAP!</p>'#13#10 +
    '  </card>'#13#10 +
    '</wml>'#13#10;

procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.ContentType := SWMLContent;
  Response.Content := SWMLDeck;
end;

end.
```

Quando a ação é chamada, o tratador de evento responde definindo as propriedades `ContentType` e `Content` do objeto `Response`. `ContentType` é definido como a string do tipo de conteúdo WML, e o conteúdo retornado é o mesmo baralho WAP simples que foi explicado anteriormente neste capítulo.

> NOTA
>
> Lembre-se de definir o `ContentType` do objeto `Response` como a string contendo o tipo MIME da variedade de conteúdo que você está retornando. Isso define a informação de tipo de conteúdo no cabeçalho HTTP. Se você retornar o tipo de conteúdo incorreto, seu conteúdo provavelmente será mal interpretado no dispositivo de destino. Alguns tipos de conteúdo WAP dignos de nota são
>
> - `text/vnd.wap.wml` para o código WML
> - `text/vnd.wap.wmlscript` para o código de script WML
> - `image/vnd.wap.wbmp` para imagens Wireless Bitmap

A Figura 24.4 mostra essa aplicação WAP simples do Delphi em ação.

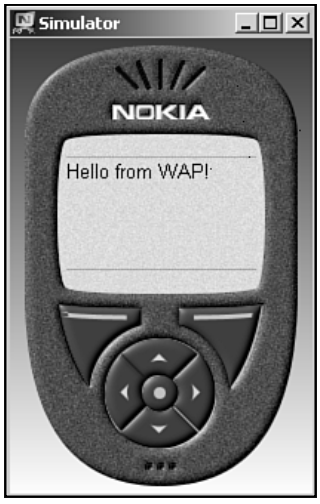

Figura 24.4 Aplicação WAP do Delphi em ação.

Informe de erros

O tratador de exceção default para aplicações WebSnap envia uma mensagem HTML ao cliente com informações sobre o erro. Naturalmente, a maioria dos dispositivos WAP não poderá entender um erro HTML, de modo que é importante garantir que quaisquer erros que possam ocorrer em sua aplicação WAP sejam expostos ao cliente como mensagens WML, em vez de HTML. Isso pode ser feito envolvendo-se cada tratador de evento `OnAction` com um bloco `try..except` que chama uma rotina de formatação de mensagem de erro. Isso aparece na Listagem 24.2.

Wireless Bitmaps

Embora o WAP ainda não tenha suporte para os elegantes gráficos JPEG e GIF, comuns na Web, a maioria dos dispositivos WAP aceita imagens monocromáticas na forma de Wireless bitmaps (wbmp). A Listagem 24.2 acrescenta uma nova ação ao módulo da Web, para dar suporte à geração de um wbmp. Essa ação gera um gráfico de aparência oficial, porém bastante aleatória, para a exibição no dispositivo de destino. Embora não entremos em detalhes sobre o formato binário dos arquivos wbmp neste texto, você pode ver que não é preciso muita coisa para gerar wbmps manualmente em suas aplicações WAP. A Figura 24.5 mostra como ficaria um WBMP em uma tela de telefone.

> NOTA
> Nem todos os browsers, dispositivos e simuladores WAP oferecem suporte para imagens wbmp. Não se esqueça de testar antes de assumir que existe suporte.

Listagem 24.2 `Main.pas` – mais uma vez, com conteúdo

```
unit Main;

interface

uses
  SysUtils, Classes, HTTPApp;

type
  TWebModule1 = class(TWebModule)
    procedure WebModule1WebActionItem1Action(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
    procedure WebModule1GraphActionAction(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  private
    procedure CreateWirelessBitmap(MemStrm: TMemoryStream);
    procedure HandleException(e: Exception; Response: TWebResponse);
  end;

var
  WebModule1: TWebModule1;

implementation

{$R *.DFM}

const
  SWMLContent = 'text/vnd.wap.wml';
  SWBMPContent = 'image/vnd.wap.wbmp';
  SWMLDeck =
```

Listagem 24.2 Continuação

```
    '<?xml version="1.0"?>'#13#10 +
    '<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"'#13#10 +
    '"http://www.WAPforum.org/DTD/wml_1.1.xml">'#13#10 +
    '<wml>'#13#10 +
    '  <card>'#13#10 +
    '    <do type="accept">'#13#10 +
    '      <go href="#hello"/>'#13#10 +
    '    </do>'#13#10 +
    '    <p>Punch the Button</p>'#13#10 +
    '  </card>'#13#10 +
    '  <card id="hello">'#13#10 +
    '    <p>Hello from WAP!</p>'#13#10 +
    '  </card>'#13#10 +
    '</wml>'#13#10;

