Delphi for Visual Basic Programmers

This version of Delphi includes a special set of productivity tools for Visual Basic programmers. Delphi offers the performance of a native code compiler with a powerful visual development environment. Delphi can be used with Visual Basic, or as a replacement for Visual Basic.

This help file includes:

Visual Basic commands to Delphi equivalents

Visual Basic Controls and Delphi Components

Component Pages Reference

Keystroke Mappings

Refer to your Delphi CD ROM for:

- Visual Basic to Delphi, A Technical Discussion (VB2Delphi.doc).
- A directory containing Visual Basic sample files translated into Delphi. (/VB/CODE)

Visual Basic command to Delphi equivalent

This Visual Basic to Delphi command reference includes all Visual Basic commands, the Delphi equivalent, descriptions of each Delphi command, the appropriate syntax for usage, and sample code. The reference is organized in alphabetical order by Visual Basic command. To use this list, select the appropriate Visual Basic command from following list:

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

#Const Directive

Delphi equivalent

```
{$...}
```

Description

Conditional directives control compilation of parts of the source text, based on evaluation of a symbol following the directive. You can define your own symbols or you can use the Object Pascal predefined symbols.

Conditional directive	Meaning
$DEFINE	Defines a conditional symbol
$ELSE	Compiles or ignores a portion of source text
$ENDIF	Ends the conditional section
$IFDEF	Compiles source text if Name is defined
$IFNDEF	Compiles source text if Name is NOT defined
$IFOPT	Compiles source text if a compiler switch is in a specified state (+ or -)
$UNDEF	Undefines a previously defined conditional symbol

Example

```
{$MINSTACKSIZE $00004000}
{$MAXSTACKSIZE $00100000}
{$IMAGEBASE $00400000}
{$APPTYPE GUI}
```

#If... Then ... #Else Directive

Delphi equivalent

`$If... $Else...$Endif Directive`

Description

Conditional directives produce different code from the same source text, based on the state of conditional symbols. Object Pascal identifiers cannot be used in conditional directives.

Note Changing your conditional defines should generally be followed by rebuilding your program.

There are two possible conditional constructs:

```
{$IFxxx} ... {$ENDIF}
      {$IFxxx} ... {$ELSE} ... {$ENDIF}
```

or

```
$IF ... $ENDIF
```

The $IFxxx ... $ENDIF construct compiles the source code between $IFxxx and $ENDIF only if the condition specified in $IFxxx evaluates to True.

If the condition is False, the source text between the two directives is ignored.

Example

```
{$ifdef BUILD_EXE}
program Hello_EXE
{$else}
library Hello_DLL
{$endif}
```

ALTER TABLE Statement (SQL)

Delphi equivalent
ALTER TABLE Statement (SQL)

Description
SQL Syntax

Example
The following example will Drop the LAST_NAME and FIRST_NAME fields while adding the FULL_NAME field to the employee DBASE Table.

```
ALTER TABLE "employee.dbf" DROP LAST_NAME, DROP FIRST_NAME, ADD FULL_NAME
CHAR[30]
```

Abs Function

Delphi equivalent
Abs Function

Declaration
```
function Abs(X);
```

Description
The Abs function returns the absolute value of the argument.

X is an integer-type or real-type expression.

Example
```
var
  r: Real;
  i: Integer;
begin
  r := Abs(-2.3); { 2.3 }
  i := Abs(-157); { 157 }
end;
```

AbsolutePosition Property

Delphi equivalent
RecNo Property

Declaration
```
property RecNo: Longint;
```

Description
Run-time and read-only. The RecNo property returns a record number for the current record in the dataset. Record numbers are only available for Paradox and dBASE tables; the concept isn't supported by most SQL servers so a table in a SQL database always returns -1 for RecNo.

Example
```
Num:= Table1.RecNo;     {Only on Paradox and dBASE Tables}
```

ActiveControl Property

Delphi equivalent
`ActiveControl`

Declaration
```
property ActiveControl: TWinControl;
```

Description
For forms, the ActiveControl property indicates which control has focus, or has focus initially when the form becomes active. Your application can use the ActiveControl property to access methods of the active control. Only one control, the active control, can have focus at a given time in an application.

Example
The following event handler responds to timer events by moving the active control one pixel to the right:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  ActiveControl.Left := ActiveControl.Left + 1;
end;
```

ActiveForm Property

Delphi equivalent
`ActiveForm`

Declaration
```
property ActiveForm: TForm;
```

Description
Run-time and read only. The ActiveForm property indicates which form currently has focus, or will have focus when the application becomes active again after another Windows application has been active.

Example
This example changes the color of the current form.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Screen.ActiveForm := clBlue;
end;
```

Add Method

Delphi equivalent
`Add Method`

Description
The Add method adds an item to a component.

Example
This code uses a button and a list box on a form. When the user clicks the button, the code adds a new string to a list box.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items.Add('New string');
end;
```

This code uses a list box, a button, and a label on a form. When the user clicks the button, the code adds a new string to the list box and reports its position in the list box as the caption of the label.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Position: Integer;
begin
  Position:= ListBox1.Items.Add('New item');
  Label1.Caption := IntToStr(Position);
end;
```

AddItem Method

Delphi equivalent
`Add Method; AddObject Method`

Description
The Add method adds an item to a component.

Example
The following code adds a new item at the top level of the outline. The new item is identified by the text 'New item':

```
Outline1.Add(0, 'New item');
```

The following code defines a record type of TMyRec and a record pointer type of PMyRec.

```
type
  PMyRec = ^TMyRec;
  TMyRec = record
    FName: string;
    LName: string;
  end;
```

Assuming these types are used, the following code adds an outline node to Outline1. A TMyRec record is associated with the added item. The FName and LName fields are obtained from edit boxes Edit1 and Edit2. The Index parameter is obtained from edit box Edit3. The item is added only if the Index is a valid value.

```
var
  MyRecPtr: PMyRec;
  OutlineIndex: LongInt;
begin
  New(MyRecPtr);
  MyRecPtr^.FName := Edit1.Text;
  MyRecPtr^.LName := Edit2.Text;
  OutlineIndex := StrToInt(Edit3.Text);
  if (OutlineIndex <= Outline1.ItemCount) and (OutlineIndex >= 0) then
    Outline1.AddObject(OutlineIndex, 'New item', MyRecPtr);
end;
```

After an item containing a TMyRec record has been added, the following code retrieves the FName and LName values associated with the item and displays the values in labels.

```
Label4.Caption := PMyRec(Outline1.Items[Outline1.SelectedItem].Data)^.FName;
Label5.Caption := PMyRec(Outline1.Items[Outline1.SelectedItem].Data)^.LName;
```

AddNew Method

Delphi equivalent

Insert Method

Declaration

```
procedure Insert;
```

Description

The Insert method puts the dataset into Insert state and opens a new, empty record at the current cursor location. When an application calls Post, the new record will be inserted in the dataset in a position based on its index, if defined. To discard the new record, use Cancel.

Example

```
with Table1 do
  begin
{ Move to the end ot the component }
  Last;
  Insert;
  FieldByName('CustNo').AsString := '9999';
  { Fill in other fields here }
  if { you are sure you want to do this} then Post
  else { if you changed your mind } Cancel;
  end.
```

AfterColUpdate Event

Delphi equivalent
`OnColExit Event`

Declaration
```
property OnColExit: TNotifyEvent;
```

Description
The OnColExit event occurs when the user uses the Tab key to move out of a column or clicks a cell in another column. Use the OnColExit event to specify any special processing you want to occur when exiting the column.

Example
The following code deletes the first two characters from the display label of the selected field when exiting a column. Note that FirstTime is a Boolean field that prevents characters from being deleted the first time a column is exited. Use this code in conjunction with code in the example of OnColEnter to modify the appearance of the display label of columns while they are entered.

```
procedure TForm1.DBGrid1ColExit(Sender: TObject);
var
  TheLabel: string;
begin
  if FirstTime then
    FirstTime := False
  else
  begin
    with DBGrid1.SelectedField do
    begin
      TheLabel := DisplayLabel;
      Delete(TheLabel, 1, 2);
      DisplayLabel := TheLabel;
    end;
  end;
end;
```

AfterDelete Event

Delphi equivalent
AfterDelete Event

Declaration
```
property AfterDelete: TDataSetNotifyEvent;
```

Description
The AfterDelete event is activated when the dataset finishes a call to the Delete method. This event is the last action before Delete returns to the caller. When AfterDelete is called, the deleted record has already been removed from the dataset, and the dataset cursor will be positioned on the following record.

Example
This example displays a message on the form's status bar indicating the table's record count after a record is deleted.

```
procedure TForm1.Table1AfterDelete(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := Format('There are now %d records in the table',
    [DataSet.RecordCount]);
end;
```

AfterInsert Event

Delphi equivalent
`AfterInsert Event`

Declaration
```
property AfterInsert: TDataSetNotifyEvent;
```

Description
The AfterInsert event is activated when a dataset finishes a call to the Insert or Append methods. This event is the last action before Insert or Append returns to the caller.

Example
This example updates the form's status bar with a message when an AfterInsert event occurs.

```
procedure TForm1.Table1AfterInsert(DataSet: TDataSet);
begin
   StatusBar1.SimpleText := 'Inserting new record';
end;
```

Align Property

Delphi equivalent
`Align Property`

Declaration
```
property Align: TAlign;
```

Description
The Align property determines how the controls align within their container (or parent control).

Example
The following code will change the alignment of a TDBMemo component when a button is clicked.

```
procedure Button1Click( Sender : TObject );
begin
  DBMemo1.Align := alClient;
end;
```

Alignment Property

Delphi equivalent
`Alignment Property`

Declaration
```
property Alignment: TLeftRight;
```

Description
Positions the caption to the right or left of a check box or radio button.

Example
This code aligns text to the right side of a label named Label1 in response to a click on a button named RightAlign:

```
procedure TForm1.RightAlignClick(Sender: TObject);
begin
  Label1.Alignment := taRightJustify;
end;
```

All, DISTINCT, DISTINCTROW, TOP Predicates (SQL)

Delphi equivalent
`All, DISTINCT Predicates (SQL)`

Description
SQL Syntax, select non unique or unique values from a table

Example
```
Select Distinct LastName, City from Customer
```

AllowDelete Property

Delphi equivalent
`AllowDelete Property`

Declaration
```
property AllowDelete: Boolean;
```

Description
If True (the default), the user can delete the current record by pressing Ctrl+Delete. (You can get the same effect in code by calling DoKey with the rkDelete parameter.) If False, TDBCtrlGrid won't delete records, though you can still do so by calling the attached dataset's Delete method.

Example
```
DBCtrlGrid.AllowDelete := TRUE;
```

AllowSizing Property

Delphi equivalent
`Options....dgColumnResize`

Declaration
```
property Options: TDBGridOptions;
```

Description
When True, the columns can be resized. A column can't be resized, however, until its field has been added to the grid. To add a field to the grid, choose Add from the Fields editor.

Example
This line of code displays column titles, makes the column indicator visible, and permits the user to edit the data displayed in the data grid:

```
procedure TForm1.FormClick( Sender: TObject );
begin
  DBGrid1.Options := [dgColumnResize, dgColLines, dgIndicator, dgEditing,
dgTitles];
end;
```

AllowUpdate Property

Delphi equivalent
`Options....dgEditing`

Declaration
```
property Options: TDBGridOptions;
```

Description
When True, allows the user to edit data in the data grid.

Example
This line of code displays column titles, makes the column indicator visible, and permits the user to edit the data displayed in the data grid:

```
procedure TForm1.FormClick( Sender: TObject );
begin
  DBGrid1.Options := [dgColumnResize, dgColLines, dgIndicator, dgEditing,
dgTitles];
end;
```

AppActivate Statement

Delphi equivalent
ShowWindow (API)

Description
See Win32.hlp

Example
The following code will activate the NotePad Application.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  h : HWnd;
begin
  h := findWindow( nil, 'Untitled - Notepad' );
  showWindow( h, SW_SHOWNORMAL );
end;
```

Appearance Property

Delphi equivalent
CTL3D Property

Declaration
```
property Ctl3D: Boolean;
```

Description
The Ctl3D property determines whether a control has a three-dimensional (3-D) or two-dimensional look. If Ctl3D is True, the control has a 3-D appearance. If Ctl3D is False, the control appears normal or flat. The default value of Ctl3D is True.

Example
The following code toggles the 3-D look of a memo control when the user clicks a button named Toggle:

```
procedure TForm1.ToggleClick(Sender: TObject);
begin
  Memo1.Ctl3D := not Memo1.Ctl3D; {Toggles the Ctl3D property of Memo1}
end;
```

Append Method

Delphi equivalent

Append Method

Declaration

```
procedure Append(const S: string);
```

Description

The Append method performs the same function as the Add method for strings and string lists but doesn't return the result. Use this method rather than Add when you need it as a parameter for a function requiring a TGetStrProc or when you don't need to know where the string is added.

Example

This example appends a new record to a table when the user clicks a button. The two fields ALPHANUMERIC and INTEGER are filled from the contents of two edit controls.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Append;
  Table1['ALPHANUMERIC'] := Edit1.text;
  Table1['INTEGER'] := StrToInt(Edit2.text);
  Table1.Post;
end;
```

AppIsRunning Property

Delphi equivalent
`DoVerb Method`

Declaration
```
procedure DoVerb(Verb: Integer);
```

Description
Requests the OLE object to perform some action. OLE defines several verbs, such as ovShow (to display the OLE object) and ovPrimary (the default action, usually to activate the OLE object). OLE objects can define their own custom verbs. You can use the ObjectVerbs property to get a list of those custom verbs.

Example
```
OleContainer1.DoVerb( ovPrimary );
```

Archive, Hidden, Normal, System Properties

Delphi equivalent

`FileType Property`

Declaration

```
property FileType: TFileType;
```

Description

The FileType property determines which files are displayed in the file list box based on the attributes of the files. Because FileType is of type TFileType, which is a set of file attributes, FileType can contain multiple values. For example, if the value of FileType is a set containing the values ftReadOnly and ftHidden, only files that have the read-only and hidden attributes are displayed in the list box. Refer to the main Delphi help system for FileType values.

Example

This example uses a file list box on a form. When the application runs, only read-only files, directories, volume IDs, and files with no attributes appear in the list box.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileType := [ftReadOnly, ftDirectory, ftVolumeID, ftNormal];
end;
```

Arrange Method

Delphi equivalent

`Cascade, Tile and ArrageIcons`

Declaration

```
procedure Cascade; Tile; ArrangeIcons
```

Description

The Cascade method rearranges the child forms in your application so they overlap. The Tile method arranges the child forms of a parent form in your application so that the forms are all the same size.

Example

This code arranges all MDI children of the current MDI parent form in a cascade pattern when the user chooses the Cascade menu command:

```
procedure TForm1.Cascade1Click(Sender: TObject);
begin
  Cascade;
end;
```

Array Function

Delphi equivalent
`VarArrayOf`

Declaration
```
function VarArrayOf(const Values: array of Variant): Variant;
```

Description
The VarArrayOf function returns a one-dimensional variant array with the elements given by the Values parameter. The low bound of the returned array is zero, the high bound is the number of values given by the Values parameter less one, and the element type is Variant.

Asc Function

Delphi equivalent
`Ord`

Description:
Returns the element's numerical ordering within the set.

Example

```
uses Dialogs;
type
   Colors = (RED,BLUE,GREEN);
var
  S: string;
 begin
   S := 'BLUE has an ordinal value of ' + IntToStr(Ord(BLUE)) + #13#10;
   S := 'The ASCII code for "c" is ' + IntToStr(Ord('c')) +  ' decimal';
   MessageDlg(S, mtInformation, [mbOk], 0);
 end;
```

Atn Function

Delphi equivalent

ArcTan Function

Declaration

```
function ArcTan(X: Extended): Extended;
```

Description

The ArcTan function returns the resulting arctangent of the argument.

Example

```
var
  R: Extended;
begin
  R := ArcTan(Pi);
end;
```

Attributes Property

Delphi equivalent

`TableDefs and FieldDefs`

Declaration

```
property FieldDefs: TFieldDefs;
```

Description

Run-time only. The FieldDefs property holds information about each TFieldDef in the dataset. You can use this property to determine which fields are in the dataset, their name, type, and size.

AutoActive Property

Delphi equivalent

`AutoActivate Property`

Declaration

```
TAutoActivate = (aaManual, aaGetFocus, aaDoubleClick);
property AutoActivate: TAutoActivate;
```

Description

The AutoActivate property determines how an object in an OLE container can be activated, as defined in the following table:

Values

aaManual	The user can't activate the object. You can activate the OLE object in code by calling DoVerb(ovShow).
aaGetFocus	The OLE object is activated whenever the OLE container gets the focus (by clicking on it with the mouse or pressing Tab to move the focus to it).
aaDoubleClick	(Default) The OLE object is activated by doubleclicking it or pressing Enter while the container has the focus.

Example

```
OleContainer1.AutoActivate := aaGetFocus;
```

AutoRedraw Property

Delphi equivalent
`Invalidate Method`

Declaration
```
procedure Invalidate;
```

Description
The Invalidate method forces a control to repaint as soon as possible.

Example
The following code invalidates Form1.

```
Form1.Invalidate;
```

AutoShowChildren Property

Delphi equivalent

`Project | Options | Forms`

Description:

Use the Forms page of the Project Options dialog box to select the main form for your applications, and to choose which of the available forms are automatically created and in which order.

AutoSize Property

Delphi equivalent
`AutoSize Property`

Declaration
```
property AutoSize: Boolean;
```

Description
When the AutoSize property is True, the image control resizes to accommodate the image it contains (specified by the Picture property). When AutoSize is False, the image control remains the same size, regardless of the size of the image. If the image control is smaller than the image, only the portion of the picture that fits inside the image component will be visible.

The default value is False.

Example
The following code keeps the size of the label control constant, even though the length of the label's caption changes. As a result, the caption of the label is probably too long to display in the label when the user clicks the button:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption := 'This string is too long as the caption of this label';
end;
```

AutoVerbMenu Property

Delphi equivalent
`AutoVerbMenu Property`

Declaration
```
property AutoVerbMenu: Boolean;
```

Description
Determines whether TOleContainer automatically creates a popup menu containing the OLE object's verbs. If AutoVerbMenu is True (the default), TOleContainer replaces any existing PopupMenu property. If AutoVerbMenu is False, no popup menu is automatically created.

Example
```
OleContainer1.AutoVerbMenu := TRUE;
```

Avg Function (SQL)

Delphi equivalent

Avg Function (SQL)

Description

SQL Syntax

Example

```
Select Avg( Total ) from "Sales.db"
```

BackColor, ForeColor Properties

Delphi equivalent
`Color, Font.Color Properties`

Declaration
```
property Color: TColor;
```

Description
For all components or objects except the Color dialog box, the Color property determines the background color of a form or the color of a control or graphics object.

Example
This code will set the label's color and font color to the current windows settings.

```
Label1.Color := clWindow;
Label1.Font.Color := clWindowText;
```

BackStyle Property

Delphi equivalent
`Transparent Property`

Declaration
```
property Transparent: Boolean;
```

Description
The Transparent property determines if a label or database text control is transparent. You could place a transparent label or text control on top of a bitmap, and the control won't hide part of the bitmap. For example, if you have placed a bitmap of the world on a form, you could label the South American continent with a label control, and you would still see the continent in the label space.

Example
This code makes a label transparent:

```
Label1.Transparent := True;
```

Beep Statement

Delphi equivalent
```
Beep
```

Declaration
```
procedure Beep;
```

Description
The Beep procedure calls the Windows API MessageBeep with a parameter of zero.

Example
This code will cause the machine to beep when Button1 is pressed.

```
Procedure TForm1.Button1Click( Sender : TObject );
begin
  beep;
end;
```

Before Delete Event

Delphi equivalent
`BeforeDelete Event`

Declaration
```
property BeforeDelete: TDataSetNotifyEvent;
```

Description
The BeforeDelete event is activated when the dataset begins a call to Delete. This event is the first action taken by the Delete method.

By assigning a method to this property, you can take any special actions required by the event. By raising an exception in this event handler, you can prevent the Delete operation from occurring.

Example
The following code will delete the child records in the linked Table2 prior to deleting the master record.

```
procedure TForm3.Table1BeforeDelete(DataSet: TDataSet);
var
 i : integer;
begin
  for i := 0 to Table2.RecordCount - 1 do
  begin
    Table2.Delete;
  end;
end;
```

Before Update Event

Delphi equivalent

```
OnEnter Event
```

Declaration

```
property OnEnter: TNotifyEvent;
```

Description

The OnEnter event occurs when a component becomes active. Use the OnEnter event handler to specify any special processing you want to occur when a component becomes active.

BeforeColUpdate Event

Delphi equivalent
OnColEnter Event

Declaration
```
property OnColEnter: TNotifyEvent;
```

Description
The OnColEnter event occurs when the user clicks a cell in a column or moves to a column with the Tab key within the data grid. Use the OnColEnter event to specify any processing you want to occur as soon as a column is entered.

Example
The following code concatenates an asterisk to the display label of a field when the column is entered.

```
procedure TForm1.DBGrid1ColEnter(Sender: TObject);
begin
  with DBGrid1.SelectedField do
    DisplayLabel := '* ' + DisplayLabel;
end;
```

BeforeInsert Event

Delphi equivalent
`BeforeInsert Event`

Declaration
```
property BeforeInsert: TDataSetNotifyEvent;
```

Description
The BeforeInsert event is activated when the dataset begins a call to the Insert or Append methods. This event is the first action taken by Insert or Append.

By assigning a method to this property, you can take any special actions required by the event. By raising an exception (such as by calling the Abort procedure) in this event handler, you can prevent the Insert operation from occurring.

Example
This example uses the BeforeInsert event to do data validation; if the StrToInt function raises an exception, the edit control's contents are set to a valid value so the assignment to the INTEGER field in the table will succeed.

```
procedure TForm1.Table1BeforeInsert(DataSet: TDataSet);
begin
  try
    {Make sure edit field can be converted to integer --
     will throw an exception if it can't }
    StrToInt(Edit1.Text);
  except
    Edit1.Text := '1000';
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Insert;
  Table1.FieldByName('INTEGER').AsInteger := StrToInt(Edit1.Text);
  Table1.Post;
end;
```

BeginTrans, CommitTrans, Rollback Methods

Delphi equivalent

`StartTransaction, Commit, Rollback Methods`

Declaration

```
procedure StartTransaction;
```

Description

The StartTransaction method begins a transaction at the isolation level specified by the TransIsolation property. If a transaction is currently active, Delphi raises an exception.

Modifications made to the database are not stored permanently until the Commit method is called to commit the changes. Modifications are discarded if the Rollback method is called to cancel the changes.

Example

```
with Database1 do
  begin
    StartTransaction;
  { Update one or more records in tables linked to Database1 }
...
    Commit;
  end;
```

BeginTrans, CommitTrans, Rollback Statements

Delphi equivalent

`StartTransaction, Commit, Rollback Methods`

Declaration

```
procedure StartTransaction;
```

Description

The StartTransaction method begins a transaction at the isolation level specified by the TransIsolation property. If a transaction is currently active, Delphi raises an exception.

Modifications made to the database are not stored permanently until the Commit method is called to commit the changes. Modifications are discarded if the Rollback method is called to cancel the changes.

Example

```
with Database1 do
  begin
    StartTransaction;
  { Update one or more records in tables linked to Database1 }
...
    Commit;
  end;
```

Between...And Operator (SQL)

Delphi equivalent
Between...And Operator (SQL)

Description
SQL Syntax

Example

```
Select * from Orders
where OrderDate Between '08/01/96' and '08/31/96'
```

Bof, EOF Properties

Delphi equivalent
EOF Properties

Declaration
```
property EOF: Boolean;
```

Description
Run-time and read-only. BOF is a Boolean property that indicates whether a dataset is known to be at its first row.EOF is a Boolean property that indicates whether a dataset is known to be at its last row. The EOF property returns a value of True after:

An application opens an empty dataset
A call to a table's Last method
A call to a table's Next fails because the cursor is on the last row

Example
This example uses TDataSource's OnDataChange event to detect when the user moves to another record. If the end of file is reached (EOF property becomes True), a message is displayed on the form's status bar.

```
procedure TForm1.DataSource1DataChange(Sender: TObject; Field: TField);
begin
  if Table1.EOF then
    StatusBar1.SimpleText := 'You''re already at the end of the table';
end;
```

Bold Property

Delphi equivalent

`Style Property`

Declaration

```
property Style: TFontStyles;
```

Description

The Style property determines whether the font is normal, italic, underlined, bold, and so on. See Delphi.hlp for further details on the Style property

Example

The following code boldfaces the font used in the memo..

```
Memo1.Font.Style := [fsBold];
```

Bookmark Property

Delphi equivalent

```
Bookmark Property
```

Declaration

```
property Bookmark: TBookmarkStr;
```

Description

Run-time only. The Bookmark property returns a bookmark for the current row. Assigning to the property moves the cursor to the record corresponding to the bookmark.

BorderColor Property

Delphi equivalent
`BorderColor Property`

Declaration
```
property BorderColor: TColor;
```

Description
The BorderColor property is used to color the border of a shape component. For a complete list of the values the BorderColor property can have, see the Color property.

Example
This example changes the border color of a shape component at run time:

```
Shape1.BorderColor := clBlack;
```

BorderStyle Property

Delphi equivalent
`BorderStyle Property`

Applies to
TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupList, TDBMemo, TDrawGrid, TEdit, TForm, THeader, TListBox, TListView, TMaskEdit, TMemo, TOutline, TPanel, TRichEdit, TScrollBox, TStringGrid, TTreeView components

Description
The BorderStyle property controls the kind of border the component displays.

Example
This example creates a form with a single-line border that the user can't resize:

```
Form1.BorderStyle := bsSingle;
```

BorderWidth Property

Delphi equivalent

`BorderWidth Property`

Declaration

```
property BorderWidth: TBorderWidth;
```

Description

The BorderWidth property determines the width in pixels of the border around a panel. The default value is 0, which means no border.

Example

This example uses a panel component and a button named CreateStatusLine on a form. The code moves the panel to the bottom of the form when the user clicks the button, and gives the panel the appearance of a status line by changing the value of the BevelInner, BevelOuter, BevelWidth, and BorderWidth properties:

```
procedure TForm1.CreateStatusLineClick(Sender: TObject);
begin
  with Panel1 do
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

BoundText Property

Delphi equivalent

`TField.value Property`

Declaration

```
property Value: Variant      {All field components}
property Value: string;      {TStringField, TBlobField}
property Value: Longint;     {TAutoIncField, TIntegerField, TSmallintField,
TWordField}
property Value: Double;      {TBCDField, TCurrencyField, TFloatField}
property Value: Boolean;     {TBooleanField}
property Value: TDateTime    {TDateField, TDateTimeField, TTimeField}
```

Description

Run-time only. Value is the actual data in a TField. Use Value to read data directly from and write data directly to a TField.

Example

The following expression will return the value of the field associated with the DBEdit control.

```
( Sender As TDBText).DataSource.DataSet.fieldbyname( (Sender As
TDBText).DAtaField ).Value
```

CREATE INDEX Statement (SQL)

Delphi equivalent
CREATE INDEX Statement (SQL)

Description
SQL Syntax

Example
```
CREATE INDEX index_name ON table_name (column [, column ...])
```

Using CREATE INDEX is the only way to create indexes for dBASE tables. For example, the following statement creates an index on a dBASE table:

```
CREATE INDEX NAMEX ON "employee.dbf" (LAST_NAME)
```

Paradox users can create only secondary indexes with CREATE INDEX. Primary Paradox indexes can be created only by specifying a PRIMARY KEY constraint when creating a new table with CREATE TABLE.

CREATE TABLE Statement (SQL)

Delphi equivalent
CREATE TABLE Statement (SQL)

Description
SQL Syntax

Example
For example, the following statement creates a Paradox table with a PRIMARY KEY constraint on the LAST_NAME and FIRST_NAME columns:

```
CREATE TABLE "employee.db"
      (
      LAST_NAME CHAR(20),
      FIRST_NAME CHAR(15),
      SALARY NUMERIC(10,2),
      DEPT_NO SMALLINT,
      PRIMARY KEY(LAST_NAME, FIRST_NAME)
      )
```

Cancel Property

Delphi equivalent

```
Cancel Property
```

Declaration

```
property Cancel: Boolean;
```

Description

The Cancel property indicates whether a button or a bitmap button is a Cancel button. If Cancel is True, any time the user presses Esc, the OnClick event handler for the button executes. Although your application can have more than one button designated as a Cancel button, the form calls the OnClick event handler only for the first button in the tab order that is visible.

Example

The following code designates a button called Button1 as a Cancel button:

```
Button1.Cancel := True;
```

CancelUpdate Method

Delphi equivalent
`CancelUpdates Method`

Declaration
```
procedure CancelUpdates;
```

Description
CancelUpdates discards all pending cached updates. CancelUpdates is always successful, so no errors will occur. The dataset returns to the state it was at before cached updates were enabled.

Example
```
Table1.CancelUpdates;
```

Caption Property

Delphi equivalent

Caption Property

Declaration

```
property Caption: string;
```

Description

The Caption property is the text that appears in the form's title bar; this text also appears as the icon label when the form is minimized. For components other than forms, the Caption property contains the text string that labels the component. To underline a character in a string, include an ampersand (&) before the character.

Example

This code changes the caption of a group box:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  GroupBox1.Caption := 'Fancy options';
end;
```

Cbool Function

Delphi equivalent
`AsBoolean`

Declaration
```
property AsBoolean: Boolean;
```

Description
Run-time only. This is a conversion property. For a TBooleanField, AsBoolean can be used to read or set the value of the field, but Value should be used for this purpose instead.

Cbyte Function

Delphi equivalent
`Byte`

Description
See Delphi.hlp BYTE

Ccur Function

Delphi equivalent
`Format Strings`

Description
Set Format string to m to display money. Format strings passed to the string formatting routines contain two types of objects--plain characters and format specifiers. Plain characters are copied verbatim to the resulting string. Format specifiers fetch arguments from the argument list and apply formatting to them.

Cdate Function

Delphi equivalent
`StrToDate`

Declaration
```
function StrToDate(const S: string): TDateTime;
```

Description
The StrToDate function converts a string to date format. The date in the string must be a valid date.

Cdbl Function

Delphi equivalent
`AsFloat`

Declaration
```
property AsFloat: Double;
```

Description
Run-time only. This is a conversion property. For a TFloatField, TBCDField or TCurrencyField, AsFloat can be used to read or set the value of the field as a Double, but Value should be used for this purpose instead.

For a TStringField, AsFloat converts a float to a string on assigning a value to the field, and converts a string to a float when reading from the field.

CellText Method

Delphi equivalent
Cells Property

Declaration
```
property Cells[ACol, ARow: Integer]: string;
```

Description
Run-time only. The Cells property is an array of strings, one string for each cell in the grid. Use the Cells property to access a string within a particular cell. ACol is the column coordinate of the cell, and ARow is the row coordinate of the cell. The first row is row zero, and the first column is column zero.

Example
This code fills each cell of a grid with the same string.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J: Integer;
begin
  with StringGrid1 do
    for I := 0 to ColCount - 1 do
      for J:= 0 to RowCount - 1 do
        Cells[I,J] := 'Delphi';
end;
```

CellValue Method

Delphi equivalent
Cells Property

Declaration
```
property Cells[ACol, ARow: Integer]: string;
```

Description
Run-time only. The Cells property is an array of strings, one string for each cell in the grid. Use the Cells property to access a string within a particular cell. ACol is the column coordinate of the cell, and ARow is the row coordinate of the cell. The first row is row zero, and the first column is column zero.

Example
This code fills each cell of a grid with the same string.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J: Integer;
begin
  with StringGrid1 do
    for I := 0 to ColCount - 1 do
      for J:= 0 to RowCount - 1 do
        Cells[I,J] := 'Delphi';
end;
```

ChDir Statement

Delphi equivalent
ChDir

Declaration
```
procedure ChDir(S: string);
```

Description
The ChDir procedure changes the current directory to the path specified by S.

If S specifies a drive letter, the current drive is also changed.

Example
```
begin
  {$I-}
  { Change to directory specified in Edit1 }
  ChDir(Edit1.Text);
  if IOResult <> 0 then
    MessageDlg('Cannot find directory', mtWarning, [mbOk], 0);
end;
```

ChDrive Statement

Delphi equivalent
`ChDir`

Declaration
```
procedure ChDir(S: string);
```

Description
The ChDir procedure changes the current directory to the path specified by S.

If S specifies a drive letter, the current drive is also changed.

Change Event

Delphi equivalent
`OnChange Event`

Applies to
TBitmap, TBrush, TCanvas, TFont, TGraphic, TGraphicsObject, TMetafile, TPen, TPicture, TStringList, TConversion objects; TComboBox, TDirectoryListBox, TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TMaskEdit, TMemo, TPageControl, TRichEdit, TScrollBar, TTrackBar, components

Declaration
```
property OnChange: TNotifyEvent;
```

Description
The OnChange event specifies which event handler should execute when the contents of a component or object changes.

For graphics objects, OnChange occurs when the specific graphics item encapsulated by the object changes. For example, the OnChange event for a pen occurs when the Color, Mode, Style, or Width properties of the TPen object are modified.

For components, OnChange occurs when the main value or values of the component are modified. For example, OnChange occurs when the Text property of an edit box is modified.

For combo boxes, the OnChange event also occurs when an item is selected in the drop down list.

For string list objects, the OnChange event occurs when a change to a string stored in the list of strings changes.

Example
This example uses a color grid on a form. The color grid is a component on the Samples page of the Component palette. When the user clicks a color rectangle or drags the mouse cursor across the color grid, the color of the form changes.

```
procedure TForm1.ColorGrid1Change(Sender: TObject);
begin
  Color := ColorGrid1.ForegroundColor;
end;
```

CheckBox Control

Delphi equivalent
`TCheckBox Component`

Description
See Standard Page Components

Checked Property

Delphi equivalent
`Checked Property`

Declaration
```
property Checked: Boolean;
```

Description
Run-time only. The Checked property determines whether an option is selected.

Example
This example fills in a radio button at run time:

```
RadioButton1.Checked := True;
```

This example uses a main menu component that contains a menu item named SnapToGrid1 on a form. When the user chooses the Snap To Grid command, a check mark appears next to the command. When the user chooses the Snap To Grid command again, the check marks disappears:

```
procedure TForm1.SnapToGrid1Click(Sender: TObject);
begin
  SnapToGrid1.Checked := not SnapToGrid1.Checked;
end;
```

Choose Function

Delphi equivalent

```
Use Index of TStringList Object
```

Description

The TStringList object maintains a list of strings. Use a string list object when you are managing a list of strings that is not maintained by a control.

You can add, delete, insert, move, and exchange strings using the Add, Append, Delete, Insert, Move, and Exchange methods. The Clear method clears all the strings in the list of strings. The Count property contains the number of strings in the list. Each string list object has a Strings property that lets you access a particular string by its position in the list of strings. To find the position of a string in the list, use the IndexOf method.

Chr Function

Delphi equivalent

Chr Function

Declaration

```
function Chr(X: Byte): Char;
```

Description

The Chr function returns the character with the ordinal value (ASCII value) of the byte-type expression, X.

Example

```
begin
  Canvas.TextOut(10, 10, Chr(65));  { The letter 'A'}
end;
```

Cint Function

Delphi equivalent
`AsInteger`

Declaration
```
property AsInteger: Longint;
```

Description
Run-time only. This is a conversion property. For a TIntegerField, TSmallintField or TWordField, AsInteger can be used to read or set the value of the field as a Longint, but Value should be used for this purpose instead.

For a TStringField, AsInteger converts an integer to a string on assigning a value to the field, and converts a string to an integer when reading from the field.

Run-time only. This is a conversion property. For a TIntegerField, TSmallintField or TWordField, AsInteger can be used to read or set the value of the field as a Longint, but Value should be used for this purpose instead.

For a TStringField, AsInteger converts an integer to a string on assigning a value to the field, and converts a string to an integer when reading from the field.

