Томский государственный университет

Факультет информатики

А.В. Скворцов, Т.Н. Поддубная

Панели инструментов, списки, деревья

(методические указания к лабораторной работе № 7)

Томск – 1999

Указания РАССМОТРЕНЫ и УТВЕРЖДЕНЫ методической комиссией факультета информатики.

Протокол №____ от «____»_____1999 г.

Председатель методической комиссии факультета информатики,

профессор _____ Поддубный В.В.

Утверждаю. Зав. кафедрой прикладной информатики, доцент

_____ Сущенко С.П. «____» _____ 1999 г.

Методические указания посвящены программной среде Borland Delphi компании Inprise Corporation – одному из мощнейших современных средств разработки приложений для Windows.

Указания разработаны для студентов межфакультетской специализации факультета информатики Томского государственного университета. Содержат подробное описание лабораторной работы: цель, задачи, описание работы и используемых компонентов Delphi. В конце работы приведены тексты программ.

Резенцент – канд. техн. наук, доцент **Ю.Л. Костюк**.

Разработка приложений неотрывно связана с вопросом создания удобного интерфейса. В настоящее время пользователи уже привыкли к меню с пиктограммами, характеризующими соответствующие команды, панелям инструментов, которые можно свободно перемещать по экрану, спискам с элементами, имеющими индивидуальные пиктограммы, и т.д.

Данная лабораторная работа посвящена знакомству с компонентами Delphi, позволяющими создавать современные приложения, красиво оформленные пиктограммами. Наше приложение будет внешне несколько похоже на проводник Windows. В левой стороне формы будет отображаться дерево каталогов для всех доступных дисков, справа – состав выбранного слева каталога, а внизу – текст выбранного файла.

Кроме того, в данной работе нужно будет освоить концепцию действий, используемую при программировании в Delphi. Использование невизуальных компонентов типа `TAction` позволяет централизованно определять различные команды, которые потом просто указываются для использования в качестве элементов различных меню, кнопок на панели инструментов и т.д. Если нужно выполнить такие действия, как: изменить название команды, назначить горячую клавишу, запретить или разрешить команду и др., достаточно указать это только для компонента `TAction`, а он автоматически распространит изменения на все связанные с ним пункты меню и кнопки.

Цель работы

Изучение визуальных компонентов Delphi для представления различной информации, снабженной пиктограммами.

Задачи работы

1. Знакомство с компонентом для хранения списков изображений `TImageList`.
2. Ознакомление с концепцией действий Delphi, компонентами типа `TAction` и их применением.
3. Знакомство с компонентами для представления иерархической информации в виде деревьев `TTreeView` и в виде списков с пиктограммами `TListView`,
4. Знакомство с компонентом для создания панелей инструментов `TToolbar` и компонентом для перетаскивания `TControlBar`.
5. Создание приложения для просмотра дерева каталогов и файлов на дисках.

Описание работы

В данной работе наше приложение должно быть красиво оформлено с помощью пиктограмм. Поэтому рекомендуется начать работу с их создания.

В Delphi для хранения списка пиктограмм используется компонент типа `TImageList` (список картинок), позволяющий хранить внутри себя изображения одинакового размера. Стандартным размером является 16x16. После помещения его на форму необходимо дважды щелкнуть на компоненте, при этом откроется редактор списка картинок, с помощью которого можно добавить в список новые картинки или удалить существующие.

Для нашей программы необходимо создать в любом графическом редакторе 8 изображений формата BMP размером 16 x 16. Примерный внешний вид таких изображения приведен на рис. 21.

Обратите внимание, что картинки специально нарисованы на фоне такого цвета, который не используется в изображениях отдельных пиктограмм. Когда эти изображения попадают в компоненту `TImageList`, все пикселы, совпадающие по цвету с самым левым нижним пикселом, помечаются прозрачными. Это значит, что при последующем отображении картинок на экране, прозрачные пикселы рисоваться не будут, т.е. на их месте будет сохраняться цвет фона.

Замечание

Картинки можно быстро получить, если их вырезать с экрана других работающих приложений. Нажмите клавиши PrintScrn или Alt+PrintScrn, и в буфере обмена окажется требуемое изображение, которое затем можно вставить в любой графический редактор, например в PaintBrush.