  SWMLError =
    '<?xml version="1.0"?>'#13#10 +
    '<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"'#13#10 +
    '"http://www.wapforum.org/DTD/wml_1.1.xml">'#13#10 +
    '<wml>'#13#10 +
     '  <card id="error" title="SimpWAP">'#13#10 +
     '    <p>Error: %s'#13#10 +
     '      <do type="prev" label="Back">'#13#10 +
     '        <prev/>'#13#10 +
     '      </do>'#13#10 +
     '    </p>'#13#10 +
     '  </card>'#13#10 +
     '</wml>'#13#10;



procedure TWebModule1.HandleException(e: Exception; Response: TWebResponse);
begin
  Response.ContentType := SWMLContent;
  Response.Content := Format(SWMLError, [e.Message]);
end;

procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  try
    Response.ContentType := SWMLContent;
    Response.Content := SWMLDeck;
  except
    on e: Exception do
      HandleException(e, Response);
  end;
end;

procedure TWebModule1.WebModule1GraphActionAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var
  MemStream: TMemoryStream;
begin
  try
    MemStream := TMemoryStream.Create;
    try
```

Listagem 24.2 Continuação

```
      CreateWirelessBitmap(MemStream);
      MemStream.Position := 0;
      with Response do
      begin
        ContentType := SWBMPContent;
        ContentStream := MemStream;
        SendResponse;
      end;
    finally
      MemStream.Free;
    end;
  except
    on e: Exception do
      HandleException(e, Response);
  end;
end;

procedure TWebModule1.CreateWirelessBitmap(MemStrm: TMemoryStream);
const
  Header : Array[0..3] of Char = #0#0#104#20;
var
  Bmp: array[1..104,1..20] of Boolean;
  X, Y, Dir, Bit: Integer;
  B: Byte;
begin
  { limpa o bitmap }
  FillChar(Bmp,SizeOf(Bmp),0);
  { desenha eixos X e Y }
  for X := 1 to 104 do Bmp[X, 20] := True;
  for Y := 1 to 20 do Bmp[1, Y] := True;
  { desenha dados aleatórios }
  Y := Random(20) + 1;
  Dir := Random(10);
  for X := 1 to 104 do
  begin
    Bmp[X,Y] := True;
    if (Dir > 4) then Y := Y+Random(2)+1
    else Y := Y - Random(2) - 1;
    if (Y > 20) then Y := 20;
    if (Y < 1) then Y := 1;
    Dir := Random(10);
  end;
  { cria dados WBMP }
  MemStrm.Write(Header, SizeOf(Header));
  Bit := 7;
  B := 0;
  for Y := 1 to 20 do
  begin
    for X := 1 to 104 do
    begin
      if Bmp[X,Y] = True then
        B := B or (1 shl Bit);
      Dec(Bit);
      if (Bit < 0) then begin
        B := not B;
        MemStrm.Write(B, SizeOf(B));
        Bit := 7;
```

Listagem 24.2 Continuação

```
        B := 0;
      end;
    end;
  end;
end;