Circle Method

Delphi equivalent
`Ellipse Method`

Declaration
```
procedure Ellipse(X1, Y1, X2, Y2: Integer);
```

Description
The Ellipse method draws an ellipse defined by a bounding rectangle on the canvas. The top left point of the bounding rectangle is at pixel coordinates (X1, Y1) and the bottom right point is at (X2, Y2). If the points of the rectangle form a square, a circle is drawn.

Example
The following code draws an ellipse filling the background of a form:

```
procedure TForm1.FormPaint(Sender: TObject);
begin
  Canvas.Ellipse(0, 0, ClientWidth, ClientHeight);
end;
```

Class Property

Delphi equivalent
`OLEClassName Method`

Declaration
```
property OleClassName: string;
```

Description
Runtime and readonly. Returns the class name of the OLE object. An OLE object must already be loaded in the container before accessing the OleClassName property.

Example
```
Caption := OleContainer1.OLEClassName;
```

ClassModule Object

Delphi equivalent

```
Type section in a Unit
```

Description

A type Declaration specifies an identifier that denotes a type. A variable's type defines the set of values it can have and the operations that can be performed on it.

See Delphi.hlp UNIT

Example

```
Type
  MyObject = Class( TObject );
```

Clear Method

Delphi equivalent
`Clear Method`

Applies to
TClipboard, TCollection, TFieldDefs, THeaderSections, TIndexDefs, TList, TListItems, TParam, TParams, TStatusPanels, TStringList, TStrings, TDBGridColumns, TListColumns objects; TBCDField, TBlobField, TBooleanField, TBytesField, TComboBox, TDBComboBox, TCurrencyField, TDateField, TDateTimeField, TDBEdit, TDBListBox, TDBMemo, TDirectoryListBox, TDriveComboBox, TEdit, TFileListBox, TFilterComboBox, TFloatField, TGraphicField, TImageList, TIndexDefs, TIntegerField, TListBox, TMaskEdit, TMemo, TMemoField, TOutline, TSmallintField, TStringField, TTimeField, TVarBytesField, TWordField components

Description
See Delphi.hlp for more information

Example
The following code will clear the contents of the field named date.

```
Table1.FieldByName( 'Date' ).Clear;
```

Clear Method (Clipboard, ComboBox, ListBox)

Delphi equivalent

`Clear Method (Clipboard, ComboBox, ListBox)`

Declaration

```
procedure Clear;
```

Description

The Clear method deletes all text from the control, or, in the case of collection, list and string objects or outlines, deletes all items. For the Clipboard object, Clear deletes the contents of the Clipboard; this happens automatically each time data is added to the Clipboard (cut and copy operations).

Example

The following code removes the text from an edit box control called NameField:

```
NameField.Clear;
```

This example uses a list box and a button on a form. When the form is created, strings are added to the list box. When the user clicks the button, all the strings contained in the Items property, a TStrings object, are cleared.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('One');
  ListBox1.Items.Add('Two');
  ListBox1.Items.Add('Three');
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items.Clear;
end;
```

Click Event

Delphi equivalent
`OnClick Event`

Declaration
```
procedure Click;
```

Description
The Click method simulates a mouse click, as if the user had clicked a menu item or button, executing any code attached to the OnClick event.

Example
The form in this example changes color each time the user clicks it:

```
procedure TForm1.FormClick(Sender: TObject);
begin
  Randomize;
  Color := Random(65535);
end;
```

Clip Property

Delphi equivalent
`Cells[]`

Declaration
```
property Cells[ACol, ARow: Integer]: string;
```

Description
Run-time only. The Cells property is an array of strings, one string for each cell in the grid. Use the Cells property to access a string within a particular cell. ACol is the column coordinate of the cell, and ARow is the row coordinate of the cell. The first row is row zero, and the first column is column zero.

Example
This code fills each cell of a grid with the same string.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J: Integer;
begin
  with StringGrid1 do
    for I := 0 to ColCount - 1 do
      for J:= 0 to RowCount - 1 do
        Cells[I,J] := 'Delphi';
end;
```

Clipboard Object Clipboard Object

Delphi equivalent

`Clipboard Object`

Description

The TClipboard object encapsulates the Windows Clipboard. Whenever you cut, copy, or paste text or graphics objects within a Delphi application or between a Delphi application and another Windows application, you are using the TClipboard object.

The Clipbrd unit declares the variable Clipboard as an instance of TClipboard. Use the Clipboard function instead of creating your own instance of TClipboard.

You can place text in and retrieve text from the Clipboard using the AsText property. If you want to place pictures in and retrieve pictures from the Clipboard, use the Assign property. To add or retrieve a component object to the Clipboard, call the GetComponent and SetComponent methods.

Close Method

Delphi equivalent
Close Method

Declaration
```
procedure Close;
```

Description
The Close method closes the object. See Delphi.hlp for application.

Example
The following method closes a form when a button called Done is clicked:

```
procedure TForm1.DoneButtonClick(Sender: TObject);
begin
  Close;
```

Close Method (OLE Container)

Delphi equivalent
`Close Method`

Declaration
```
procedure Close
```

Description
Deactivates the OLE object and terminates its server application, but doesn't remove it from the container. Any changes the user made to the OLE object are automatically saved. There must be an OLE object in the container before calling Close. After calling Close, the container's State property is osLoaded.

Example
```
OleContainer1.Close;
```

Close Statement

Delphi equivalent

CloseFile Procedure

Declaration

```
procedure CloseFile(var F);
```

Description

Due to naming conflicts, the CloseFile procedure replaces the Borland Pascal Close procedure. Use the CloseFile procedure instead of Close to terminate the association between the file variable and an external disk file.

F is a file variable of any file type opened using Reset, Rewrite, or Append. The external file associated with F is completely updated and then closed, freeing the file handle for reuse.

Example

```
var
  F: TextFile;
begin
  if OpenDialog1.Execute then  { Bring up open file dialog }
  begin
    AssignFile(F, OpenDialog1.FileName);
            { File selected in dialog }
    Reset(F);
    Edit1.Text := IntToStr(FileSize(F);
        { Put file size string in a TEdit control }
    CloseFile(F);   { Close file }
  end;
end;
```

Cls Method

Delphi equivalent
`TCanvas.FillRect`

Declaration
```
procedure FillRect(const Rect: TRect);
```

Description
The FillRect method fills the specified rectangle on the canvas using the current brush.

Example
The following code will fill Form1's canvas with the current brush color when Button1 is clicked

```
Procedure TForm1.Button1Click( Sender : TObject );
begin
  Canvas.Brush.Color := Color;
  Canvas.FillRect( Canvas.ClipRect );
end;
```

Col, Row Properties

Delphi equivalent
`Col, Row Properties`

Declaration
```
property Col: Longint;
```

Description
Run-time only. The value of the Col property indicates the current column of the cell that has input focus. You can use the Col property along with the Row property to determine which cell is selected at run time.

Example
The following line of code adds the string 'Hello' to the end of the list of strings in column four of the string grid named StringGrid1:

```
StringGrid1.Cols[3].Add('Hello');
```

ColAlignment Property

Delphi equivalent
`TField.Alignment Property`

Declaration
```
property Alignment: TAlignment;
```

Description
The Alignment property is used by some data-aware controls to center, left-, or right-align the data in a field. Data-aware controls that support alignment include TDBGrid and TDBEdit.

Example
The following line of code will center the display of the Name field.

```
Table1.FieldByName( 'Name' ).Alignment := taCenter;
```

ColContaining Method

Delphi equivalent

```
Use ColWidths Property (Protected)
```

Declaration

```
property ColWidths[Index: Longint]: Integer;
```

Description

Run-time only. The ColWidths property determines the width in pixels of all the cells within the column referenced by the Index parameter.

By default, all the columns are the same width, the value of the DefaultColWidth property. To change the width of all columns within a grid, change the DefaultColWidth property value.

To change the width of one column without affecting others, change the ColWidths property. Specify the column you want to change as the value of the Index parameter. Remember the first column always has an Index value of 0.

ColIndex Property

Delphi equivalent
ColumnMoved Method (Protected)

Declaration
```
procedure ColumnMoved(FromIndex, ToIndex: Longint); dynamic;
```

Description
The ColumnMoved method is the protected implementation for a custom grid's OnColumnMoved event handler. The ColumnMoved method does nothing except call any event handler attached to the OnColumnMoved event. You can override ColumnMoved to provide other responses in addition to the inherited event-handler call.

The ColumnMoved method is called when the user moves a column in a grid that has goColMoving in the Options property set. The FromIndex parameter specifies the original column index (corresponding to the Col property) of the column being moved. The ToIndex parameter specifies the destination column index. Use the ColumnMoved method to move any data that underlies a column when a column is moved.

Example
The following code will move the first column of DBGrid1 to the second column when Button1 is clicked.

```
Type
  TMyDBGrid = class( TDBGrid );

Procedure TForm1.Button1Click( Sender : TObject );
begin
  TMyDBGrid( DBGrid1 ).ColumnMoved( 1, 2 );
end;
```

ColPos Property

Delphi equivalent

`Use DefaultWidth Method`

Declaration

```
function DefaultWidth: Integer;
```

Description

Returns the default width of a column in a TDBGrid component. The default column width is based on several factors. If you create a field object using the Fields editor, the field's DisplayWidth property is used to calculate the width in pixels. If the grid has a title row and the width of the column title is bigger than that of DisplayWidth, the title width becomes the default column width.

ColResize Event

Delphi equivalent
`ColWidths Property (Protected)`

Description
See ColWidth Property

ColWidth Property

Delphi equivalent

ColWidths Property (Protected)

Declaration

```
property ColWidths[Index: Longint]: Integer;
```

Description

Run-time only. The ColWidths property determines the width in pixels of all the cells within the column referenced by the Index parameter.

By default, all the columns are the same width, the value of the DefaultColWidth property. To change the width of all columns within a grid, change the DefaultColWidth property value.

To change the width of one column without affecting others, change the ColWidths property. Specify the column you want to change as the value of the Index parameter. Remember the first column always has an Index value of 0.

Example

```
Type
  TMyDBGrid= class( TDBGrid );

Procedure TForm1.ButtonClick( Sender : TObject );
begin
  TMyDBGrid( DBGrid1 ).ColWidths[0] := TMyDBGrid( DBGrid1 ).DefaultColWidth *
2;
end;
```

Collection Object

Delphi equivalent

`TCollection Object`

Description

The TCollection object is used to maintain a collection of TCollectionItem objects. TCollection maintains an index of the collection items in its Items array. The Count property contains the number of items in the collection. Use the Add and Clear properties to add new items to the collection or to delete items from the collection. Objects descended from TCollection can contain objects descended from TCollectionItem. Examples are in the TStatusBar and THeaderControl controls which use TStatusPanels and THeaderSections collection objects to contain TStatusPanel and THeaderSection collection items.

In addition to these properties, methods, and events, the TCollection object also has the methods that apply to all objects and is a direct descendent of TPersistent.

CollsVisible Property

Delphi equivalent

Use TField.Visible Property

Declaration

```
property Visible: Boolean;
```

Description

The Visible property determines whether the component appears onscreen. If Visible is True, the component appears. If Visible is False, the component is not visible.

For controls, calling the Show method makes the control's Visible property True, but it also performs other actions to ensure that the user can see the control.

Example

The following code will toggle the visiblity of the Name field.

```
TForm1.Button1Click( Sender : TObject );
begin
  With TAble1 do
    FieldByName( 'Name' ).Visible := NOT FieldByName( 'Name' ).Visible;
end;
```

Color Property

Delphi equivalent

```
Color Property
```

Description

When you use the Color dialog box(TcolorDialog) to select a color, you are assigning a new color value to the dialog box's Color property. You can then use the value within the Color property and assign it to the Color property of another control.

Example

This code colors a form red:

```
Form1.Color := clRed;
```

ColorMode Property

Delphi equivalent

```
Use ADeviceMode Parameter of SetPrinter
```

Declaration

```
procedure SetPrinter(ADevvice, ADriver, APort: PChar; ADeviceMode: THandle);
```

Description

The SetPrinter method specifies a printer as the current printer. You should seldom, if ever, need to call this method, but instead should access the printer you want in the Printers property array. For more information, see the Windows API CreateDC function.

Cols, Rows Properties

Delphi equivalent

Cols, Rows Properties

Declaration

```
property Cols[Index: Integer]: TStrings;
```

Description

The Cols property is an array of the strings and their associated objects in a column. The number of strings and associated objects is always equal to the value of the ColCount property, the number of columns in the grid. Use the Cols property to access the strings and their associated objects within a particular column in the grid. The Index parameter is the number of the column you want to access; the Index value of the first column in the grid is zero.

The Rows property is an array of the strings and their associated objects in a row. The number of strings and associated objects is always the value of the RowCount property, the number of rows in the grid.

Example

The following line of code adds the string 'Hello' to the end of the list of strings in column four of the string grid named StringGrid1:

```
StringGrid1.Cols[3].Add('Hello');
```

Column Object, Columns Collection

Delphi equivalent

`Columns Property`

Declaration

```
property Columns: TDBGridColumns;
```

Description

The Columns property returns the TDBGridColumns collection object holding the TColumn objects representing the columns in the grid. Use Columns to access TDBGridColumns methods and properties for manipulating the attributes of one or more columns or to add or delete columns from the grid.

Example

The following line of code will set the width of the first column in DBGrid1 to 45 pixels;

```
DBGrid1.Columns[0].Width := 45;
```

Columns Property (DBGrid)

Delphi equivalent

```
Columns Property
```

Declaration

```
property Columns: TDBGridColumns;
```

Description

The Columns property returns the TDBGridColumns collection object holding the TColumn objects representing the columns in the grid. Use Columns to access TDBGridColumns methods and properties for manipulating the attributes of one or more columns or to add or delete columns from the grid.

Example

The following line of code will set the width of the first column in DBGrid1 to 45 pixels;

```
DBGrid1.Columns[0].Width := 45;
```

Columns Property (ListBox)

Delphi equivalent
`Columns Property`

Declaration
```
property Columns: Longint;
```

Description
The Columns property denotes the number of columns in the list box or radio group box. Specify the number of columns you want for the list box or radio group box as the value of Columns.

Example
```
ListBox1.Columns := 2;
```

ComboBox Control

Delphi equivalent
`ComboBox Component`

Description
See Standard Page Components

Command Function

Delphi equivalent
`ParamStr Function`

Declaration
```
function ParamStr(Index: Integer): string;
```

Description
The ParamStr function returns a specified command-line parameter.

Index is an expression of type Integer. ParamStr returns the parameter from the command line that corresponds to Index, or an empty string if Index is greater than ParamCount. For example, an Index value of 2 returns the second command-line parameter.

ParamStr(0) returns the path and file name of the executing program (for example, C:\TEST\MYPROG.EXE).

Example
```
var
  I: Word;
   Y: Integer;
begin
  Y := 10;
  for I := 1 to ParamCount do begin
    Canvas.TextOut(5, Y, ParamStr(I));
    Y := Y + Canvas.TextHeight(ParamStr(I)) + 5;
  end;
end;
```

CommandButton Control

Delphi equivalent
`TButton Component`

Description
See Standard Page Components

Comments Property

Delphi equivalent
`VersionInfo API`

Description
See Win32.HLP

CommitTrans Method

Delphi equivalent

Commit Method

Declaration

```
procedure Commit;
```

Description

The Commit method writes all modifications since the last call to StartTransaction to the database and ends the current transaction. If no transaction is active, Delphi raises an exception.

Example

```
with Database1 do
  begin
    StartTransaction;
{ Update one or more records in tables linked to Database1 }
...
    Commit;
  end;
```

CommitTrans Statement

Delphi equivalent

Commit Method

Declaration

```
procedure Commit;
```

Description

The Commit method writes all modifications since the last call to StartTransaction to the database and ends the current transaction. If no transaction is active, Delphi raises an exception.

Example

```
with Database1 do
  begin
    StartTransaction;
{ Update one or more records in tables linked to Database1 }
...
    Commit;
  end;
```

CommonDialog Control

Delphi equivalent
`TOpenDialog`

Description
See Dialogs Page Components

CompactDatabase Method

Delphi equivalent
`DBIPackTable (BDE)`

Example
The following code will pack the Table associated with Table1 when Button1 is clicked. Table1.Exclusive must be true.

```
Procedure TForm1.Button1Click( Sender : TObject );
begin
  DBIPackTable( Table1.DBHandle, Table1.Handle, Nil, szParadox, TRUE );
end;
```

CompactDatabase Statement

Delphi equivalent
`DBIPackTable (BDE)`

Example
The following code will pack the Table associated with Table1 when Button1 is clicked. Table1.Exclusive must be true.

```
Procedure TForm1.Button1Click( Sender : TObject );
begin
  DBIPackTable( Table1.DBHandle, Table1.Handle, Nil, szParadox, TRUE );
end;
```

CompanyName Property

Delphi equivalent
VersionInfo API

Description
See WinAPI.HLP

Const Statement

Delphi equivalent

Const Reserved Word

Description

The const reserved word defines an identifier whose value cannot change within the block containing the Declaration. A constant identifier cannot be included in its own Declaration. Delphi allows constant expressions.

Expressions used in constant Declarations must be written such that the compiler can evaluate them at compile time.

Examples

(* Constant Declarations *)

```
const
  MaxData  = 1024 * 64 - 16;
  NumChars = Ord('Z') - Ord('A') + 1;
  Message  = 'Hello world...';

(* Typed constants *)
const
  identifier: type = value;
  ...
  identifier: type = value;
```

Container Object, Containers Collection

Delphi equivalent

`TSession.DataBases, TDataBase.Datasets Properties`

Declaration

```
property Databases[Index: Integer]: TDatabase;
```

Description

Run-time and read-only. The Databases property holds a list of all of the currently active TDatabase components.

Example

```
{ Check to see if any record associated with this database has pending
updates }
Changed := False;
with Database1 do
  for I := 0 to DatasetCount - 1 do
    Changed := Changed or DataSets[I].Modified;
```

Container Property

Delphi equivalent
`Parent Property`

Declaration
```
property Parent: TWinControl;
```

Description
The Parent property contains the name of the parent of the control. The parent of a control is the windowed control that contains the control. If one control (parent) contains others, the contained controls are child controls of the parent. For example, if your application includes three radio buttons in a group box, the group box is the parent of the three radio buttons, and the radio buttons are the child controls of the group box.

Example
To set up the form for this example, put a group box on the form and add a radio button to the group box. Put two labels and a button on the form. This code displays the name of the parent of the radio button and the class name of the owner of the radio button in the captions of the two labels when the user clicks the button:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := RadioButton1.Parent.Name + ' is the parent';
  Label2.Caption := RadioButton1.Owner.ClassName +
    ' is the class name of the owner';
end;
```

This example uses a button and a group box on a form. When the user clicks the button, the button moves inside the group box, because the group box is now the parent of the button.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Parent := GroupBox1;
end;
```

Container Property (Data Access)

Delphi equivalent
```
Database Name, Session Name Properties
```

Declaration
```
property DatabaseName: TFileName;
```

Description
Set the DatabaseName property to specify the database to access. This property can specify:

A defined BDE alias,
A directory path for desktop database files,
An application-specific alias defined by a TDatabase component

ControlBox Property

Delphi equivalent

`Use BorderIcons Property`

Declaration

```
property BorderIcons: TBorderIcons;
```

Description

The BorderIcons property is a set whose values determine which icons appear on the title bar of a form. These are the possible values that the BorderIcons set can contain:

Value	Meaning
biSystemMenu	The form has a Control menu (also known as a System menu)
biMinimize	The form has a Minimize button
biMaximize	The form has a Maximize button
biHelp	The form shows help in pop-up windows, rather than starting Windows Help (WINHELP.EXE). If biMinimize and biMaximize are excluded, a question mark appears in the form's title bar; otherwise, it is hidden.

Example

The following code removes a form's System menu when the user clicks a button:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  BorderIcons := BorderIcons - [biSystemMenu];
end;
```

Controls Collection

Delphi equivalent
`Controls Property`

Declaration
```
property Controls[Index: Integer]: TControl;
```

Description
Run-time and read only. The Controls property is an array of all controls that are children of the control. The Controls property is most useful if you have a need to refer to the children of a control by number rather than name.

Don't confuse the Controls property with the Components property. The Components property lists all components that are owned by the component, while the Controls property lists all the controls that are child windows of the control. All components put on a form are owned by the form, and therefore, they appear in the form's Components property list.

Example
This example uses a group box on a form, with several controls contained within the group box. The form also has an edit box and a button outside of the group box. The code counts each control's child controls turning each of them invisible as they are counted. The total number of controls counted displays in the edit box.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I:= 0 to GroupBox1.ControlCount -1 do
     GroupBox1.Controls[I].Visible := False;
  Edit1.Text := IntToStr(GroupBox1.ControlCount) + ' controls';
end;
```

Copies Property

Delphi equivalent

```
Copies Property
```

Declaration

```
property Copies: Integer;
```

Description

The value of the Copies property determines the number of copies of the print job to print. If you change the value of Copies at design time, the value you specify is the default value in the edit box control when the Print dialog box appears. The default value is 0.

Example

The following code sets the default number of copies for the print dialog box, PrintDialog1, to 3 before displaying the dialog box:

```
PrintDialog1.Copies := 3;
PrintDialog1.Execute;
```

Copy Method

Delphi equivalent
Copy Method

Declaration
```
procedure Copy;
```

Description
Copies the OLE object to the Windows clipboard. There must be an OLE object in the container before calling Copy.

Example
```
Procedure TForm1.Button1Click( Sender : TObject );
begin
  OleContainer1.copy;
end;
```
Note There must be an OLE object in the container before calling Copy.

Cos Function

Delphi equivalent
`Cos Function`

Declaration
```
function Cos(X: Extended): Extended;
```

Description
The Cos function returns the cosine of the angle X, in radians.

Count Function (SQL)

Delphi equivalent
Count Function (SQL)

Description
SQL Syntax

Example
```
Select Count( OrderNo ) from Orders
```

Count Property (Data Access)

Delphi equivalent

DataBaseCount, DataSetCount Properties

Declaration

```
property DatabaseCount: Integer;
```

Description

Run-time and read-only. DatabaseCount is the number of TDataBase components currently attached to the Session. DatasetCount is the number of dataset components (TTable, TQuery, and TStoredProc) that are currently using the TDatabase component. Read-only and run-time only.

Example

```
{ Check to see if any record associated with this database has pending
updates }
Changed := False;
with Database1 do
  for I := 0 to DatasetCount - 1 do
    Changed := Changed or DataSets[I].Modified;
```

Count Property (VB Collections)

Delphi equivalent
Count Property

Declaration
```
property Count: Integer;
```

Description
Run-time and read only. The Count property contains the number of items in a collection, list, menu item, tree nodes object, tree node, header section, list column or a status bar.

Example
The following code displays the number of items in a list box in the caption of a label when the user clicks the CountItems button:

```
procedure TForm1.CountItemsClick(Sender: TObject);
begin
  Label1.Caption := 'There are ' + IntToStr(ListBox1.Items.Count) +
    ' items in the listbox.';
end;
```

The following example assumes the form contains a main menu component, which includes a File menu and a label. This code displays the number of menu items that make up the File menu.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := IntToStr(FileMenu.Count):
end;
```

CreateDatabase Function

Delphi equivalent

`TDatabase.Create Method`

Declaration

```
constructor Create(AOwner: TComponent);
Tdatabase Component
```

For information on TDatabase Component see Data Access Page Components

Description

The Create method allocates memory to create the component and initializes its data as needed. Each object can have a Create method customized to create that particular kind of object. The owner of the created component is passed in the AOwner parameter.

Usually you don't need to create objects manually. Objects you design in Delphi are automatically created for you when you run the application and destroyed when you close the application.

Example

```
MyDatabase := TDatabase.create( self );
```

CreateDatabase Method

Delphi equivalent
`TDatabase.Create Method`

Description
See CreateDatabase Function

```
ExampleMyDatabase := TDatabase.create( self );
```

CreateDynaset Method

Delphi equivalent

`TQuery.Create Method`

Declaration

```
constructor Create(AOwner: TComponent);
Tquery Component
```

For information on TQuery Component see Data Access Page Components

Description

The Create method allocates memory to create the component and initializes its data as needed. Each object can have a Create method customized to create that particular kind of object. The owner of the created component is passed in the AOwner parameter.

Usually you don't need to create objects manually. Objects you design in Delphi are automatically created for you when you run the application and destroyed when you close the application.

Example

```
MyQuery := TQuery.Create( Self );
```

CreateEmbed Method

Delphi equivalent
`CreateObject Method`

Declaration
```
procedure CreateObject(const OleClassName: string; Iconic: Boolean);
```

Description
Creates an embedded OLE object given its class name (also known as programmatic identifier). Iconic indicates whether the object is shown as an icon (True) or displayed as it would be in the server application (False). If there's already an OLE object in the container, it's destroyed and any changes the user made to it are discarded.

Example
```
OleContainer1.CreateObject( 'PdoxWin7Table', TRUE );
```

CreateField Method

Delphi equivalent
`TField.Create Method`

Declaration
```
constructor Create;
Tfield Component
```

For information on Tfield Component see Data Access Page Components

Description
The Create method constructs a new object instance. Create returns an instance of the type being created, allocated on the global heap.

Example
The following example creates a new TStringField named Query1CO_NAME.

```
procedure TForm1.Button2Click(Sender: TObject);
var
  T: TStringField;
begin
  Query1.Close;

  T := TStringField.Create(Self);

  T.FieldName := 'CO_NAME';
  T.Name := Query1.Name + T.FieldName;
  T.Index := Query1.FieldCount;
  T.DataSet := Query1;

  Query1.FieldDefs.UpDate;
  Query1.Open;
end;
```

CreateIndex Method

Delphi command

AddIndex Method

Declaration

```
procedure AddIndex(const Name, Fields: string; Options: TIndexOptions);
```

Description

The AddIndex method creates a new index for the TTable. Name is the name of the new index. Fields is a list of the fields to include in the index. Separate the field names by a semicolon. Options is a set of values from the TIndexOptions type.

Example

```
Table1.AddIndex('NewIndex', 'CustNo;CustName', [ixUnique,
ixCaseInsensitive]);
```

CreateLink Method

Delphi equivalent
`CreateLinkToFile Method`

Declaration
```
procedure CreateLinkToFile(const FileName: string; Iconic: Boolean);
```

Description
Creates a linked OLE object from the contents of the given file. Iconic indicates whether the object is shown as an icon (True) or displayed as it would be in the server application (False). If there's already an OLE object in the container, it's destroyed and any changes the user made to it are discarded.

Example
```
OleContainer1.CreateLinkToFile( 'TITAN.BMP', FALSE );
```

CreateObject Function

Delphi equivalent

`CreateOleObject Function`

Declaration

```
function CreateOleObject(const ClassName: string): Variant;
```

Description

The CreateOleObject function creates an OLE automation object of the specified class and returns it in a variant. This is the way to get a new instance of an OLE automation server object for your automation controller.

Example

```
Procedure TForm1.Button1Click( Sender : TObject )
begin
  MyVariant := CreateOleObject( 'MyAutomationProgID' );
end;
```

CreateQueryDef Method

Delphi equivalent

`TQuery.Create Method`

Declaration

```
constructor Create(AOwner: TComponent);
Tquery Component
```

For more information on TQuery Component see Data Access Page Components

Description

The Create method allocates memory to create the component and initializes its data as needed. Each object can have a Create method customized to create that particular kind of object.

Example

```
MyQuery := TQuery.create( Self );
```

CreateSnapshot Method

Delphi equivalent

`TQuery.Create Method`

Declaration

```
constructor Create(AOwner: TComponent);
Tquery Component
```

For more information on TQuery Component see Data Access Page Components.

Description

The Create method allocates memory to create the component and initializes its data as needed. Each object can have a Create method customized to create that particular kind of object.

Example

```
MyQuery := TQuery.create( Self );
```

CreateTableDef Method

Delphi equivalent
`TTable.Create Method`

Declaration
```
constructor Create(AOwner: TComponent);
TTable Component
```

For more information on TTable Component see Data Access PageComponents.

Description
The Create method allocates memory to create the component and initializes its data as needed. Each object can have a Create method customized to create that particular kind of object.

Example
```
MyTable := TTAble.Create( Self );
```

CurDir Function

Delphi equivalent
`GetDir Procedure`

Declaration
```
procedure GetDir(D: Byte; var S: string);
```

Description
The GetDir procedure returns the current directory of a specified drive.

D can be set to any of the following values:

Value	Drive
0	Default
1	A
2	B
3	C

Performs no error checking. If the drive specified by D is invalid, S returns X:\ as if it were the root directory of the invalid drive.

Example
```
var
   s : string;
 begin
   GetDir(0,s); { 0 = Current drive }
   MessageDlg('Current drive and directory: ' + s, mtInformation, [mbOk] ,
0);
 end;
```

CurrentX, CurrentY Properties

Delphi equivalent

PenPos Property

Declaration

```
property PenPos: TPoint;
```

Description

The PenPos property is the current drawing position of the pen. You should use the MoveTo method to set the drawing position, rather than changing PenPos directly.

Example

The following code will move the current pen and display the its position.

```
procedure TForm1.Button1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  pt : TPoint;
begin
  Canvas.MoveTo( x,y );
  pt := Canvas.PenPos;
  Canvas.TextOut( 0, 0 , 'The Current Pen Position is at : '+
IntToStr( pt.x ) + ', ' + IntToStr( pt.y ));
end;
```

DBCombo Control

Delphi equivalent
`DBComboBox Component`

Description
See Data Controls Page Components

DBEngine Object

Delphi equivalent

`TSession Component`

Description

The TSession component provides global control over database connections for an application. Delphi automatically creates a default TSession component at runtime for applications which use database controls. This component may be accessed at runtime through the global variable Session. Do not close or destroy the default session.

There is also a TSession component on the data access page of the component palette. This component is only needed if you are writing a multi-threaded application which will perform simultaneous data access (such as running two queries at the same time). The global variable Sessions is a TSessionList which provides a way to manage applications which use multiple sessions.

TSession provides global control over database connections for an application. The Databases property of TSession is an array of all the active databases in the session. The DatabaseCount property is an integer specifying the number of active databases in the Session.

DBGrid Control

Delphi equivalent
`DBGrid Component`

Description
See Data Controls Page Components

DBList Control

Delphi equivalent
`DBListBox Component`

Description
See Data Controls Page Component

DDB Function

Delphi equivalent
`DoubleDecliningBalance Function`

Declaration
```
function DoubleDecliningBalance(Cost, Salvage: Extended;
  Life, Period: Integer): Extended;
```

Description
The DoubleDecliningBalance function determines accelerated depreciation values for an asset, given the initial cost, life expectancy, end value, and depreciation period. It calculates depreciation using the double-declining balance method.

DELETE Statement (SQL)

Delphi equivalent
DELETE Statement (SQL)

Description
SQL Syntax

Example
```
Delete From Customer
Where IDField = 1234
```

DISTINCT, DISTINCTROW Predicates (SQL)

Delphi equivalent
DISTINCT, DISTINCTROW Predicates (SQL)

Description
SQL Syntax

Example
The following statement retrieves data from a Paradox table and a dBASE table:

```
SELECT DISTINCT C.CUST_NO, C.STATE, O.ORDER_NO
      FROM "CUSTOMER.DB" C, "ORDER.DBF" O
      WHERE C.CUST_NO = O.CUST_NO
```

DROP Statement (SQL)

Delphi equivalent
`DROP Statement (SQL)`

Description
SQL Syntax

Example
```
DROP TABLE "employee.db"
```

Data Control

Delphi equivalent
`Datasource Component`

Description
See Data Access Page Components

DataChanged Property

Delphi equivalent
Modified Property

Declaration
```
property Modified: Boolean;
```

Description
Run-time only. The Modified property determines whether the text of an edit box or memo control was changed since it was created or since the last time the Modified property was set to False. If Modified is True, the text was changed. If Modified is False, the text was not changed.

Example
```
procedure TForm1.Button1Click(Sender: TObject);
begin
if DBEdit1.Modified = True then
  begin
    MessageDlg('DBEdit box text was modified',
      mtInformation, [mbOK], 0);
    DBEdit1.Modified := False;
  end
  else
    MessageDlg('DBEdit box text was not modified',
      mtInformation, [mbOK], 0);
end;
```

DataField Property

Delphi equivalent
`DataField Property`

Declaration
```
property DataField: string;
```

Description
The DataField property identifies the field from which the data-aware control displays data. The dataset the field is located in is specified in a data source component (TDataSource). The DataSource property of the data-aware control specifies which data source component.

If the DataField value of a database edit box (TDBEdit) is an integer or floating-point value, only characters that are valid in such a field can be entered in the edit box. Characters that are not legal are not accepted.

Example
The following code specifies that the DataField of DBEdit1 is 'FNAME'.

```
DBEdit1.DataField := 'FNAME';
```

DataSource Property

Delphi equivalent

```
DataSource Property
```

Declaration

```
property DataSource: TDataSource;
```

Description

The DataSource property determines where the component obtains the data to display. Specify the data source component that identifies the dataset the data is found in.

Example

The following code specifies DataSource1 to be the DataSource of DBGrid1.

```
DBGrid1.DataSource := DataSource1;
```

DataUpdatable Property

Delphi equivalent
CanModify Property

Declaration
```
property CanModify: Boolean;
```

Description
Run-time and read only. Specifies if a field can be modified for any reason, such as during a SetKey operation. CanModify is True if the value of the field can be modified. If the ReadOnly property of the field is True, or the ReadOnly property of the dataset is True, then CanModify is False.

Example
```
If Table1.Fields[0].CanModify then
  Table1.Fields[0].Clear;
```

Database Object, Database Collection

Delphi equivalent
`TDatabase Component`

Description
See Data Access Page Components

Database Property

Delphi equivalent

Database Property

Declaration

```
property Database: TDatabase;
```

Description

Run-time and read-only. Database specifies the database (TDatabase) component associated with the dataset component. If you did not create a TDatabase at design time, then Delphi will create one at run time. Use the Database property to reference the properties and methods of the database.

Example

```
{ Do a transaction }
with Table1.Database do
  begin
    StartTransAction;
  { Post some records with Table1 }
    Commit;
  end;
```

DatabaseName Property

Delphi equivalent

```
DatabaseName Property
```

Declaration

```
property DatabaseName: TFileName;
```

Description

Set the DatabaseName property to specify the database to access. This property can specify:

A defined BDE alias,
A directory path for desktop database files,
An application-specific alias defined by a TDatabase component

Example

```
Database1.DatabaseName := 'Delphi_Demos';
```

Date Function

Delphi equivalent
Date Function

Declaration
```
function Date: TDateTime;
```

Description
The Date function returns the current date.

Example
This example uses a label and a button on a form. When the user clicks the button, the current date is displayed in the caption of the label:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := 'Today is  ' + DateToStr(Date);
end;
```

Date Statement

Delphi equivalent
`SetLocalTime (API)`

Description
See Win32.hlp

DatePart Function

Delphi equivalent
DecodeDate Function

Declaration
```
procedure DecodeDate(Date: TDateTime; var Year, Month, Day: Word);
```

Description
The DecodeDate procedure breaks the value specified as the Date parameter into Year, Month, and Day values. If the given TDateTime value is less than or equal to zero, the year, month, and day return parameters are all set to zero.