Использование списка картинок `TImageList` вместо отдельных изображений позволяет значительно экономить ресурсы Windows, т.к. для списка на самом деле хранится одна большая картинка, имеющая в качестве составных частей все необходимые картинки. Экономия особенно заметна при использовании больших наборов изображений.

В нашей программе пиктограммы будут назначаться следующим визуальным элементам: пунктам меню, кнопкам панелей инструментов, вершинам дерева каталогов, элементам списка файлов.

Для того чтобы можно было назначить картинку любому из этих элементов, необходимо вначале установить в свойстве `ImageList` содержащего его компонента используемый список картинок, а затем в свойстве `ImageIndex` каждого элемента указать номер картинки из списка. Однако если

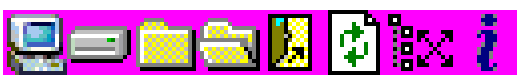

Рис. 21. Картинки для приложения

наша программа основана на концепции действий, рассматриваемой ниже, вместо прямого задания значения свойства `ImageIndex` пункта меню или кнопки панели инструментов следует указать значение свойства `ImageIndex` соответствующего компонента `TAction`.

> ***Замечание***
>
> *Не используйте компоненты* `TSpeedButton` *и* `TBitBtn`*. Они устарели и оставлены в Delphi только для совместимости со старыми программами. Вместо* `TBitBtn` *надо использовать* `TButton`*, а вместо* `TSpeedButton` *– панель инструментов* `TToolbar` *с кнопками* `TToolButton`*.*

Для визуального контроля назначаемых действий следует указать свойство `ImageList` списка действий `TAction`. Тогда в редакторе действий слева от названий действий будет появляться назначенная пиктограмма.

Теперь перейдем к вопросу о создании действий. Как было сказано ранее, действия (`TAction`) в Delphi предназначены для однократного централизованного определения разнообразных команд. Условно схема использования действий приведена на рис. 22.

Для определения действия необходимо на форму поместить компонент типа `TActionList`, который может содержать в себе множество действий. Затем нужно дважды щелкнуть на нём мышкой, при этом откроется редактор действий. Нажатием на клавишу Insert можно добавлять новые действия, а на клавишу Delete – удалять. При выборе мышкой действия в списке оно становится доступным в инспекторе объектов. Список основных свойств и событий действий приведен в табл. 16-17.

В табл. 18 приведен список действий, который необходимо создать в нашей программе.

После того как созданы все необходимые действия, их необходимо назначить соответствующим пунктам меню и кнопкам. У этих компонен-

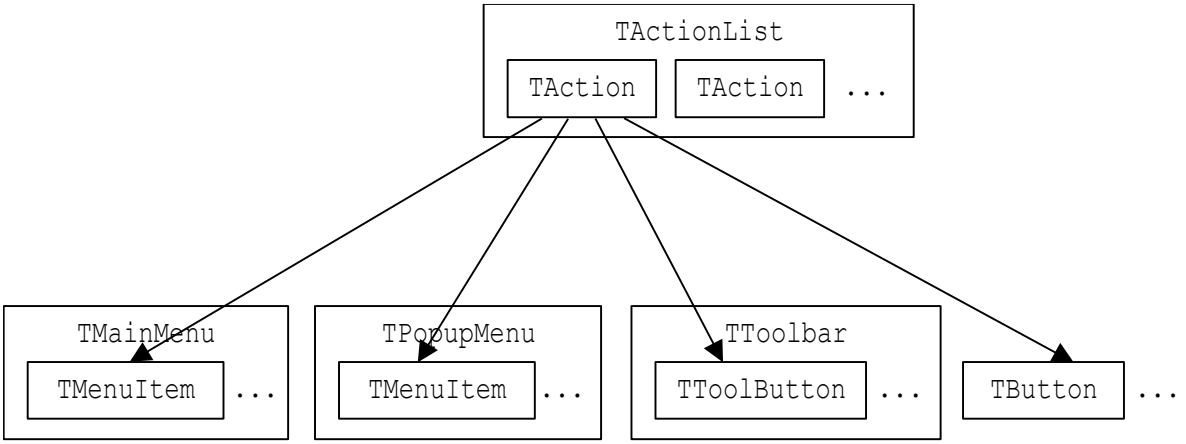

Рис. 22. Схема использования действий в Delphi

тов имеется свойство Action, значение которого можно установить в инспекторе объектов с помощью выпадающего списка всех доступных в форме действий.