initialization
  Randomize;
end.
```

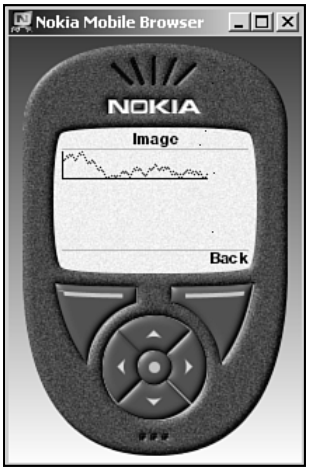

Figura 24.5 Exibindo o WBMP no simulador Nokia.

I-mode

I-mode é uma tecnologia própria para conteúdo da Internet em telefones portáteis, desenvolvida pela NTT DoCoMo, gigante de telecomunicações do Japão. I-mode tem muito sucesso no Japão, com mais de 20 milhões de assinantes e crescendo. De várias maneiras, i-mode é tudo o que o WAP não é: aceita gráficos ricos, com 256 cores, utiliza telas de telefone coloridas e uma conexão TCP/IP "sempre ligada". Além do mais, a DoCoMo desenvolveu um modelo de compartilhamento de receitas que permite ao sites i-mode obterem uma fatia da torta financeira com base no uso. No entanto, o i-mode está dificultado pela aspecto da tecnologia própria, e a disponibilidade fora do Japão ainda é escassa; os serviços do i-mode estarão sendo distribuídos nos Estados Unidos, Reino Unido e Europa continental a partir deste ano.

De um ponto de vista do desenvolvedor, o direcionamento de telefones i-mode não é muito mais difícil do que a criação de conteúdo para a Web, pois o conteúdo i-mode é desenvolvido usando-se um subconjunto da HTML, conhecido como Compact HTML (cHTML). As tags cHTML aceitas e as regras estão disponíveis na DoCoMo, em `http://www.nttdocomo.com/i/tagindex.html`. Observe que, além de usar apenas as tags aceitas pela cHTML, os sites i-mode também precisam garantir que será usado o conjunto de caracteres S-JIS, a imagens são no formato GIF e as páginas não possuem qualquer conteúdo de script ou Java.

PQA

Palm Query Applications (PQA) são essencialmente páginas HTML normais, sem acréscimos como scripting e imagens projetadas para exibição na tela de um dispositivo PalmOS sem fio. Os dispositivos PalmOS sem fio, como os Palm VIIx ou aqueles equipados com modems Novatel, atualmente estão limitados à América do Norte. Assim como o i-mode, PQAs são desenvolvidas usando um sub-

conjunto da HTML, exceto que o Palm incluiu algumas extensões próprias. Os desenvolvedores interessados em criar PQAs deverão baixar o Web Clipping Developer's Guide do Palm, em `http://www.palmos.com/dev/tech/docs/`.

Em geral, o ramo PQA da HTML inclui tudo o que existe na HTML 3.2, com a exceção dos applets (miniaplicativos), JavaScript, tabelas aninhadas, mapas de imagem e as tags `VSPACE`, `SUB`, `SUP`, `LINK` e `ISINDEX`. PQAs também incluem vários acréscimos interessantes à HTML. Os mais interessantes são a meta tag `palmcomputingplatform` e as tags `%zipcode` e `%deviceid`. Quando um documento HTML contém a meta tag `palmcomputingplatform`, isso serve como um indicador para o proxy Palm.net da `Palm Computing` (que atua como intermediário entre um servidor Web e o dispositivo Palm, semelhante a um gateway WAP) de que o documento está otimizado para exibição em um portátil PalmOS e não exige análise para que seu conteúdo inválido seja removido. Quando a tag `%zipcode` aparece em um pedido postado pelo cliente, o proxy Palm.net substitui a tag pelo código postal de onde o dispositivo está localizado (com base nas informações da torre de rádio). A tag `%deviceid`, semelhantemente, envia o ID exclusivo do dispositivo PalmOS para o servidor. Isso é prático principalmente porque a HTML PQA não oferece suporte para cookies, mas um meio semelhante de gerenciamento de estado pode ser preparado com o uso da tag `%deviceid`.

Com o Web clipping, o Palm utiliza uma técnica diferente da maioria dos outros players nesse espaço. Em vez de usar uma entidade tipo browser para navegar até um site e apanhar seu conteúdo, as PQAs existem localmente no dispositivo PalmOS. PQAs são criadas passando-se um arquivo .HTML padrão por um compilador especial, que vincula a HTML com arquivos gráficos referenciados e outras dependências. Os usuários instalam PQAs como aplicações PRC normais do PalmOS. As PQAs ganham eficiência incluindo partes da aplicação local e só indo para a rede para as páginas de "resultados".

Cliente PQA

A primeira etapa para o desenvolvimento de uma aplicação PQA é criar a parte que residirá fisicamente no dispositivo cliente. Isso é feito criando-se um documento HTML e compilando-o com a ferramenta PQA Builder da Palm Computing. Um exemplo de um documento HTML para uma PQA aparece na Listagem 24.3.

Listagem 24.3 Um documento HTML para uma PQA