Example
This example uses a button and two labels on a form. When the user clicks the button, the current date and time are reported in the captions of the two labels.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Present: TDateTime;
  Year, Month, Day, Hour, Min, Sec, MSec: Word;
 begin
  Present:= Now;
  DecodeDate(Present, Year, Month, Day);
  Label1.Caption := 'Today is Day ' + IntToStr(Day) + ' of Month '
    + IntToStr(Month) + ' of Year ' + IntToStr(Year);
  DecodeTime(Present, Hour, Min, Sec, MSec);
  Label2.Caption := 'The time is Minute ' + IntToStr(Min) + ' of Hour '
    + IntToStr(Hour);
end;
```

DateSerial Function

Delphi equivalent
`EncodeDate Function`

Declaration
```
function EncodeDate(Year, Month, Day: Word): TDateTime;
```

Description
The EncodeDate function returns a TDateTime type from the values specified as the Year, Month, and Day parameters.

The year must be between 1 and 9999.

Valid Month values are 1 through 12.

Valid Day values are 1 through 28, 29, 30, or 31, depending on the Month value. For example, the possible Day values for month 2 (February) are 1 through 28 or 1 through 29, depending on whether or not the Year value specifies a leap year.

If the specified values are not within range, an EConvertError exception is raised. The resulting value is one plus the number of days between 12/30/1899 and the given date.

Example
This example uses a button and a label on a form. When the user clicks the button, a specified date is encoded as a MyDate variable of type TDateTime. MyDate is then displayed as a string in the caption of the label.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyDate: TDateTime;
begin
  MyDate := EncodeDate(83, 12, 31);
  Label1.Caption := DateToStr(MyDate);
end;
```

DateValue Function

Delphi equivalent
`StrToDate Function`

Declaration
```
function StrToDate(const S: string): TDateTime;
```

Description
The StrToDate function converts a string to date format. The date in the string must be a valid date.

The string must consist of two or three numbers, separated by the character defined by the DateSeparator global variable. The order for month, day, and year is determined by the ShortDateFormat global variable--possible combinations are m/d/y, d/m/y, and y/m/d.

If the string contains only two numbers, it is interpreted as a date (m/d or d/m) in the current year. Year values between 0 and 99 are assumed to be in the current century.

If the given string does not contain a valid date, an EConvertError exception is raised.

Example
This example uses an edit box, a label, and a button on a form. When the user enters a date in the edit box in the MM/DD/YY format, the string entered is converted to a TDateTime value. This value is then converted back to a string value so it can appear as the caption of the label:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ADate: TDateTime;
begin
  ADate := StrToDate(Edit1.Text);
  Label1.Caption := DateToStr(ADate);
end;
```

Day Function

Delphi equivalent
DecodeDate Function

Declaration
```
procedure DecodeDate(Date: TDateTime; var Year, Month, Day: Word);
```

Description
The DecodeDate procedure breaks the value specified as the Date parameter into Year, Month, and Day values. If the given TDateTime value is less than or equal to zero, the year, month, and day return parameters are all set to zero.

Example
This example uses a button and two labels on a form. When the user clicks the button, the current date and time are reported in the captions of the two labels.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Present: TDateTime;
  Year, Month, Day, Hour, Min, Sec, MSec: Word;
 begin
  Present:= Now;
  DecodeDate(Present, Year, Month, Day);
  Label1.Caption := 'Today is Day ' + IntToStr(Day) + ' of Month '
    + IntToStr(Month) + ' of Year ' + IntToStr(Year);
  DecodeTime(Present, Hour, Min, Sec, MSec);
  Label2.Caption := 'The time is Minute ' + IntToStr(Min) + ' of Hour '
    + IntToStr(Hour);
end;
```

DblClick Event

Delphi equivalent
`OnDblClick Event`

Declaration
```
property OnDblClick: TNotifyEvent;
```

Description
The OnDblClick event occurs when the user double-clicks the mouse button while the mouse pointer is over the component.

Example
This example notifies the user that the form was double-clicked.

```
procedure TForm1.FormClick(Sender: TObject);
begin
  MessageDlg('You double-clicked the form', mtInformation, [mbOk], 0);
end;
```

Deactivate Event

Delphi equivalent
`OnDeactivate Event`

Declaration
```
property OnDeactivate: TNotifyEvent;
```

Description
The OnDeactivate event occurs when the user switches from your application to another Windows application. Use the OnDeactive event to do any special processing you want to occur before your application is deactivated.

Note See Handling Application Events for more information about creating event handlers for application events.

Example
The following code minimizes an application when it's deactivated. Note that AppDeactivate should be declared a method of TForm1.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnDeactivate := AppDeactivate;
end;
procedure TForm1.AppDeactivate(Sender: TObject);
begin
  Application.Minimize;
end;
```

DefColWidth Property

Delphi equivalent
`DefaultColWidth (Protected)`

Declaration
```
property DefaultColWidth: Integer;
```

Description
The DefaultColWidth property determines the width of all the columns within the grid.

If you want to change the width of a single column within a grid without changing other columns, use the ColWidths property during run time. If you change the DefaultColWidth property value after changing the width of specified columns, all the columns become the height specified in the DefaultColWidth property once again.

The default value is 64 pixels.

Example
```
Type
  TMyDBGrid= class( TDBGrid );

Procedure TForm1.ButtonClick( Sender : TObject );
begin
  TMyDBGrid( DBGrid1 ).ColWidths[0] := TMyDBGrid( DBGrid1 ).DefaultColWidth *
2;
end;
```

Default Property

Delphi equivalent
`Default Property`

Declaration
```
property Default: Boolean;
```

Description
The Default property indicates whether a push or bitmap button is the default button. If Default is True, any time the user presses Enter, the OnClick event handler for that button runs. The only exception to this is if the user selects another button before pressing Enter, in which case the OnClick event handler for that button runs. Although your application can have more than one button designated as a default button, the form calls the OnClick event handler for the first button in the tab order.

Whenever any button has focus, it becomes the default button temporarily. When the focus moves to a control that isn't a button, the button with its Default property set to True becomes the default button once again.

Example
This example makes the button named OK the default button:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  OK.Default := True;
end;
```

DefaultExt Property

Delphi equivalent
`DefaultExt Property`

Declaration
```
property DefaultExt: TFileExt;
```

Description
The DefaultExt property specifies the extension that is added to the file name the user types in the File Name edit box if the user doesn't include a file-name extension in the filename. If the user specifies an extension for the filename, the value of the DefaultExt property is ignored. If the DefaultExt value remains blank, no extension is added to the filename entered in the File Name edit box.

Legal property values include strings up to 3 characters in length. Don't include the period (.) that divides the filename and its extension.

Example
This example sets the default file extension to TXT, displays the Open dialog box, then assigns the filename the user selects with the dialog box to a variable the application can use to open a file:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  NameOfFile : TFileName;
begin
  OpenDialog1.DefaultExt := 'TXT';
  if OpenDialog1.Execute then
    NameOfFile := OpenDialog1.FileName;
end;
```

When this code runs, if the user types a filename in the File Name edit box in the Open dialog box, but doesn't specify an extension, the TXT extension is added to the filename, and the entire filename is saved in the NameOfFile variable. For example, if the user types MYNOTES as the filename, the string saved in the NameOfFile variable is MYNOTES.TXT.

DefaultValue Property

Delphi equivalent

```
Use OnEnter Event
```

Declaration

```
property OnEnter: TNotifyEvent;
```

Description

The OnEnter event occurs when a component becomes active. Use the OnEnter event handler to specify any special processing you want to occur when a component becomes active.

Delete Method

Delphi equivalent
`Delete Method`

Declaration
```
procedure Delete;
```

Description
The Delete method deletes the current record from the dataset. The next record then becomes the new current record. If the record deleted was the last record in the dataset, then the previous record becomes the current record.

This method is valid only for datasets that return a live result set.

Example
```
Table1.delete;
```

Delete Method (OLE Container)

Delphi equivalent
`Close Method`

Declaration
```
procedure Close
```

Description
Deactivates the OLE object and terminates its server application, but doesn't remove it from the container. Any changes the user made to the OLE object are automatically saved. There must be an OLE object in the container before calling Close. After calling Close, the container's State property is osLoaded.

Example
```
OleContainer1.Close;
```

DeleteQueryDef Method

Delphi equivalent
`Free Method`

Declaration
```
procedure Free;
```

Description
The Free method destroys the object and frees its associated memory. If you created the object yourself using the Create method, you should use Free to destroy and release memory. Free is successful even if the object is nil, so if the object was never initialized, for example, calling Free won't result in an error.

Example
```
MyTable.Free;
```

DeleteSetting Statement

Delphi equivalent
DeleteKey Method

Declaration
```
function DeleteKey(const Key: string): Boolean;
```

Description
DeleteKey removes a specified key and its associated data from the registry. If the key has subkeys, the subkeys and any associated data are also removed.

DeleteKey returns True if key deletion is successful. On error, DeleteKey returns False.

Example
```
var
  MyReg : TRegistry;
begin
  MyReg := TRegistry.create;
  MyReg.DeleteKey( 'Software\Borland\MyApplicationKey' );
  MyReg.Free;
end;
```

Description Property (Data Access)

Delphi equivalent

Message Property

Example

```
try
  Table1.open;
on
 E : EDBEngineError do
  showMessage( E.Message );
end;
```

DeviceName Property

Delphi equivalent
`Printers Property`

Declaration
```
property Printers: TStrings;
```

Description
Run-time and read-only. The Printers property is a list of all printers installed in Windows.

Example
```
listbox1.items := printer.printers;
```

DialogTitle Property

Delphi equivalent
`Title Property`

Declaration
```
property Title: string;
```

Description
The Title property determines the text that appears in the dialog box's title bar.

Example
This code displays the Open dialog box with the text "Open Pascal files" in its title bar and lists only Pascal files in the list box:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  OpenDialog1.Filter := 'Pascal files (*.PAS)|*.PAS';
  OpenDialog1.Title := 'Open Pascal files';
  OpenDialog1.Execute;
end;
```

Dim Statement

Delphi equivalent

Var

Description

A variable (var) Declaration associates an identifier and a type with a location in memory where values of that type can be stored.
An absolute clause can be used to specify an absolute memory address.

The var reserved word is also used to declare variable parameters.

Example

{ Variable Declarations }

```
  var
    X, Y, Z: real;
    I, J, K: Integer;
    Done, Error: Boolean;
    Vector: array[1..10] of real;
    Name: string[15];
    InFile, OutFile: Text;
    Letters: set of 'A'..'Z';
```

Dir Function

Delphi equivalent
`FileExists`

Declaration
```
function FileExists(const FileName: string): Boolean;
```

Description
The FileExists function returns True if the file specified by FileName exists. If the file does not exist, FileExists returns False.

Example
The following code prompts you for confirmation before deleting a file:

```
if FileExists(FileName) then
  if MsgBox('Do you really want to delete ' + ExtractFileName(FileName)
    + '?'), []) = IDYes then DeleteFile(FileName);
```

DirListBox Control

Delphi equivalent
`DirectoryListBox Component`

Description
See System Page Components

DisplayType Property

Delphi equivalent
`Iconic Property`

Declaration
```
property Iconic: Boolean;
```

Description
Determines how the OLE object is displayed. If Iconic is True, it's displayed as an icon; if False, it's displayed as it would be in the server application.

Do...Loop Statement

Delphi equivalent

`Repeat...Until Statement`

Description

The statements between repeat and until are executed in sequence while the Boolean expression in the until statement evaluates to True.

Using this loop ensures that the sequence is executed at least once because the Boolean expression is evaluated after the execution of each sequence.

Example

```
{ Repeat Statements }
  repeat Ch := GetChar until Ch <> ' ';
  repeat
    Write('Enter value: ');
    ReadLn(I);
  until (I >= 0) and (I <= '9');
```

DoEvents Function

Delphi equivalent
`Application.ProcessMessages`

Declaration
```
procedure ProcessMessages;
```

Description
The ProcessMessages method interrupts the execution of your application so that Windows can respond to events. In Win32, neglecting message processing doesn't affect other applications, it only affects the responsiveness of your own process. By calling ProcessMessages, your application permits Windows to process these events at the time ProcessMessages is called. The ProcessMessages method cycles the Windows message loop until it is empty and then returns control to your application.

Example
This example uses two buttons that are long enough to accommodate lengthy captions on a form. When the user clicks the button with the caption Ignore Messages, the code begins to generate a long series of random numbers. If the user tries to resize the form while the handler is running, nothing happens until the handler is finished. When the user clicks the button with the caption Process Messages, more random numbers are generated, but Windows can still respond to a series of mouse events, such as resizing the form.

Note How quickly these event handlers run depends on the microprocessor of your computer. A message appears on the form informing you when the handler has finished executing.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Button1.Caption := 'Ignore Messages';
  Button2.Caption := 'Process Messages';
end;
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, X, Y: Word;
begin
  I := 0;
  J := 0;
  while I < 64000 do
  begin
    Randomize;
    while J < 64000 do
    begin
      Y := Random(J);
      Inc(J);
     end;

    X := Random(I);
    Inc(I);
  end;
  Canvas.TextOut(10, 10, 'The Button1Click handler is finished');
end;
procedure TForm1.Button2Click(Sender: TObject);
var
  I, J, X, Y: Word;
begin
  I := 0;
  J := 0;
  while I < 64000 do
```

```
  begin
    Randomize;
    while J < 64000 do
    begin
      Y := Random(J);
      Inc(J);
      Application.ProcessMessages;
    end;

  X := Random(I);
  Inc(I);
  end;
  Canvas.TextOut(10, 10, 'The Button2Click handler is finished');
end;
```

DoVerb Method

Delphi equivalent
DoVerb Method

Declaration
```
procedure DoVerb(Verb: Integer);
```

Description
Requests the OLE object to perform some action. OLE defines several verbs, such as ovShow (to display the OLE object) and ovPrimary (the default action, usually to activate the OLE object). OLE objects can define their own custom verbs. You can use the ObjectVerbs property to get a list of those custom verbs.

Example
```
OleContainer1.DoVerb( ovPrimary );
```

Drag Method

Delphi equivalent
`BeginDrag Method`

Declaration
```
procedure BeginDrag(Immediate: Boolean);
```

Description
The BeginDrag method starts the dragging of a control. If the Immediate parameter is True, the mouse pointer changes to the value of the DragCursor property and dragging begins immediately. If Immediate is False, the mouse pointer doesn't change to the value of the DragCursor property and dragging doesn't begin until the user moves the mouse pointer a short distance (5 pixels). This allows the control to accept mouse clicks without beginning a drag operation.

Your application needs to call the BeginDrag method to begin dragging only when the DragMode property value for the control is dmManual.

Example
The following code handles a mouse-down event on a file list box by beginning dragging only if it was the left mouse button pressed:

```
procedure TFMForm.FileListBox1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft then     { only drag if left button pressed }
    with Sender as TFileListBox do  { treat Sender as TFileListBox }
    begin
      if ItemAtPos(Point(X, Y), True) >= 0 then { is there an item here? }
        BeginDrag(True);      { if so, drag it }
    end;
end;
```

DragDrop Event

Delphi equivalent
`OnDragDrop Event`

Declaration
```
TDragDropEvent = procedure(Sender, Source: TObject; X, Y: Integer) of object;
property OnDragDrop: TDragDropEvent;
```

Description
The OnDragDrop event occurs when the user drops an object being dragged. Use the OnDragDrop event handler to specify what you want to happen when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

The TDragDropEvent type points to a method that handles the dropping of a dragged object. The Source parameter is the object being dragged, Sender is the object the Source is being dropped on, and X and Y are screen coordinates in pixels.

Example
This code comes from an application that contains a list box and three labels, each with a different font and color. The user can select a label and drag it to a list box and drop it. When the label is dropped, the items in the list box assume the color and font of the dropped label. This is the OnDragDrop event handler.

```
procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    (Sender as TListBox).Font := (Source as TLabel).Font;
  end;
end;
```

The Source in this example is the label, and the Sender is the list box.

DragIcon Property

Delphi equivalent
`DragCursor Property`

Declaration
```
property DragCursor: TCursor;
```

Description
The DragCursor property determines the shape of the mouse pointer when the pointer is over a component that will accept an object being dragged. See Delphi.hlp (DragCursor) for further infomation.

Example
The following code changes the DragCursor of Memo1 to crIBeam.

```
Memo1.DragCursor := crDrag;
```

DragMode Property

Delphi equivalent
`DragMode Property`

Declaration
```
property DragMode: TDragMode;
```

Description
The DragMode property determines the drag and drop behavior of a control. These are the possible values:

Value	Meaning
dmAutomatic	If dmAutomatic is selected, the control is ready to be dragged; the user just clicks and drags it.
dmManual	If dmManual is selected, the control can't be dragged until the application calls the BeginDrag method.

If a control's DragMode property value is dmAutomatic, the application can disable the drag and drop capability at run time by changing the DragMode property value to dmManual.

Example
This example determines whether the drag mode of the button on the form is manual. If it is, the dragging the button becomes possible.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Button1.DragMode = dmManual then
    Button1.BeginDrag(True);
end;
```

DragOver Event

Delphi equivalent
`OnDragOver Event`

Declaration
```
TDragOverEvent = procedure(Sender, Source: TObject; X, Y: Integer; State:
TDragState; var Accept: Boolean) of object;
property OnDragOver: TDragOverEvent;
```

Description
The OnDragOver event occurs when the user drags an object over a component. You use an OnDragOver event to accept an object so the user can drop it.

The OnDragOver event accepts an object when its Accept parameter is True.

Usually, you will want the cursor to change shape, indicating that the control can accept the dragged object if the user drops it. You can change the shape of the cursor by changing the value of the DragCursor property for the control either at design time or at run time before an OnDragOver event occurs.

The TDragOverEvent type points to a method that handles the dragging of one object over another. The Source parameter is the object being dragged. Sender is the object the Source is being dragged over. X and Y are the coordinates within the control in pixels. State is the state of the drag object in relationship to the object being dragged over. And Accept determines whether the Sender recognizes the drag object. Accept defaults to False.

Example
This OnDragOver event handler permits the list box to accept a dropped label:

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;
```

The Source parameter identifies what is being dragged. The Sender is the control being dragged over.

This code permits the list box to accept any dropped control:

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := True;
end;
```

DrawMode Property

Delphi equivalent
`Pen.Mode Property`

Declaration
```
property Mode: TPenMode;
```

Description
The Mode property determines how the pen draws lines on the canvas. See Delphi.hlp for a table that describes the behavior for each pen mode.

Example
The following code sets the mode of the pen of the Canvas of Form1 to the inverse of the pen Color.

```
Form1.Canvas.Pen.Mode := pmNotCopy;
```

DrawStyle Property

Delphi equivalent
`Pen.Style Property`

Declaration
`property Style: TPenStyle;`

Description
The Style property determines the style in which the pen draws lines. See Delphi.hlp for a table that shows the different style values.

Example
This example uses two radio buttons on a form. When the user drags the mouse pointer across the form, lines are drawn. The user can use the two radio buttons to choose between two pen styles. Selecting the first radio button draws a dotted line. Selecting the second radio button draws a solid line.

```
var
  Drawing: Boolean;
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  Drawing := True;
  Canvas.MoveTo(X, Y);
end;
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  if Drawing then
    Canvas.LineTo(X, Y);
end;
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;

  Shift: TShiftState; X, Y: Integer);
begin
  Canvas.LineTo(X, Y);
  Drawing := False;
end;
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
  Canvas.Pen.Style := psDot;
end;
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
  Canvas.Pen.Style := psSolid;
end;
```

DrawWidth Property

Delphi equivalent
`Pen.Width Property`

Declaration
```
property Width: Integer;
```

Description
The Width property determines the maximum width of the graphics object in pixels.

Example
To set the pen width to a random value from 1 to 10,

```
Canvas.Pen.Width := 1 + Random(10);
```

Drive Property

Delphi equivalent
`Drive Property`

Declaration
```
property Drive: Char;
```

Description
Run-time only. For the drive combo box, the Drive property determines which drive is displayed in the edit control of the combo box. When the user uses the drive combo box to select a new drive, the selected drive becomes the value of the Drive property. The value of the Text property also changes to the new volume name when the Drive property value changes.

For the directory list box, the Drive property determines which drive the list box displays the directory structure on. When the value of Drive changes, the Directory value changes also to the current directory on the specified drive.

For the file list box, the Drive property determines which drive the list box displayed the files on. When the value of Drive changes, the Directory value also changes to the current directory on the specified drive.

Example
The following example assumes that a drive combo box, a file list box, and a directory list box are on a form. This code changes the drive displayed in the drive combo box, displays the current directory of the selected drive in the directory list box, and displays the files in the current directory of the selected drive in the file list box when the user selects a drive in the drive combo box:

```
procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

DriveListBox Control

Delphi equivalent
`DriveComboBox Component`

Description
See System Page Components

DriverName Property

Delphi equivalent
`GetPrinter Method`

Declaration
```
procedure GetPrinter (ADevice, ADriver, APort: PChar; var ADeviceMode:
THandle);
```

Description
The GetPrinter method retrieves the current printer. You should rarely need to call this method and should instead access the printer you want in the Printers property array. For more information, see the CreateDC function in the Win32 Developer's Reference (WIN32.HLP).

DropDown Event

Delphi equivalent
`OnDropDown Event`

Declaration
```
property OnDropDown: TNotifyEvent;
```

Description
The OnDropDown event occurs when the user opens (drops down) a combo box or list box.

Example
The following code doesn't sort the items in a combo box until the user opens it.

```
procedure TForm1.ComboBox1DropDown(Sender: TObject);
begin
  ComboBox1.Sorted := True;
end;
```

Duplex Property

Delphi equivalent

```
Use ADeviceMode Parameter of SetPrinter/GetPrinter Methods
```

Declaration

```
procedure GetPrinter (ADevice, ADriver, APort: PChar; var ADeviceMode:
THandle);
```

Description

The GetPrinter method retrieves the current printer. You should rarely need to call this method and should instead access the printer you want in the Printers property array. For more information, see the CreateDC function in the Win32 Developer's Reference (WIN32.HLP).

Dynaset Object

Delphi equivalent
`TQuery Component`

Description
See Data Access Components Page

EOF Function

Delphi equivalent
EOF Function

Declaration
Typed or untyped files:

```
function Eof(var F): Boolean;
```

Text files:

```
function Eof [ (var F: Text) ]: Boolean;
```

Description
The Eof function tests whether or not the current file position is the end-of-file.

F is a text file variable. If F is omitted, the standard file variable Input is assumed.

Eof(F) returns True if the current file position is beyond the last character of the file or if the file contains no components; otherwise, Eof(F) returns False.

{$I+} lets you handle run-time errors using exceptions. For more information on handling run-time library exceptions, see Handling RTL Exceptions. (Delphi.hlp)

Example

```
var
  F1, F2: TextFile;
  Ch: Char;
begin
  if OpenDialog1.Execute then begin
    AssignFile(F1, OpenDialog1.Filename);
    Reset(F1);
    if SaveDialog1.Execute then begin
      AssignFile(F2, OpenDialog1.Filename);
      Rewrite(F2);
      while not Eof(F1) do
      begin
        Read(F1, Ch);
        Write(F2, Ch);
      end;
      CloseFile(F2);
    end;
    CloseFile(F1);

  end;
end;
```

EOF Property

Delphi equivalent
`EOF Property`

Declaration
```
property EOF: Boolean;
```

Description
Run-time and read-only. EOF is a Boolean property that indicates whether a dataset is known to be at its last row. The EOF property returns a value of True after:

An application opens an empty dataset
A call to a table's Last method
A call to a table's Next fails because the cursor is on the last row

Example
This example uses TDataSource's OnDataChange event to detect when the user moves to another record. If the end of file is reached (EOF property becomes True), a message is displayed on the form's status bar.

```
procedure TForm1.DataSource1DataChange(Sender: TObject; Field: TField);
begin
  if Table1.EOF then
    StatusBar1.SimpleText := 'You''re already at the end of the table';
end;
```

EXEName Property

Delphi equivalent
`EXEName Property`

Declaration
```
property ExeName: string;
```

Description
Run-time and read-only. The ExeName property contains the name of the executable application including path information. The name of the application is the name you gave the project file with an .EXE extension. If you haven't specified a name, the default name is PROJECT1.EXE.

Example
This code displays the current name of the application's .EXE file in a label control when the user clicks the button:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := Application.ExeName;
end;
```

For example, if the application name is C:\DELPHI\WORK\MYAPP.EXE, that entire string appears in the label control.

Edit Method

Delphi equivalent
`Edit Method`

Declaration
```
procedure Edit;
```

Description
The Edit method prepares the current record of the dataset for changes and puts the dataset in Edit mode, setting the State property to dsEdit. Data-aware controls cannot modify existing records unless the dataset is in Edit mode.

Calling this method for a dataset that cannot be modified raises an exception. The CanModify property will be True for datasets that can be modified. This method is valid only for datasets that return a live result set.

Example
```
if Table1.State <> dsEdit then
  Table1.edit;
Table1.FieldByName( 'Name' ).AsString := Randy;
Table1.post;
```

EditMode Property

Delphi equivalent
`State Property`

Declaration
```
property State: TDataSetState;
```

Description
Run-time and read-only. The State property specifies the current state of the dataset. The possible values are those of the TDataSetState type:

dsInactive when the dataset is closed
dsBrowse when the dataset is in Browse state
dsEdit when the dataset is in Edit state
dsInsert when the dataset is in Insert state
dsSetKey when the dataset is in SetKey state
dsCalcFields when the OnCalcFields event is called.

Example
```
if Table1.State <> dsEdit then
  Table1.edit;
Table1.FieldByName( 'Name' ).AsString := Randy;
Table1.post;
```

Enabled Property

Delphi equivalent

`Enabled Property`

Description

The Enabled property determines if the control responds to mouse, keyboard, or timer events, or if the data-aware controls update each time the dataset they are connected to changes.

Example

To disable a button called FormatDiskButton,

```
FormatDiskButton.Enabled := False;
```

This code alternately dims or enables a menu command when a user clicks the button:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if OpenCommand.Enabled then
    OpenCommand.Enabled := False
  else
    OpenCommand.Enabled := True;
end;
```

End Statement

Delphi equivalent

End Statement

Desciption

The reserved word end marks the end of a block. End can be used with:

begin to form compound statements
case to form case statements
record to declare record types
object to declare object types
asm to call the built-in assembler
except to end an exception list
finally to end a finally block

The final end of a module is followed by a period to denote that there is nothing after it.

Examples

```
(* with begin to form compound statement *)
if First < Last then
begin
  Temp := First;
  First := Last;
  Last := Temp;
end;
(* with case statement *)
case Ch of
  'A'..'Z', 'a'..'z':  WriteLn('Letter');
  '0'..'9':            WriteLn('Digit');
  '+', '-', '*', '/':  WriteLn('Operator');
else
  WriteLn('Special character');
end;
(* in record type definitions *)
type
  MyClass  = (Num, Dat, Str);

  Date   = record
    D, M, Y: Integer;
  end;
  Facts = record
    Name: string[10];
    case Kind: MyClass of
      Num: (N: real);
      Dat: (D: Date);
      Str: (S: string);
  end;
(* in object type definitions *)
type
Location = object
  X, Y: Integer;
  procedure Init(PX, PY: Integer);
  function GetX: Integer;
  function GetY: Integer;
end;
(* with asm *)
```

```
asm
  mov ax,1
  mov cx, 100
end;
```

EndDoc Method

Delphi equivalent

EndDoc Method

Declaration

```
procedure EndDoc;
```

Description

The EndDoc method ends the current print job and closes the text file variable. After the application calls EndDoc, the printer begins printing. Use EndDoc after successfully sending a print job to the printer. If the print job isn't successful, use the Abort method.

The Close procedure calls the EndDoc method.

Example

This example uses a button on a form. When the user clicks it, the event handler prints a rectangle on the printer and displays a message on the form.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Printer do
  begin
    BeginDoc;
    Canvas.Rectangle(20, 20, 400, 300);
    EndDoc;
  end;
  Canvas.TextOut(10, 10, 'Printed');
end;
```

To use the EndDoc method, you must add the Printers unit to the uses clause of your unit.

Environ Function

Delphi equivalent
`GetEnvironmentVariable (API)`

Description
See Win32.HLP

Err Object

Delphi equivalent
`Exception Class`

Declaration
```
const ExceptionClass: TClass;
```

Description
ExceptionClass is a class reference variable that determines what exception classes will be reported by the debugger. ExceptionClass is set to Exception by default, so only objects descended from Exception and raised in the Raise statement will be reported by the debugger during a debug session.

Error Event

Delphi equivalent

```
EDBException Object (EDatabaseError)
```

Declaration

```
EDatabaseError = class(Exception);
```

Description

The EDatabaseError type is the exception type raised when a database error is detected by a component. Use EDatabaseError with an exception handling block or to create a database exception. With an exception handling block, you can detect the condition and handle it yourself. If something in your code encounters an error, you can create and raise the exception yourself.

Error Function

Delphi equivalent
`Exception Class`

Declaration
```
const ExceptionClass: TClass;
```

Description
See Exception Handling (Delphi.HLP)

Each statement in the except part of a try..except block defines code to execute to handle a particular kind of exception. The form of these exception-handling statements is as follows:

on <type of exception> do <statement>;

By using exceptions, you can spell out the "normal" expression of your algorithm, then provide for those exceptional cases when it doesn't apply. Without exceptions, you have to test every single time to make sure you're allowed to proceed with each step in the calculation.

Error Object, Errors Collection

Delphi equivalent
`Exception Class`

Declaration
```
const ExceptionClass: TClass;
```

Description
See Exception Handling (Delphi.HLP)

Each statement in the except part of a try..except block defines code to execute to handle a particular kind of exception. The form of these exception-handling statements is as follows:

on <type of exception> do <statement>;

By using exceptions, you can spell out the "normal" expression of your algorithm, then provide for those exceptional cases when it doesn't apply. Without exceptions, you have to test every single time to make sure you're allowed to proceed with each step in the calculation.

Error Statement

Delphi equivalent

`Raise`

Description

When an exception occurs, you might want to display some sort of message to the user, then proceed with the standard handling. To do that, you declare a local exception handler that displays the message then calls the reserved word raise.

Examples

For example, given the following Declaration,

```
type
  EPasswordInvalid = class(Exception);
```

You can raise a "password invalid" exception at any time by calling raise with an instance of EPasswordInvalid, like this:

```
if Password <> CorrectPassword then
  raise EPasswordInvalid.Create('Incorrect password entered');
```

Exclusive Property

Delphi equivalent

Exclusive Property

Declaration

```
property Exclusive: Boolean;
```

Description

Set the Exclusive property to True to prevent any other user from accessing the table. If other users are accessing the table when you try to open it, your exception handler will have to wait for those users to release it. If you do not provide an exception handler and another user already has the table open, your application will be terminated.

Example

```
{ Try to open Table1 with Exclusive True }
{ First, close Table1 }
Table1.Active := False;
repeat { until successful or Cancel button is pressed }
  try
    Table1.Exclusive := True; { See if it will open }
    Table1.Active := True;
    Break; { If no error, exit the loop }
  except
    on EDatabaseError do
      { Ask if it is OK to retry }
      if MessageDlg('Could not open Table1 exclusively - OK to retry?',
mtError,

        [mbOK, mbCancel], 0) <> mrOK then raise; { If not, reraise to abort }
      { Otherwise resume the repeat loop }
  end;
  until False;
```

Execute Method

Delphi equivalent

`Open Method`

Declaration

```
procedure Open;
```

Description

The Open method opens the dataset, putting it in Browse state. It is equivalent to setting the Active property to True.

For TQuery, Open executes the SELECT statement in the SQL property. If the statement does not return a result set (for example, an INSERT or UPDATE statement), then use ExecSQL instead of Open.

For TStoredProc, use Open to execute the stored procedure if the procedure returns a result set. If the stored procedure returns a single row, use ExecProc instead.

Example

```
try
  Query1.Open;
except
  on EDataBaseError do { The dataset could not be opened };
end;
```

ExecuteSQL Method

Delphi equivalent
`ExecSQL Method`

Declaration
```
procedure ExecSQL;
```

Description
Use the ExecSQL method to execute an SQL statement assigned to the SQL property of a TQuery if the statement does not return a result set. If the SQL statement is an INSERT, UPDATE, DELETE, or any DDL statement, then use this method.

If the SQL statement is a SELECT statement, use Open instead.

Example
```
Query1.Close;
Query1.SQL.Clear;
Query1.SQL.Add('Delete from Country where Name = 'Argentina');
Query1.ExecSQL;
```

Exit Statement

Delphi equivalent
```
Exit Procedure
```

Declaration
```
procedure Exit;
```

Description
The Exit procedure immediately passes control away from the current procedure.

If the current procedure is the main program, Exit causes the program to terminate.

Exit will cause the calling procedure to continue with the statement after the point at which the procedure was called.

Exp Function

Delphi equivalent

Exp Function

Declaration

```
function Exp(X: Real): Real;
```

Description

The Exp function returns the exponential of X.

The return value is e raised to the power of X, where e is the base of the natural logarithms.

Example

```
 var
  S: string;
begin
  S := 'e = ' + FloatToStr(Exp(1.0));
  Canvas.TextOut(10, 10, S);
end;
```

FROM Clause (SQL)

Delphi equivalent
FROM Clause (SQL)

Description
SQL Syntax

Example
The FROM clause specifies the table or tables from which to retrieve data. Table_reference can be a single table, a comma-delimited list of tables, or can be an inner or outer join as specified in the SQL-92 standard. For example, the following statement specifies a single table:

```
SELECT PART_NO
FROM "PARTS.DBF"
```

The next statement specifies a left outer join for table_reference:

```
SELECT * FROM PARTS LEFT OUTER JOIN INVENTORY
ON PARTS.PART_NO = INVENTORY.PART_NO
```

FV Function

Delphi equivalent

`FutureValue Function`

Declaration

```
function FutureValue(Rate: Extended; NPeriods: Integer; Payment,
PresentValue:
  Extended; PaymentTime: TPaymentTime): Extended;
```

Description

The FutureValue function returns the future value of an investment of PresentValue where Payment is invested for NPeriods at the rate of Rate per period. The PaymentTime parameter indicates whether the investment is an ordinary annuity or an annuity due (enter 0 if payments are at the end of each period, 1 if they are at the beginning).

FetchVerbs Method

Delphi equivalent
`UpdateVerbs Method`

Declaration
```
procedure UpdateVerbs;
```

Description
Refreshes the list of verbs the OLE object responds to. TOleContainer automatically calls UpdateVerbs when you first access the ObjectVerbs property, but some OLE objects change their verbs as they perform some actions. The Media Clip OLE object, for example, changes its "Play" verb to "Stop" while running. An OLE object must already be loaded in the container before calling UpdateVerbs.

Example
```
ListBox1.Items := OleContainer1.ObjectVerbs;
```

Field Object, Fields Collection

Delphi equivalent
`TField Component`

Description
See Data Access Page Components

FieldSize Method

Delphi equivalent
`Size Property`

Declaration
```
property Size: Integer;
```

Description
For a TStringField, Size is the number of bytes reserved for the field in the dataset. For a TBCDField, it is the number of digits following the decimal point. For a TBlobField, TBytesField, TVarBytesField, TMemoField, or TGraphicField, it is the size of the field as stored in the table.

Fields Property

Delphi equivalent
IndexFields Property

Declaration
```
property IndexFields[Index: Integer]: TField;
```

Description
Run-time only. The IndexFields property gives you access to information about each field of the current index for the dataset. The Active property must be True or the information will not be valid.

Example
```
S := '';
with Table1 do
{ Create a composite string with the index's names separated by "@" }
for I := 0 to IndexFieldCount - 1 do
    S := S + '@' + IndexFields[I].FieldName;
```

FileAttr Function

Delphi equivalent

FileGetAttr Function

Declaration

```
function FileGetAttr(const FileName: string): Integer;
```

Description

The FileGetAttr function returns the file attributes of the file given by FileName.

Example

```
Procedure TForm1.FormActivate( Sender : TObject );
begin
  If ( FileGetAttr( 'C:\Autoexec.bat' ) AND faReadOnly ) = 1 then
    CheckBox1.Checked := TRUE;
end;
```

FileCopy Statement

Delphi equivalent
`CopyFile (API)`

Description
See Win32.HLP

FileDateTime Function

Delphi equivalent
`FileGetDate Function`

Declaration
```
function FileGetDate(Handle: Integer): Integer;
```

Description
The FileGetDate function returns the DOS date-and-time stamp of the file given by Handle. The return value is -1 if the handle is invalid. The FileDateToDateTime function can be used to convert the returned value to a TDateTime value.

FileDescription Property

Delphi equivalent
`VerInfo (API)`

Description
See Win32.hlp

FileLen Function

Delphi equivalent
`FileSize Function`

Declaration
```
function FileSize(var F): Integer;
```

Description
The FileSize function returns the size in bytes of file F. However, if F is a record file FileSize will return the number of records in the file.