Таблица 16. Основные свойства объектов типа TAction

Свойство	Тип	Комментарий
Caption	String	Название действия в меню
Category	String	Категория – используется для внутреннего упорядочивания действий внутри TActionList
Checked	Boolean	Отмечены ли галочкой пункты меню и нажаты ли соответствующие кнопки
Enabled	Boolean	Разрешена ли команда
Hint	String	Всплывающая подсказка для кнопок
ImageIndex	Integer	Код картинки в связанном списке картинок
ShortCut	TShortCut	Код горячей клавиши для вызова действия
Visible	Boolean	Видимы ли пункты меню и кнопки

Таблица 17. Основные события объектов типа TAction

Свойство	Комментарий
OnExecute	Выполнение действия
OnUpdate	Обновление информации о действии. Здесь можно изменить любые свойства действия в зависимости от текущего состояния программы. Обычно с помощью этого события запрещают недоступные команды (а следовательно, связанные с действием пункты меню и кнопки)

Таблица 18. Список действий, создаваемых в нашем приложении

Действие	Описание
ActionRefresh	«Обновить дерево». Приводит к повторному считыванию информации о каталогах
ActionExpandAll	«Раскрыть все». Вызывает раскрытие всех нераскрытых вершин дерева, а следовательно построение полного дерева каталогов дисков
ActionExit	«Выход». Выход из программы
ActionAbout	«О программе…». Выдает краткую информацию о программе

В нашем приложении необходимо сделать назначение действий для главного меню TMainMenu, локального (выпадающего) меню TPopupMenu и кнопкам панелей инструментов, рассматриваемых ниже.

Замечание

Не забудьте указать свойство ImageList *компонентов меню и панелей инструментов из выпадающего списка в инспекторе объектов для отображения в них картинок.*

При создании элементов меню, соответствующих действиям, не нужно указывать заголовки. Нужно задать свойство Action элемента меню, при этом заголовок, код картинки, горячая клавиша и обработчик события будут автоматически взяты из компонента TAction.

Локальное меню обычно появляется на экране при нажатии правой кнопки мышки на визуальных компонентах. Для этого у этих компонентов должно быть установлено свойство PopupMenu в инспекторе объектов из выпадающего списка доступных локальных меню TPopupMenu. В нашем случае достаточно назначить свойство PopupMenu только для формы. Тогда нажатие правой кнопки мышки в любом месте формы будет вызывать локальное меню.

Для редактирования состава локального меню необходимо, как и для главного меню формы, дважды щелкнуть на компоненте TPopupMenu, при этом появится редактор меню, работа с которым описана в предыдущих лабораторных работах. В нашем приложении локальное меню должно состоять из 4 пунктов, которым должны быть назначены все 4 действия из списка действий TActionList.

Для создания панели инструментов с кнопками в Delphi используется компонент типа TToolbar. Для добавления в него новых кнопок в дизай-

нере форм необходимо нажать правую кнопку мыши на панели инструментов для вызова локального меню и выбрать соответствующий пункт. Добавляемые кнопки при этом имеют тип TToolButton. Как и пункты меню, кнопки имеют аналогичное свойство Action, при установке значения которого из выпадающего списка кнопки автоматически получат все остальные необходимые свойства (текст надписи, всплывающую подсказку, код картинки, обработчик события нажатия). По умолчании кнопки на панели инструментов отображаются только с картинкой без сопроводительной надписи. Для отображения с надписью (как в пункте меню) необходимо указать свойство ShowCaptions равным True.

Замечание

Для отображения всплывающих подсказок для визуальных компонентов необходимо указать значение свойства ShowHint *формы равным* True.

В большинстве современных приложений, например в Microsoft Office, панели инструментов можно свободно перетаскивать по экрану с помощью мышки.

Для того чтобы это можно было делать, в Delphi имеется компонент типа TControlBar, который дополняет все помещаемые на него другие компоненты рамочкой и двумя вертикальными полосками слева для перетаскивания. Обычно в TControlBar помещают панели инструментов TToolbar. Поэтому перед размещением панелей инструментов на форме сначала необходимо поместить компонент TControlBar, а в него затем – панели инструментов.