```
<html>
<head>
<title>DDG PQA Test</title>
<meta name="palmcomputingplatform" content="true">
</head>
<body>
<p>This is a sample PQA for DDG</p>
<img src="image.gif">
<form method="post" action="http://128.64.162.164/scripts/pqatest.dll">
<input type="hidden" value="%zipcode" name="zip">
<input type="hidden" value="%deviceid" name="id">
<input type="submit">
</form>
</body>
```

Você pode ver que esse documento HTML simples contém uma referência a uma imagem, algum texto, um formulário com um botão de envio e campos ocultos usados para passar o código postal e o ID do dispositivo para o servidor. A Figura 24.6 mostra esse documento sendo compilado no PQA Builder.

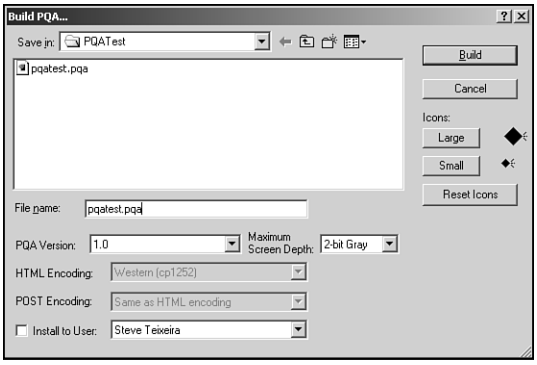

Figura 24.6 Compilando com o PQA Builder.

Uma vez compilado, é gerado um arquivo com uma extensão `.pqa`. Esse arquivo contém o documento HTML, além de quaisquer imagens referenciadas. Esse arquivo pode ser instalado no dispositivo PalmOS da mesma forma que qualquer outra aplicação PalmOS.

Servidor PQA

A parte do servidor, como WAP, é uma aplicação WebSnap que cuida dos pedidos de página de clientes e retorna páginas. No entanto, ao contrário do WAP com sua WML, PQAs se comunicam usando a variante da HTML descrita anteriormente. A Listagem 24.4 mostra a unidade principal de uma aplicação WebBroker, criada para realizar o papel de servidor para o cliente PQA descrito anteriormente.

Listagem 24.4 `Main.pas` – a unidade principal para a aplicação PQATest

```
unit Main;

interface

uses
  SysUtils, Classes, HTTPApp;

type
  TWebModule1 = class(TWebModule)
    procedure WebModule1WebActionItem1Action(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  private
    { Declarações privadas }
  public
    { Declarações públicas }
  end;

var
  WebModule1: TWebModule1;

implementation

{$R *.DFM}

const
  SPQAResp =
    '<html><head><meta name="palmcomputingplatform" content="true"></head>'+
    #13#10 +
    '<body>Hello from a Delphi server<br>Your zipcode is: %s<br>'#13#10 +
```

Listagem 24.4 Continuação

```
    'Your device ID is: %s<br><img src="file:pqatest.pqa/image.gif"></body>+
    '</html>';

procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.Content := Format(SPQAResp, [Request.ContentFields.Values['zip'],
    Request.ContentFields.Values['id']]);
end;

