22.1 Introduction

This chapter shows the real power of Delphi. This comes from using the full power of Microsoft Windows, especially when using 32-bit operating systems such as Microsoft Windows 3.1 with Win32s, Microsoft Windows 95/98/NT/2000. These use Win32 which provides a standard programming model to allow a Windows program to run as a full 32-bit program. It also gives access to a great deal of advanced Windows functions (known as APIs – Application Programming Interface). Win32 has the following advantages:

- The ability to produce an application program which can be used with Microsoft Windows 95/98/NT/2000, and Windows 3.1x.
- Full OLE (object linking and embedding) support, including 16-bit/32-bit interoperability. OLE allows application programs to share data where an OLE server provides information for an OLE client.
- Improved performance with 32-bit operations.
- Access to a large amount of Win32 APIs (application programming interface) for Microsoft Windows 95/98/NT/2000 (such as Windows, Menus, Resources, Memory Management, Graphics, File Compression, and so on).
- Win32 semantics for the application programming interface (API).

Win32s is an operating system extension that allows Win32 applications for Microsoft Windows 95/98/NT/2000 to run on Windows version 3.1x. At the heart of Win32s is a virtual device driver (VxD) and a number of dynamic-link libraries (DLLs). These extend Windows 3.1x to support Win32-based applications. On Microsoft Windows 95/98/NT/2000 there is no need to distribute extra files, while Windows 3.x requires the installation of the Win32 files.

22.1.1 Win32 APIs

There are a great deal of extra APIs that can be used with Win32, these can be classified with:
• Windows operations (creating windows, scrolling bars, menus, dialog boxes, and so on).
• Graphics functions (painting, drawing, colours, palettes and so on).
• OLE and DDE (dynamic data exchange).
• TrueType fonts.
• Common dialogs.
• Network support (NetBios and Windows sockets 1.1 APIs).
• Multimedia support (sound APIs).
• File compression.
• Support for help files, bitmaps, icons and metafiles.
• Multiple document interface (MDI).

These Windows APIs are defined through the WinProcs unit.

### 22.2 Running Pascal programs

Delphi enhances Pascal in that it allows the use of components to enter and display data in a program. Delphi is also event driven. Functions and procedures in Pascal such as `Readln` and `Writeln` can also be supported. When a new project is standard the initial unit template that is created is of the form:

```
Program 22.1
unit Unit1;
interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls, Forms, Dialogs;

type
  TForm1 = class(TForm)
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

end.
```

244 Mastering Pascal
Functions such as Readln, Writeln, KeyPressed, ReadKey are defined in the System unit (or the WinCrt unit in Delphi 1). There is no need to include this in the unit list (but in Delphi 1 the WinCrt unit must be added). The example given next shows how they can be used (bold text highlights the changes from the standard template):

**Program 22.2**

```pascal
unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs;  {add WinCrt for Delphi 1}

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  voltage, current, resistance: real;
  ch: char;

implementation

{$R *.DFM}

begin

  writeln('Enter a voltage');
  readln(voltage);
  writeln('Enter a current');
  readln(current);
  resistance := voltage / current;
  writeln('Resistance is', resistance:8:2);
  writeln('Press any key to exit');
  ch := readkey;

end.

A sample run of this program is shown in Figure 22.1

Delphi 2 and later. In order to use a console window (to use writeln and readln), the Generate console application checkbox on the Linker page of the Project|Options dialog must be checked. Delphi 1. Programs which use writeln/readln require that the WinCrt unit is added in the uses list.
22.3 Message and dialog boxes

22.3.1 ShowMessage

The ShowMessage procedure displays a message box with an OK button. The Msg parameter is the message string that appears within the message box. The name of your application’s executable file appears as the caption of the message box. It is defined in the Dialogs unit and its syntax is:

```pascal
procedure ShowMessage(const Msg: string);
```

Program 22.3 shows a simple example where the ShowMessage procedure is used to display the message ‘Hello, Goodbye’. Figure 22.2 shows a sample run.