To use FileSize the file must be open and it can't be used on a text file.

F is a file variable.

If the file is empty, FileSize(F) returns 0.

Example
```
var
   f: file of Byte;
   size : Longint;
   S: string;
   y: integer;
 begin
   if OpenDialog1.Execute then begin
     AssignFile(f, OpenDialog1.FileName);
     Reset(f);
     size := FileSize(f);
     S := 'File size in bytes: ' + IntToStr(size);
     y := 10;
     Canvas.TextOut(5, y, S);
     y := y + Canvas.TextHeight(S) + 5;
     S := 'Seeking halfway into file...';
     Canvas.TextOut(5, y, S);
     y := y + Canvas.TextHeight(S) + 5;

     Seek(f,size div 2);
     S := 'Position is now ' + IntToStr(FilePos(f));
     Canvas.TextOut(5, y, S);
     CloseFile(f);
   end;
 end;
```

FileListBox Control

Delphi equivalent
`FileListBox Component`

Description
See System Page Components

FileName Property

Delphi equivalent
`FileName Property`

Declaration
```
property FileName: string;
```

Description
Run-time only. The FileName property contains the name of the selected file in the list box, including the path name.

Example
This example displays an Open dialog box and suggests the filename LIST.PAS to the user. Once the user selects a filename, the code displays that name in a label on the form:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  OpenDialog1.FileName := 'LIST.PAS';
  if OpenDialog1.Execute then
    Label1.Caption := OpenDialog1.FileName;
end;
```

FileTitle Property

Delphi equivalent
`Title Property`

Applies to
TOpenDialog, TSaveDialog components

Declaration
```
property Title: string;
```

Description
The Title property determines the text that appears in the dialog box's title bar.

Example
This code displays the Open dialog box with the text "Open Pascal files" in its title bar and lists only Pascal files in the list box:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  OpenDialog1.Filter := 'Pascal files (*.PAS)|*.PAS';
  OpenDialog1.Title := 'Open Pascal files';
  OpenDialog1.Execute;
end;
```

FillColor Property

Delphi equivalent
`Brush.Color Property`

Description
A TBrush object is used when filling solid shapes, such as rectangles and ellipses. The interior of the shape is filled with a color or pattern. TBrush encapsulates a Windows HBRUSH.

The color of the brush is specified by the Color property.

Example
This example displays a rectangle filled with red horizontal stripes whenever a form OnPaint event occurs.

```
procedure TForm1.FormPaint(Sender: TObject);
begin
  with Canvas do
  begin
    Brush.Style := bsHorizontal;
    Brush.Color := clRed;
    Rectangle(12, 50, 100, 200);
  end;
end;
```

FillStyle Property

Delphi equivalent
`Brush.Style Property`

Description
A TBrush object is used when filling solid shapes, such as rectangles and ellipses. The pattern is specified by the Style property.

Example
This example displays a rectangle filled with red horizontal stripes whenever a form OnPaint event occurs.

```
procedure TForm1.FormPaint(Sender: TObject);
begin
  with Canvas do
  begin
    Brush.Style := bsHorizontal;
    Brush.Color := clRed;
    Rectangle(12, 50, 100, 200);
  end;
end;
```

Filter Property

Delphi equivalent
`Filter Property`

Applies to
TTable, TQuery, TStoredProc components

Declaration
```
property Filter: string;
```

Description
The Filter property is a string that lets you specify which records you want to see in the data set. Filters are similar to, though less powerful than, queries, with the benefit that filters work on the data set itself, meaning that the result is always "live" (unlike queries which sometimes produce result sets that can't be modified). You can turn a filter on and off by changing the Filtered property.

The syntax for the filter string is very similar to that used in the WHERE clause of an SQL statement. See Delphi.HLP for more information.

Example
```
Procedure TForm1.Button1Click( Sender : TObject );
begin
  Table1.Filter := 'Company = ''Unisco''';
  Table1.Filtered := True;
end;
```

Filter Property (Common Dialog)

Delphi equivalent
`Filter Property (Dialogs)`

Applies to
TOpenDialog, TSaveDialog components

Declaration
```
property Filter: string;
```

Description
The Filter property determines the file masks available to the user for use in determining which files display in the dialog box's list box.

Example
This code sets the value of the Filter property, displays the dialog box, and assigns the file name the user selects to a variable:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  NameOfFile : TFileName;
begin
  OpenDialog1.Filter := 'Text files (*.TXT)|*.TXT|Pascal files (*.PAS)' +
    '|*.PAS|Quattro Pro files (*.WB1)|*.WB1';
  if OpenDialog1.Execute then
    NameOfFile := OpenDialog1.FileName;

end;
```

FilterIndex Property

Delphi equivalent
`FilterIndex Property`

Applies to
TOpenDialog, TSaveDialog components

Declaration
```
property FilterIndex: Integer;
```

Description
The FilterIndex property determines which file filter specified in the Filter property appears as the default file filter in the List Files of Type drop-down list box. For example, if you set the FilterIndex value to 2, the second file filter listed in the Filter property becomes the default filter when the dialog box appears. The default FilterIndex value is 1. If you specify a value greater than the number of file filters in the Filter property, the first filter is chosen.

The default value is 1.

Example
This code specifies three file filters as the value of the Filter property, sets the FilterIndex to 2 so that the second file filter is the default file filter, and displays the Open dialog box. Once the user selects a file with the dialog box and chooses OK, the filename the user selected appears in a label on the form.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  OpenDialog1.Filter := 'Text files (*.TXT)|*.TXT|Pascal files (*.PAS)' +
    '|*.PAS|dBASE program files (*.PRG)|*.PRG';
  OpenDialog1.FilterIndex := 2;
  if OpenDialog1.Execute then
    Label1.Caption := OpenDialog1.FileName;
end;
```

FindFirst, FindLast, FindNext, FindPrevious Method

Delphi equivalent
`Lookup`

Applies to
TTable, TQuery, TStoredProc components

Declaration
```
function Lookup(const KeyFields: string; const KeyValues: Variant;
     const ResultFields: string): Variant;
```

Description
The Lookup method locates and moves to the first record matching the supplied search criteria. KeyFields lists the field or fields you want to search; to search more than one, separate each field name with a semicolon. KeyValues is a variant specifying the field value to match, or an array of field values, if KeyFields lists more than one field.

Lookup uses the fastest possible method to locate a matching record; if the dataset has an index on the specified fields, the index will be used. Otherwise, Lookup sets up a Borland Database Engine filter for efficient searching.

If a matching record couldn't be found, Lookup returns Null. If one could be found, Lookup performs the lookup operations you set up in the lookup fields defined for the dataset (if any). It then extracts the value of the field specified in ResultFields and uses it as its return value. You can list multiple fields in ResultFields by separating their names with semicolons; if you do so, the field values will be returned in a variant array.

FirstRow Property

Delphi equivalent
`TopRow Property`

Declaration
```
property TopRow: Longint;
```

Description
Run-time only. The TopRow property determines which row in the grid appears at the top of the grid.

If you have one or more nonscrolling rows in the grid, they remain at the top, regardless of the value of the TopRow property. In this case, the row you specify as the top row will be the first row below the nonscrolling rows.

Example
This code uses a string grid and a button on a form. When the user clicks the button, the last row of the string grid becomes the top row:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  StringGrid1.TopRow := StringGrid1.RowCount;
end;
```

Fix Function

Delphi equivalent
`Trunc Function`

Declaration
```
function Trunc(X: Extended): Longint;
```

Description
The Trunc function truncates a real-type value to an integer-type value.

X is a real-type expression. Trunc returns a Longint value that is the value of X rounded toward zero.

If the truncated value of X is not within the Longint range, an error occurs, which you can handle using the EInvalidOp exception. If you do not handle it, you will receive a run-time error.

Example
```
var
   S, T: string;
begin
   Str(1.4:2:1, T);
   S := T + ' Truncs to ' + IntToStr(Trunc(1.4)) + #13#10;
   Str(1.5:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(1.5)) + #13#10;
   Str(-1.4:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(-1.4)) + #13#10;
   Str(-1.5:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(-1.5));
   MessageDlg(S, mtInformation, [mbOk], 0);
end;
```

FixedCols, FixedRows Properties

Delphi equivalent
`FixedCols, FixedRows Properties`

Applies to
TDrawGrid, TStringGrid components

Declaration
```
property FixedCols: Integer;
```

Description
The FixedCols property determines the number of nonscrolling columns within a grid. The default value is 1. Nonscrolling columns remain fixed at the far left of the grid, even when the user scrolls the other columns. Nonscrolling columns are useful for displaying row titles that need to remain visible in the grid at all times.

Each grid must have a least one column that isn't fixed. In other words, the value of the FixedCols property must always be at least one less than the value of the ColCount property, which contains the number of columns in the grid.

Example
This example uses a string grid and a button. When the user clicks the button, a message dialog box appears informing the user that a fixed column number of 2 is recommended. The dialog box also offers the user an opportunity to accept the recommended number if the number of fixed columns isn't already 2. If the user chooses Yes, the number of fixed columns changes to 2.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Check: Integer;
begin
  if StringGrid1.FixedCols <> 2 then
  begin
    Check := MessageDlg('2 fixed columns are recommended!  Change?',
        mtWarning, mbYesNoCancel, 0);
    if Check = IdYes then
       StringGrid1.FixedCols := 2;
  end;
end;
```

Flags Property (Color Dialog)

Delphi command

Options Property

Applies to

TColorDialog component

Declaration

```
property Options: TColorDialogOptions;
```

Description

These are the possible values that can be included in the Options set:

Value	Meaning
cdFullOpen	Displays the custom coloring options when the Color dialog opens
cdPreventFullOpen	Disables the Create Custom Colors button in the Color dialog box so the user cannot create their own custom colors.
cdShowHelp	Adds a Help button to the Color dialog box.
cdSolidColor	Directs the dialog box to display only solid colors.
cdAnyColor	Directs the dialog box to display any color.

The default value is [ ], the empty set, meaning all of these values are False and none of the options are in effect.

Example

This example displays the Color dialog box with a Help button and the Create Custom Colors button dimmed. The form is colored whatever color the user chooses.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ColorDialog1.Options := [cdPreventFullOpen, cdShowHelp];
  if ColorDialog1.Execute then
    Color := ColorDialog1.Color;
end;
```

Flags Property (File Dialog)

Delphi equivalent
`Options Property`

Applies to
TOpenDialog, TSaveDialog component

Declaration
```
property Options: TOpenOptions;
```

Description
See Delphi.HLP for the possible values that can be included in the Options set for the Open and Save dialog boxes.

Example
This example uses an Open dialog box and a button on a form. The code forces the user to enter valid filename characters, prevents the read-only check box from appearing in the dialog box, and let's the user choose to overwrite a file if the user selects a file that doesn't exist; the selected filename appears in a label on the form:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  OpenDialog1.Options := [ofNoValidate, ofHideReadOnly, ofCreatePrompt];
  if OpenDialog1.Execute then
    Label1.Caption := OpenDialog1.FileName;
end;
```

Flags Property (Font Dialog)

Delphi equivalent
`Options Property`

Applies to
TFontDialog component

Declaration
```
property Options: TFontDialogOptions
```

Description
See Delphi.hlp for the possible values that can be included in the Options set for the Fonts dialog box.

Example
This example sets the options of the Font dialog box so that when the dialog box displays, only TrueType fonts show in the list of fonts and no font size is selected:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  FontDialog1.Options := [fdTrueTypeOnly, fdNoSizeSel];
  if FontDialog1.Execute then
    Memo1.Font := FontDialog1.Font;
end;
```

Flags Property (Print Dialog)

Delphi equivalent
`Options Property`

Applies to
TPrintDialog component

Declaration
```
property Options: TPrintDialogOptions;
```

Description
See Delphi.hlp for the possible values that can be included in the Options set for the Print dialog box.

Example
This example displays the Printer dialog box that includes a Help button. If users try to print when no printer is installed, they will see a warning message.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  PrinterDialog1.Options := [poHelp, poWarning];
  if PrinterDialog1.Execute then
    ...
end;
```

Font Object

Delphi equivalent
`TFont Object`

Description
A TFont object defines the appearance of text. TFont encapsulates a Windows HFONT.

A TFont object defines a set of characters by specifying their height, font family (typeface) name, and so on. The height is specified by the Height property. The typeface is specified by the Name property. The size in points is specified by the Size property. The color is specified by the Color property. The attributes of the font (bold, italic, and so on) are specified by the Style property.

When a font is modified, an OnChange event occurs.

Font Property

Delphi equivalent

```
Font Property
```

Declaration

```
property Font: TFont;
```

Description

The Font property is a font object that controls the attributes of text written on or in the component or object or sent to the printer. To modify a font, you change the value of the Color, Name, Size, or Style properties of the font object.

Example

This code changes color of text in a memo control to dark blue:

```
Memo1.Font.Color := clNavy;
```

FontBold, FontItalic, FontStrikethru, FontUnderline Property

Delphi equivalent
`Style Property`

Applies to
TFont objects

Declaration
```
property Style: TFontStyles;
```

Description
The Style property determines whether the font is normal, italic, underlined, bold, and so on. These are the possible values:

Value	Meaning
fsBold	The font is boldfaced.
fsItalic	The font is italicized.
fsUnderline	The font is underlined.
fsStrikeout	The font is displayed with a horizontal line through it.

The Style property is a set, so it can contain multiple values. For example, a font could be both boldfaced and italicized.

Example
The following code boldfaces the font used in the memo..

```
Memo1.Font.Style := [fsBold];
```

FontName Property

Delphi equivalent
`Name Property`

Applies to
TFont objects

Declaration
```
property Name: TFontName;
```

Description
The Name property of a font object determines the name of the font contained within the font object.

Example
This code sets the font for all text that appears on the form to Times New Roman. If the controls on the form have their ParentFont property set to True, text on these controls will also be in Times New Roman.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Font.Name := 'Times New Roman';
end;
```

FontSize Property

Delphi equivalent
`Size Property`

Applies to
TFont component

Declaration
```
property Size: Integer;
```

Description
The Size property is the size of the font in points. It is the height of the font plus the font's excluding internal leading. If you are concerned with the height of the font on the screen--the number of pixels the font needs--use the Height property instead. If you want to specify a font's height using pixels, use the Size property.

Delphi calculates Size using this formula:

Font.Size = -Font.Height * Font.PixelsPerInch / 72

Therefore, whenever you enter a point size in the Size property, you'll notice the Height property changes to a negative value. Conversely, if you enter a positive Height value, the Size property value changes to a negative value.

Example
This examples uses a button on a form. When the user clicks the button, the size of the font used by the button changes to 24 points.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Font.Size := 24;
end;
```

Fonts Property

Delphi equivalent
`Fonts Property`

Applies to
TPrinter object; TScreen component

Declaration
```
property Fonts: TStrings;
```

Description
Run-time and read-only. The Fonts property for the screen component returns a list of fonts supported by the screen.

The Fonts property for a printer object holds a list of fonts supported by the printer. The list contains TrueType fonts even if the printer doesn't support them natively because the Windows Graphics Device Interface (GDI) can draw TrueType fonts accurately when a print job uses them.

Example
This code displays the fonts supported by the screen in a FontList list box when the user clicks the ListFonts button:

```
procedure TForm1.ListFontsClick(Sender: TObject);
begin
  FontList.Clear;
  FontList.Sorted := True;
  FontList.Items := Screen.Fonts;
end;
```

For...Next Statement

Delphi equivalent

For..To, For...Downto Reserved Words

Description

The for statement causes the statements after do to be executed once for each value between the initial value of the range and final value, inclusive. For loops are useful if you know beforehand exactly how many times you want the loop to be executed.

The control variable always starts off at initial value.

to	Increments the control variable by 1 for each loop. The initial value must be less than the final value.
downto	Decrements the control variable by 1 for each loop. The initial value must be greater than the final value.

The following rules apply to the control variable:

It must be a variable identifier that is local in scope to the block containing the for statement.
It must be of an ordinal type.
The initial and final values must be of a type assignment compatible with the ordinal type of the control variable.
After a for statement is executed, the value of the control variable is undefined, unless execution of the for statement was interrupted by a goto statement.

Example

```
(* for ... to, for ... downto *)
for I := 1 to ParamCount do
  WriteLn(ParamStr(I);
for I := 1 to 10 do
  for J := 1 to 10 do
  begin
    X := 0;
    for K := 1 to 10 do
      X := X + Mat1[I, K] * Mat2[K, J];
    Mat[I, J] := X;
  end;
```

ForeColor Property

Delphi equivalent
`Font.Color Property`

Description
See BackColor

Example
This code will set the label's color and font color to the current windows settings.

```
Label1.Color := clWindow;
Label1.Font.Color := clWindowText;
```

ForeignName Property

Delphi equivalent
`Use MasterSource, MasterFields Properties`

Applies to
TTable component

Declaration
```
property MasterSource: TDataSource;
```

Description
When linking a detail table to a master table, use the MasterSource property to specify the TDataSource from which the TTable will get data for the master table.

Example
Suppose you have a master table named Customer that contains a CustNo field, and you also have a detail table named Orders that also has a CustNo field. To display only those records in Orders that have the same CustNo value as the current record in Customer, write this code:

```
Orders.MasterSource := 'CustSource';
Orders.MasterFields := 'CustNo';
```

If you want to display only the records in the detail table that match more than one field value in the master table, specify each field and separate them with a semicolon.

```
Orders.MasterFields := 'CustNo;SaleDate';
```

ForeignTable Property

Delphi equivalent
`Use MasterSource, MasterFields Properties`

Applies to
TTable component

Declaration
```
property MasterSource: TDataSource;
```

Description
When linking a detail table to a master table, use the MasterSource property to specify the TDataSource from which the TTable will get data for the master table.

Example
Suppose you have a master table named Customer that contains a CustNo field, and you also have a detail table named Orders that also has a CustNo field. To display only those records in Orders that have the same CustNo value as the current record in Customer, write this code:

```
Orders.MasterSource := 'CustSource';
Orders.MasterFields := 'CustNo';
```

If you want to display only the records in the detail table that match more than one field value in the master table, specify each field and separate them with a semicolon.

```
Orders.MasterFields := 'CustNo;SaleDate';
```

Form Object, Forms Collection

Delphi equivalent

`TForm Component, TScreen.Forms Property`

Description

The Form component is at the center of Delphi applications. You design your application by putting other components on a form. Forms can be used as windows, dialog boxes, or simply as forms, such as data-entry forms.

Example

The following code adds the name of all forms on the screen to ListBox1 when Button1 is clicked.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  For I := 0 to Screen.FormCount-1 do
    ListBox1.Items.Add(Screen.Forms[I].Name);
end;
```

Format Function

Delphi equivalent

```
Format Function
```

Declaration

```
function Format(const Format: string; const Args: array of const): string;
```

Description

This function formats the series of arguments in the open array Args. Formatting is controlled by the Object Pascal format string Format; the results are returned in the function result as a Pascal string.

For information on the format strings, see Format Strings in DELPHI.HLP.

Example

```
Format('%d %d %0:d %d', [10, 20]) = '10 20 10 20'.
```

Format Property

Delphi equivalent
`EditMask Property`

Applies to
TDateField, TDateTimeField, TMaskEdit, TStringField, TTimeField components

Declaration
```
property EditMask: string;
```

Description
The EditMask property is the mask that is used to limit the data that can be put into a masked edit box or entered into a data field. A mask restricts the characters the user can enter to valid characters and formats. If the user attempts to enter a character that is not valid, the edit box does not accept the character. Validation is performed on a character-by-character basis. Use an OnValidate event to validate the entire input.

Example
This example assigns an edit mask to the masked edit box on the form. The edit mask makes it easy to enter American telephone numbers in the edit box.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  MaskEdit1.EditMask := '!\(999\)000-0000;1;
  MaskEdit1.Text := '';
  MaskEdit1.AutoSelect := False;
end;
```

Frame Control

Delphi equivalent
`GroupBox Component`

Description
Standard Page Components

FreeLocks Statement

Delphi equivalent

DBISaveChanges (BDE)

Example

```
Check( DBISaveChanges( Table1.Handle ));
```

FromPage, ToPage Properties

Delphi equivalent
`FromPage, ToPage Properties`

Applies to
TPrintDialog component

Declaration
```
property FromPage: Integer;
```

Description
The value of the FromPage property determines on which page the print job begins. The default value is 0.

Example
This example uses a Print dialog box on a form. These lines set up the Print dialog box so that when it appears, the default values of 1 and 1 are the default starting and ending values for the Pages From and To edit boxes.

```
PrintDialog1.Options := [poPageNums];
PrintDialog1.FromPage := 1;
PrintDialog1.ToPage := 1;
```

Function Statement

Delphi equivalent
`Function Reserved Word`

Description
The reserved word function defines a block that computes and returns a value.

The function heading specifies the identifier for the function, the formal parameters (if any), and the function result type.

Functions can return values of any type except file types.

A function can have Declaration parts following the function heading.
The function heading is followed by:

Declarations of local variables, types, labels, constants, procedures, or functions
Statements that execute when the function is called

Example

(* Function Declaration *)

```
function UpCaseStr(S: string): string;
var
  I: Integer;
begin
  for I := 1 to Length(S) do
  if (S[I] >= 'a') and (S[I] <= 'z') then
  Dec(S[I], 32);
  UpCaseStr := S;
end;
```

GROUPBY Clause (SQL)

Delphi equivalent
GROUP BY Clause (SQL)

Description
SQL Syntax

Example

```
SELECT DISTINCT CustNo, COUNT(*)
FROM "Orders.db"
GROUP BY CustNo
ORDER BY CustNo
```

Get Statement

Delphi equivalent

`Read Procedure`

Declaration

Typed files:

```
procedure Read(F , V1 [, V2,...,Vn ] );
```

Text files:

```
procedure Read( [ var F: Text; ] V1 [, V2,...,Vn ] );
```

Description

The Read procedure can be used in the following ways.

For typed files, it reads a file component into a variable.
For text files, it reads one or more values into one or more variables.

Example

```
uses Dialogs;
 var
  F1, F2: TextFile;
  Ch: Char;
begin
  if OpenDialog1.Execute then begin
    AssignFile(F1, OpenDialog1.Filename);
    Reset(F1);
    if SaveDialog1.Execute then begin
      AssignFile(F2, OpenDialog1.Filename);
      Rewrite(F2);
      while not Eof(F1) do
      begin
        Read(F1, Ch);
        Write(F2, Ch);
      end;
      CloseFile(F2);
    end;
    CloseFile(F1);

  end;
end.
```

GetAllSettings Function

Delphi command
`ReadSection Method`

Applies to
TIniFile object

Declaration
```
procedure ReadSection (const Section: string; Strings: TStrings);
```

Description
The ReadSection method reads all the variables of a section of an .INI file into a string object. The Strings parameter specifies the string list object. If you want to use a string list that is maintained by a component such as a list box, Strings should specify the property of the component that contains the string list. If you want to maintain the string list independent of any components, use a TStringList object.

GetAttr Function

Delphi command
FileGetAttr Function

Declaration
```
function FileGetAttr(const FileName: string): Integer;
```

Description
The FileGetAttr function returns the file attributes of the file given by FileName.

Example
```
Procedure TForm1.FormActivate( Sender : TObject );
begin
  If ( FileGetAttr( 'C:\Autoexec.bat' ) AND faReadOnly ) = 1 then
    CheckBox1.Checked := TRUE;
end;
```

GetBookmark Method

Delphi command
`GetBookmark Method`

Applies to
TTable, TQuery, TStoredProc components

Declaration
```
function GetBookmark: TBookmark;
```

Description
The GetBookmark method saves the current record information of the dataset to allow you to return to that record with a later call to the GotoBookmark method. The bookmark should be eventually be passed to the FreeBookmark method to release the resources reserved during the call to GetBookmark. If the dataset is empty or not in Browse state, GetBookmark will return nil.

Example
```
var MyBookmark: TBookmark;
...
with Table1 do
  begin
{ Save the current record position in MyBookmark }
  MyBookmark := GetBookmark;
  ... { Other code here }
{ Return to the record associated with MyBookmark }
  GotoBookmark(MyBookmark);
{ Release the resources for MyBookmark }
  FreeBookmark(MyBookmark);
  end;
```

GetData Method

Delphi command
`GetAsHandle Method`

Applies to
TClipboard object

Declaration
```
function GetAsHandle (Format: Word): THandle;
```

Description
The GetAsHandle method returns the data from the Clipboard in a windows handle for the format specified in the Format parameter. For more information, see the Win32 Developer's Reference (WIN32.HLP) for information about the available formats.

Example
The following code locks the memory for text on the Clipboard, then converts the text to a Pascal-style string.

```
var
  MyHandle: THandle;
  TextPtr: PChar;
  MyString: string;
begin
  ClipBoard.Open
  Try
    MyHandle := Clipboard.GetAsHandle(CF_TEXT);
    TextPtr := GlobalLock(MyHandle);
    MyString := StrPas(TextPtr);
    GlobalUnlock(MyHandle);
  finally
    Clipboad.Close;
  end;
end;
```

GetFormat Method

Delphi command
`HasFormat Method`

Applies to
TClipboard object

Declaration
```
procedure HasFormat(Format: Word): Boolean;
```

Description
The HasFormat method determines if the Clipboard object contains a particular format. If asFormat is True, the format is present; if False, the format is absent. The Clipboard object keeps a list of available formats in the Formats array property.

Example
This example uses a button on a form. When the user clicks the button, a message box appears if there is no text on the Clipboard; otherwise, you don't see anything happen.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not Clipboard.HasFormat(CF_TEXT) then
      MessageDlg('There is no text on the Clipboard', mtInformation,
        [mbOK],0);
end;
```

GetSetting Function

Delphi command

```
See TRegistry and TRegIniFile Objects
```

Description

A TRegistry object is a low-level wrapper for the Microsoft Windows95/NT system registry and functions that operate on the registry. The registry is a database that your applications can use to store and retrieve configuration information. Configuration information is stored in a hierarchical tree. Each node in the tree is called a key. Every key can contain subkeys and data values that represent part of the configuration information for an application.

GetText Method

Delphi command
`AsText Property`

Applies to
TClipboard object

Declaration
```
property AsText: string;
```

Description
Run-time only. The AsText property returns the current contents of the Clipboard as a string. The Clipboard must contain a string or an exception occurs. The string value of the AsText property is limited to 255 characters. If you need to set and retrieve more than 255 characters, use the SetTextBuf and GetTextBuf Clipboard methods.

Example
The following code retrieves the contents of the Clipboard as a string and displays the value in a label:

```
begin
  Label1.Caption := Clipboard.AsText;
end;
```

GoTo Statement

Delphi command

GoTo Reserved Word

Description

The reserved word goto transfers program execution to the statement prefixed by the label referenced in the statement.

Example

```
label 1, 2;
goto 1
   .
   .
   .
1: WriteLn ('Abnormal program termination');
2: WriteLn ('Normal program termination');
```

GotFocus Event

Delphi command
`OnEnter Event`

Declaration
```
property OnEnter: TNotifyEvent;
```

Description
The OnEnter event occurs when a component becomes active. Use the OnEnter event handler to specify any special processing you want to occur when a component becomes active.

Example
This example uses an edit box and a memo control on a form. When either the edit box or the memo is the active control, it is colored yellow. When the active control becomes inactive, the color of the control returns to the Windows system color for a window.

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;
procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;
procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

Grid Control

Delphi command
`TStringGrid, TDrawGrid Components`

Description
See Additional Page Components

GridLineWidth Property

Delphi command
`GridLineWidth Property`

Applies to
TDrawGrid, TStringGrid components

Declaration
```
property GridLineWidth: Integer;
```

Description
The GridLineWidth property determines the width of the lines between the cells in the grid. The default value is 1 pixel. Larger values create heavier lines.

Example
This example includes a draw grid on a form. When the application runs and the form is created, the width of the lines on the draw grid changes if the default column width of the grid is over 90 pixels wide:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  with DrawGrid1 do
  begin
    if DefaultColWidth > 90 then
      GridLineWidth := 2
    else
      GridLineWidth := 1;
   end;
end;
```

GridLines Property

Delphi command
`Options Property`

Applies to
TDrawGrid, TStringGrid component

Declaration
```
property Options: TGridOptions;
```

Description
See Delphi.HLP for Options properties on Grid controls

Example
This code changes the look of the grid; only horizontal lines appear in both the body of the grid and in the nonscrolling regions when the user clicks the ChangeGridStyle button:

```
procedure TForm1.ChangeGridStyleClick(Sender: TObject);
begin
  DrawGrid1.Options := [goFixedHorzLine, goHorzLine, goVertLine];
end;
```

Handle Property

Delphi command
`Handle Property`

Applies to
TBitmap, TBrush, TCanvas, TFont, TIcon, TMetafile, TPen, objects

Declaration
```
property Handle: HBitmap;    {for TBitmap objects}
property Handle: HBrush;     {for TBrush objects}
property Handle: HDC;        {for TCanvas objects}
property Handle: HFont;      {for TFont objects}
property Handle: HIcon;      {for TIcon objects}
property Handle: HMetafile;  {for TMetafile objects}
property Handle: HPen;       {for TPen objects}
```

Description
The Handle property lets you access the Windows GDI object handle, so you can access the GDI object. If you need to use a Windows API function that requires the handle of a pen object, you could pass the handle from the Handle property of a TPen object.

Having Clause (SQL)

Delphi command

Having Clause (SQL)

Description

SQL Syntax

Example

```
Select DeptNo, DeptName from Depts
Group By DeptNo, DeptName
Having Count(*) < 5
```

HeadFont Property

Delphi command
`TitleFont Property`

Applies to
TDBGrid component

Declaration
```
property TitleFont: TFont;
```

Description
The TitleFont property determines which font and text attributes are used to display the title of a column in a DBGrid component. At design time, the TitleFont property is set by bringing up the Font dialog box. To modify the TitleFont at runtime, change the value of the Color, Name, Size, or Style properties of the font object

Example
```
DBGrid1.TitleFont.Name := 'MS Sans Serif';
DBGrid1.TitleFont.Style := [fsBold];
```

Height, Width Properties

Delphi command
`Height, Width Properties`

Applies to
All controls; TBitmap, TFont, TGraphic, TIcon, TMetafile, TPicture, TTextAttributes objects; TForm, TImageList, TScreen components

Description
The Height property determines vertical size of a component or object. The Width property determines the horizontal size of a component or object.

Example
The following code doubles the height of a list box control:

```
ListBox1.Height := ListBox1.Height * 2;
```

HelpCommand Property

Delphi command
`HelpCommand Method`

Applies to
TApplication component

Declaration
```
function HelpCommand(Command: Word; Data: Longint): Boolean;
```

Description
The HelpCommand method gives you quick access to any of the Help commands in the WinHelp API (application programming interface). For information about the commands you can call and the data passed to them, see the WinHelp topic in the Help system.

Example
This example uses a bitmap button on a form. When the user clicks the button, the Help contents screen of the specified Help file appears.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  Application.HelpFile := 'MYHELP.HLP';
  Application.HelpCommand(HELP_CONTENTS, 0);
end;
```

HelpContext Property

Delphi command
`HelpContext Property`

Applies to
All controls

Declaration
```
property HelpContext: THelpContext;
```

Description
The HelpContext property provides a context number for use in calling context-sensitive online Help. Each screen in the Help system should have a unique context number. When a component is selected in the application, pressing F1 displays a Help screen. Which Help screen appears depends on the value of the HelpContext property.

HelpContext Property (CommonDialog)

Delphi command

`HelpContext Property`

Declaration

```
property HelpContext: THelpContext;
```

Description

The HelpContext property provides a context number for use in calling context-sensitive online Help. Each screen in the Help system should have a unique context number. When a component is selected in the application, pressing F1 displays a Help screen. Which Help screen appears depends on the value of the HelpContext property.

Example

The following code associates a Help file with the application, and makes the screen with a context number of 7 the context-sensitive Help screen for the Edit1 edit box:

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  Application.HelpFile := 'MYHELP.HLP';
  PrintDialog1.HelpContext := 7;
end;
```

HelpContext, HelpFile Properties (Data Access)

Delphi command
`HelpContext Property`

Declaration
```
property HelpContext: THelpContext;
```

Description
The HelpContext property provides a context number for use in calling context-sensitive online Help. Each screen in the Help system should have a unique context number. When a component is selected in the application, pressing F1 displays a Help screen. Which Help screen appears depends on the value of the HelpContext property.

HelpContextID Property

Delphi command
`HelpContext Property`

Declaration
```
property HelpContext: THelpContext;
```

Description
The HelpContext property provides a context number for use in calling context-sensitive online Help. Each screen in the Help system should have a unique context number. When a component is selected in the application, pressing F1 displays a Help screen. Which Help screen appears depends on the value of the HelpContext property.

Example
The following code associates a Help file with the application, and makes the screen with a context number of 7 the context-sensitive Help screen for the Edit1 edit box:

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  Application.HelpFile := 'MYHELP.HLP';
  Edit1.HelpContext := 7;
end;
```

HelpFile Property (App, CommonDialog, MenuLine)

Delphi command
`HelpFile Property`

Applies to
TApplication component

Declaration
```
property HelpFile: string;
```

Description
Run-time only. The HelpFile property holds the name of the file the application uses to display Help. By default, HelpFile is a null string, and the application's Help method ignores attempts to display Help. If HelpFile contains anything, the HelpContext method passes it to the Windows Help system as the name of the file to use for Help.

Example
To specify the MYHELP.HLP file as the Help file for your application, use this line of code:

```
Application.HelpFile := 'MYHELP.HLP';
```

Hex Function

Delphi command

`Format Function`

Declaration

```
function Format(const Format: string; const Args: array of const): string;
```

Description

This function formats the series of arguments in the open array Args. Formatting is controlled by the Object Pascal format string Format; the results are returned in the function result as a Pascal string. x is the argument for Hexadecimal. The argument must be an integer value. The value is converted to a string of hexadecimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has fewer digits, the resulting string is left-padded with zeros.

Hidden Property

Delphi command
`FileType Property`

Applies to
TFileListBox component

Declaration
```
property FileType: TFileType;
```

Description
The FileType property determines which files are displayed in the file list box based on the attributes of the files. When ftHidden is True, the list box can display files with the hidden attribute.

Example
This example uses a file list box on a form. When the application runs, only read-only files, directories, volume IDs, and files with no attributes appear in the list box.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileType := [ftReadOnly, ftDirectory, ftVolumeID, ftNormal];
end;
```

Hide Method

Delphi command
Hide Method

Declaration
```
procedure Hide;
```

Description
The Hide method makes a form or control invisible by setting the Visible property of the form or control to False. Although a form or control that is hidden is not visible, you can still set the properties of the form or control, or call its methods.

Example
This code uses a button and a timer on a form. When the user clicks the button, the form disappears for the period of time specified in the Interval property of the timer control, then the form reappears:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Timer1.Enabled := True;
  Hide;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Visible := True;
  Timer1.Enabled := False;
end;
```

HideSelection Property

Delphi command
`HideSelection Property`

Applies to
TEdit, TListView, TMemo, TRichEdit, TTreeView components

Declaration
```
property HideSelection: Boolean;
```

Description
The HideSelection property determines whether text that is selected in an edit or memo remains selected when the focus shifts to another control. If True, the text is no longer selected until the focus returns to the control. If False, the text remains selected. The default value is True.

Example
This example uses an edit box and a memo on a form. When the user jumps from one control to the other, selected text remains selected in the memo, but not in the edit box.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit1.HideSelection := True;
  Memo1.HideSelection := False;
end;
```

Hour Function

Delphi command
DecodeTime Procedure

Declaration
```
procedure DecodeTime(Time: TDateTime; var Hour, Min, Sec, MSec: Word);
```

Description
The DecodeTime procedure breaks the value specified as the Time parameter into hours, minutes, seconds, and milliseconds.