end.
```

O servidor responde ao cliente enviando o código postal do cliente e o ID do dispositivo. Uma técnica interessante no código HTML retornado pelo servidor é que ele referencia a mesma imagem que foi compilada para o arquivo `.pqa` no cliente, usando a sintaxe *`arquivo`*:<*`nomepqa`*>. Isso permite criar gráficos ricos em suas PQAs, compilando-os para o lado do cliente e referenciando-os no servidor, eliminando assim a necessidade de fazer o download de quaisquer gráficos pelo modem sem fio. As Figuras 24.7 e 24.8 mostram essa aplicação em ação, antes e depois que o botão de envio é pressionado. Observe que o código postal e o ID do dispositivo são valores `null` no simulador.

Figura 24.7 PQATest no simulador PalmOS.

Figura 24.8 PQATest no simulador PamlOS depois do clique na tecla Submit.

Experiência do usuário sem fio

A experiência do usuário é, de longe, o fator mais importante para determinar se um sistema móvel por fim será útil para os indivíduos. No entanto, a experiência do usuário normalmente tem curta duração em favor dos recursos gratuitos ou da tecnologia. Devido às limitações inerentes ao mundo móvel – principalmente conectividade e tamanho de dispositivo –, não há muito espaço para erros

quando o desenvolvedor tenta fornecer aos usuários a funcionalidade de que precisam e no momento em que precisam dela. O truque é focalizar o usuário: determine quais informações ou serviços os usuários precisam e esforce-se para conseguir isso da forma mais eficiente possível.

Redes por comutação de circuitos e comutação de pacotes

Ao considerar o telefone móvel como plataforma cliente, uma questão que pode ter um impacto dramático na facilidade de utilização é se a rede de telefone móvel é comutada por circuito ou por pacote. As redes por comutação de circuitos operam como um modem em um telefone convencional: você precisa discar para estabelecer uma conexão direta com o host, e essa conexão precisa ser mantida enquanto a troca de dados estiver ocorrendo. As redes com comutação de pacotes se comportam mais como uma conexão Ethernet fixa: a conexão está sempre ativa, e os pacotes de dados podem ser enviados e recebidos pela conexão a qualquer momento.

O overhead exigido para se estabelecer conexões em redes por comutação de circuitos geralmente é um dos principais impedimentos para a satisfação do usuário – principalmente se o usuário estiver realizando uma tarefa que deverá tomar apenas alguns segundos. Afinal, quem deseja esperar 20 segundos para se conectar para poder interagir com uma aplicação por 3 segundos? A maioria das redes móveis atualmente ainda utiliza comutação de circuitos, mas alguns exemplos notáveis de redes que utilizam a tecnologia de comutação de pacotes incluem as redes i-mode da NTT DoCoMo, iDEN da Nextel e CDPD da AT&T.

O sem fio não está na Web

Um erro de conceito comum entre os fornecedores de dispositivos portáteis e redes sem fio é que as pessoas desejam surfar na Web com esses dispositivos. Com suas pequeninas telas e largura de banda limitada, os dispositivos portáteis são um veículo extremamente inconveniente para a navegação pela Web. Cada página de informações pode levar vários segundos para serem apanhadas, devido às restrições da rede, e a entrada de dados pode ser bastante dolorosa – principalmente em um telefone portátil. Em vez disso, as aplicações portáteis precisam ser otimizadas para oferecer informações e serviços específicos com o mínimo de entrada de dados necessário da parte do usuário e com o mínimo de viagens de ida e volta possível entre cliente e servidor.

A importância do espaço

Ao criar aplicações portáteis, você sempre deverá permanecer sensível à quantidade disponível de espaço na tela. Seja criando uma aplicação baseada em WAP, vinculada a um microbrowser, ou uma interface com o usuário personalizada em J2ME, os desenvolvedores de aplicação precisam equilibrar entre os problemas de comunicar uma quantidade suficiente de informações em cada tela e manter uma interface com o usuário legível e navegável.

Técnicas de entrada de dados e navegação

Relacionado ao espaço está a questão da entrada de dados. Diferentes tipos de dispositivos contam com diferentes mecanismos para entrada de dados. Dispositivos PalmOS e Pocket PC usam uma combinação de um reconhecimento de caneta e escrita manual (com teclados portáteis opcionais); os dispositivos `RIM BlackBerry` utilizam teclados do tamanho de um polegar; telefones portáteis geralmente possuem apenas um teclado numérico e alguns botões extras. Isso significa que as aplicações projetadas para uma plataforma móvel poderiam ser difíceis de se usar em outra. Por exemplo, uma caneta de um PDA é ótima para se tocar em áreas aleatórias da tela, enquanto um BlackBerry é mais adequado para a entrada de dados de texto; um telefone, por sua vez, é mais adequado para o mínimo de entrada de dados possível!

M-Commerce

Assim como o e-commerce passou a se referir a transações comerciais realizadas pela Web, *m-commerce* refere-se a transações comercias feitas por meio de dispositivos móveis. Os desenvolvedores de sites de comércio por dispositivo móvel precisam entender que m-commerce é fundamentalmente diferente de e-commerce. Evidentemente, não é viável que os clientes de dispositivos móveis naveguem pelos itens. No caso de um telefone portátil, por exemplo, é muito demorada a entrada de toques de tecla – imagens ou não existem ou são de má qualidade, e não há espaço suficiente na tela para a descrição dos itens.

Em vez disso, os sistemas de m-commerce precisam ser projetados com a noção de que os usuários sabem o que querem; basta facilitar o modo como eles dão seu dinheiro. Lembre-se: se o usuário quiser comprar uma televisão ou um livro, não há motivo para que ele não possa simplesmente esperar até chegar em sua casa ou escritório e fazer a compra em um computador "normal". O fato de que o usuário sequer deseja se engajar no m-commerce significa que existe algum sentido na urgência de se fazer a compra, e os comerciantes de m-commerce bem-sucedidos serão aqueles que reconhecerem e tirarem proveito desse fato.

Por exemplo, um país da Europa oriental permite que os motoristas paguem a tarifa do estacionamento com seus telefones celulares. Isso pode ser uma aplicação relativamente simples na superfície, mas a proposição de valor para ambas as partes é muito atraente. O motorista não precisa se preocupar em ter moedas suficientes para colocar nos medidores, e o operador do aparelho não precisa juntar as moedas de dezenas ou centenas ou milhares de medidores, a fim de trocá-las no banco. O policial monitorando os medidores pode usar um dispositivo móvel ligado ao mesmo sistema para saber instantaneamente quanto tempo resta para determinada vaga.

Resumo

O mundo da computação para dispositivo sem fio cresceu muito nos últimos anos, e pode ser muito difícil acompanhar todas as tendências emergentes. Nossa esperança é que, nesse ponto, você já esteja armado com informações suficientes para tomar algumas decisões estratégicas e seguir adiante com um projeto de mobilidade. Além disso, você viu que o Delphi é uma ferramenta bastante capaz quando se trata da criação da próxima geração de aplicações para dispositivos sem fio.