Для того чтобы можно было вытаскивать визуальные компоненты (обычно, это панели инструментов) за пределы TControlBar, у них имеется свойство DragKind, которое необходимо установить в значение dkDock. При вытаскивании (в т.н. «плавающем» положении) панели инструментов помещаются в небольшом окне с заголовком, задаваемым в свойстве Caption панели. Для того чтобы панели инструментов могли приклеиваться не только в верхней части приложения, но и снизу и по бокам, необходимо просто разместить дополнительные компоненты типа TControlBar во всех нужных местах.

Теперь перейдем к размещению на форме компонентов. Внешний вид разрабатываемого нами приложения изображен на рис. 20. Условно окно состоит из трех основных частей. Слева размещен компонент типа TTreeView, предназначенный для представления иерархических структур, справа – список TListView, позволяющий создавать списки элементов, оформленные пиктограммами. В нижней части необходимо разместить компонент TMemo для просмотра содержимого выбранного файла. Кроме

того, в верхней части формы мы должны поместить компонент TControlBar с панелями инструментов, а в самой нижней части формы – строку статуса типа TStatusBar.

На форме также должны быть размещены два списка пиктограмм. Первый из них будет использоваться в главном и локальном меню, панелях инструментов и в дереве. Второй список будет формироваться автоматически для создания изображений около списка файлов в компоненте TListView.

Теперь рассмотрим, как можно использовать компоненты TTreeView и TListView. При запуске наша программа должна будет получить список всех доступных на компьютере дисков и поместить их в дерево. Затем, когда пользователь будет раскрывать соответствующие вершины в дереве, программа должна будет считывать содержимое соответствующего каталога и добавлять все найденные каталоги в дерево.

Для того чтобы добавить в дерево новую вершину, необходимо воспользоваться методом Add или AddChild объекта типа TTreeNodes, который доступен через свойство Items дерева TTreeView. Эти методы имеют два аргумента, первый из них задает базовую вершину, за которой необходимо добавить новую, а второй указывает название вершины. Если базовая вершина не указана (указан nil), то будет создана вершина самого верхнего уровня. Таким образом, метод Add добавляет вершину такого же уровня, как и базовая, а метод AddChild создает дочернюю.

Оба из указанных методов возвращают экземпляр объекта типа TTreeNode. Список его основных свойств приведен в табл. 19.

Компонент дерева TTreeView может порождать много разных событий, однако в нашей программе нам понадобятся только два. Событие OnExpanding происходит при попытке пользователя раскрыть вершину дерева. Мы должны в обработчике этого события проверить, раскрывалась ли эта вершина ранее, и если нет, то необходимо прочитать список всех соответствующих подкаталогов и добавить новые вершины в дерево.

Событие OnChange дерева происходит после того, как пользователь выберет в дереве другую вершину. В этом обработчике мы должны прочитать список всех файлов в выбранном каталоге и сформировать из них список в компоненте типа TListView. Кроме того, здесь же необходимо вывести в строку статуса полный путь выбранного каталога.

Таблица 19. Основные свойства объектов типа TTreeNode

Свойство	Тип	Комментарий
Count	Integer	Количество дочерних вершин
Data	Pointer	Предназначено для свободного использования программистом
HasChildren	Boolean	Имеет ли вершина в дереве слева от себя квадратик с плюсом, показывающий, что вершину можно раскрыть
Level	Integer	Уровень дерева, на котором находится вершина
ImageIndex	Integer	Код пиктограммы, появляющейся слева от вершины. Перед использованием этого свойства не забудьте задать свойство ImageList компонента TTreeView
SelectedIndex	Integer	Код пиктограммы, используемой в случае, если вершина выбрана в дереве
Text	String	Текст надписи вершины

Список типа TListView хорошо знаком любому пользователю Windows. Он используется в Проводнике Windows для представления списков файлов. Этот элемент имеет 4 стандартных режима работы: в виде списка, маленьких пиктограмм, крупных пиктограмм и в виде многоколоночного отчета. Мы будем использовать последний режим. Для его задания нужно свойство ViewStyle установить равным vsReport и затем создать список необходимых колонок. Для этого следует выбрать в инспекторе объектов свойство Columns компонента TListView и нажать на кнопку с многоточием или дважды щелкнуть на нем. При этом появится окно редактора колонок со списком колонок. В нем можно добавлять новые колонки, нажимая клавишу Insert, и удалять их, нажимая Delete. Если выделить нужную колонку в списке, то она станет доступной для редактирования в инспекторе объектов, где необходимо указать такие свойства, как текст заголовка Caption и ширину Width. Список основных свойств компонентов колонки TListColumn приведен в табл. 20.