Example
This example uses a button and two labels on a form. When the user clicks the button, the current date and time are reported in the captions of the two labels.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Present: TDateTime;
  Year, Month, Day, Hour, Min, Sec, MSec: Word;
 begin
  Present:= Now;
  DecodeDate(Present, Year, Month, Day);
  Label1.Caption := 'Today is Day ' + IntToStr(Day) + ' of Month '
    + IntToStr(Month) + ' of Year ' + IntToStr(Year);
  DecodeTime(Present, Hour, Min, Sec, MSec);
  Label2.Caption := 'The time is Minute ' + IntToStr(Min) + ' of Hour '
    + IntToStr(Hour);

end;
```

INNER JOIN Operation (SQL)

Delphi command
`Select Statement (SQL)`

Description
SQL Syntax

Example

```
SELECT * FROM PARTS LEFT INNER JOIN INVENTORY
      ON PARTS.PART_NO = INVENTORY.PART_NO
```

INSERT INTO Statement (SQL)

Delphi command
Insert Statement (SQL)

Description
SQL Syntax

Examples
The following statement adds a row to a table, assigning values to two columns:

```
INSERT INTO EMPLOYEE_PROJECT (EMP_NO, PROJ_ID) VALUES (52, "DGPII");

The next statement specifies values to insert into a table with a SELECT
statement:

INSERT INTO PROJECTS
    SELECT * FROM NEW_PROJECTS
          WHERE NEW_PROJECTS.START_DATE > "6-JUN-1994";
```

IPmt Function

Delphi command

`InterestPayment Function`

Declaration

```
function InterestPayment(Rate: Extended; Period, NPeriods: Integer;
PresentValue,
  FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

Description

The InterestPayment function calculates what portion of a loan payment is interest. Rate represents the fixed periodic interest rate, Period identifies the payment period, NPeriods is the number of periods of the loan, PresentValue represents the amount borrowed (the principal), FutureValue is the future value of the investment, and PaymentTime indicates whether the cash flows occur at the beginning or end of the period (by entering a value of 1 or 0, respectively).

IRR Function

Delphi command

`InternalRateOfReturn Function`

Declaration

```
function InternalRateOfReturn(Guess: Extended;
  const CashFlows: array of Double): Extended;
```

Description

The InternalRateOfReturn function determines the internal rate of return on an investment. It references an array that contains cash flow information and uses the supplied internal rate of return estimate to calculate results.

Before you use this function, define an array containing expected cash flow amounts over a period of time. It is assumed that the amounts are received at regular intervals. Negative amounts are interpreted as cash outflows, and positive amounts as inflows. The first amount must be a negative number, to reflect the initial investment. The following amounts can all be the same for each time period, or they can be different (including a mixture of negatives, positives, or zeros).

Icon Property

Delphi command
`Icon Property`

Applies to
TForm component

Declaration
```
property Icon: TIcon
```

Description
The Icon property determines the icon that is displayed when the window or form is minimized. If you don't assign a specific icon to Icon, the form uses the application's icon.

Example
This code assigns an icon to a form when the form is created:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Icon.LoadFromFile('MYICON.ICO');
end;
```

If...Then...Else Statement

Delphi command
`If...Then...Else Statement`

Description
If, then, and else specify the conditions under which a statement will be executed.

If the Boolean expression after if is True, the statement after then is executed.

Otherwise, if the expression evaluates to False and the else part is present, the statement after else is executed. If the else part is not present, execution continues with the next statement following the if statement.

Note No semicolon is allowed preceding an else clause.

Example

```
(* if statements *)
if (I < Min) or (I > Max) then I := 0;
if x < 1.5 then
  z := x + y
else
  z := 1.5;
```

Image Control

Delphi command
`Image Component`

Description
See Additional Page Components

Image Property

Delphi command

`TCanvas.Handle`

Description

The Handle property lets you access the Windows GDI object handle, so you can access the GDI object. If you need to use a Windows API function that requires the handle of a pen object, you could pass the handle from the Handle property of a TPen object.

InStr Function

Delphi command
Pos Function

Declaration
```
function Pos(Substr: string; S: string): Integer;
```

Description
The Pos function searches for a substring in a string.

Substr and S are string-type expressions.

Pos searches for Substr within S and returns an integer value that is the index of the first character of Substr within S.

If Substr is not found, Pos returns zero.

Example
```
var S: string;
begin
  S := '   123.5';
  { Convert spaces to zeroes }
  while Pos(' ', S) > 0 do
    S[Pos(' ', S)] := '0';
end;
```

Index Object, Indexes Collection

Delphi command
`TIndexDef, TIndexDefs Objects`

Description
The TIndexDef object describes the index for a table.

Use the Fields property to get a list of the fields in the index. Use the Name property to get the name of the index. Test the flags in the Options property for a specific characteristic of the index.

In addition to these properties and methods, this object also has the methods that apply to all objects.

Index Property (Data Access)

Delphi command
`IndexName Property`

Applies to
TTable component

Declaration
```
property IndexName: string;
```

Description
The IndexName property identifies a secondary index for the TTable. If no value is assigned to IndexName, the table's primary index will be used to order the records.

Example
```
Table1.IndexName := 'CustNoIndex';
```

InitDir Property

Delphi command
`InitialDir Property`

Applies to
TOpenDialog, TSaveDialog components

Declaration
```
property InitialDir: string;
```

Description
The InitialDir property determines the current directory when the dialog box first appears and value of the InitialDir property is shown as the current directory in the directory tree. Only files in the current directory appear in the dialog box's list box of filenames. After the dialog box appears, users can then use the directory tree to change to another directory if they want.

When specifying the initial directory, include the full path name. For example,

C:\WINDOWS\SYSTEM

If no initial directory is specified, the directory that is current when the dialog box appears remains the current directory. The same is true if you specify a directory that does not exist.

Example
This code specifies C:\WINDOWS as the initial directory when the dialog box appears, displays the dialog box, and displays the name of the file the user selects with the dialog box in a label on the form:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  OpenDialog1.InitialDir := 'C:\WINDOWS';
  if OpenDialog1.Execute then
    Label1.Caption := OpenDialog1.FileName;
end;
```

Initialize Event

Delphi command
`OnCreate Event`

Applies to
TForm component

Declaration
```
property OnCreate: TNotifyEvent;
```

Description
The OnCreate event specifies which event handler to call when the form is first created. You can write code in the event handler that sets initial values for properties and does any processing you want to occur before the user begins interacting with the form.

Delphi creates a form when the application is run by calling the Create method.

Example
This very simple OnCreate event handler assures that the form is the same color as the Windows system color of your application workspace:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Color := clAppWorkSpace;
end;
```

Note The Color property in this example is not prefaced with the name of the form. If you write the statement like this:

Form1.Color := clAppWorkSpace;

The application won't run without error, because Form1 does not yet exist at the time this code is executed

Input # Statement

Delphi command

`Read Procedure`

Declaration

Typed files:

```
procedure Read(F , V1 [, V2,...,Vn ] );
```

Text files:

```
procedure Read( [ var F: Text; ] V1 [, V2,...,Vn ] );
```

Description

The Read procedure can be used in the following ways.

For typed files, it reads a file component into a variable.
For text files, it reads one or more values into one or more variables.

Example

```
uses Dialogs;
 var
  F1, F2: TextFile;
  Ch: Char;
begin
  if OpenDialog1.Execute then begin
    AssignFile(F1, OpenDialog1.Filename);
    Reset(F1);
    if SaveDialog1.Execute then begin
      AssignFile(F2, OpenDialog1.Filename);
      Rewrite(F2);
      while not Eof(F1) do
      begin
        Read(F1, Ch);
        Write(F2, Ch);
      end;
      CloseFile(F2);
    end;
    CloseFile(F1);

  end;
end.
```

Input Function

Delphi command

`Read Procedure`

Declaration

Typed files:

```
procedure Read(F , V1 [, V2,...,Vn ] );
```

Text files:

```
procedure Read( [ var F: Text; ] V1 [, V2,...,Vn ] );
```

Description

The Read procedure can be used in the following ways.

For typed files, it reads a file component into a variable.

For text files, it reads one or more values into one or more variables.

Example

```
uses Dialogs;
 var
  F1, F2: TextFile;
  Ch: Char;
begin
  if OpenDialog1.Execute then begin
    AssignFile(F1, OpenDialog1.Filename);
    Reset(F1);
    if SaveDialog1.Execute then begin
      AssignFile(F2, OpenDialog1.Filename);
      Rewrite(F2);
      while not Eof(F1) do
      begin
        Read(F1, Ch);
        Write(F2, Ch);
      end;
      CloseFile(F2);
    end;
    CloseFile(F1);

  end;
end.
```

InputBox Function

Delphi command
`InputBox Function`

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

Description
The InputBox function displays an input dialog box ready for the user to enter a string in its edit box. The ACaption parameter is the caption of the dialog box, the APrompt parameter is the text that prompts the user to enter input in the edit box, and the ADefault parameter is the string that appears in the edit box when the dialog box first appears.

If the user chooses the Cancel button, the default string is the value returned. If the user chooses the OK button, the string in the edit box is the value returned.

Example
This example 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.

```
uses Dialogs;
procedure TForm1.Button1Click(Sender: TObject);
var
  InputString: string;
begin
  InputString:= InputBox('Input Box', 'Prompt', 'Default string');
end;
```

InsertObjDlg Method

Delphi command
`InsertObjectDialog method`

Applies to
TOleContainer component

Declaration
```
function InsertObjectDialog: Boolean;
```

Description
Executes the Insert Object OLE dialog box, which lets the user create an OLE object. The OLE object can be embedded or linked. InsertObjectDialog returns True if the dialog box was successfully display and the user chose the OK button or False otherwise. If the user chose OK and there's already an OLE object in the container, it's destroyed and any changes the user made to it are discarded.

Int, Fix Functions

Delphi command
Trunc Function

Declaration
```
function Trunc(X: Extended): Longint;
```

Description
The Trunc function truncates a real-type value to an integer-type value.

X is a real-type expression. Trunc returns a Longint value that is the value of X rounded toward zero.

If the truncated value of X is not within the Longint range, an error occurs, which you can handle using the EInvalidOp exception. If you do not handle it, you will receive a run-time error.

Example
```
var
   S, T: string;
begin
   Str(1.4:2:1, T);
   S := T + ' Truncs to ' + IntToStr(Trunc(1.4)) + #13#10;
   Str(1.5:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(1.5)) + #13#10;
   Str(-1.4:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(-1.4)) + #13#10;
   Str(-1.5:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(-1.5));
   MessageDlg(S, mtInformation, [mbOk], 0);
end;
```

IntegralHeight Property

Delphi command
`IntegralHeight Property`

Applies to
TDBListBox, TDirectoryListBox, TFileListBox, TListBox component

Declaration
```
property IntegralHeight: Boolean;
```

Description
The IntegralHeight property controls the way the list box represents itself on the form. If IntegralHeight is True, the list box shows only entries that fit completely in the vertical space, and the bottom of the list box moves up to the bottom of the last completely drawn item in the list. If IntegralHeight is False, the bottom of the list box is at the location determined by its ItemHeight property, and the bottom item visible in the list might not be complete.

If the list box has a Style property value of lbOwerDrawVariable, setting the IntegralHeight property to True has no effect.

If the Style property value of the list box is lsOwnerDrawFixed, the height of the list box at design time is always an increment of the ItemHeight value.

Example
This example uses a list box on a form. To try it, enter as many strings in the Items property as you like using the Object Inspector. When the application runs, the list box displays only entries that fit completely in the vertical space, and the bottom of the list box moves up to the bottom of the last string in the list box if the form is less than 300 pixels in height:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  if Height < 300 then
    ListBox1.IntegralHeight := True
  else
    ListBox1.IntegralHeight := False;
end;
```

Interval Property

Delphi command
`Interval Property`

Applies to
TTimer component

Declaration
```
property Interval: Cardinal;
```

Description
The Interval property determines in milliseconds the amount of time that passes before the timer component initiates another OnTimer event.

You can specify any value in the cardinal range as the interval value, but the timer component won't call an OnTimer event if the value is 0. The default value is 1000 (one second).

Example
The code in this OnTimer event handler moves a ball, the shape component (TShape) slowly across a form.

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  Shape1.Shape := stCircle;
  Shape1.Left := Shape1.Left + 1;
end;
```

IpOleObject Property

Delphi command
`OleObject Property`

Applies to
TAutoObject

Declaration
```
property OleObject: Variant;
```

Description
For TAutoObject, the OleObject property provides a variant containing the automation object. When your server needs to pass an OLE object to a controller, the controller expects it in the form of a variant. You should therefore always references to an automation object's OleObject property, rather than references to the object itself.

For TOleContainer (runtime and readonly) the OleObject property returns an OLE Automation object for the OLE object. If the OLE object doesn't support OLE Automation, an exception is raised. An OLE object must already be loaded in the container before accessing the OleObject property.

IsArray Function

Delphi command
`VarIsArray Function`

Declaration
```
function VarIsArray(const V: Variant): Boolean;
```

Description
The VarIsArray function returns True if the given variant is an array. Otherwise, the function result is False.

IsEmpty Function

Delphi command
`VarIsEmpty Function`

Declaration
```
function VarIsEmpty(const V: Variant): Boolean;
```

Description
The VarIsEmpty function returns True if the given variant contains the value Unassigned. If the variant contains any other value, the function result is False.

IsNull Function

Delphi command
`VarIsNull Function`

Declaration
```
function VarIsNull(const V: Variant): Boolean;
```

Description
The VarIsNull function returns True if the given variant contains the value Null. If the variant contains any other value, the function result is False.

Italic Property

Delphi command
`TFont.Style Property`

Applies to
TFont objecs

Declaration
```
property Style: TFontStyles;
```

Description
The Style property determines whether the font is normal, italic, underlined, bold, and so on. The font is italicized if style is set to fsItalic.

Example
The following code boldfaces the font used in the memo..

```
Memo1.Font.Style := [fsItalic];
```

Item Method

Delphi command
`Items Property`

Declaration
```
property Items[Index:Integer]:TCollectionItem;
```

Description
Run-time only. The Items property provides access to a collection item (TCollectionItem) by its position in the collection. The value of the Index parameter corresponds to the Index property and represents the position of the collection item in the Items array.

ItemData Property

Delphi command
`Use TStrings.Value or TStrings.Objects`

Applies to
TStrings, TStringList objects

Declaration
```
property Values[const Name: string]: string;
```

Description
The Values property gives you access to a specific string in a list of strings. The strings must have a unique structure before you can use the Values property array to access them:

Name=Value

The Name that identifies the string is to the left of the equal sign (=), and the current Value of the Name identifier is on the right side of the equal sign. There should be no spaces present before and after the equal sign.

Example
The following code allows the user to specify a bitmap file with the OpenDialog1 open dialog box component when Form1 is created. Then, the bitmap file specified is added to the Items list of ListBox1.

If ListBox1 is an owner-draw control (specified by a Style property of lbOwnerDrawFixed or lbOwnerDrawVariable), the second procedure is the OnDrawItem event handler for ListBox1. The bitmap in the Object property and the text of an item are retrieved and displayed in Listbox1.

```
procedure TForm1.FormCreate(Sender: TObject);
var
  TheBitmap: TBitmap;
begin
  if OpenDialog1.Execute then
  begin
    TheBitmap := TBitmap.Create;
    TheBitmap.LoadFromFile(OpenDialog1.FileName);
    ListBox1.Items.AddObject(OpenDialog1.FileName, TheBitmap);
  end;
end;
procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer;
  Rect: TRect; State: TOwnerDrawState);
var

  DrawBitmap: TBitmap;
begin
  DrawBitmap := TBitmap(ListBox1.Items.Objects[Index]);
  with ListBox1.Canvas do
  begin
    Draw(Rect.Left, Rect.Top + 4, DrawBitmap);
    TextOut(Rect.Left + 2 + DrawBitmap.Width, Rect.Top + 2,
ListBox1.Items[Index]);
  end;
end;
```

Example 2
Assume that a string that identifies the password needed to access a database exists in the Params string list. You can change the acceptable password using this code:

```
Database1.Params.Values['Password'] := 'TopSecret';
```

If there is no password string, the same code creates one at the bottom of the list of strings and assigns the 'TopSecret' string as its value.

You can also assign the value of the string to a variable. For example, this code assigns the current value of the password string to a variable called StringValue:

```
var
  StringValue: string;
StringValue := Database1.Params.Values['Password'];
```

KeyDown, KeyUp Events

Delphi command

`OnKeyDown Event`

Declaration

```
TKeyEvent = procedure (Sender: TObject; var Key: Word; Shift: TShiftState) of
object;
property OnKeyDown: TKeyEvent;
```

Description

The OnKeyDown event occurs when a user presses any key while the control has focus. Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys, and pressed mouse buttons.

Example

This event handler displays a message dialog when the user presses Alt+F10:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if ((Shift = [ssAlt]) and (Key = VK_F10)) then
    MessageDlg('Alt+F10 pressed down', mtInformation, [mbOK], 0);
end;
```

KeyPress Event

Delphi command

OnKeyPress Event

Declaration

```
TKeyPressEvent = procedure (Sender: TObject; var Key: Char) of object;
property OnKeyPress: TKeyPressEvent;
```

Description

The OnKeyPress event occurs when a user presses a single character key. Use the OnKeyPress event handler when you want something to happen as a result of a single character key press.

Example

This event handler displays a message dialog box specifying which key was pressed:

```
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

KeyPreview Property

Delphi command
KeyPreview Property

Applies to
TForm component

Declaration
```
property KeyPreview: Boolean;
```

Description
When the KeyPreview property is True, most key events (OnKeyDown event, OnKeyUp event, and OnKeyPress event) go to the form first, regardless of which control is selected on the form. This allows your application to determine how to process key events. After going to the form, key events are then passed to the control selected on the form. When KeyPreview is False, the key events go directly to the controls. The default value is False.

The exceptions are the navigation keys, such as Tab, BackTab, the arrow keys, and so on. If the selected control processes such keys, you can use KeyPreview to intercept them; otherwise, you can't.

If KeyPreview is False, all key events go to the selected control.

Example
This example changes a form's color to aqua when the user presses a key, even when a control on the form has the focus. When the user releases the key, the form returns to its original color.

```
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
end;
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  FormColor := Form1.Color;
  Form1.Color := clAqua;
end;
procedure TForm1.FormKeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  Form1.Color := FormColor;
end;
```

Kill Statement

Delphi command
`DeleteFile Function`

Declaration
```
function DeleteFile(const FileName: string): Boolean;
```

Description
The DeleteFile function erases the file named by FileName from the disk.

If the file cannot be deleted or does not exist, the function returns False but does not raise an exception.

Example
The following code erases the file DELETE.ME in the current directory:

```
DeleteFile('DELETE.ME');
```

KillDoc Method

Delphi command
`Abort Method`

Applies to
TPrinter object

Declaration
```
procedure Abort;
```

Description
The Abort procedure terminates the printing of a print job, dropping all unprinted data. The device is then set for the next print job. Use Abort to terminate the print job before it completes; otherwise, use the EndDoc method.

To use the Abort method, you must add the Printers unit to the uses clause of your unit.

Example
The following code aborts a print job if the user presses Esc. Note that you should set KeyPreview to True to ensure that the OnKeyDown event handler of Form1 is called.

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift:
TShiftState);
begin
  if (Key=VK_ESCAPE) and Printer.Printing then
  begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

LBound Function

Delphi command
Low

Declaration
```
function Low(X);
```

Description
The Low function returns the lowest value in the range of the argument.

Result type is X, or the index type of X where X is either a type identifier or a variable reference.

Type	Low returns
Ordinal type	The lowest value in the range of the type
Array type	The lowest value within the range of the index type of the array
String type	Returns 0
Open array	Returns 0
String parameter	Returns 0

Example
```
function Sum( var X: array of Double): Double;
var
  I: Word;
  S: Real;
begin
  S := 0;               { Note that open array index range is always zero-
based. }
  for I := 0 to High(X) do S := S + X[I];
  Sum := S;
end;
procedure TForm1.Button1Click(Sender: TObject);
var
  List1: array[0..3] of Double;
  List2: array[5..17] of Double;
  X: Word;

  S, TempStr: string;
begin
  for X := Low(List1) to High(List1) do
      List1[X] := X * 3.4;
  for X := Low(List2) to High(List2) do
      List2[X] := X * 0.0123;
  Str(Sum(List1):4:2, S);
  S := 'Sum of List1: ' + S + #13#10;
  S := S + 'Sum of List2: ';
  Str(Sum(List2):4:2, TempStr);
  S := S + TempStr;
  MessageDlg(S, mtInformation, [mbOk], 0);
end;
```

LCase Function

Delphi command

`LowerCase, StrLower, AnsiiLowerCase`

Declaration

```
function LowerCase(const S: string): string;
```

Description

The LowerCase function returns a string with the same text as the string passed in S, but with all letters converted to lowercase. The conversion affects only 7-bit ASCII characters between 'A' and 'Z'. To convert 8-bit international characters, use AnsiLowerCase.

Example

This example uses two edit boxes and a button on a form. When the user clicks the button, the text in the Edit1 edit box displays in the Edit2 edit box in lowercase letters.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit2.Text := LowerCase(Edit1.Text);
end;
```

LEFT JOIN, RIGHT JOIN Operations

Delphi command

LEFT JOIN, RIGHT JOIN Operations (SQL)

Description

SQL Syntax

Example

```
Select CustName, OrderNo from Customer
Left Outer Join Orders On Customer.CustNo = Orders.CustNo
```

LOF Function

Delphi command
`FileSize function`

Declaration
```
function FileSize(var F): Integer;
```

Description
The FileSize function returns the size in bytes of file F. However, if F is a record file FileSize will return the number of records in the file.

To use FileSize the file must be open and it can't be used on a text file.

F is a file variable.

If the file is empty, FileSize(F) returns 0.

Example
```
var
   f: file of Byte;
   size : Longint;
   S: string;
   y: integer;
 begin
   if OpenDialog1.Execute then begin
     AssignFile(f, OpenDialog1.FileName);
     Reset(f);
     size := FileSize(f);
     S := 'File size in bytes: ' + IntToStr(size);
     y := 10;
     Canvas.TextOut(5, y, S);
     y := y + Canvas.TextHeight(S) + 5;
     S := 'Seeking halfway into file...';
     Canvas.TextOut(5, y, S);
     y := y + Canvas.TextHeight(S) + 5;

     Seek(f,size div 2);
     S := 'Position is now ' + IntToStr(FilePos(f));
     Canvas.TextOut(5, y, S);
     CloseFile(f);
   end;
 end;
```

LTrim, RTrim, and Trim Functions

Delphi command

`TrimLeft, TrimRight, and Trim functions`

Declaration

```
function TrimLeft(const S: string): string;
```

Description

The TrimLeft function trims leading spaces and control characters from the given string S. The Trim function trims leading and trailing spaces and control characters from the given string S. The TrimRight trims trailing spaces and control characters from the given string S.

Label Control

Delphi command
`TLabel Component`

Description
See Standard Page Components

LargeChange, SmallChange Properties

Delphi command
`LargeChange, SmallChange Properties`

Applies to
TScrollBar component

Declaration
```
property LargeChange: TScrollBarInc;
property SmallChange: TScrollBarInc;
```

Description
The LargeChange property determines how far the scroll box moves when the user clicks the scroll bar on either side of the scroll box or presses PgUp or PgDn. The default value is 1 position.

For example, if the LargeChange property setting is 1000, each time the user clicks the scroll bar, the scroll box moves 1000 positions. How big the change from one position to another depends on the difference between the Max property value and the Min property value. If Max is 3000 and Min is 0, the user needs to click the scroll bar three times to move the scroll box from one end of the scroll bar to the other.

Example
This code determines that when the user clicks the scroll bar on either side of the scroll box, the scroll box moves 100 positions on the scroll bar:

```
ScrollBar1.LargeChange := 100;
ScrollBar1.SmallChange := 10;
```

LastFunction (SQL)

Delphi command
```
Last Method
```

Declaration
```
procedure Last;
```

Description
The Last method moves the cursor to the last record in the active range of records of the dataset. The active range of records is affected by the filter established with SetRangeEnd.

Example
```
Table1.Last;
```

Left Function

Delphi command
`Copy, StrLCopy`

Declaration
```
function Copy(S: string; Index, Count: Integer): string;
```

Description
The Copy function returns a substring of a string.

S is a string-type expression. Index and Count are integer-type expressions. Copy returns a string containing Count characters starting with at S[Index].

If Index is larger than the length of S, Copy returns an empty string.

If Count specifies more characters than are available, the only the characters from S[Index] to the end of S are returned.

Example
```
 var S: string;
begin
  S := 'ABCDEF';
  S := Copy(S, 2, 3);   { 'BCD' }
end;
```

Left, Top Properties

Delphi command

```
Left, Top Properties
```

Declaration

```
property Left: Integer;
property Top: Integer;
```

Description

The Left property determines the horizontal coordinate of the left edge of a component relative to the form in pixels. If the control is contained in a TPanel, the Left and Top properties will be relative to the panel. If the control is contained directly by the form, it will be relative to the form. For forms, the value of the Left property is relative to the screen in pixels.

Example

The following code moves a button 10 pixels up and to the left each time a user clicks it:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Top := Button1.Top - 10;
  Button1.Left := Button1.Left - 10;
end;
```

LeftCol Property

Delphi command
`LeftCol Property`

Applies to
TDrawGrid, TStringGrid components

Declaration
```
property LeftCol: Longint;
```

Description
Run-time only. The LeftCol property determines which column in the grid appears at the far left side of the grid.

Example
This line of code positions the fourth column of a string grid at the left edge of the grid:

```
StringGrid1.LeftCol := 3;
```

LegalCopyRight Property

Delphi command
`Version Info (API)`

Description
See Win32.hlp

LegalTrademarks Property

Delphi command
`Version Info (API)`

Description
See Win32.hlp

Len Function

Delphi command

Length, StrLen functions

Declaration

```
function StrLen(Str: PChar): Cardinal;
function Length(S: string): Integer;
```

Description

The Length function returns the number of characters actually used in the string S. The StrLen function returns the number of characters in Str, not counting the null terminator.

Example

```
var
  S: string;
begin
  S := 'The Black Knight';
  Canvas.TextOut(10, 10, 'String Length = ' + IntToStr(Length(S)));
end;
```

Like Operator (SQL)

Delphi command
`Like Operator (SQL)`

Description
SQL Syntax

Example
```
Select * from Customer
where LastName Like 'Sm%'
```

Line Control

Delphi command
`TShape Component`

Description
See Additional Page Components

Line Input # Statement

Delphi command

Readln procedure

Declaration

```
procedure Readln([ var F: Text; ] V1 [, V2, ...,Vn ]);
```

Description

The Readln procedure reads a line of text and then skips to the next line of the file.

Readln(F) with no parameters causes the current file position to advance to the beginning of the next line if there is one; otherwise, it goes to the end of the file.

Example

```
program Project1;

{$AppType ConSole}

uses windows;

var
   s : string;
begin
   Write('Enter a line of text: ');
   Readln(s);
   Writeln('You typed: ',s);
   Writeln('Hit <Enter> to exit');
   Readln;
end.
```

Line Method

Delphi command

`LineTo, Rectangle, PolyLine methods (TCanvas)`

Declaration

```
procedure LineTo(X, Y: Integer);
```

Description

The LineTo method draws a line on the canvas from the current drawing position (specified by the PenPos property) to the point specified by X and Y and sets the pen position to (X, Y). See Tcanvas Object in Delphi.hlp for more information.

Example

The following code draws a line from the upper left corner of a form to the point clicked with the mouse.

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  Canvas.MoveTo(0, 0);
  Canvas.LineTo(X, Y);
end;
```

LinkClose Event

Delphi command
`OnClose, OnClose events (TDdeClientConv, TDDeServerConv)`

Applies to
TDDEClientConv component

Declaration
```
property OnClose: TNotifyEvent;
```

Description
An OnClose event occurs when a DDE conversation is terminated. A conversation is terminated when one of the applications involved is closed, or when the CloseLink method is called.

Example
The following code displays a message when a conversation is closed.

```
procedure TForm1.DdeClientConv1Close(Sender: TObject);
begin
  MessageDlg('This conversation is finished!', mtInformation, [mbOK],0);
end;
```

LinkError Event

Delphi command
`EDdeError`

Declaration
```
EDDEError = class(Exception);
```

Description
The EDDEError exception is raised when your application can't find the specified server or conversation, or when a session is unexpectedly terminated.

LinkExecute Event

Delphi command

`OnExecuteMacro event (TDdeserver)`

Declaration

```
TMacroEvent = procedure(Sender: TObject; Msg : string) of object;
property OnExecuteMacro : TMacroEvent;
```

Description

The OnExecuteMacro event occurs when a DDE client application sends a macro to a DDE server conversation component. Write code to process the macro in the OnExecuteMacro event handler. See the DDE client application documentation for information about how it sends macros. If the DDE client is a Delphi application, a macro is sent with the ExecuteMacro method of the TDDEClientConv component.

The TMacroEvent type points to a method that handles the passing of a macro string from a DDE client to a DDE server conversation (TDDEServerConv) component. Msg contains the macro.

Example

The following code clears the contents of a memo in the server application if the appropriate message is sent from the client application.

```
procedure TForm1.DdeServerConv1ExecuteMacro(Sender: TObject; Msg: TStrings);
begin
  if Msg.Strings[0] = 'Edit|Clear' then
    Memo1.Clear;
end;
```

LinkExecute Method

Delphi command

`ExecuteMacro method (TDdeClientConv)`

Declaration

```
TMacroEvent = procedure(Sender: TObject; Msg : string) of object;
property OnExecuteMacro : TMacroEvent;
```

Description

The OnExecuteMacro event occurs when a DDE client application sends a macro to a DDE server conversation component. Write code to process the macro in the OnExecuteMacro event handler. See the DDE client application documentation for information about how it sends macros. If the DDE client is a Delphi application, a macro is sent with the ExecuteMacro method of the TDDEClientConv component.

The TMacroEvent type points to a method that handles the passing of a macro string from a DDE client to a DDE server conversation (

TDDEServerConv) component. Msg contains the macro.

Example

The following code executes the macro that is specified by the Text of Edit1. The macro sets WaitFlg to True to wait until the server has completed macro execution.

```
var
  TheMacro: PChar;
begin
  StrPCopy(TheMacro, Edit1.Text);
  DDEClientConv1.ExecuteMacro(TheMacro, True);
end;
```

LinkItem Property

Delphi command
`DdeItem property (TDdeClientItem)`

Applies to
TDDEClientItem component

Declaration
```
property DDEItem: string;
```

Description
The DDEItem property specifies the item of a DDE conversation. The value of DDEItem depends on the linked DDE server application. DDEItem is typically a selectable portion of text, such as a spreadsheet cell or a database field in an edit box. If the DDE server is a Delphi application, DDEItem is the name of the linked DDE server component. For example, to link to a DDE server component named DDEServer1, set DDEItem to 'DDEServer1'.

See the documentation for the DDE server application for the specific information about specifying DDEItem.

At design time, you can specify DDEItem either by typing the item string in the object inspector or by pasting a link using the DDE Info dialog box, which appears if you click the ellipsis (...) button for DDEService or DDETopic in the Object Inspector. After you choose Paste Link in the DDE Info dialog box, you can choose the item from a list of possible items for DDEItem in the object inspector if link information is still on the Clipboard.

Example
The following code specifies a DDE item of 'DDEServer1'.

```
DDEClientItem1.DDEItem := 'DDEServer1';
```

LinkMode Property

Delphi command
`ConnectMode property (TDdeClientItem)`

Applies to
TDDEClientConv component

Declaration
```
property ConnectMode: TDataMode;
```

Description
The ConnectMode property determines the type of connection to establish when initiating a link with a DDE server application. These are the possible values:

Value	Meaning
ddeAutomatic	The link is automatically established when the form containing the TDDEClient component is created at run time. This is the default value.
ddeManual	The link is established only when the OpenLink method is called.

Example
The following code sets the connect mode of DDEClientConv1 to manual.

```
DDEClientConv1.ConnectMode := ddeManual;
```

LinkOpen Event

Delphi command
`OnOpen event  (TDdeClientConv, TDdeServerConv)`

Applies to
TDDEClientConv, TDDEServerConv components

Declaration
```
property OnOpen: TNotifyEvent;
```

Description
An OnOpen event occurs when a DDE conversation is opened. A DDE conversation can be initiated automatically or manually. Automatically open a conversation by setting the value of the ConnectMode property to ddeAutomatic. When the form containing the DDE client conversation component is created at run time, the DDE conversation opens. Manually open a conversation by setting the value of ConnectMode to ddeManual and calling the OpenLink method.

Example
The following code sends a macro to the server and closes the link immediately after opening it.

```
procedure TForm1.DdeClientConv1Open(Sender: TObject);
begin
  with DDEClientConv1 do
  begin
    ExecuteMacro('File|New', False);
    CloseLink;
  end;
end;
```

LinkPoke Method

Delphi command
`PokeData, PokeDataLines methods (TDdeClientConv)`

Applies to
TDDEClientConv component

Declaration
```
function PokeData(Item: string; Data: PChar): Boolean;
function PokeDataLines(Item: string; Data: TStrings): Boolean;
```

Description
The PokeData method sends data to a DDE server application. Text data from a linked control in the DDE client application is transferred to the linked section of the DDE server application. Item specifies the linked item in the DDE server. Data is a null-terminated string that specifies the text data to transfer to the DDE server.

The PokeDataLines method sends data to a DDE server application. Text data from a linked control in the DDE client application is transferred to the linked section of the DDE server application. Item specifies the linked item in the DDE server. Data is a TStrings object that specifies the text data to transfer to the DDE server.

Example
The following code pokes the data that is in Edit1 to the DDE server. The DDE item of the conversation is specified in the DDEItem property of DDEClientItem1. TheData is a PChar variable.

```
DDEClientConv1.PokeData(DDEClientItem1.DDEItem, StrPCopy(TheData,
Edit1.Text));
```

LinkRequest Method

Delphi command
`RequestData method (TDdeClientConv)`

Applies to
TDDEClientConv component

Declaration
function RequestData(const Item: string): PChar;

Description
The RequestData method requests data from a DDE server. Call RequestData when you want your DDE client application to receive data from the server once, instead of being updated continually. Another reason to use RequestData is that some DDE servers contain DDE items that can't be continually updated; the only way for your client to access these items is to explicitly request the data.

Example
The following code requests data from the DDE server and displays it in Label1. The DDE item of the conversation is specified in the DDEItem property of DDEClientItem1.

```
var
  TheData: PChar;
begin
  TheData := DDEClientConv1.RequestData(DDEClientItem1.DDEItem);
  Label1.Caption := StrPas(TheData);
end;
```

LinkTopic Property

Delphi command
`DdeTopic property (TDdeClientConv)`

Applies to
TDDEClientConv component

Declaration
```
property DDETopic: string;
```

Description
The DDETopic property specifies the topic of a DDE conversation. Typically, DDETopic is a filename (and path, if necessary) used by the application specified in DDEService. If the DDE server is an Delphi application, by default DDETopic is the caption of the form containing the linked component. For example, to link to a component on a form named Form1, set DDETopic to 'Form1'. However, if the DDE client is linked to a TDDEServerConv component, DDETopic is the name of the server conversation component instead of the form caption. For example, to link to DDEServerConv1, set DDETopic to 'DDEServerConv1'.

See the documentation for the DDE server application for the specific information about specifying DDETopic.

Example
The following code specifies a DDE topic of 'Form1'.

```
DDEClientConv1.DDETopic := 'Form1';
```

List Property

Delphi command
`Items property (TListBox)`

Applies to
TComboBox, TDBRadioGroup, TDirectoryListBox, TDriveComboBox, TFileListBox, TFilterComboBox, TListBox, TRadioGroup components

Declaration
```
property Items: TStrings;
```

Description
The Items property contains the strings that appear in the list box or combo box, or as radio buttons in a radio group box. Because Items is an object of type TStrings, you can add, delete, insert, and move items using the Add, Delete, Insert, Exchange, and Move methods of the TStrings object.

The ItemIndex property determines which item is selected, if any.

To determine if a particular item in the list of strings that makes up the Items property for a list box or combo box is selected, use the Selected property.