```pascal
Program 22.3

unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs;

type
  TForm1 = class(TForm)
    private
      \{ Private declarations }\n    public
```

246 Mastering Pascal
22.3.2 Message Box

The MessageBox function displays a message in a dialog box with specified buttons and then waits for the user to select a button. The value returned indicates the chosen button. The basic format is:

```
function MessageBox(Text: PChar; Caption: PChar; Flags: Word): Integer;
```

where:

- **Text** Is a string of characters which is the message to be displayed.
- **Caption** Is a string of characters which is the title of the dialog box.
- **Flags** A numeric value that is the sum of values that specifies the number, the type of buttons to display, the icon style and the default button. Table 22.1 outlines these values and if it is omitted then the default value for buttons is 0.

Table 22.1 defines the button settings. The values from 0 to 5 define the type of the button to be displayed. For example, a value of 5 will have two buttons, which are Retry and Cancel. The values 16, 32, 48 and 64 identify the icon to be displayed. For example, a value of 32 will display a question bubble. The values 0, 256 and 512 define which button is the default. Each of these values can be added together to create the required set of buttons, icons and default...
button. For example, to create a dialog box with the OK and Cancel buttons, a Critical icon and the Cancel button to be the default, then the setting would be:

\[ \text{setting} = 1 + 16 + 256 \]

which is 273. Note that to aid documentation in the program then the predefined constant values can be used, so for the previous example:

\[ \text{setting} = \text{MB_OKCANCEL} + \text{MB_ICONHAND} + \text{MB_DEFBUTTON2} \]

**Table 22.1** Button settings

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MB_OK</td>
<td>0</td>
<td>Display OK button only</td>
</tr>
<tr>
<td>MB_OKCANCEL</td>
<td>1</td>
<td>Display OK and Cancel buttons. See example 1 in Figure 22.3</td>
</tr>
<tr>
<td>MB_ABORTRETRYIGNORE</td>
<td>2</td>
<td>Display Abort, Retry, and Ignore buttons. See example 2 in Figure 22.3</td>
</tr>
<tr>
<td>MB_YESNOCANCEL</td>
<td>3</td>
<td>Display Yes, No, and Cancel buttons. See example 3 in Figure 22.3</td>
</tr>
<tr>
<td>MB_YESNO</td>
<td>4</td>
<td>Display Yes and No buttons. See example 4 in Figure 22.3</td>
</tr>
<tr>
<td>MB_RETRYCANCEL</td>
<td>5</td>
<td>Display Retry and Cancel buttons. See example 5 in Figure 22.3</td>
</tr>
<tr>
<td>MB_ICONHAND</td>
<td>16</td>
<td>Display Critical Message icon. See example 1 in Figure 22.4</td>
</tr>
<tr>
<td>MB_ICONQUESTION</td>
<td>32</td>
<td>Display Warning Query icon. See example 2 in Figure 22.4</td>
</tr>
<tr>
<td>MB_ICONEXCLAMATION</td>
<td>48</td>
<td>Display Warning Message icon. See example 3 in Figure 22.4</td>
</tr>
<tr>
<td>MB_ICONINFORMATION</td>
<td>64</td>
<td>Display Information Message icon. See example 4 in Figure 22.4</td>
</tr>
<tr>
<td>MB_DEFBUTTON1</td>
<td>0</td>
<td>First button is default</td>
</tr>
<tr>
<td>MB_DEFBUTTON2</td>
<td>256</td>
<td>Second button is default</td>
</tr>
<tr>
<td>MB_DEFBUTTON3</td>
<td>512</td>
<td>Third button is default</td>
</tr>
</tbody>
</table>
Figure 22.3 Buttons for MessageBox

Figure 22.4 Icons for MessageBox
The `MessageBox` function returns a value depending on the button pressed; these return values are outlined in Table 22.2. For example, if the user presses the OK button then the return value will be 1. If the dialog box has a Cancel button then the user pressing ESC has the same effect as choosing Cancel.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Button chosen</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDOK</td>
<td>1</td>
<td>OK</td>
</tr>
<tr>
<td>IDCANCEL</td>
<td>2</td>
<td>Cancel</td>
</tr>
<tr>
<td>IDABORT</td>
<td>3</td>
<td>Abort</td>
</tr>
<tr>
<td>IDRETRY</td>
<td>4</td>
<td>Retry</td>
</tr>
<tr>
<td>IDIGNORE</td>
<td>5</td>
<td>Ignore</td>
</tr>
<tr>
<td>IDYES</td>
<td>6</td>
<td>Yes</td>
</tr>
<tr>
<td>IDNO</td>
<td>7</td>
<td>No</td>
</tr>
</tbody>
</table>

Program 22.4 gives an example of a program which displays a dialog box with Yes and No buttons, and a question mark icon (the highlighted text shows the parts that have been added to the default unit text). The response will thus either be a 6 (if the Yes button is selected) or a 7 (if the No button is selected). Figure 22.5 shows a sample run.