Для добавления в список новых элементов необходимо воспользоваться вызовом ListView.Items.Add. Он возвращает элемент списка типа TListItem. Список его основных свойств приведен в табл. 21.

Таблица 20. Основные свойства компонентов типа `TListColumn`

Свойство	Тип	Комментарий
`Caption`	`String`	Заголовок колонки
`ImageIndex`	`Integer`	Код пиктограммы, появляющейся слева от вершины. Перед использованием этого свойства не забудьте задать свойство `ImageList` компонента `TListView`
`Width`	`Integer`	Ширина колонки
`MinWidth`	`Integer`	Минимальная допустимая ширина колонки
`MaxWidth`	`Integer`	Максимальная допустимая ширина колонки
`AutoSize`	`Boolean`	Автоматический подбор ширины колонки. При изменении размеров списка (например, если пользователь растянет форму, а следовательно, и список) все колонки со значением этого свойства, равным `True`, сами изменят свою ширину так, чтобы суммарная ширина всех колонок была точно равна ширине самого списка

Таблица 21. Основные свойства объектов типа `TListItem`

Свойство	Тип	Комментарий
`Caption`	`String`	Текст надписи элемента списка
`Data`	`Pointer`	Предназначено для свободного использования программистом
`ImageIndex`	`Integer`	Код пиктограммы, появляющейся слева от вершины. Перед использованием этого свойства не забудьте задать свойство `ImageList` компонента `TListView`
`SubItems`	`TStrings`	Текст надписей во 2-й, 3-й и т.д. колонке

В Windows для каждого файла на диске можно получить пиктограмму, представляющую его в Проводнике. Для этого необходимо воспользоваться функцией Windows `ExtractAssociatedIcon`. После получения пиктограммы её необходимо поместить в список картинок `ListImageList` для списка файлов `ListView`. Так как функция `ExtractAssociatedIcon` возвращает пиктограмму размером 32x32, то в компоненте `ListImageList` необходимо указать эти же размеры для составляющих картинок.