Example
This example uses an edit box, a list box, and a button on a form. When the user clicks the button, the text in the edit box is added to the list box:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items.Add(Edit1.Text);
end;
```

ListBox Control

Delphi command
`TListBox Component`

Description
See Standard Page Components

ListCount Property

Delphi command
`Count property (TListbox.Items)`

Declaration
```
property Count: Integer;
```

Description
Run-time and read only. The Count property contains the number of items in a collection, list, menu item, tree nodes object, tree node, header section, list column or a status bar.

For list objects, Count is the number of items in the list.

Example
The following code displays the number of items in a list box in the caption of a label when the user clicks the CountItems button:

```
procedure TForm1.CountItemsClick(Sender: TObject);
begin
  Label1.Caption := 'There are ' + IntToStr(ListBox1.Items.Count) +
    ' items in the listbox.';
end;
```

The following example assumes the form contains a main menu component, which includes a File menu and a label. This code displays the number of menu items that make up the File menu.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := IntToStr(FileMenu.Count):
end;
```

ListField Property

Delphi command
`DataField property (data aware controls)`

Declaration
```
property DataField: string;
```

Description
The DataField property identifies the field from which the data-aware control displays data. The dataset the field is located in is specified in a data source component (TDataSource). The DataSource property of the data-aware control specifies which data source component.

If the DataField value of a database edit box (TDBEdit) is an integer or floating-point value, only characters that are valid in such a field can be entered in the edit box. Characters that are not legal are not accepted.

Example
The following code specifies that the DataField of DBEdit1 is 'FNAME'.

```
DBEdit1.DataField := 'FNAME';
```

ListFields Method

Delphi command
`Use Fields[], FieldCount properties (TDataset)`

Declaration
```
property FieldCount: Integer;
```

Description
Run-time and read-only. The FieldCount property specifies the number of fields (columns) in a dataset. It may not be the same as the number of fields in the underlying data set, because you can add calculated fields and remove fields with the fields editor.

For the data grid and database lookup list box, the value of the FieldCount property is the number of fields in the dataset displayed in the control.

ListIndex Property

Delphi command
`ItemIndex property`

Applies to
`TComboBox, TDBComboBox, TDBRadioGroup, TDirectoryListBox, TDriveComboBox, TFileListBox, TFilterComboBox, TListBox, TRadioGroup components`

Declaration
```
property ItemIndex: Integer;
```

Description
Run-time only except for the TRadioGroup component. The value of the ItemIndex property is the ordinal number of the selected item in the control's item list. If no item is selected, the value is -1, which is the default value unless MultiSelect is True. To select an item at run time, set the value of ItemIndex to the index of the item in the list you want selected, with 0 being the first item in the list.

For list boxes and combo boxes, if the value of the MultiSelect property is True and the user selects more than one item in the list box or combo box, the ItemIndex value is the index of the selected item that has focus. If MultiSelect is True, ItemIndex defaults to 0.

Example
This example uses a drive combo box on a form. When the user selects a drive in the combo box, the index value of the selected item appears in the caption of the label:

```
procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  Label1.Caption := 'Index value ' + IntToStr(DriveComboBox1.ItemIndex);
end;
```

ListIndexes Method

Delphi command
Use IndexDefs, IndexFields, IndexFieldCount (TTable)

Applies to
TTable component

Declaration
```
property IndexDefs: TIndexDefs;
property IndexFields[Index: Integer]: TField;
property IndexFieldCount: Integer;
```

Description
Run-time and read-only. The IndexDefs property holds information about all the indexes for the TTable.

Run-time only. The IndexFields property gives you access to information about each field of the current index for the dataset. The Active property must be True or the information will not be valid.

Run-time only. The IndexFieldCount property is the number of actual fields for the current index. If you are using the primary index for the component, this value will be one. If the component is not Active, the value of IndexFieldCount will be zero.

Example
```
{ Get the current available indicies }
Table1.IndexDefs.Update;
{ Find one which combines Customer Number ('CustNo') and Order Number
('OrderNo') }
for I := 0 to Table1.IndexDefs.Count - 1 do
  if Table1.IndexDefs.Items[I].Fields = 'CustNo;OrderNo' then
```

ListParameters Method

Delphi command
`Use Params property (TQuery )`

Applies to
TQuery component

Declaration
```
property Params[Index: Word]: TParam;
```

Description
When you enter a query, Delphi creates a Params array for the parameters of a dynamic SQL statement. Params is a zero-based array of TParam objects with an element for each parameter in the query; that is, the first parameter is Params[0], the second Params[1], and so on. The number of parameters is specified by ParamCount. Read-only and run-time only.

Note Use the ParamByName method instead of Params to avoid dependencies on the order of the parameters.

Example
For example, suppose a TQuery component named Query2 has the following statement for its SQL property:

```
INSERT
  INTO COUNTRY (NAME, CAPITAL, POPULATION)
  VALUES (:Name, :Capital, :Population)
```

An application could use Params to specify the values of the parameters as follows:

```
Query2.Params[0].AsString := 'Lichtenstein';
Query2.Params[1].AsString := 'Vaduz';
Query2.Params[2].AsInteger := 420000;
```

These statements would bind the value "Lichtenstein" to the :Name parameter, "Vaduz" to the :Capital parameter, and 420000 to the :Population parameter.

ListTables Method

Delphi command
`Use Datasets[], DatasetCount (TDatabase)`

Applies to
TDataBase component

Declaration
```
property DatasetCount: Integer;
```

Description
DatasetCount is the number of dataset components (TTable, TQuery, and TStoredProc) that are currently using the TDatabase component. Read-only and run-time only.

Load Event

Delphi command
`OnCreate event (TForm)`

Applies to
TForm component

Declaration
```
property OnCreate: TNotifyEvent;
```

Description
The OnCreate event specifies which event handler to call when the form is first created. You can write code in the event handler that sets initial values for properties and does any processing you want to occur before the user begins interacting with the form.

Delphi creates a form when the application is run by calling the Create method.

Example
This very simple OnCreate event handler assures that the form is the same color as the Windows system color of your application workspace:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Color := clAppWorkSpace;
end;
```

Note The Color property in this example is not prefaced with the name of the form. If you write the statement like this,

```
Form1.Color := clAppWorkSpace;
```

The application won't run without error, because Form1 does not yet exist at the time this code is executed

Load Statement

Delphi command
`Create method (TForm)`

Applies to
TForm component

Declaration
```
property OnCreate: TNotifyEvent;
```

Description
The OnCreate event specifies which event handler to call when the form is first created. You can write code in the event handler that sets initial values for properties and does any processing you want to occur before the user begins interacting with the form.

Delphi creates a form when the application is run by calling the Create method.

Example
This example will create an instance (Form2) of the type TForm2.

```
Form2 := TForm2.create( Application );
```

LoadPicture Function

Delphi command
`LoadFromFile method`

Applies to
TBlobField, TGraphicField, TMemoField components

Declaration
```
procedure LoadFromFile(const FileName: string);
```

Description
The LoadFromFile method reads a file with the name passed in FileName and loads the contents in TBlobField, TMemoField, or TGraphicField.

Note For TMemoField and TGraphicField, the file should have been created by the SaveToFile or SaveToStream method.

Example
This example loads a bitmap into an image component.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Picture.Bitmap.LoadFromFile( 'TARTAN.BMP' );
end;
```

LoadResData Function

Delphi command
`LoadResource (API)`

Description
See Win32.hlp

LoadResPicture Function

Delphi command
`LoadResource (API)`

Description
See Win32.hlp

LoadResString Function

Delphi command
`LoadString Function`

Description
See Win32.hlp

Loc Function

Delphi command
FilePos Function

Declaration
```
function FilePos(var F): Longint;
```

Description
The FilePos function returns the current file position within a file.

To use FilePos the file must be open and it can't be used on a text file.

F is a file variable.

Position	Result
Beginning of file	FilePos(F) = 0
Middle of file	FilePos(F) = current file position
End of file	Eof(F) = True

Example
```
var
   f: file of Byte;
   size : Longint;
   S: string;
   y: integer;
 begin
   if OpenDialog1.Execute then begin
     AssignFile(f, OpenDialog1.FileName);
     Reset(f);
     size := FileSize(f);
     S := 'File size in bytes: ' + IntToStr(size);
     y := 10;
     Canvas.TextOut(5, y, S);
     y := y + Canvas.TextHeight(S) + 5;
     S := 'Seeking halfway into file...';
     Canvas.TextOut(5, y, S);
     y := y + Canvas.TextHeight(S) + 5;

     Seek(f,size div 2);
     S := 'Position is now ' + IntToStr(FilePos(f));
     Canvas.TextOut(5, y, S);
     CloseFile(f);
   end;
 end;
```

Locked Property

Delphi command
`ReadOnly property`

Applies to
TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupComboBox, TDBLookupList, TDBLookupListBox, TDBMemo, TDBRadioGroup, TEdit, TListView, TMaskEdit, TMemo, TRichEdit, TTable, TTreeViewcomponents

Declaration
```
property ReadOnly: Boolean;
```

Description
The ReadOnly property determines if the user can change the contents of the control. If ReadOnly is True, the user can't change the contents. If ReadOnly is False, the user can modify the contents. The default value is False.

For data-aware controls, the ReadOnly property determines whether the user can use the data-aware control to change the value of the field of the current record, or if the user can use the control only to display data. If ReadOnly is False, the user can change the field's value as long as the dataset is in edit mode.

When the ReadOnly property of a data grid is True, the user can no longer use the Insert key to insert a new row in the grid, nor can the user append a new row at the end of the data grid with the Down Arrow key.

Example
This code toggles the read-only state of an edit box each time the user double-clicks the form:

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  Edit1.Left := 2;
  Edit1.Top := 2;
  Edit1.ReadOnly := True;
  Edit1.Text := 'Change Me';
  Canvas.TextOut(10, 40, 'Double-click form to toggle read-only state');
end;
procedure TForm1.FormDblClick(Sender: TObject);
begin
  Edit1.ReadOnly := not Edit1.ReadOnly;
end;
```

Log Function

Delphi command
Ln function

Declaration
```
function Ln(X: Real): Real;
```

Description
The Ln function returns the natural logarithm (Ln(e) = 1) of the real-type expression X.

Example
```
var
   e : real;
   S : string;
begin
   e := Exp(1.0);
   Str(ln(e):3:2, S);
   S := 'ln(e) = ' + S;
   Canvas.TextOut(10, 10, S);
end;
```

LogMessages Property

Delphi command
`Trace Mode setting (BDE)  Also TDatabase, Tession TraceFlags`

Description
See Delphi.hlp

LoginTimeout Property

Delphi command

Time Out setting (BDE)

LostFocus Event

Delphi command
`OnExit event`

Applies to
All windowed controls

Declaration
```
property OnExit: TNotifyEvent;
```

Description
The OnExit event occurs when the input focus shifts away from one control to another. Use the OnExit event handler when you want special processing to occur when this control ceases to be active.

Example
This example uses an edit box and a memo control on a form. When either the edit box or the memo is the active control, it is colored yellow. When the active control becomes inactive, the color of the control returns to the Windows system color for a window.

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;
procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;
procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;
procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

MDIChild Property

Delphi command
`FormStyle property of fsMDIChild (TForm )`

Declaration
```
property FormStyle: TFormStyle;
```

Description
The FormStyle property determines the style of the form.

Example
This example ensures the main form of the application is an MDI parent form:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  if FormStyle <> fsMDIForm then
    FormStyle := fsMDIForm;
  if FormStyle = fsMDIForm then
    Edit1.Text := 'MDI form'
  else
    Edit1.Text := 'Not an MDI form';   {This line never runs}
end;
```

MDIForm Object

Delphi command
`FormStyle property set to fsMDIForm (TForm )`

Declaration
```
property FormStyle: TFormStyle;
```

Description
The FormStyle property determines the style of the form.

Example
This example ensures the main form of the application is an MDI parent form:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  if FormStyle <> fsMDIForm then
    FormStyle := fsMDIForm;
  if FormStyle = fsMDIForm then
    Edit1.Text := 'MDI form'
  else
    Edit1.Text := 'Not an MDI form';   {This line never runs}
end;
```

MIRR Function

Delphi command

```
InternalRateOfReturn (Math.Pas unit)
```

Declaration

```
function InternalRateOfReturn(Guess: Extended;
  const CashFlows: array of Double): Extended;
```

Description

The InternalRateOfReturn function determines the internal rate of return on an investment. It references an array that contains cash flow information and uses the supplied internal rate of return estimate to calculate results.

Before you use this function, define an array containing expected cash flow amounts over a period of time. It is assumed that the amounts are received at regular intervals. Negative amounts are interpreted as cash outflows, and positive amounts as inflows. The first amount must be a negative number, to reflect the initial investment. The following amounts can all be the same for each time period, or they can be different (including a mixture of negatives, positives, or zeros).

Max, Min Properties (CommonDialog)

Delphi command
`MaxFontSize, MinFontSize properties (TFontDialog )`

Applies to
TFontDialog component

Declaration
```
property MaxFontSize: Integer;
property MinFontSize: Integer;
```

Description
The MaxFontSize/MinFontSize propertes determines the largest/smallest font sizes available in the Font dialog box. Use the MaxFontSize property when you want to limit the font sizes available to the user. To limit the font sizes available, the Options set property of the Font dialog box must also contain the value fdLimitSize. If fdLimitSize is False, setting the MaxFontSize property has no affect on number of fonts available in the Font dialog box.

The default value is 0, which means there is no maximum/minium font size specified.

Example
This example uses a Font dialog box, a button, and a label on a form. When the user clicks the button, the Font dialog box appears. The font sizes available are within the range of 10 to 14. When the user chooses OK, the selected font is applied to the caption of the label.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  FontDialog1.Options := [fdLimitSize];
  FontDialog1.MaxFontSize := 14;
  FontDialog1.MinFontSize := 10;
  if FontDialog1.Execute then
    Label1.Font := FontDialog1.Font;
end;
```

Max, Min Properties (Scroll Bar)

Delphi command
`Max, Min properties (TScrollBar )`

Applies to
TScrollBar components

Declaration
```
property Max: Integer;
```

Description
The Max property along with the Min property determines the number of possible positions the scroll box can have on the scroll bar. The LargeChange and SmallChange properties use the number of positions to determine how far to move the scroll box when the user uses the scroll bar.

Example
The following code sets the minimum position to the value specified in an edit box, and sets the maximum position to 1000 more than the minimum position.

```
ScrollBar1.Min := StrToInt(Edit1.Text);
ScrollBar1.Max := ScrollBar1.Min + 1000;
```

MaxButton Property

Delphi command
`biMaximize in BorderIcons set (TForm)`

Applies to
TForm component

Declaration
```
property BorderIcons: TBorderIcons;
```

Description
The BorderIcons property is a set whose values determine which icons appear on the title bar of a form. With BorderIcons set to biMaximize, the form has a Maximize button

Example
The following code removes a form's Maximize button when the user clicks a button:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  BorderIcons := BorderIcons - [biMaximize];
end;
```

MaxLength Property

Delphi command
`MaxLength (TMaskEdit)`

Declaration
```
property MaxLength: Integer;
```

Description
The MaxLength property specifies the maximum number of characters the user can enter in an edit box, memo, or combo box. The default setting for MaxLength is 0, which means that there is no limit on the number of characters the control can contain. Any other number limits the number of characters the control accepts.

Example
The following example sets the maximum number of characters for an edit box to 80:

```
MaskEdit1.MaxLength := 80;
```

Menu Control

Delphi command
`TMainMenu, TPopUpMenu Components`

Description
See Standard Page Components

Mid Function

Delphi command
Copy

Declaration
```
function Copy(S: string; Index, Count: Integer): string;
```

Description
The Copy function returns a substring of a string.

S is a string-type expression. Index and Count are integer-type expressions. Copy returns a string containing Count characters starting with at S[Index].

If Index is larger than the length of S, Copy returns an empty string.

If Count specifies more characters than are available, the only the characters from S[Index] to the end of S are returned.

Example
```
 var S: string;
begin
  S := 'ABCDEF';
  S := Copy(S, 2, 3);    { 'BCD' }
end;
```

Min, Max Functions (SQL)

Delphi command
`Min, Max functions (SQL)`

Description
SQL Syntax

Example
```
Select Max(OrderNo) from Orders
```

MinButton Property

Delphi command
`biMinimize in BorderIcons set (TForm)`

Applies to
TForm component

Declaration
```
property BorderIcons: TBorderIcons;
```

Description
The BorderIcons property is a set whose values determine which icons appear on the title bar of a form.

Example
The following code removes a form's Maximize button when the user clicks a button:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  BorderIcons := BorderIcons - [biMinimize];
end;
```

Minute Function

Delphi command
DecodeTime procedure

Declaration
```
procedure DecodeTime(Time: TDateTime; var Hour, Min, Sec, MSec: Word);
```

Description
The DecodeTime procedure breaks the value specified as the Time parameter into hours, minutes, seconds, and milliseconds.

Example
This example uses a button and two labels on a form. When the user clicks the button, the current date and time are reported in the captions of the two labels.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Present: TDateTime;
  Year, Month, Day, Hour, Min, Sec, MSec: Word;
 begin
  Present:= Now;
  DecodeDate(Present, Year, Month, Day);
  Label1.Caption := 'Today is Day ' + IntToStr(Day) + ' of Month '
    + IntToStr(Month) + ' of Year ' + IntToStr(Year);
  DecodeTime(Present, Hour, Min, Sec, MSec);
  Label2.Caption := 'The time is Minute ' + IntToStr(Min) + ' of Hour '
    + IntToStr(Hour);

end;
```

MkDir Statement

Delphi command
MkDir procedure

Declaration
```
procedure MkDir(S: string);
```

Description
The MkDir procedure creates a new subdirectory with the path specified by string S. The last item in the path cannot be an existing file name.

{$I+} lets you handle run-time errors using exceptions. For more information on handling run-time library exceptions, see Handling RTL Exceptions.

Example
```
uses Dialogs;
begin
  {$I-}
  { Get directory name from TEdit control }
  MkDir(Edit1.Text);
  if IOResult <> 0 then
    MessageDlg('Cannot create directory', mtWarning, [mbOk], 0)
  else
    MessageDlg('New directory created', mtInformation, [mbOk], 0);
end;
```

Month Function

Delphi command
DecodeDate procedure

Declaration
```
procedure DecodeDate(Date: TDateTime; var Year, Month, Day: Word);
```

Description
The DecodeDate procedure breaks the value specified as the Date parameter into Year, Month, and Day values. If the given TDateTime value is less than or equal to zero, the year, month, and day return parameters are all set to zero.

Example
This example uses a button and two labels on a form. When the user clicks the button, the current date and time are reported in the captions of the two labels.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Present: TDateTime;
  Year, Month, Day, Hour, Min, Sec, MSec: Word;
 begin
  Present:= Now;
  DecodeDate(Present, Year, Month, Day);
  Label1.Caption := 'Today is Day ' + IntToStr(Day) + ' of Month '
    + IntToStr(Month) + ' of Year ' + IntToStr(Year);
  DecodeTime(Present, Hour, Min, Sec, MSec);
  Label2.Caption := 'The time is Minute ' + IntToStr(Min) + ' of Hour '
    + IntToStr(Hour);

end;
```

MouseDown, MouseUp Events

Delphi command

`OnMouseDown, OnMouseUp events`

Declaration

```
TMouseEvent = procedure (Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer) of object;
property OnMouseDown: TMouseEvent;
property OnMouseUp: TMouseEvent;
```

Description

The OnMouseDown event occurs when the user presses a mouse button with the mouse pointer over a control. Use the OnMouseDown event handler when you want some processing to occur as a result of pressing a mouse button.

The OnMouseUp event occurs when the user releases a mouse button that was pressed with the mouse pointer over a component. Use the OnMouseUp event handler when you want processing to occur when the user releases a mouse button.

Example

The following code creates and displays a label when a mouse button is pressed. If you attach this event handler to the OnMouseDown event of a form, a label specifying the coordinates of the mouse pointer appears when the user clicks the mouse button. Note that the StdCtrls unit must be added to the uses clause of the interface section of the form's unit to be able to create labels dynamically.

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  NewLabel: TLabel;
begin
  NewLabel := TLabel.Create(Form1);
  NewLabel.Parent := Self;
  NewLabel.Left := X;
  NewLabel.Top := Y;
  NewLabel.Caption := '(' + IntToStr(X) + ',' + IntToStr(Y) + ')';
  NewLabel.Visible := True;
end;
```

MouseIcon Property

Delphi command
`Screen.Cursor Property`

Applies to
TScreen component

Declaration
```
property Cursor: TCursor;
```

Description
The Screen object's Cursor property controls the mouse cursor shape at a global level. Assigning any value but crDefault to the Screen object's Cursor property sets the mouse cursor shape for all windows belonging to the application. The global mouse cursor shape remains in effect until you assign crDefault to the Screen object's Cursor property, at which point normal cursor behavior is restored.

To see a list of possible cursor shapes, see the Cursor property for all controls.

Example
Assignments to the Screen object's cursor property are typically guarded by a try...finally statement to ensure that normal cursor behavior is restored, for example:

```
Screen.Cursor := crHourglass;                 { Show hourglass cursor }
try
  { Do some lengthy operation }
finally
  Screen.Cursor := crDefault;                  { Always restore to normal }
end;
```

MouseMove Event

Delphi command
`OnMouseMove event`

Declaration
```
TMouseMoveEvent = procedure(Sender: TObject; Shift: TShiftState; X, Y:
Integer) of object;
property OnMouseMove: TMouseMoveEvent;
```

Description
The OnMouseMove event occurs when the user moves the mouse pointer while the mouse pointer is over a control. Use the OnMouseMove event handler when you want something to happen when the mouse pointer moves within the control.

By using the Shift parameter of the OnMouseDown event handler, you can respond to the state of the mouse buttons and shift keys. Shift keys are the Shift, Ctrl, and Alt keys.

Example
The following code updates two labels when the mouse pointer is moved. The code assumes you have two labels on the form, lblHorz and lblVert. If you attach this code to the OnMouseMove event of a form, lblHorz continually displays the horizontal position of the mouse pointer, and lblVert continually displays the vertical position of the mouse pointer while the pointer is over the form.

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y:
Integer);
begin
  lblHorz.Caption := IntToStr(X);
  lblVert.Caption := IntToStr(Y);
end;
```

MousePointer Property

Delphi command
`Screen.Cursor Property`

Declaration
```
property Cursor: TCursor;
```

Description
The Screen object's Cursor property controls the mouse cursor shape at a global level. Assigning any value but crDefault to the Screen object's Cursor property sets the mouse cursor shape for all windows belonging to the application. The global mouse cursor shape remains in effect until you assign crDefault to the Screen object's Cursor property, at which point normal cursor behavior is restored.

To see a list of possible cursor shapes, see the Cursor property for all controls.

Example
Assignments to the Screen object's cursor property are typically guarded by a try...finally statement to ensure that normal cursor behavior is restored, for example:

```
Screen.Cursor := crHourglass;                  { Show hourglass cursor }
try
  { Do some lengthy operation }
finally
  Screen.Cursor := crDefault;                   { Always restore to normal }
end;
```

Move Method

Delphi command
`SetBounds method`

Declaration
```
procedure Setbounds(ALeft, ATop, AWidth, AHeight: Integer);
```

Description
The SetBounds method sets the component's boundary properties, Left, Top, Width, and Height, to the values passed in ALeft, ATop, AWidth, and AHeight, respectively.

SetBounds enables you to set more than one of the component's boundary properties at a time. Although you can always set the individual boundaries, using SetBounds enables you to make several changes at once without repainting the control for each change.

Example
The following code doubles the size of a button control when the user clicks it:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.SetBounds(Left, Top, Height * 2, Width * 2);
end;
```

Note that you could use the following code instead, but each click would result in the button being redrawn twice: once to change the height, and once to change the width:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Height := Button1.Height * 2;
  Button1.Width := Button1.Width * 2;
end;
```

Move Method (Data Access)

Delphi command
`MoveBy (TTable, TQuery, TStoredProc)`

Declaration
```
function MoveBy(Distance: Integer): Integer;
```

Description
The MoveBy method moves the dataset cursor by Distance records. If Distance is negative, the move is backward. If Distance is positive, the movement is forward. If Distance is zero, no move is done. MoveBy returns the number of records that were traversed.

If the dataset is in Insert or Edit state, MoveBy will perform an implicit Post of any pending data, even if Distance is 0.

Example
The following code skips three records forward:

```
Table1.MoveBy(3);
```

MoveFirst, MoveLast, MoveNext, MovePrevious Methods

Delphi command

`First, Last, Next, Prior methods`

Declaration

```
procedure First;
procedure Last;
procedure Next;
procedure Prior;
```

Description

The First method moves the cursor to the first record in the active range of records of the dataset.

The Last method moves the cursor to the last record in the active range of records of the dataset.

The Prior method moves the current record position of the dataset backward by one record.

The Next method moves the cursor forward by one record. If the cursor is already on the last record, it does not move.

The active range of records is affected by the filter established with ApplyRange.

If the dataset is in Edit or Insert state, all will perform an implicit Post of any pending data.

Example

```
{ Move to the next record }
Table1.Next;
if Table1.Eof then { No more records };
```

Example 2

```
Table1.Last;
```

MsgBox Function

Delphi command
`MessageBox (TApplication)`

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

Description
See MessageBox in Delphi.hlp for message box formatting

The MessageBox method is an encapsulation of the Windows API MessageBox function except that you don't need to supply a window handle.

The MessageBox method displays a generic dialog box that displays a message and one or more buttons. The value of the Text parameter is the message, which can be longer than 255 characters if necessary. Long messages are automatically wrapped in the message box. The value of the Caption property is the caption that appears in the title bar of the dialog box. Captions can be longer than 255 characters, but they don't wrap. A long caption results in a wide message box.

Example
This example uses a button and a label on a form. When the user clicks the button, a message box appears. When the user responds to the message box, the button selected is reported in the caption of the label:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Button: Integer;
begin
  Button := Application.MessageBox('Welcome to Delphi!', 'Message Box',
mb_OKCancel +
    mb_DefButton1);
 if Button = IDOK then
   Label1.Caption := 'You chose OK';
 if Button = IDCANCEL then
   Label1.Caption := 'You chose Cancel';
end;
```

MultiLine Property

Delphi command

```
Use TEdit control for single line
```

Description

See Standard Page Components

MultiSelect Property

Delphi command
`MultiSelect property`

Applies to
TFileListBox, TListBox, TListView components

Declaration
```
property MultiSelect: Boolean;
```

Description
The MultiSelect property determines whether the user can select more than one element at a time from the list. If MultiSelect is True, the user can select multiple items. If MultiSelect if False, multiple items can be selected in the list box at the same time. The default value is False.

Example
This line of code ensures that the user can select multiple items in a list box:

```
ListBox1.MultiSelect := True;
```

NPV Function

Delphi command

`NetPresentValue function (Math.pas unit)`

Declaration

```
function NetPresentValue(Rate: Extended; const CashFlows: array of Double;
  PaymentTime: TPaymentTime): Extended;
```

Description

The NetPresentValue function calculates the current value of an array of estimated cash flow values, discounted at the given interest rate of Rate. PaymentTime indicates whether the cash flows occur at the beginning or end of the period (by entering a value of 1or 0, respectively). NetPresentValue helps you determine how much an investment is currently worth, based on expected earnings, although its accuracy depends on the accuracy of the cash flows in the array.

NPer Function

Delphi command

`NumberOfPeriods function (Math.pas unit)`

Declaration

```
function NumberOfPeriods(Rate, Payment, PresentValue, FutureValue: Extended;
 PaymentTime: TPaymentTime): Extended;
```

Description

The NumberOfPeriods function computes the number of payment periods required for an investment of PresentValue to reach a value of FutureValue, while making regular payments of Payment and accruing interest at the rate of Rate per compounding period. PaymentTime indicates whether the cash flows occur at the beginning or end of the period (by entering a value of 1or 0, respectively).

Name Property

Delphi command
`Name property`

Applies to
All components

Declaration
```
property Name: TComponentName;
```

Description
The Name property contains the name of the component as referenced by other components. By default, Delphi assigns sequential names based on the type of the component, such as 'Button1', 'Button2', and so on. You may change these to suit your needs.

Example
The following code lists the names of all the components of Form1 in a list box.

```
var
  I: Integer;
begin
  for I := 0 to Form1.ComponentCount-1 do
    ListBox1.Items.Add(Form1.Components[I].Name);
end;
```

Name Property (Data Access)

Delphi command
`Name property`

Applies to
All components

Declaration
```
property Name: TComponentName;
```

Description
The Name property contains the name of the component as referenced by other components. By default, Delphi assigns sequential names based on the type of the component, such as 'Button1', 'Button2', and so on. You may change these to suit your needs.

Example
The following code lists the names of all the components of Form1 in a list box.

```
var
  I: Integer;
begin
  for I := 0 to Form1.ComponentCount-1 do
    ListBox1.Items.Add(Form1.Components[I].Name);
end;
```

Name Statement

Delphi command
`RenameFile function`

Declaration
```
function RenameFile(const OldName, NewName: string): Boolean;
```

Description
The RenameFile function attempts to change the name of the file specified by OldFile to NewFile. If the operation succeeds, RenameFile returns True. If it cannot rename the file (for example, if a file called NewName already exists), it returns False.

Example
The following code renames a file:

```
if not RenameFile('OLDNAME.TXT','NEWNAME.TXT') then
  ErrorMsg('Error renaming file!');
```

Named Date/TIme Formats (Format Function)

Delphi command

`ShortDateFormat, Format Strings (DateToStr function, Format)`

Declaration

```
function DateToStr(Date: TDateTime): string;
```

Description

The DateToStr function converts a variable of type TDateTime to a string. The conversion uses the format specified by the ShortDateFormat global variable.

Named Numeric Formats (Format Function)

Delphi command

```
Format Strings (Format function)
```

Declaration

```
function Format(const Format: string; const Args: array of const): string;
```

Description

This function formats the series of arguments in the open array Args. Formatting is controlled by the Object Pascal format string Format; the results are returned in the function result as a Pascal string.

For information on the format strings, see Format Strings in Delphi.HLP.

NegotiateMenus Property

Delphi command
`AutoMerge property (TMainMenu)`

Applies to
TMainMenu component

Declaration
```
property AutoMerge: Boolean;
```

Description
The AutoMerge property determines if the main menus (TMainMenu) of forms other than the main form merge with the main menu of the main form in non-MDI applications at run time. The default value is False. To merge the form's menus with the main menu in the main form, set the AutoMerge property of each main menu you want merged to True. Make sure that the AutoMerge property of the main menu you are merging with other menus remains False. How menus merge depends on the value of the GroupIndex property for each menu item.

If the application is an MDI application (the FormStyle properties are set so the main form is a parent form and subsequent forms are child forms), menu merging occurs automatically and you don't need to use the AutoMerge property.

Example
This example uses two forms with a main menu and a button on each form. Using the Object Inspector, set the GroupIndex value for each menu item on the menu bar in the second form to a number greater than 0. When the application runs and the user clicks the button on the first form, the main menu on the second form merges with the main menu of the first form. When the user clicks the button on the second form, the form closes.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.MainMenu1.AutoMerge := True;
  Form2.Show;
end;
```

This is the code for the button-click event handler on the second form:

```
procedure TForm2.Button1Click(Sender: TObject);
begin
  Close;
end;
```

To run this example, you must add Unit2 to the uses clause of Unit1.

NegotiatePosition Property

Delphi command
`GroupIndex (TmenuItem)`

Applies to
TMenuItem component

Declaration
```
property GroupIndex: Byte;
```

Description
If your application has multiple forms, you'll probably want your application's main menu to change as different forms become active. The alternative is for each form to display its own menu within itself. MDI applications always merge the menus of child windows with the main menu of the parent window. By using the GroupIndex property for menu items, you can determine how menus are merged. You can choose to replace or insert menu items in a menu bar.

Each menu item has a GroupIndex property value. By default, all menu items in a menu bar have the same GroupIndex value, unless you explicitly change them. Each successive menu item in a menu bar must have a GroupIndex value equal to or greater than the previous menu item.

NewPage Method

Delphi command
`NewPage method (TPrinter)`

Declaration
```
procedure NewPage;
```

Description
The NewPage method forces the current print job to begin printing on a new page in the printer. It also increments the value of the PageNumber property and resets the value of the Pen property of the Canvas back to (0, 0).

Example
This example uses a button on a form. When the user clicks the button, a rectangle is printed twice, one per page.

To run this example successfully, you must add the Printers unit to the uses clause of your unit.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Printer do
  begin
    BeginDoc;
    Canvas.Rectangle(10, 10, 200, 200);
    NewPage;
    Canvas.Rectangle(10, 10, 200, 200);
    EndDoc;
  end;
end;
```

NewPassword Method

Delphi command

```
Use AddPassword method for Paradox tables
```

Declaration

```
procedure AddPassword(const Password: string);
```

Description

The AddPassword method is used to add a new password to the current TSession component for use with Paradox tables. When an application opens a Paradox table that requires a password, the user will be prompted to enter a password unless the Session has a valid password for the table.

NoMatch Property

Delphi command
`FindKey , GoToKey, FindFirst, FindNext, etc. Methods`

Applies to
TTable component

Declaration
```
function FindKey(const KeyValues: array of const): Boolean;
```

Description
The FindKey method searches the database table to find a record whose index fields match those passed in KeyValues. FindKey takes a comma-delimited array of values as its argument, where each value corresponds to a index column in the underlying table. The values can be literals, variables, null, or nil. If the number of values supplied is less than the number of columns in the database table, then the remaining values are assumed to be null. FindKey will search for values specified in the array in the current index.

Example
This example will search for CustNo = '1234':

```
if Table1.FindKey(['1234']) then
   ShowMessage('Customer Found');
```

Normal Property

Delphi command
`FileType Property`

Applies to
TFileListBox component

Declaration
```
property FileType: TFileType;
```

Description
The FileType property determines which files are displayed in the file list box based on the attributes of the files. Because FileType is of type TFileType, which is a set of file attributes, FileType can contain multiple values. For example, if the value of FileType is a set containing the values ftReadOnly and ftHidden, only files that have the read-only and hidden attributes are displayed in the list box

When ftNormal is True, the list box can display files with no attributes.

Example
This example uses a file list box on a form. When the application runs, only read-only files, directories, volume IDs, and files with no attributes appear in the list box.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileType := [ftReadOnly, ftDirectory, ftVolumeID, ftNormal];
end;
```

Now Function

Delphi command
Now Function

Declaration
```
function Now: TDateTime;
```

Description
The Now function returns the current date and time, corresponding to Date + Time.