```pascal
Program 22.4
unit Unit1;

interface

uses SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls, Forms, Dialogs;

type TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
end;

var Form1: TForm1;
  response:integer;
  style:integer;

implementation

{$R *.DFM}

begin
  style:=MB_YESNO + MB_ICONQUESTION;
```

Mastering Pascal
response:=Application.MessageBox('Press a button',
    'Button demonstration',style);
if (response=IDYES) then
    Application.MessageBox('Yes','Result',MB_ICONEXCLAMATION)
else
    Application.MessageBox('No','Result',MB_ICONEXCLAMATION);
end.

Figure 22.5  Example run

22.3.3  Message Dialog

The MessageDlg function displays a message dialog box in the centre of the screen. The message box displays the value of the Msg string constant. Its syntax is:

function MessageDlg(const Msg: string; AType: TMsgDlgType;
    AButtons: TMsgDlgButtons;   HelpCtx: Longint): Word;

Atype  Determines the type of message box that appears. These are the possible values as defined in Table 22.3.

Abuttons  Determines which buttons appear in the message box. The buttons that are to be displayed are included in brackets. These are defined in Table 22.4. For example to show the buttons Yes, No and OK then [mbYes, mbNo, mbOK] is used.

HelpCtx  Determines which Help screen is available for the message box.

Table 22.3  Atype values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>mtWarning</td>
<td>Message box containing a yellow exclamation point symbol</td>
</tr>
<tr>
<td>mtError</td>
<td>Message box containing a red stop sign</td>
</tr>
<tr>
<td>mtInformation</td>
<td>Message box containing a blue 'i'</td>
</tr>
<tr>
<td>mtConfirmation</td>
<td>Message box containing a green question mark</td>
</tr>
<tr>
<td>mtCustom</td>
<td>No bitmap</td>
</tr>
</tbody>
</table>

Delphi input/output  251
Table 22.4  Button values

<table>
<thead>
<tr>
<th>Value</th>
<th>Button type</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbYes</td>
<td>Yes (Delphi 1: Green check mark and the text ‘Yes’)</td>
</tr>
<tr>
<td>mbNo</td>
<td>No (Delphi 1: Red circle and slash mark through the circle and the text ‘No’)</td>
</tr>
<tr>
<td>mbOK</td>
<td>OK (Delphi 1: Green check mark and the text ‘OK’)</td>
</tr>
<tr>
<td>mbCancel</td>
<td>Cancel (Delphi 1: Red X and the text ‘Cancel’)</td>
</tr>
<tr>
<td>mbHelp</td>
<td>Help (Delphi 1: Cyan question mark and the text ‘Help’)</td>
</tr>
<tr>
<td>mbAbort</td>
<td>Abort (Delphi 1: Red check mark and the text ‘Abort’)</td>
</tr>
<tr>
<td>mbRetry</td>
<td>Retry (Delphi 1: Two green circular arrows and the text ‘Retry’)</td>
</tr>
<tr>
<td>mbIgnore</td>
<td>Ignore (Delphi 1: Green man walking away and the text ‘Ignore’)</td>
</tr>
<tr>
<td>mbAll</td>
<td>All (Delphi 1: Green double check marks and the text ‘All’)</td>
</tr>
</tbody>
</table>

The function returns the value of the button the user selected. Valid return values are:

mrNone  mrOk     mrCancel  mrAbort
mrRetry  mrIgnore  mrYes   mrNo
mrAbort  mrRetry   mrIgnore mrAll

As an example, place a button on a form, as shown in Figure 22.6. Change the caption on the button’s properties so that it is ‘Exit’. Next, either select the events tab from the Object Inspector or double click on the button. The screen in Figure 22.7 shows the unit window. Into here the code can be added. In this case the added code (which is highlighted) is:

```pascal
procedure TForm1.Button1Click(Sender: TObject);
begin
  if MessageDlg('Do you wish to exit?', mtInformation, [mbYes, mbNo], 0) = mrYes then
    begin
      MessageDlg('Exiting program', mtInformation, [mbOk], 0);
      Close;
    end;
end;
```

Figure 22.9 shows a sample run. It can be seen that the dialog box has two buttons, Yes and No. If the selected button is Yes then another dialog is shown with the message ‘Exiting program’.

252 Mastering Pascal
Figure 22.6  Example button

![Example button](Image)

Figure 22.7  Unit window

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   if MessageDlg('Do you wish to exit?', mtInformation, [mbYes, mbNo], 0) = mbYes then
       MessageDlg('Exiting program', mtInformation, [mbOk], 0);
end;
end.
```

Figure 22.8  Addition of code
22.4 Input dialog

22.4.1 Input Box

The InputBox function prompts the user either to input text or to choose a button. A value is then returned from the contents of the text box. Its syntax is:

```pascal
function InputBox(const ACaption, APrompt, ADefault: string): string;
```

where

- **ACaption** String of text which is displayed in the dialog box.
- **APrompt** Text that prompts the user to enter input in the edit box.
- **ADefault** String which is displayed in the text box and is the default response if no other input is provided. If this field is omitted then the text box is initially empty.

The InputBox function has a Cancel and an OK button. If the user chooses the OK button then the string in the edit box is the value returned, else the default value is return. The program thus does not know if the user has pressed the OK or the Cancel buttons (this is overcome with the InputQuery function).
Program 22.5 displays an input dialog box when the user clicks the button on the form. The input dialog box includes a prompt string and a default string. The string the user enters in the dialog box is stored in the `InputString` variable. Figure 22.10 shows an example run.

```
Program 22.5
unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var instr:string;
begin
  instr:=InputBox('Enter a value','Value','10');
  close;
end;
end.
```

22.4.2 Input Query

The `InputQuery` function is similar to `InputBox` but the button pressed is returned through the function header. Its syntax is:

```
function InputQuery(const ACaption, APrompt: string; var Value: string): Boolean;
```

A TRUE return indicates that the OK button was pressed, else a FALSE is returned. The entered string is returned from the `Value` string.
22.5 Exercises

22.5.1 Write a Delphi program using the `MessageDialog` function which displays the following dialog boxes (Note output taken from Delphi 1):
22.5.2 Using Pascal programs written in previous programs, modify them so that they run within Delphi.

22.5.3 Write a Delphi program in which the user enters their name and the program displays it in a dialog box, which asks the user if entered name is correct. Sample input and dialog boxes are shown next.

An outline of the code is given next.

```pascal
var
  Form1: TForm1;
  instr: string;

implementation
begin
  {$R *.DFM}
  instr:=InputBox('Enter your name','Name','fred');
  MessageDlg('Is our name '+instr+' ?', mtInformation, [mbYes, mbNo], 0)
end.
```

Delphi input/output  257