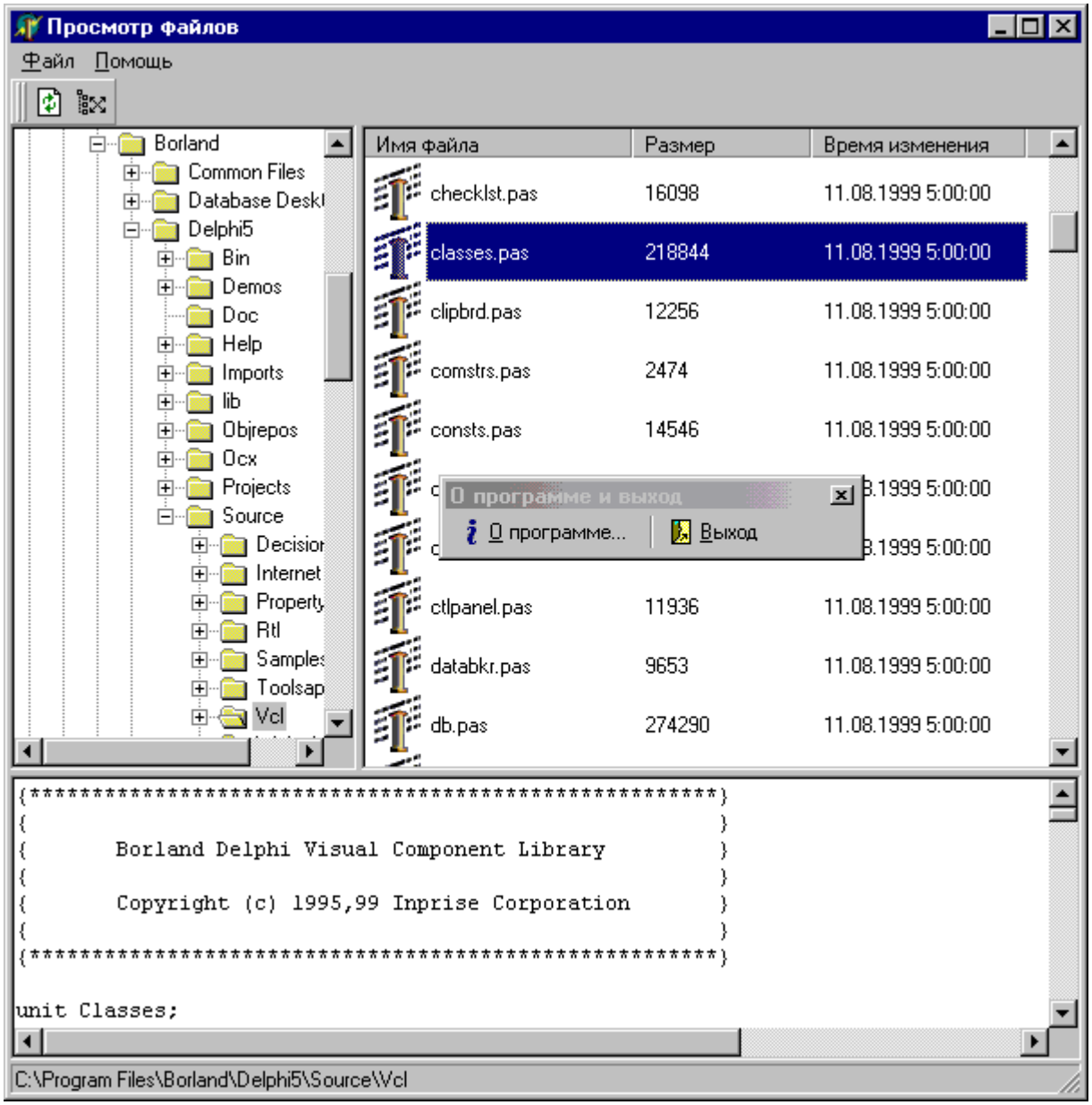

Рис. 23. Внешний вид приложения

При выборе элемента в списке `TListView` возникает событие `OnClick`, в ответ на которое мы должны считать содержимое выбранного файла и отобразить его в компоненте `TMemo`.

В заключение отметим, что в нашей форме размещено два компонента типа `TSplitter`: вертикально между деревом и списком и под ними горизонтально над текстом файла. Эти компоненты позволяют пользователю с помощью мышки изменять соотношение размеров разделяемых компонентов.

Замечание

Признаком хорошего пользовательского интерфейса является возможность легкой настройки размеров рабочих зон на форме. Для этого используйте компоненты типа `TSplitter`.

Для этого необходимо, чтобы один из разделяемых компонентов имел свойство `Align` равным `alLeft`, `alRight`, `alTop` или `alBottom`, а второй – `alClient`. При этом разделитель `TSplitter` помещается между ними и его свойство `Align` устанавливается равным свойству `Align` первого компонента.

На рис. 23 приведен внешний вид запущенного приложения. В листинге 14 приведен текст файла модуля.