Example
This example uses a label and a button on a form. When the user clicks the button, the current date and time appear as the caption of the label.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := 'The date and time is  ' + DateTimeToStr(Now);
end;
```

Number Property

See Exception Handling in Users Guide.

NumberFormat Property

Delphi command
`DisplayFormat Property`

Applies to
TDateField, TDateTimeField, TIntegerField, TSmallintField, TTimeField, TWordField, TBCDField, TCurrencyField, TFloatField components

Declaration
```
property DisplayFormat: string
```

Description
The DisplayFormat property is used to format the value of the field for display purposes.

For TIntegerField, TSmallintField, and TWordField, formatting is performed by FloatToTextFmt. If DisplayFormat is not assigned a string, the value is formatted by Str.

For TDateField, TDateTimeField, and TTimeField, formatting is performed by DateTimeToStr. If DisplayFormat is not assigned a string, the value is formatted according to the default Windows specifications in the [International] section of the WIN.INI file.

For TBCDField, TCurrencyField, and TFloatField, formatting is performed by FloatToTextFmt. If DisplayFormat is not assigned a string, the value is formatted according to the value of the Currency property.

OLE Container Control

Delphi command

`OLEContainer Component`

Description

The TOleContainer component lets you embed or link OLE objects in your Delphi 2.0 application. TOleContainer handles many of the complexities of OLE 2.0 for you. Letting the user choose an OLE object to insert is as simple as calling the InsertObjectDialog method. Call the CreateObject or CreateObjectFromFile methods to create an embedded OLE object; use the CreateLinkToFile method to create a linked OLE object.

OLEDropAllowed Property

Delphi command
See OnDragOver Event in Delphi.hlp

Example
This OnDragOver event handler permits the list box to accept a dropped label:

```
procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

The Source parameter identifies what is being dragged. The Sender is the
control being dragged over.
This code permits the list box to accept any dropped control:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := True;
end;
```

OLEType Property

Delphi command
`Linked Property`

Applies to
TOleContainer component

Declaration
```
property Linked: Boolean;
```

Description
Runtime and readonly. Indicates whether the OLE object is linked. If True, the OLE object is linked; if False, it's embedded. An OLE object must already be loaded in the container before accessing the Linked property.

ORDER BY Clause (SQL)

Delphi command
`ORDER BY Clause (SQL)`

Description
SQL Syntax

Example
The ORDER BY clause specifies the order of retrieved rows. For example, the following query retrieves a list of all parts listed in alphabetical order by part name:

```
SELECT * FROM PARTS
ORDER BY PART_NAME ASC
```

The next query retrieves all part information ordered in descending numeric order by part number:

```
SELECT * FROM PARTS
ORDER BY PART_NO DESC
```

Calculated fields can be ordered by correlation name or ordinal position. For example, the following query orders rows by FULL_NAME, a calculated field:

```
SELECT LAST_NAME || ', ' || FIRST_NAME AS FULL_NAME, PHONE,
FROM CUSTOMER
ORDER BY FULL_NAME
```

Projection of all grouping or ordering columns is not required.

ObjectMove Event

Delphi command
`OnObjectMove Event`

Applies to
TOleContainer component

Declaration
```
TObjectMoveEvent = procedure(OleContainer: TOleContainer;const Bounds: TRect)
of object;
property OnObjectMove: TObjectMoveEvent;
```

Description
Occurs when the users moves or resizes the OLE object (by moving or sizing the hatched frame around the OLE object). You must handle this event to let the user move or resize the OLE object; you must set the OLE container's BoundsRect to the Bounds parameter of the event handler or the OLE object won't change its location or size.

Example:
```
procedure TMDIChildForm.BobOleContainerObjectMove(OleContainer:
TOleContainer;
const Bounds: TRect);
begin
  OleContainer.BoundsRect := Bounds;
end;
```

ObjectVerbs Property

Delphi command
`ObjectVerbs Property`

Applies to
TOleContainer component

Declaration
```
property ObjectVerbs: TStrings;
```

Description
Runtime and readonly. Returns a string list containing the names of the verbs the OLE object supports. The verb names can have embedded ampersand ('&') characters to indicate shortcut keys. An OLE object must already be loaded in the container before accessing the ObjectVerbs property.

ObjectVerbsCount Property

Delphi command
`ObjectVerbs.Count Property`

Description
Runtime and readonly. This function should return the number of verbs the OLE Object supports.

On Error Statement

Delphi command
See Exception Handling

Description
Object Pascal makes it easy to incorporate error handling into your applications because exceptions don't get in the way of the normal flow of your code. In fact, by moving error checking and error handling out of the main flow of your algorithms, exceptions can simplify the code you write.

When you declare a protected block, you define specific responses to exceptions that might occur within that block. When an exception occurs in that block, execution immediately jumps to the response you defined, then leaves the block.

On..GoSub, On...GoTo Statements

Delphi command

GoTo Statement

When using goto statements, you must observe the following rules:

The label referenced by the goto statement must be in the same block as the goto statement. You cannot jump into or out of a procedure or statement.

Jumping into a structured statement from outside that structured statement can have undefined effects, although the compiler does not indicate an error.

Good programming practices recommend that you use goto statements as little as possible.

Example

```
label 1, 2;
goto 1
   .
   .
   .
1: WriteLn ('Abnormal program termination');
2: WriteLn ('Normal program termination');
```

Open Statement

Delphi command
`FileOpen Function`

Declaration
```
function FileOpen(const FileName: string; Mode: Integer): Integer;
```

Description
The FileOpen function opens the specified file using the specified access mode.

Example
```
procedure OpenForShare(const FileName: String);
var
  FileHandle : Integer;
begin
  FileHandle := FileOpen(FileName, fmOpenWrite or fmShareDenyNone);
  if FileHandle > 0 then
    [valid file handle}
  else
    [Open error: FileHandle = negative DOS error code]
end;
```

Note Use of the non-native Pascal file variable handlers such as FileOpen is discouraged. These routines map to the Windows API functions and return file handles, not normal Pascal file variables. These are low-level file access routines. For normal file operations use the normal AssignFile, Rewrite, Reset operations instead of FileOpen.

OpenDatabase Function

Delphi command
`Open Method`

Applies to
TQuery, TStoredProc, TTable components

Declaration
```
procedure Open;
```

Description
The Open method opens the dataset, putting it in Browse state. It is equivalent to setting the Active property to True.

For TQuery, Open executes the SELECT statement in the SQL property. If the statement does not return a result set (for example, an INSERT or UPDATE statement), then use ExecSQL instead of Open.

Example
```
Database1.Open;
```

OpenDatabase Method

Delphi command
`Open Method`

Applies to
`TQuery, TStoredProc, TTable components`

Declaration
```
procedure Open;
```

Description
The Open method opens the dataset, putting it in Browse state. It is equivalent to setting the Active property to True.

For TQuery, Open executes the SELECT statement in the SQL property. If the statement does not return a result set (for example, an INSERT or UPDATE statement), then use ExecSQL instead of Open.

Example
```
Database1.Open;
```

OpenQueryDef Method

Delphi command
`Open Method`

Applies to
TQuery, TStoredProc, TTable components

Declaration
```
procedure Open;
```

Description
The Open method opens the dataset, putting it in Browse state. It is equivalent to setting the Active property to True.

For TQuery, Open executes the SELECT statement in the SQL property. If the statement does not return a result set (for example, an INSERT or UPDATE statement), then use ExecSQL instead of Open.

Example
```
try
  Query1.Open;
except
  on EDataBaseError do { The dataset could not be opened };
end;
```

OpenRecordset Method

Delphi command
`Open Method`

Applies to
TQuery, TStoredProc, TTable components

Declaration
```
procedure Open;
```

Description
The Open method opens the dataset, putting it in Browse state. It is equivalent to setting the Active property to True.

For TQuery, Open executes the SELECT statement in the SQL property. If the statement does not return a result set (for example, an INSERT or UPDATE statement), then use ExecSQL instead of Open.

Example
```
try
  Table1.Open;
except
  on EDataBaseError do { The dataset could not be opened };
end;
```

OpenTable Method

Delphi command
`Open Method`

Applies to
TQuery, TStoredProc, TTable components

Declaration
```
procedure Open;
```

Description
The Open method opens the dataset, putting it in Browse state. It is equivalent to setting the Active property to True.

For TQuery, Open executes the SELECT statement in the SQL property. If the statement does not return a result set (for example, an INSERT or UPDATE statement), then use ExecSQL instead of Open.

Example
```
try
  Table1.Open;
except
  on EDataBaseError do { The dataset could not be opened };
end;
```

OptionButton Control

Delphi command
See GroupIndex Property of TSpeedButton Component

Applies to
TSpeedButton component

Declaration
```
property GroupIndex: Integer;
```

Description
The GroupIndex property determines which speed buttons work together as a group.

By default, speed buttons have a GroupIndex property value of 0, indicating that they do not belong to a group. When the user clicks such a speed button, the button appears "pressed," or in its down state, then the button returns to its normal up state when the user releases the mouse button.

Speed buttons with the same GroupIndex property value (other than 0), work together as a group. When the user clicks one of these speed buttons, it remains "pressed," or in its down state, until the user clicks another speed button belonging to the same group. Speed buttons used in this way can present mutually exclusive choices to the user.

The default value is 0.

Example
This code assures that the three speed buttons work together as a group:

```
SpeedButton1.GroupIndex := 1;
SpeedButton2.GroupIndex := 1;
SpeedButton3.GroupIndex := 1;
```

OrdinalPosition Property

Delphi command
`FieldNo Property`

Declaration
```
property FieldNo: Integer;
```

Description
Run-time and read-only. FieldNo is the ordinal of the TField component in its dataset. This property is available for programs that make direct calls to the Borland Database Engine.

Orientation Property

Delphi command
`Orientation Property`

Applies to
TPrinter object;

Declaration
```
property Orientation: TPrinterOrientation;
```

Description
Run-time only. The value of the Orientation property determines if the print job prints vertically or horizontally on a page. These are the possible values:

Value	Meaning
poPortrait	The print job prints vertically on the page.
poLandscape	The print job prints horizontally on the page.

Example
This example uses two radio buttons on a form named Landscape and Portrait. The form also includes a button. When the user selects an orientation by clicking one of the radio buttons and then clicks the button to print one line of text, the print job prints using the selected orientation:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Printer.BeginDoc;
  Printer.Canvas.TextOut(100,100,'Hi there');
  Printer.EndDoc;
end;
procedure TForm1.PortraitClick(Sender: TObject);
begin
  Printer.Orientation := poPortrait;
end;
procedure TForm1.LandscapeClick(Sender: TObject);
begin
  Printer.Orientation := poLandscape;
end;
```

PPmt Function

Delphi command

```
Payment Function
```

Declaration

```
function Payment(Rate: Extended; NPeriods: Integer;
  PresentValue, FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

Description

The Payment function calculates the fully amortized payment of borrowing PresentValue dollars at Rate percent per period over NPeriods. It assumes that interest is paid at the end of each period.

FutureValue is the value that the investment will reach at some point. PaymentTime indicates whether the cash flows occur at the beginning or end of the period (enter a value of 1 for the beginning or 0 for the end).

PSet Method

Delphi command
`TCanvas.Pixels Properrty`

Applies to
TCanvas object

Declaration
```
property Pixels[X, Y: Longint]: TColor;
```

Description
The Pixels array enables you to access any pixel on the canvas directly, to either set or read the color there. Each element in Pixels contains the color of the corresponding pixel in the canvas. The array indexes, X and Y, specify the horizontal and vertical coordinates of the pixel, respectively.

Note This is the slowest way to draw an image.

Example
This example draws a red line (very slowly) when the form becomes active. Attach the following code to the OnActivate event handler:

```
procedure TForm1.FormActivate(Sender: TObject);
var
  W: Word;
begin
  for W := 10 to 200 do
    Canvas.Pixels[W, 10] := clRed;
end;
```

PV Function

Delphi command

`PresentValue Function`

Declaration

```
function PresentValue(Rate: Extended; NPeriods: Integer;
  Payment, FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

Description

The PresentValue function calculates the present value of an investment where Payment is received for NPeriods and is discounted at the rate of Rate per period. FutureValue is the value the investment may reach at some point. PaymentTime indicates whether the cash flows occur at the beginning or end of the period (enter a value of 1 for the beginning or 0 for the end).

Page Property

Delphi command
`PageNumber Property`

Declaration
```
property PageNumber: Integer;
```

Description
Run-time and read-only. The PageNumber property contains the number of the current page. Each time an application calls the NewPage method, NewPage increments the value of PageNumber.

Example
This example uses a button on a form. When the user clicks the button, one line of text is printed on six separate pages. As each page is printed, a message indicating the number of the page being printed appears on the form.

To run this example successfully, you must add Printers to the uses clause of your unit.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
begin
  Printer.BeginDoc;
  X := 10;
  Y := 10;
    for I := 1 to 6 do
    begin
      Printer.Canvas.TextOut(100, 100, 'Object Pascal is great');
      Canvas.TextOut(X, Y, 'Printing page ' + IntToStr(Printer.PageNumber));
      Printer.NewPage;
      Y := Y + 20;
    end;
  Printer.EndDoc;
end;
```

Paint Event

Delphi command
`OnPaint Event`

Applies to
TForm, TPaintBox component

Declaration
```
property OnPaint: TNotifyEvent;
```

Description
The OnPaint event occurs when Windows requires the form or paint box to paint, such as when the form or paint box receives focus or becomes visible when it wasn't previously. Your application can use this event to draw on the canvas of the form or paint box.

Example
The following code is an entire unit that loads a background bitmap onto the Canvas of the main form in the OnPaint event handler.

```
unit Unit1;
interface
uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls, Forms,
Dialogs;
type
  TForm1 = class(TForm)
    procedure FormPaint(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    TheGraphic: TBitmap; { Add this Declaration for the graphic}
  public
    { Public Declarations }
  end;
var
  Form1: TForm1;
implementation

{$R *.DFM}
procedure TForm1.FormPaint(Sender: TObject); { OnPaint event handler}
begin
  Form1.Canvas.Draw(0, 0, TheGraphic); { Draw the graphic on the Canvas }
end;
procedure TForm1.FormCreate(Sender: TObject); { OnCreate event handler }
begin
  TheGraphic := TBitmap.Create; { Create the bitmap object }
  TheGraphic.LoadFromFile('C:\APP\BKGRND.BMP'); { Load the bitmap from a
file}
end;
end.
```

PaintPicture Method

Delphi command
`TCanvas.Draw Method`

Applies to
TCanvas objects

Declaration
```
procedure Draw(X, Y: Integer; Graphic: TGraphic);
```

Description
The Draw method draws the graphic specified by the Graphic parameter on the canvas at the location given in the screen pixel coordinates (X, Y). Graphics can be bitmaps, icons, or metafiles.

Example
The following code draws the graphic in C:\WINDOWS\TARTAN.BMP centered in Form1 when the user clicks Button1. Attach this code to the OnClick event handler of Button1.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap1: TBitmap;
begin
  Bitmap1 := TBitmap.Create;
  Bitmap1.LoadFromFile('c:\windows\tartan.bmp');
  Form1.Canvas.Draw((Form1.Width div 2)-(Bitmap1.Width div 2),
    (Form1.Height div 2) - (Bitmap1.Height div 2), Bitmap1);
end;
```

PaperBin Property

Delphi command
`ADeviceMode parameter of SetPrinter Method`

Applies to
TPrinter object

Declaration
```
procedure SetPrinter(ADevvice, ADriver, APort: PChar; ADeviceMode: THandle);
```

Description
The SetPrinter method specifies a printer as the current printer. You should seldom, if ever, need to call this method, but instead should access the printer you want in the Printers property array. For more information, see the Windows API CreateDC function.

PaperSize Property

Delphi command
`ADeviceMode Parameter of SetPrinter Method`

Applies to
TPrinter object

Declaration
```
procedure SetPrinter(ADevvice, ADriver, APort: PChar; ADeviceMode: THandle);
```

Description
The SetPrinter method specifies a printer as the current printer. You should seldom, if ever, need to call this method, but instead should access the printer you want in the Printers property array. For more information, see the Windows API CreateDC function.

Parameter Object, Parameters Collection

Delphi command
`TQuery.Params Property`

Applies to
TQuery component

Declaration
```
property Params[Index: Word]: TParam;
```

Description
When you enter a query, Delphi creates a Params array for the parameters of a dynamic SQL statement. Params is a zero-based array of TParam objects with an element for each parameter in the query; that is, the first parameter is Params[0], the second Params[1], and so on. The number of parameters is specified by ParamCount. Read-only and run-time only.

Note Use the ParamByName method instead of Params to avoid dependencies on the order of the parameters.

Example
For example, suppose a TQuery component named Query2 has the following statement for its SQL property:

```
INSERT
  INTO COUNTRY (NAME, CAPITAL, POPULATION)
  VALUES (:Name, :Capital, :Population)
```

An application could use Params to specify the values of the parameters as follows:

```
Query2.Params[0].AsString := 'Lichtenstein';
Query2.Params[1].AsString := 'Vaduz';
Query2.Params[2].AsInteger := 420000;
```

These statements would bind the value "Lichtenstein" to the :Name parameter, "Vaduz" to the :Capital parameter, and 420000 to the :Population parameter.

Parent Property

Delphi command
`Parent Property`

Applies to
All controls

Declaration
```
property Parent: TWinControl;
```

Description
The Parent property contains the name of the parent of the control. The parent of a control is the windowed control that contains the control. If one control (parent) contains others, the contained controls are child controls of the parent. For example, if your application includes three radio buttons in a group box, the group box is the parent of the three radio buttons, and the radio buttons are the child controls of the group box.

Don't confuse the Parent property with the Owner property. A form is the owner of all the components on it, whether or not they are windowed controls. A child control is always a windowed control contained within another windowed control (its parent). If you put three radio buttons in a group box on a form, the owner of the radio buttons is still the form, while the parent is the group box.

PasswordChar Property

Delphi command
`PasswordChar Property`

Applies to
TDBEdit, TEdit, TMaskEdit components

Declaration
```
property PasswordChar: Char;
```

Description
The PasswordChar property lets you create an edit box that displays special characters in place of the entered text. By default, PasswordChar is the null character (ANSI character zero), meaning that the control displays its text normally. If you set PasswordChar to any other character, the control displays that character in place of each character in the control's text.

Example
The following code displays asterisks for each character in an edit box called PasswordField:

```
PasswordField.PasswordChar := '*';
```

Paste Method

`Delphi command`

Paste Method

Applies to

TOleContainer component

Declaration

```
procedure Paste;
```

Description

Pastes the contents of the Windows clipboard as an embedded object. If there's already an OLE object in the container, it's destroyed and any changes the user made to it are discarded.

PasteOK Property

Delphi command
`CanPaste Property`

Applies to
TOleContainer component

Declaration
```
property CanPaste: Boolean;
```

Description
Runtime and readonly. Indicates whether the data in the Windows clipboard is suitable for pasting as an embedded object.

PasteSpecialDlg Method

Delphi command
`PasteSpecialDialog Method`

Applies to
TOleContainer component

Declaration
```
function PasteSpecialDialog: Boolean;
```

Description
Executes the Paste Special OLE dialog to give the user more control over how the contents of the Windows clipboard are pasted into the container. The Paste Special dialog box lets the user select the format of the data, whether it should be embedded or linked, and whether to display the OLE object should be displayed as an icon (and if so, to choose a different icon). PasteSpecialDialog returns True if the dialog box was successfully display and the user chose the OK button or False otherwise. If the user chose the OK button and there's already an OLE object in the container, it's destroyed and any changes the user made to it are discarded.

Path Property

Delphi command
`Directory Property`

Applies to
TDirectoryListBox, TFileListBox components

Declaration
```
property Directory: string;
```

Description
The value of the Directory property determines the current directory for the file list box and directory list box components. The file list box displays the files in the directory specified in the Directory property. The directory list box displays the value of the Directory property as the current directory in the list box.

Examine the example to see how a directory list box and a file list box can work together through their Directory properties.

Example
If you have a file list box and a directory list box on a form, this code changes the current directory in the directory list box and displays the files in that directory in the file list box when the user changes directories using the directory list box:

```
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
end;
```

PathChange Event

Delphi command
`OnChange Event`

Declaration
```
property OnChange: TNotifyEvent;
```

The OnChange event specifies which event handler should execute when the contents of a component or object changes.

Pattern Property

Delphi command
`Mask Property`

Applies to
TFileListBox components

Declaration
```
property Mask: string;
```

Description
The Mask property determines which files are displayed in the file list box. A file mask or file filter is a filename that usually includes wildcard characters (*.PAS, for example). Only files that match the mask are displayed in list box. The file mask *.* displays all files, which is the default value.

You can specify multiple file masks. Separate the file mask specifications with semicolons. For example, *.PAS; *.EXE.

Example
This example uses a file list box on a form. When the application runs, the list box displays only files with a .PAS file extension:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.Mask := '*.PAS';
end;
```

PatternChange Event

Delphi command
`OnChange Event`

Applies to
TFileListBox components

Declaration
```
property OnChange: TNotifyEvent;
property Mask: string;
```

Description
The OnChange event specifies which event handler should execute when the contents of a component or object changes.

The Mask property determines which files are displayed in the file list box. A file mask or file filter is a filename that usually includes wildcard characters (*.PAS, for example). Only files that match the mask are displayed in list box. The file mask *.* displays all files, which is the default value.

You can specify multiple file masks. Separate the file mask specifications with semicolons. For example, *.PAS; *.EXE.

Percent Position Property

See RecNo and RecordCount Properties in Delphi.hlp

Picture Object

Delphi command
`TPicture Object`

Description
See Additional Components Page

Picture Property

Delphi command
`Picture Property`

Applies to
TDBImage, TImage components

Declaration
```
property Picture: TPicture;
```

Description
The Picture property determines the image that appears on the image control. The property value is a TPicture object which can contain an icon, metafile, bitmap graphic, or user-defined graphic object.

Example
This example uses two picture components. When the form first appears, two bitmaps are loaded into the picture components and stretched to fit the size of the components. To try this code, substitute names of bitmaps you have available.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Stretch := True;
  Image2.Stretch := True;
  Image1.Picture.LoadFromFile('BITMAP1.BMP');
  Image2.Picture.LoadFromFile('BITMAP2.BMP');
end;
```

PictureBox Control

Delphi command
`TImage Component`

Description
See Additional Page Components

Pmt Function

Delphi command

```
Payment Function
```

Declaration

```
function Payment(Rate: Extended; NPeriods: Integer;
  PresentValue, FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

Description

The Payment function calculates the fully amortized payment of borrowing PresentValue dollars at Rate percent per period over NPeriods. It assumes that interest is paid at the end of each period.

FutureValue is the value that the investment will reach at some point. PaymentTime indicates whether the cash flows occur at the beginning or end of the period (enter a value of 1 for the beginning or 0 for the end).

Point Method

Delphi command
`TCanvas.Pixels Property`

Applies to
TCanvas object

Declaration
```
property Pixels[X, Y: Longint]: TColor;
```

Description
The Pixels array enables you to access any pixel on the canvas directly, to either set or read the color there. Each element in Pixels contains the color of the corresponding pixel in the canvas. The array indexes, X and Y, specify the horizontal and vertical coordinates of the pixel, respectively.

Note This is the slowest way to draw an image.

Example
This example draws a red line (very slowly) when the form becomes active. Attach the following code to the OnActivate event handler:

```
procedure TForm1.FormActivate(Sender: TObject);
var
  W: Word;
begin
  for W := 10 to 200 do
    Canvas.Pixels[W, 10] := clRed;
end;
```

PopupMenu Method

Delphi command
`TPopupMenu Component`

Description
See Standard Page Components

Port Property

Delphi command
`GetPrinter Method`

Applies to
TPrinter object

Declaration
```
procedure GetPrinter (ADevice, ADriver, APort: PChar; var ADeviceMode:
THandle);
```

Description
The GetPrinter method retrieves the current printer. You should rarely need to call this method and should instead access the printer you want in the Printers property array. For more information, see the CreateDC function in the Win32 Developer's Reference (WIN32.HLP).

Primary Property

Delphi command
`TIndexDef.Options Property`

Applies to
TIndexDef object

Declaration
```
property Options: TIndexOptions;
```

Description
Run-time and read-only. Options is the set of characteristics of the index. Possible elements are those of the TIndexOptions type: ixPrimary, ixUnique, ixDescending, ixNonMaintained, and ixCaseInsensitive.

Example
```
Table1.AddIndex('Key', 'CustNo', [ixPrimary]);
```

Print # Statement

Delphi command
`WriteLn Procedure`

Declaration
```
procedure Writeln([ var F: Text; ] P1 [, P2, ...,Pn ] );
```

Description
The Writeln procedure is an extension to the Write procedure, as it is defined for text files.

After executing Write, Writeln writes an end-of-line marker (carriage-return/linefeed) to the file. Writeln(F) with no parameters writes an end-of-line marker to the file. (Writeln with no parameter list corresponds to Writeln(Output).)

The file must be open for output.

Example
```
var
   s : string;
 begin
   Write('Enter a line of text: ');
   Readln(s);
   Writeln('You typed: ',s);
   Writeln('Hit <Enter> to exit');
   Readln;
 end;
```

Print Method

Delphi command
`BeginDoc, EndDoc Methods`

Applies to
TPrinter object

Declaration
```
procedure BeginDoc;
```

Description
The BeginDoc method sends a print job to the printer. If the print job is sent successfully, the application should call EndDoc to end the print job. The print job won't actually start printing until EndDoc is called.

To use the BeginDoc method, you must add the Printers unit to the uses clause of your unit.

PrintForm Method

Delphi command
`Print Method`

Applies to
TForm component

Declaration
```
procedure Print;
```

Description
The Print method prints the form.

Example
This example uses a button named PrintButton on a form. When the user chooses the button, the form prints.

```
procedure TForm1.PrintButtonClick(Sender: TObject);
begin
  Print;
end;
```

PrintQuality Property

Delphi command
`ADeviceMode parameter of SetPrinter Method`

Applies to
TPrinter object

Declaration
```
procedure SetPrinter(ADevvice, ADriver, APort: PChar; ADeviceMode: THandle);
```

Description
The SetPrinter method specifies a printer as the current printer. You should seldom, if ever, need to call this method, but instead should access the printer you want in the Printers property array. For more information, see the Windows API CreateDC function.

Printer Object, Printers Collection

Delphi command

`TPrinter Object; Printers Property`

Description

The TPrinter object encapsulates the printer interface of Windows. Within the Printers unit, the Printer function will create an instance of TPrinter, ready for you to use.

To start a print job, call the BeginDoc method. To end a print job that is sent successfully to the printer, call the EndDoc method. If a problem occurs and you need to terminate a print job that was not sent to the printer successfully, call the Abort method.

You can determine if a job is printing by checking the value of the Printing property. If the job aborted, the Aborted property is True.

The printing surface of a page is represented by the Canvas property. You can use the Brush, Font, and Pen properties of the Canvas object to determine how drawing or text appears on the page.

The list of installed printers is found in the Printers property. The value of the PrinterIndex property is the currently selected printer. The list of fonts supported by the current printer is found in the Fonts property.

You can determine if a print job prints in landscape or portrait orientation using the Orientation property.

You height and width of the current page is found in the PageHeight and PageWidth properties. The current page is the value of the PageNumber property.

The Title property determines the text that appears listed in the Print Manager and on network header pages.

Using the PrintScale property of a TForm component, you determine how the printed image of the form appears.

Whenever you use a TPrinter object, you must add Printers to the uses clause of the unit that implements the properties or methods of a TPrinter object.

The TPrinter object is a direct descendent of TObject. In addition to these properties and methods, this object also has the methods that apply to all objects.

Example

The following code displays the names of all printers in ListBox1.

```
begin
  ListBox1.Items := Printer1.Printers;
end;
```

Property Get Statement

Delphi command

```
Property Reserved Word
```

Description

The reserved word property enables you to declare properties. A property definition in a class declares a named attribute for objects of the class and the actions associated with reading and writing the attribute.

Property Let Statement

Delphi command

```
Property Reserved Word
```

Description

The reserved word property enables you to declare properties. A property definition in a class declares a named attribute for objects of the class and the actions associated with reading and writing the attribute.

Public Property

Delphi command

```
Public Standard Directive
```

Description

The public directive is used within class type Declarations.

Component identifiers declared in public component parts have no special restrictions on their scope.

Put Statement

Delphi command
`Write Procedure`

Declaration
```
procedure Write( [var F: Text; ] P1 [ , P2,..., Pn] );
```

Description
The Write procedure writes one or more values to a text file. F, if specified, is a text file variable. If F is omitted, the standard file variable Output is assumed. Each P is a write parameter. Each write parameter includes an output expression whose value is to be written to the file. A write parameter can also contain the specifications of a field width and a number of decimal places. Each output expression must be of a type Char, one of the Integer types (Byte, Shortint, Word, Longint, Cardinal) , one of the floating-point types (Single, Real, Double, Extended, Currency) , one of the string types (PChar, AnsiString, ShortString) , a packed string, or one of the Boolean types (Boolean, Bool).

QBColor Function

Delphi command
`ColorToRGB Function`

Declaration
```
function ColorToRGB(Color: TColor): Longint;
```

Description
The ColorToRGB function returns the RGB value that Windows uses from a TColor type used by Delphi. If the color represents a system color, the current RGB value for that system color is returned.

Example
The following code converts the color of the current form, Form1, to a Windows RGB value:

```
var
  L : Longint;
begin
  L := ColorToRGB(Form1.Color);
end;
```

QueryDef Object, QueryDefs Collection

Delphi command
`TQuery Component`

Description
See Data Access Page Components

QueryUnload Event

Delphi command
`OnCloseQuery Event`

Applies to
TForm component

Declaration
```
TCloseQueryEvent = procedure(Sender: TObject; var CanClose: Boolean) of
object;
property OnCloseQuery: TCloseQueryEvent;
```

Description
The OnCloseQuery event occurs when an action to close the form takes place--that is, when the Close method is called or when the user chooses Close from the form's System menu. An OnCloseQuery event handler contains a Boolean CanClose variable that determines whether a form is allowed to close. It's default value is True.

You can use an OnCloseQuery event handler to ask users if they are sure they really want the form closed immediately. For example, you can use the handler to display a message box that prompts the user to save a file before closing the form.

The TCloseQueryEvent type points to the method that determines whether a form can be closed. The value of the CanClose parameter determines if the form can close or not.

Example
When the user attempts to close the form in this example, a message dialog appears that asks the user if it is OK to close the form. If the user chooses the OK button, the form closes. If the user chooses Cancel, the form doesn't close.

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if MessageDlg('Close the form?', mtConfirmation,
    [mbOk, mbCancel], 0) = mrCancel then
     CanClose := False;
end;
```

RGB Function

Delphi command
`RGB (API)`

Description
See Win32.hlp

RSet Statement

Delphi command
`Format Function`

Declaration
```
function Format(const Format: string; const Args: array of const): string;
```

Description
This function formats the series of arguments in the open array Args. Formatting is controlled by the Object Pascal format string Format; the results are returned in the function result as a Pascal string.

For information on the format strings, see Format Strings in DELPHI.HLP.

Example
```
Format('%d %d %0:d %d', [10, 20]) = '10 20 10 20'.
```

RTrim Function

Delphi command
`TrimRight Function`

Declaration
```
function TrimRight(const S: string): string;
```

Description
The Trim function trims trailing spaces and control characters from the given string S.

Raise Method

Delphi command
`Raise Reserved Word`

Description
To indicate an error condition in an application, you can raise an exception which involves constructing an instance of that type and calling the reserved word raise

To raise an exception, call the reserved word raise, followed by an instance of an exception object.

When an exception handler actually handles the exception, it finishes by destroying the exception instance, so you never need to do that yourself.

Setting the exception address
Raising an exception sets the ErrorAddr variable in the System unit to the address where the application raised the exception. You can refer to ErrorAddr in your exception handlers, for example, to notify the user of where the error occurred. You can also specify a value for ErrorAddr when you raise an exception.

To specify an error address for an exception, add the reserved word at after the exception instance, followed by and address expression such as an identifier.

Examples
For example, given the following Declaration,

```
type
  EPasswordInvalid = class(Exception);
```

you can raise a "password invalid" exception at any time by calling raise with an instance of EPasswordInvalid, like this:

```
if Password <> CorrectPassword then
  raise EPasswordInvalid.Create('Incorrect password entered');
```

Randomize Statement

Delphi command
Randomize Procedure

Declaration
```
procedure Randomize;
```

Description
The Randomize procedure initializes the built-in random number generator with a random value (obtained from the system clock).

The random number generator should be initialized by making a call to Randomize, or by assigning a value to RandSeed.

Example
```
var
   I: Integer;
 begin
   Randomize;
   for I := 1 to 50 do begin
     { Write to window at random locations }
     Canvas.TextOut(Random(Width), Random(Height), 'Boo!');
   end;
 end;
```

Rate Function

Delphi command

`InterestRate Function`

Declaration

```
function InterestRate(NPeriods: Integer;
  Payment, PresentValue, FutureValue: Extended; PaymentTime: TPaymentTime):
Extended;
```

Description

The InterestRate function calculates the interest rate required in order for an investment of PresentValue, with periodic payments of Payment, to be worth FutureValue within NPeriods compounding periods. If NPeriods represents years, an annual interest rate results; if NPeriods represents months, a monthly interest rate results, and so on. The PaymentTime parameter indicates whether the cash flows occur at the beginning or end of the period (by entering a value of 1or 0, respectively).

ReadFromFile Method

Delphi command
`LoadFromFile Method`

Applies to
TOleContainer component

Declaration
```
procedure LoadFromFile(const FileName: string);
```

Description
Loads an OLE object from the specified file. If there's already an OLE object in the container, it's destroyed and any changes the user made to it are discarded. If OldStreamFormat is False, LoadFromFile only loads files saved with Delphi 2.0's TOleContainer component; if OldStreamFormat is True, LoadFromFile loads files saved with Delphi 1.0's TOLEContainer component as well as Delphi 2.0's TOleContainer.

ReadOnly Property

Delphi command
`ReadOnly Property`

Applies to
TDBCheckBox, TDBComboBox, TDBEdit, TDBGrid, TDBImage, TDBListBox, TDBLookupCombo, TDBLookupComboBox, TDBLookupList, TDBLookupListBox, TDBMemo, TDBRadioGroup, TEdit, TListView, TMaskEdit, TMemo, TRichEdit, TTable, TTreeView components

Declaration
```
property ReadOnly: Boolean;
```

Description
The ReadOnly property determines if the user can change the contents of the control. If ReadOnly is True, the user can't change the contents. If ReadOnly is False, the user can modify the contents. The default value is False.

For data-aware controls, the ReadOnly property determines whether the user can use the data-aware control to change the value of the field of the current record, or if the user can use the control only to display data. If ReadOnly is False, the user can change the field's value as long as the dataset is in edit mode.

When the ReadOnly property of a data grid is True, the user can no longer use the Insert key to insert a new row in the grid, nor can the user append a new row at the end of the data grid with the Down Arrow key.

Example
This code toggles the read-only state of an edit box each time the user double-clicks the form:

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  Edit1.Left := 2;
  Edit1.Top := 2;
  Edit1.ReadOnly := True;
  Edit1.Text := 'Change Me';
  Canvas.TextOut(10, 40, 'Double-click form to toggle read-only state');
end;
procedure TForm1.FormDblClick(Sender: TObject);
begin
  Edit1.ReadOnly := not Edit1.ReadOnly;
end;
```

ReadOnly Property (Data Access)

Delphi command
`ReadOnly Property`

Applies to
TTable components

Declaration
```
property ReadOnly: Boolean;
```

Description
Use the ReadOnly property to prevent users from changing data in the table.

Note Set the Active property to False before changing ReadOnly.

RecordCount Property

Delphi command
`RecordCount Property`

Applies to
TQuery, TStoredProc, TTable components

Declaration
```
property RecordCount: Longint;
```

Description
Run-time and read-only. The RecordCount property specifies the number of records in the dataset. The number of records reported may depend on the server and whether a range limitation is in effect.

Note If the TTable is connected to a dBASE table, RecordCount only reports the total number of records in the table. If you try to limit the number of records in the dataset by calling SetRange, RecordCount will be unaffected; RecordCount reports the total number of records, not the number of records in the limited range.

RecordSelectors Property

Delphi command
`Options Property`

Applies to
TDBGrid component

Declaration
```
property Options: TDBGridOptions;
```

Description
dgRowSelect and dbAlwaysShowSelection control the display of record selection.

When dgRowSelect is True, the user can select whole rows only instead of individual cells. Mutually exclusive with dgAlwaysShowEditor.

When dgAlwaysShowSelection is True, the cell selected in the grid continues to display as selected even if the data grid doesn't have the focus.

Example
This line of code displays column titles, makes the column indicator visible, and permits the user to edit the data displayed in the data grid:

```
procedure TForm1.FormClick(Sender: TObject);
begin
  DBGrid1.Options := [dgIndicator, dgEditing, dgTitles];
end;
```

RecordSource Property

Delphi command
`DataSet Property`

Applies to
TDataSource component

Declaration
```
property DataSet: TDataSet
```

Description
DataSet specifies the dataset component (TTable, TQuery, and TStoredProc) that is providing data to the data source. Usually you set DataSet at design time with the Object Inspector, but you can also set it programmatically. The advantage of this interface approach to connecting data components is that the dataset, data source, and data-aware controls can be connected and disconnected from each other through the TDataSource component. In addition, these components can belong to different forms.

Example
```
DataSource1.DataSet := Table1; {get data from this form's Table1}
DataSource1.DataSet := Form2.Table1; {get data from Form2's Table1}
```

Recordset Object, Recordsets Collection

Delphi command
`DataSet Components`

Applies to
TDataSource component

Declaration
```
property DataSet: TDataSet
```

Description
DataSet specifies the dataset component (TTable, TQuery, and TStoredProc) that is providing data to the data source. Usually you set DataSet at design time with the Object Inspector, but you can also set it programmatically. The advantage of this interface approach to connecting data components is that the dataset, data source, and data-aware controls can be connected and disconnected from each other through the TDataSource component. In addition, these components can belong to different forms.

Recordset Property

Delphi command
See DataSet Property

Applies to
TDataSource component

Declaration
```
property DataSet: TDataSet
```

Description
DataSet specifies the dataset component (TTable, TQuery, and TStoredProc) that is providing data to the data source. Usually you set DataSet at design time with the Object Inspector, but you can also set it programmatically. The advantage of this interface approach to connecting data components is that the dataset, data source, and data-aware controls can be connected and disconnected from each other through the TDataSource component. In addition, these components can belong to different forms.

Example
```
DataSource1.DataSet := Table1; {get data from this form's Table1}
DataSource1.DataSet := Form2.Table1; {get data from Form2's Table1}
```

RecordsetType Property

Delphi command
See RequestLive Property in Delphi.HLP

Refresh Method

Delphi command
`Refresh Method`

Applies to
All controls components

Declaration
```
procedure Refresh;
```

Description
The Refresh method erases whatever image is on the screen and then repaints the entire control. Within the implementation of Refresh, the Invalidate and then the Update methods are called.

Example
The following code refreshes all windowed controls of Form1, then refreshes Form1.