Листинг 14. Текст главного модуля MainUnit.pas

```
unit MainUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ComCtrls, ExtCtrls, ToolWin, Menus, ImgList, ActnList;

type
  TFileViewerForm = class(TForm)
    MainMenu: TMainMenu;
    MenuFile: TMenuItem;
    ItemExit: TMenuItem;
    ImageList: TImageList;
    StatusBar: TStatusBar;
    ListImageList: TImageList;
    ItemSeparator: TMenuItem;
    ItemExpandAll: TMenuItem;
    ItemRefresh: TMenuItem;
    ControlBar1: TControlBar;
    ControlBar2: TControlBar;
    ControlBar3: TControlBar;
    ToolBar1: TToolBar;
    ToolButton1: TToolButton;
    ToolButton2: TToolButton;
    PanelOverClient: TPanel;
    PanelClient: TPanel;
    SplitterVertical: TSplitter;
    SplitterHorizontal: TSplitter;
    TreeView: TTreeView;
    ListView: TListView;
    Memo: TMemo;
    ControlBar4: TControlBar;
    ActionList: TActionList;
    PopupMenu: TPopupMenu;
    N1: TMenuItem;
    ActionRefresh: TAction;
    ActionExpandAll: TAction;
    ActionExit: TAction;
    ActionAbout: TAction;
    ToolBar2: TToolBar;
    ToolButton3: TToolButton;
    ToolButton4: TToolButton;
    ToolButton5: TToolButton;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    N5: TMenuItem;
    N6: TMenuItem;
    N7: TMenuItem;
    N8: TMenuItem;
    procedure FormCreate(Sender: TObject);
    procedure TreeViewCollapsing(Sender: TObject; Node: TTreeNode;
      var AllowCollapse: Boolean);
    procedure TreeViewChange(Sender: TObject; Node: TTreeNode);
    procedure TreeViewExpanding(Sender: TObject; Node: TTreeNode;
      var AllowExpansion: Boolean);
    procedure ListViewClick(Sender: TObject);
    procedure ActionRefreshExecute(Sender: TObject);
```

```
    procedure ActionExpandAllExecute(Sender: TObject);
    procedure ActionExitExecute(Sender: TObject);
    procedure ActionAboutExecute(Sender: TObject);
  private
    // Очистить дерево и прочитать список доступных дисков
    procedure UpdateTree;
    // Получить текущий выбранный в дереве каталог
    function GetNodeFolder(N: TTreeNode): string;
  end;

var
  FileViewerForm: TFileViewerForm;

implementation

{$R *.DFM}

uses
 ShellApi; // Используется при получении пиктограмм для файлов в списке

procedure TFileViewerForm.FormCreate(Sender: TObject);
begin // Прочитать список доступных дисков
  UpdateTree;
end;

procedure TFileViewerForm.UpdateTree;
var c: char; Root, N: TTreeNode; S: Int64;
begin // Очистить дерево и прочитать список доступных дисков
  TreeView.Items.BeginUpdate; // Запретить перерисовку дерева в процессе его изменения
  try
    TreeView.Items.Clear; // Очистить дерево
    Root:=TreeView.Items.Add(nil,'Диски');
    Root.Data:=Pointer(1);
    Root.ImageIndex:=0;
    Root.SelectedIndex:=0;
    for c:='C' to 'Z' do
    begin
      S:=DiskSize(byte(c)-byte('A')+1);
      if S>=0 then // Если диск доступен, то размер всегда неотрицательный
      begin
        N:=TreeView.Items.AddChild(Root,c+':');
        N.ImageIndex:=1;
        N.SelectedIndex:=1;
        N.HasChildren:=True; // Выставить значок [+] слева от вершины дерева
      end;
    end;
    Root.Expanded:=True; // Раскрыть дерево
  finally
    TreeView.Items.EndUpdate; // Обновление дерева закончено, его можно перерисовать
  end;
end;

procedure TFileViewerForm.TreeViewCollapsing(Sender: TObject;
  Node: TTreeNode; var AllowCollapse: Boolean);
begin // Разрешить сворачивание всех вершин, кроме первой
  AllowCollapse:=Node.Level>0;
end;

function TFileViewerForm.GetNodeFolder(N: TTreeNode): string;
begin // Получить текущий каталог в дереве
  Result:='';
```

```
  while (N<>nil) and (N.Level>0) do
  begin
    if Result<>'' then Result:='\'+Result;
    Result:=N.Text+Result;
    N:=N.Parent; // Перейти от текущей вершины к родительской
  end;
end;

procedure TFileViewerForm.TreeViewChange(Sender: TObject; Node: TTreeNode);
var s: string; sr: TSearchRec; Icon: TIcon;
  procedure AddFile;
  var LI: TListItem; w: word;
  begin // Добавить в список файл
    LI:=ListView.Items.Add;
    LI.Caption:=sr.Name;
    LI.SubItems.Add(IntToStr(sr.Size));
    LI.SubItems.Add(DateTimeToStr(FileDateToDateTime(sr.Time)));
    Icon.Handle:=ExtractAssociatedIcon(HInstance, PChar(s+'\'+sr.Name), w);
    LI.ImageIndex:=ListImageList.AddIcon(Icon);
    if GetAsyncKeyState(VK_ESCAPE)<0 then // Если нажата клавиша Escape, то прервать чтение
      Abort;
  end;
begin // Прочитать список элементов в текущем каталоге
  StatusBar.SimpleText:=GetNodeFolder(TreeView.Selected);
  ListView.Items.BeginUpdate; // Запретить перерисовку списка в процессе его изменения
  try
    ListView.Items.Clear; // Очистить список
    Memo.Lines.Clear; // Очистить зону загруженного файла
    ListImageList.Clear; // Очистить список пиктограмм
    s:=GetNodeFolder(Node);
    if (s<>'') and (FindFirst(s+'\*.*', 0, sr)=0) then
    try
      try
        Icon:=TIcon.Create; // Создать пиктограмму для временного использования
        try
          AddFile;
          while FindNext(sr)=0 do
            AddFile;
        finally
          Icon.Free; // Удалить временную пиктограмму
        end;
      finally
        FindClose(sr);
      end;
    except
    end;
  finally
    ListView.Items.EndUpdate; // Обновление списка закончено, его можно перерисовать
  end;
end;

procedure TFileViewerForm.TreeViewExpanding(Sender: TObject;
  Node: TTreeNode; var AllowExpansion: Boolean);
var s: string; sr: TSearchRec;
  procedure AddFolder(AText: string);
  var N: TTreeNode;
  begin
    if (sr.Attr and faDirectory<>0) and (AText<>'.') and (AText<>'..') then
    begin
      N:=TreeView.Items.AddChild(Node, AText);
      N.ImageIndex:=2; N.SelectedIndex:=3; N.HasChildren:=True;
```

```
    end;
    if GetAsyncKeyState(VK_ESCAPE)<0 then // Если нажата клавиша Escape, то прервать чтение
      Abort;
  end;
begin
  if Node.HasChildren and (Node.Count=0) then
  try // Раскрытие еще не прочитанной папки
    s:=GetNodeFolder(Node);
    if (s<>'') and (FindFirst(s+'\*.*', faDirectory, sr)=0) then
    begin
      try
        AddFolder(sr.Name);
        while FindNext(sr)=0 do
          AddFolder(sr.Name);
      finally
        FindClose(sr);
      end;
    end;
    Node.HasChildren:=Node.Count>0;
    Node.AlphaSort; // Сортировка элементов по алфавиту
  except // При ошибке чтения (например, нет прав доступа к каталогу) стереть подэлементы
    Node.DeleteChildren;
    Node.HasChildren:=True;
    raise;
  end;
end;

procedure TFileViewerForm.ListViewClick(Sender: TObject);
begin // Загрузить выбранный в списке файл
  Memo.Lines.Clear;
  if ListView.Selected<>nil then
    try
      Memo.Lines.LoadFromFile(GetNodeFolder(TreeView.Selected)+'\'+ListView.Selected.Caption);
    except
    end;
end;

procedure TFileViewerForm.ActionRefreshExecute(Sender: TObject);
begin // Обновить дерево (перечитать заново)
  UpdateTree;
end;

procedure TFileViewerForm.ActionExpandAllExecute(Sender: TObject);
begin // Развернуть все дерево
  TreeView.Items.BeginUpdate; // Запретить перерисовку дерева в процессе его изменения
  try
    Screen.Cursor:=crHourGlass; // Сделать курсор в виде песочных часов
    try
      TreeView.FullExpand; // Раскрыть все дерево
    finally
      Screen.Cursor:=crDefault; // Восстановить курсор
    end;
  finally
    TreeView.Items.EndUpdate; // Обновление дерева закончено, его можно перерисовать
  end;
end;

procedure TFileViewerForm.ActionExitExecute(Sender: TObject);
begin // Выход из программы
  Close;
end;
```

```
procedure TFileViewerForm.ActionAboutExecute(Sender: TObject);
begin // Краткая информация о программе
  ShowMessage('Программа просмотра файлов'#13'Авторское право (c) 2000 А.В. Скворцов');
end;

end.
```

Вопросы и задания для самостоятельной работы

1. Для чего предназначены действия в Delphi? Как они связываются с пунктами меню, кнопками и т.д.?
2. Почему нежелательно использование компонентов типов `TSpeedButton` и `TBitBtn`?
3. Как создаются изображения с прозрачными частями для `TImage-List`?
4. В чем заключается выгода от использования списка картинок `TImageList` в отличие от отдельных изображений?
5. Добавьте 4 кнопки для выбора режима просмотра (свойство `ViewStyle`) списка `ListView`.
6. Как создать перетаскиваемые панели инструментов?
7. Как добавить кнопки в панель инструментов `TToolbar`?
8. Как добавить колонки в `TListView`?
9. Для чего и как используются компоненты `TSplitter`?

Учебное издание

Алексей Владимирович Скворцов
Поддубная Тамара Николаевна

Панели инструментов, списки, деревья

(методические указания к
лабораторной работе № 7)

Томский государственный университет. Томск, 1999. – 20 с.