```
var
  I: Integer;
begin
  for I := 0 to Form1.ComponentCount-1 do
    if Form1.Components[i] is TWinControl then
      with Form1.Components[i] as TWinControl do
        Refresh;
  Form1.Refresh;
end;
```

Refresh Method (Data Access)

Delphi command
`Refresh Method`

Applies to
TTable, TQuery, TStoredProc components

Declaration
```
procedure Refresh;
```

Description
The Refresh method rereads all records from the dataset. Use Refresh to be certain that data controls display the latest information from the dataset. If used with a TTable, the corresponding table must have a unique index. If used with a TQuery, the result set must be live, the tables must be local (Paradox or dBASE) and have a unique index.

Relations Object, Relations Collection

Delphi command
See MasterSource, MasterFields, DataSource Properties

Applies to
TTable component

Declaration
```
property MasterSource: TDataSource;
```

Description
When linking a detail table to a master table, use the MasterSource property to specify the TDataSource from which the TTable will get data for the master table.

Example
Suppose you have a master table named Customer that contains a CustNo field, and you also have a detail table named Orders that also has a CustNo field. To display only those records in Orders that have the same CustNo value as the current record in Customer, write this code:

```
Orders.MasterSource := 'CustSource';
Orders.MasterFields := 'CustNo';
```

If you want to display only the records in the detail table that match more than one field value in the master table, specify each field and separate them with a semicolon.

```
Orders.MasterFields := 'CustNo;SaleDate';
```

Rem Statement

Delphi command

```
{}; //
```

Description

Adds comments to code

Examples

```
{ Any text not containing right brace }
(* Any text not containing asterisk/right parenthesis *)
// Any text from a double-slash to the end of the line
```

RemoveItem Method

Delphi command
`Delete Method`

Applies to
TListItems object

Declaration
```
procedure Delete(Index: Integer);
```

Description
The Delete method removes the item (TListItem object) in the list view specified with the Index parameter. The index is zero-based, so the first item has an index of 0, the second item has an index on 1, and so on.

Example
The following code deletes the selected item from Outline1.

```
Outline1.Delete(Outline1.SelectedItem);
```

Reposition Event

Delphi command
`OnDataChange Event`

Applies to
TDataSource component

Declaration
```
TDataChangeEvent = procedure(Sender: TObject; Field: TField) of object;
property OnDataChange: TDataChangeEvent;
```

Description
The OnDataChange occurs when the State property changes from dsInactive, or when a data-aware control notifies the TDataSource that something has changed.

Notification occurs when the following items change because of field modification or scrolling to a new record: field component, record, dataset component, content, and layout. The Field parameter to the method may be nil if more than one of the fields changed simultaneously (as in a move to a different record). Otherwise, Field is the field which changed.

The TDataChangeEvent type points to a method that handles the changing of data in a data source component (TDataSource). The Field parameter is the field in which the data is changing. It is used by the OnDataChange event of the data source.

Requery Method

Delphi command
`Refresh Method`

Applies to
TTable, TQuery, TStoredProc components

Declaration
```
procedure Refresh;
```

Description
The Refresh method rereads all records from the dataset. Use Refresh to be certain that data controls display the latest information from the dataset. If used with a TTable, the corresponding table must have a unique index. If used with a TQuery, the result set must be live, the tables must be local (Paradox or dBASE) and have a unique index.

Required Property

Delphi command
`Required Property`

Applies to
TBCDField, TBooleanField, TBytesField, TCurrencyField, TDateField, TDateTimeField, TFloatField, TIntegerField, TSmallintField, TStringField, TTimeField, TVarBytesField, TWordField components

Declaration
```
property Required: Boolean;
```

Description
Specifies whether a non-nil value for a field is required. The default value is False, meaning a field does not require a value. If a field is created with the Fields Editor, then this property is set based on the underlying table. Set Required to True for fields that must get values (for example, a password or part number), and write an OnValidate event handler for the field. Before a record is posted, exceptions are raised for any required fields that have nil values.

Reset Statement

Delphi command
`CloseFile Procedure`

Declaration
```
procedure CloseFile(var F);
```

Description
Due to naming conflicts, the CloseFile procedure replaces the Borland Pascal Close procedure. Use the CloseFile procedure instead of Close to terminate the association between the file variable and an external disk file.

F is a file variable of any file type opened using Reset, Rewrite, or Append. The external file associated with F is completely updated and then closed, freeing the file handle for reuse.

Example
```
var
  F: TextFile;
begin
  if OpenDialog1.Execute then  { Bring up open file dialog }
  begin
    AssignFile(F, OpenDialog1.FileName);
            { File selected in dialog }
    Reset(F);
    Edit1.Text := IntToStr(FileSize(F);
        { Put file size string in a TEdit control }
    CloseFile(F);   { Close file }
  end;
end;
```

Resize Event

Delphi command
`OnResize Event`

Applies to
TDBNavigator, TForm, TPanel, TScrollBox components

Declaration
```
property OnResize: TNotifyEvent;
```

Description
The OnResize event occurs whenever the form is resized while an application is running. Use the OnResize event handler when you want something to happen in your application when the form is resized.

Restartable Property

Delphi command
See RequestLive Property in Delphi.hlp

ReturnsRecords Property

Delphi command
See ExecSQL Method

Example

```
Query1.Close;
Query1.SQL.Clear;
Query1.SQL.Add('Delete from Country where Name = 'Argentina');
Query1.ExecSQL;
```

Right Function

Delphi command
Copy Function

Declaration
```
function Copy(S: string; Index, Count: Integer): string;
```

Description
The Copy function returns a substring of a string.

S is a string-type expression. Index and Count are integer-type expressions. Copy returns a string containing Count characters starting with at S[Index].

If Index is larger than the length of S, Copy returns an empty string.

If Count specifies more characters than are available, the only the characters from S[Index] to the end of S are returned.

Example
```
 var S: string;
begin
  S := 'ABCDEF';
  S := Copy(S, 2, 3);   { 'BCD' }
end;
```

RmDir Statement

Delphi command
RmDir Procedure

Declaration
```
procedure RmDir(S: string);
```

Description
The RmDir procedure deletes an empty subdirectory.

RmDir removes the subdirectory with the path specified by S. If the path does not exist, is non-empty, or is the currently logged directory, an I/O error occurs.

{$I+} lets you handle run-time errors using exceptions. For more information on handling run-time library exceptions, see Handling RTL Exceptions.

Example
```
uses Dialogs;
begin
   {$I-}
   { Get directory name from TEdit control }
   RmDir(Edit1.Text);
   if IOResult <> 0 then
     MessageDlg('Cannot remove directory', mtWarning, [mbOk], 0)
   else
     MessageDlg('Directory removed', mtInformation, [mbOk], 0);
 end;
```

Rnd Function

Delphi command
Random Function

Declaration
```
function Random [ ( Range: Integer) ];
```

Description
The Random function returns a random number within the range 0 <= X < Range.

If Range is not specified, the result is a real-type random number within the range

0 <= X < 1.

To initialize the random number generator, call Randomize, or assign a value to the RandSeed variable.

Example
```
var
   I: Integer;
 begin
   Randomize;
   for I := 1 to 50 do begin
     { Write to window at random locations }
     Canvas.TextOut(Random(Width), Random(Height), 'Boo!');
   end;
 end;
```

Note Because the implementation of the Random function may change between compiler versions, we do not recommend using Random for encryption or other purposes that require reproducible sequences of pseudo-random numbers.

Rollback Method

Delphi command
`Rollback Method`

Applies to
TDataBase component

Declaration
```
procedure Rollback;
```

Description
The Rollback method undoes any modifications made within the current transaction, those changes made since the last call to StartTransaction. If a transaction is not active when this method is called, an exception is raised.

Example
```
with Database1 do
  begin
    StartTransaction;
  { Update one or more records in tables linked to Database1 }
...
    Rollback;
  end;
```

Rollback Statement

Delphi command
`Rollback Method`

Applies to
TDataBase component

Declaration
```
procedure Rollback;
```

Description
The Rollback method undoes any modifications made within the current transaction, those changes made since the last call to StartTransaction. If a transaction is not active when this method is called, an exception is raised.

Example
```
with Database1 do
  begin
    StartTransaction;
  { Update one or more records in tables linked to Database1 }
...
    Rollback;
  end;
```

Row Property

Delphi command
`Row Property (Protected)`

Applies to
TDrawGrid, TOutline, TStringGrid components

Declaration
```
property Row: Longint;
```

Description
Run-time only. The value of the Row property indicates which row of the control has focus. For outlines, you can use the Row property to determine which item is selected at run time. For the grid components, you can use Row along with the Col property to determine which cell is selected.

Example
This examples uses a string grid with a label above it on a form. When the user clicks a cell in the grid, the location of the cursor is displayed in the caption of the label.

```
procedure TForm1.StringGrid1Click(Sender: TObject);
begin
  Label1.Caption := 'The cursor is in column ' + IntToStr(StringGrid1.Col +
1)
     +  ', row ' + IntToStr(StringGrid1.Row + 1);
end;
```

RowColChange Event

Delphi command
See OnColEnter and OnColExit Events in Delphi.hlp

Example
The following code concatenates an asterisk to the display label of a field when the column is entered.

```
procedure TForm1.DBGrid1ColEnter(Sender: TObject);
begin
  with DBGrid1.SelectedField do
    DisplayLabel := '* ' + DisplayLabel;
end;
```

SELECT Statement (SQL)

Delphi command
`SELECT Statement (SQL)`

Description
SQL Syntax

Example
```
Select * from Customer
```

SLN Function

Delphi command
`SLNDepreciation Function`

Declaration
```
function SLNDepreciation(Cost, Salvage: Extended; Life: Integer): Extended;
```

Description
The SLNDepreciation function calculates the straight-line depreciation allowance for an asset over one period of its life. The function divides the Cost minus the Salvage by the number of years of useful Life of the asset. Cost is the amount initially paid for the asset. Salvage is the value of the asset at the end of its useful life.

To compute accelerated depreciation (allowing higher depreciation values in the first years of the assets life), use the SYDDepreciation function.

SQL Aggregate Functions (SQL Only)

Delphi command

```
SQL Aggregate Functions (SQL Only)
```

Description

SQL Syntax

Example

```
Select Count(CustNo) from Customer
```

SQL Property

Delphi command
`SQL Property`

Applies to
TQuery component

Declaration
```
property SQL: TStrings;
```

Description
The SQL property holds the text of the SQL statement that will be executed when Open or ExecSQL is called. Once a query has been executed by Open, you must call the Close method before you can change the SQL text.

Example
```
Query1.Close;
Query1.SQL.Clear;
Query1.SQL.Add('Delete from Country where Name = 'Argentina');
Query1.ExecSQL;
```

SQL Subqueries

Delphi command
SQL Subqueries

Description
SQL syntax

Example
```
select p.part_no from parts p
where p.quantity in
(select i.quantity from inventory i
where  i.part_no = 'aa9393')
```

SYD Function

Delphi command
`SYDDepreciation Function`

Declaration
```
function SYDDepreciation(Cost, Salvage: Extended; Life, Period: Integer):
Extended;
```

Description
The SYDDepreciation function (for "sum-of-years-digits depreciation") calculates depreciation amounts for an asset using an accelerated depreciation method. This allows for higher depreciation in the earlier years of an asset's life.

Cost is the initial cost of the asset. Salvage is the value of the asset at the end of its life expectancy. Life is the length of the asset's life expectancy. Period is the period for which you want to calculate the depreciation.

SavePicture Statement

Delphi command
`SaveToFile Method`

Applies to
TCustomMemoryStream

Declaration
```
procedure SaveToFile(const FileName: string);
```

Description
The SaveToFile method writes the entire contents of the memory stream to the file specified by FileName. If the specified file already exists, SaveToFile overwrites it. The file becomes a binary copy of the memory stream contents.

SaveSetting Statement

Delphi command
See TRegistry, TRegIniFile Objects in Delphi.hlp

Description
A TRegistry object is a low-level wrapper for the Microsoft Windows95/NT system registry and functions that operate on the registry. The registry is a database that your applications can use to store and retrieve configuration information. Configuration information is stored in a hierarchical tree. Each node in the tree is called a key. Every key can contain subkeys and data values that represent part of the configuration information for an application.

SaveToFile Method

Delphi command
`SaveToFile Method`

Declaration
```
procedure SaveToFile(const FileName: string);
```

Description
The SaveToFile method saves an object to the file specified in FileName. The graphic objects save a graphic to the file, the outline, tree view and string objects save text to the file, and the field components save the contents of the field to the file.

ScaleX, ScaleY Methods

Delphi command
`ScaleBy Method`

Declaration
```
procedure ScaleBy(M, D: Integer);
```

Description
The ScaleBy method scales a control to a percentage of its former size. The M parameter is the multiplier and the D parameter is the divisor. For example, if you want a control to be 75% of its original size, specify the value of M as 75, and the value of D as 100 (75/100). You could also obtain the same results by specifying the value of M as 3, and the value of D as 4 (3/4). Both fractions are equal and result in the control being scaled by the same amount, 75%.

If you want the control to be 33% larger than its previous size, specify the value of M as 133, and the value of D as 100 (133/100). You can also obtain the same results by specifying the value of M as 4, and the value of D as 3 (4/3), as the fraction 133/100 is approximately equal to 4/3.

Example
This example makes your form 50 percent larger:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ScaleBy(150, 100);
end;
```

Screen Object

Delphi command

`TScreen Component`

Description

The TScreen component represents the state of the screen as your application runs. A Screen variable of type TScreen is already declared, ready for you to use as an instance of TScreen.

The screen component lists all forms displayed on the screen in the Forms property array. The number of forms is kept as the value of the FormCount property. You can find out which form currently has the focus by checking the value of the ActiveForm property. Similarly, the control that has the focus is the value of the ActiveControl property.

The screen component lists all data modules that are currently created in the application in the DataModules property. The number of created data modules is kept as the value of the DataModuleCount property.

Scroll Event

Delphi command
`OnScroll Event`

Applies to
TScrollBar component

Declaration
```
TScrollEvent = procedure(Sender: TObject; ScrollCode: TScrollCode; var
ScrollPos: Integer) of object;
property OnScroll: TScrollEvent;
```

Description
The OnScroll event occurs whenever the user uses the scroll bar control. Use the OnScroll event handler if you want something to happen when the user uses the scroll bar control. Within the handler, write the code that responds to the user using the scroll bar.

Refer to Delphi.hlp for scollbar parameters.

Example
The following code repositions the thumb tab position by varying amounts. If Page Up was pressed, the box moves up only one. If Page Down was pressed, the box moves down 10. This shows how you can use the OnScroll event handler to move the thumb tab by different increments than specified by the LargeChange and SmallChange properties.

```
procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  if ScrollCode = scPageUp then ScrollPos := ScrollPos - 1
    else if ScrollCode = scPageDown then ScrollPos := ScrollPos + 10;
  Label1.Caption := IntToStr(ScrollPos);
end;
```

Scroll Method

Delphi command
`ScrollBy Method`

Applies to
All controls; TForm component

Declaration
```
procedure ScrollBy(DeltaX, DeltaY: Integer);
```

Description
The ScrollBy method scrolls the contents of a form or windowed control. You will seldom need to call the ScrollBy method unless you want to write your own scrolling logic rather than use a scroll bar.

The DeltaX parameter is the change in pixels along the X axis. A positive DeltaX value scrolls the contents to the right; a negative value scrolls the contents to the left. The DeltaY parameter is the change in pixels along the Y axis. A positive DeltaY value scrolls the contents down; a negative value scrolls the contents up.

Example
This example uses a timer and several controls of your choosing on a form. When the application runs, the controls on the form appear to slide down and off to the right. This is because the contents of the form are scrolling both down and to the right by one pixel each time a timer event occurs:

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  Timer1.Interval := 1;
end;
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  ScrollBy(1,1);
end;
```

ScrollBars Property

Delphi command
`Scrollbars Property`

Applies to
TDBMemo, TDrawGrid, TMemo, TRichEdit, TStringGrid components

Declaration
```
property ScrollBars: TScrollStyle;
```

Description
The ScrollBars property controls whether a memo control, a rich edit control, or a grid control has any scroll bars. You can set ScrollBars to any of the following values:

Value	Meaning
ssNone	No scroll bar
ssHorizontal	Puts a scroll bar on the right edge
ssVertical	Puts a scroll bar on the bottom edge
ssBoth	Puts a scroll bar on both the right and bottom edges

By default, grids have both vertical and horizontal scroll bars, while memo controls have none.

Example
The following example adds a scroll bar to the bottom of memo control Memo1:

```
Memo1.ScrollBars := scHorizontal;
```

Second Function

Delphi command
FormatDateTime Function

Declaration
```
function FormatDateTime(const Format: string; DateTime: TDateTime): string;
```

Description
FormatDateTime formats the date-and-time value given by DateTime using the format given by Format. S displays the second without a leading zero (0-59). ss displays the second with a leading zero (00-59).

Example
The following example assigns 'The meeting is on Wednesday, February 15, 1995 at 10:30 AM' to the string variable S.

```
S := FormatDateTime('"The meeting is on" dddd, mmmm d, yyyy, ' +
  '"at" hh:mm AM/PM', StrToDateTime('2/15/95 10:30am'));
```

Seek Function

Delphi command
`Seek Procedure`

Declaration
```
procedure Seek(var F; N: Longint);
```

Description
The Seek procedure moves the current position of a file to a specified component. You can use Seek only on open typed or untyped files.

In the above syntax, F is a typed or untyped file variable, and N is an expression of type Longint.

The current file position of F moves to component number N. The number of the first component of a file is 0.

To expand a file, you can seek one component beyond the last component; that is, the statement Seek(F, FileSize(F)) moves the current file position to the end of the file.

{$I+} lets you handle run-time errors using exceptions. For more information on handling run-time library exceptions, see Handling RTL Exceptions.

Example
```
uses Dialogs;
var
   f: file of Byte;
   size : Longint;
   S: string;
   y: integer;
 begin
   if OpenDialog1.Execute then begin
     AssignFile(f, OpenDialog1.FileName);
     Reset(f);
     size := FileSize(f);
     S := 'File size in bytes: ' + IntToStr(size);
     y := 10;
     Canvas.TextOut(5, y, S);
     y := y + Canvas.TextHeight(S) + 5;
     S := 'Seeking halfway into file...';
     Canvas.TextOut(5, y, S);

     y := y + Canvas.TextHeight(S) + 5;
     Seek(f,size div 2);
     S := 'Position is now ' + IntToStr(FilePos(f));
     Canvas.TextOut(5, y, S);
     CloseFile(f);
   end;
 end;
```

Seek Method

Delphi command
`FindKey, GoToKey Methods`

Applies to
TTable component

Declaration
```
function FindKey(const KeyValues: array of const): Boolean;
```

Description
The FindKey method searches the database table to find a record whose index fields match those passed in KeyValues. FindKey takes a comma-delimited array of values as its argument, where each value corresponds to a index column in the underlying table. The values can be literals, variables, null, or nil. If the number of values supplied is less than the number of columns in the database table, then the remaining values are assumed to be null. FindKey will search for values specified in the array in the current index.

Example
This example will search for CustNo = '1234':

```
if Table1.FindKey(['1234']) then
   ShowMessage('Customer Found');
```

Seek Statement

Delphi command
Seek Procedure

Declaration
```
procedure Seek(var F; N: Longint);
```

Description
The Seek procedure moves the current position of a file to a specified component. You can use Seek only on open typed or untyped files.

In the above syntax, F is a typed or untyped file variable, and N is an expression of type Longint.

The current file position of F moves to component number N. The number of the first component of a file is 0.

To expand a file, you can seek one component beyond the last component; that is, the statement Seek(F, FileSize(F)) moves the current file position to the end of the file.

{$I+} lets you handle run-time errors using exceptions. For more information on handling run-time library exceptions, see Handling RTL Exceptions.

Example
```
uses Dialogs;
var
   f: file of Byte;
   size : Longint;
   S: string;
   y: integer;
 begin
   if OpenDialog1.Execute then begin
     AssignFile(f, OpenDialog1.FileName);
     Reset(f);
     size := FileSize(f);
     S := 'File size in bytes: ' + IntToStr(size);
     y := 10;
     Canvas.TextOut(5, y, S);
     y := y + Canvas.TextHeight(S) + 5;
     S := 'Seeking halfway into file...';
     Canvas.TextOut(5, y, S);

     y := y + Canvas.TextHeight(S) + 5;
     Seek(f,size div 2);
     S := 'Position is now ' + IntToStr(FilePos(f));
     Canvas.TextOut(5, y, S);
     CloseFile(f);
   end;
 end;
```

SelBookmarks Collection

Delphi command

```
TBookMarkList Object
```

SelBookmarks Property

Delphi command

SelectedRows Property

SelChange Event

Delphi command
`OnColEnter, OnColExit Events`

Applies to
TDBGrid component

Declaration
```
property OnColEnter: TNotifyEvent;
property OnColExit: TNotifyEvent
```

Description
The OnColEnter event occurs when the user clicks a cell in a column or moves to a column with the Tab key within the data grid. Use the OnColEnter event to specify any processing you want to occur as soon as a column is entered.

Example
The following code concatenates an asterisk to the display label of a field when the column is entered.

```
procedure TForm1.DBGrid1ColEnter(Sender: TObject);
begin
  with DBGrid1.SelectedField do
    DisplayLabel := '* ' + DisplayLabel;
end;
```

SelCount Property

Delphi command
`SelCount Property`

Applies to
TDBListBox, TDirectoryListBox, TFileListBox, TListBox components

Declaration
```
property SelCount: Integer;
```

Description
Run-time and read-only. The SelCount property reports the number of items that are selected in a list box when the value of the MultiSelect property is True. When MultiSelect property is False, only one item can be selected. If no items are selected, the value of SelCount is 0.

Example
This example uses a list box, a label, and a button on a form. Enter several strings in the list box as the value of the Items property. When the user selects items in the list box and clicks the button, the number of items selected in the list box is displayed in the caption of the label:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := IntToStr(ListBox1.SelCount) + ' items are selected';
end;
```

SelEndCol, SelStartCol, SelEndRow, SelStartRow Properties

Delphi command
`Selection Property`

Applies to
TDrawGrid, TStringGrid components

Declaration
```
property Selection: TGridRect;
```

Description
The Selection property contains the column and row coordinates of the cell or cells selected in the grid.

Example
The following code selects the rectangle containing rows 1 and 2, and columns 3 and 4.

```
StringGrid1.Selection := Rect(3,1,2,4);
```

SelLength, SelStart, SelText Properties

Delphi command
`SelLength, SelStart, SelText Properties`

Applies to
TComboBox, TDBComboBox, TDBEdit, TDBMemo, TDriveComboBox, TEdit, TFilterComboBox, TMaskEdit, TMemo components

Declaration
```
property SelLength: Integer;
property SelStart: Integer;
property SelText: string;
```

Description
The SelLength property returns the length (in characters) of the control's selected text. By using SelLength along with the SelStart property, you specify which part of the text in the control is selected. You can change the number of selected characters by changing the value of SelLength. When the SelStart value changes, the SelLength value changes accordingly.

The edit box or memo must be the active control when you change the value of SelLength, or nothing appears to happen.

The SelStart property returns the starting position of the selected part of the control's text, with the first character in the text having a value of 0. The SelText property contains the selected part of the control's text.

Example
This example uses an edit box and a label on a form. When the user selects text in the edit box, the number of selected characters is reported in the caption of the label:

```
procedure TForm1.Edit1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  Label1.Caption := 'Selected length = ' +
    IntToStr(Edit1.SelLength);
end;
```

Select Case Statement

Delphi command

`Case Reserved Word`

Description

Case statements are used to branch code depending on the results or values the code encounters.

A case statement consists of an expression (the selector) and a list of statements, each prefixed with one or more constants (called case constants) or with the reserved word else. The selector must be of an ordinal type, so string types are invalid selector types.

All case constants must be unique and of an ordinal type compatible with the selector type.

When the program enters a case statement, it evaluates each expression until a match is found. The program then performs the actions associated with that expression. If no match is found program defaults to the else statement. If there is no else part, execution continues with the next statement following the case statement.

Example

```
case Ch of
  'A'..'Z', 'a'..'z':  WriteLn('Letter');
  '0'..'9':            WriteLn('Digit');
  '+', '-', '*', '/':  WriteLn('Operator');
else
  WriteLn('Special character');
end;
```

Selected Property

Delphi command
`Selected Property`

Applies to
TDBListBox, TDirectoryListBox, TFileListBox components

Declaration
```
property Selected[X: Integer]: Boolean;
```

Description
The Selected property determines whether a particular item is selected in the list box. The X parameter is the item referenced by its position in the list box, with the first item having an X value of 0. If the specified item is selected in the list box, the value of the Selected property is True. If the specified item is not selected, Selected is False.

If you want the user to be able to select more than one item in the list box, use the MultiSelect property

Example
This example uses a list box on a form. When the form is first created, 3 items are added to the list box. When the user selects an item in the list box, the list box color changes to reflect the item selected:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  ListBox1.Items.Add('Blue');
  ListBox1.Items.Add('Yellow');
  ListBox1.Items.Add('Red');
end;
procedure TForm1.ListBox1Click(Sender: TObject);
begin
  if ListBox1.Selected[0] then
    ListBox1.Color := clBlue;
  if ListBox1.Selected[1] then
    ListBox1.Color := clYellow;
  if ListBox1.Selected[2] then

    ListBox1.Color := clRed;
end;
```

SelectedItem Property

Delphi command
`ItemIndex Property`

Applies to
TComboBox, TDBComboBox, TDBRadioGroup, TDirectoryListBox, TDriveComboBox, TFileListBox, TFilterComboBox, TListBox, TRadioGroup components

Declaration
```
property ItemIndex: Integer;
```

Description
Run-time only except for the TRadioGroup component. The value of the ItemIndex property is the ordinal number of the selected item in the control's item list. If no item is selected, the value is -1, which is the default value unless MultiSelect is True. To select an item at run time, set the value of ItemIndex to the index of the item in the list you want selected, with 0 being the first item in the list.

For list boxes and combo boxes, if the value of the MultiSelect property is True and the user selects more than one item in the list box or combo box, the ItemIndex value is the index of the selected item that has focus. If MultiSelect is True, ItemIndex defaults to 0.

Example
This example uses a drive combo box on a form. When the user selects a drive in the combo box, the index value of the selected item appears in the caption of the label:

```
procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  Label1.Caption := 'Index value ' + IntToStr(DriveComboBox1.ItemIndex);
end;
```

Set Statement

Delphi command

:=

SetAttr Statement

Delphi command
`FileSetAttr Function`

Declaration
```
function FileSetAttr(const FileName: string; Attr: Integer): Integer;
```

Description
The FileSetAttr function sets the file attributes of the file given by FileName to the value given by Attr.

SetData Method

Delphi command
`Assign Method`

Declaration
```
procedure Assign(Source: TPersistent);
```

Description
Assign copies data from one field to another. Both fields must be valid and have the same DataType and Size, and the DataSize of Source must be 255 bytes or less.

Example
The following code copies the bitmap of a speed button named SpeedButton1 to the Clipboard:

```
Clipboard.Assign(SpeedButton1.Glyph);
```

SetDataAccessOption Statement

Delphi command
`NetFileDir Property`

Applies to
TSession component

Declaration
```
property NetFileDir: string;
```

Description
The NetFileDir property specifies the directory that contains the BDE network control file, PDOXUSRS.NET. This property enables multiple users to share Paradox tables on network drives. NetFileDir overrides the specification defined for the Paradox driver in the BDE Configuration Utility.

All applications that need to share the same Paradox database must specify the same directory, and all must have read/write/create rights for the directory.

SetDefaultWorkspace Statement

Delphi command
`AddPassword Method`

Applies to
TSession component

Declaration
```
procedure AddPassword(const Password: string);
```

Description
The AddPassword method is used to add a new password to the current TSession component for use with Paradox tables. When an application opens a Paradox table that requires a password, the user will be prompted to enter a password unless the Session has a valid password for the table.

Example
```
Session.AddPassword('ASecret');
```

SetFocus Method

Delphi command
`SetFocus Method`

Applies to
All controls; TForm component

Declaration
```
procedure SetFocus;
```

Description
The SetFocus method gives the input focus to the control. If the control is a form, the form calls the SetFocus method of its active control.

Example
When the user clicks the button on this form, the list box becomes the active control and receives the input focus:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.SetFocus;
end;
```

SetText Method

Delphi command
`AsText Property; Assign Method`

Applies to
TClipboard object

Declaration
```
property AsText: string;
```

Description
Run-time only. The AsText property returns the current contents of the Clipboard as a string. The Clipboard must contain a string or an exception occurs.

You can also use the AsText property to place a copy of a string on the Clipboard. Assign a string as the value of AsText.

The string value of the AsText property is limited to 255 characters. If you need to set and retrieve more than 255 characters, use the SetTextBuf and GetTextBuf Clipboard methods.

If the Clipboard contains a string, this expression is True:

Clipboard.HasFormat(CF_TEXT)

Example
The following code retrieves the contents of the Clipboard as a string and displays the value in a label:

```
begin
  Label1.Caption := Clipboard.AsText;
end;
```

Shape Control

Delphi command
`Shape Component`

Description
See Additional Page Components

Shape Property

Delphi command
`Shape Property`

Applies to
TShape components

Declaration
```
property Shape: TShapeType;
```

Description
The Shape property determines how a TShape component appears on a form. These are the possible values and their meanings:

Value	Meaning
stEllipse	The shape is an ellipse.
stRectangle	The shape is a rectangle.
stRoundRect	The shape is a rectangle with rounded corners.
stRoundSquare	The shape is a square with rounded corners.
stSquare	The shape is a square.
stCircle	The shape is a circle.

Example
This example uses a shape component on a form. When the user clicks the shape, it becomes a ball with red stripes:

```
procedure TForm1.Shape1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
with Shape1 do
  begin
    Shape := stCircle;
    Brush.Color := clRed;
    Brush.Style := bsVertical;
  end;
end;
```

Shell Function

Delphi command
`WinExec (API)`

Description
See Win32.hlp

Shortcut Property

Delphi command
`ShortCut Property`

Applies to
TMenuItem component

Declaration
```
property ShortCut: TShortCut;
```

Description
The ShortCut property determines the key strokes users can use to access a menu item. The key combination the user can use appears to the right of the menu item in the menu. To see an example of menu shortcuts, pull down the Delphi Edit menu and note the menu shortcuts on the right side of some of the editing commands.

Example
This code creates a shortcut, Ctrl+C, at run time and assigns it to the Close command on a File menu.

```
begin
  CloseCommand.ShortCut := ShortCut(Word('C'), [ssCtrl]);
end;
```

Show Method

Delphi command
`Show Method`

Applies to
All controls; TForm component

Declaration

```
procedure Show;
```

Description
The Show method makes a form or control visible by setting its Visible property to True. If the Show method of a form is called and the form is somehow obscured, Show tries to make the form visible by bringing it to the front with the BringToFront method.

If the control is in an auto-scrolling control, such as a TForm or TScrollBox, Show will make sure that the form is also scrolled into view. If the control is contained in a pages control such as the TPageControl, the page the control is contained on will be brought to the front.

Example
This code puts away the current form and displays another:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Form1.Hide;
  Unit2.Form2.Show;
end;
```

ShowColor Method

Delphi command
`Execute Method`

Applies to
TColorDialog, TFontDialog, TOpenDialog, TPrintDialog, TPrinterSetupDialog, TSaveDialog components

Declaration
```
function Execute: Boolean;
```

Description
The Execute method displays the dialog box in the application and returns True when the user closes the dialog box by choosing the OK button. If the user chooses Cancel or closes the dialog box by using the system menu, Execute returns False.

ShowFont Method

Delphi command
`Execute Method`

Applies to
TColorDialog, TFontDialog, TOpenDialog, TPrintDialog, TPrinterSetupDialog, TSaveDialog components

Declaration
```
function Execute: Boolean;
```

Description
The Execute method displays the dialog box in the application and returns True when the user closes the dialog box by choosing the OK button. If the user chooses Cancel or closes the dialog box by using the system menu, Execute returns False.

ShowHelp Method

Delphi command
`HelpCommand Method`

Applies to
TApplication component

Declaration
```
function HelpCommand(Command: Word; Data: Longint): Boolean;
```

Description
The HelpCommand method gives you quick access to any of the Help commands in the WinHelp API (application programming interface). For information about the commands you can call and the data passed to them, see the WinHelp topic in the Help system.

Example
This example uses a bitmap button on a form. When the user clicks the button, the Help contents screen of the specified Help file appears.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  Application.HelpFile := 'MYHELP.HLP';
  Application.HelpCommand(HELP_CONTENTS, 0);
end;
```

ShowOpen Method

Delphi command
`Execute Method`

Applies to
TColorDialog, TFontDialog, TOpenDialog, TPrintDialog, TPrinterSetupDialog, TSaveDialog components

Declaration
```
function Execute: Boolean;
```

Description
The Execute method displays the dialog box in the application and returns True when the user closes the dialog box by choosing the OK button. If the user chooses Cancel or closes the dialog box by using the system menu, Execute returns False.

Example
This example uses a main menu component, a memo, an Open dialog box, and a Save dialog box on a form. To use it, you need to create a File menu that includes an Open command. This code is an event handler for the OnClick event of the Open command on the File menu. If the user has selected a filename by choosing the Open dialog box's OK button, the code sets the Save dialog box Filename property to the same filename, and displays the selected filename as the caption of the form.

```
procedure TForm1.Open1Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
  begin
    Memo1.Lines.LoadfromFile(OpenDialog1.FileName);
    SaveDialog1.Filename := OpenDialog1.FileName;
    Caption := OpenDialog1.FileName;
  end;
end;
```

ShowPrinter Method

Delphi command
`Execute Method`

Applies to
TColorDialog, TFontDialog, TOpenDialog, TPrintDialog, TPrinterSetupDialog, TSaveDialog components

Declaration
```
function Execute: Boolean;
```

Description
The Execute method displays the dialog box in the application and returns True when the user closes the dialog box by choosing the OK button. If the user chooses Cancel or closes the dialog box by using the system menu, Execute returns False.

ShowSave Method

Delphi command
`Execute Method`

Applies to
TColorDialog, TFontDialog, TOpenDialog, TPrintDialog, TPrinterSetupDialog, TSaveDialog components

Declaration
```
function Execute: Boolean;
```

Description
The Execute method displays the dialog box in the application and returns True when the user closes the dialog box by choosing the OK button. If the user chooses Cancel or closes the dialog box by using the system menu, Execute returns False.

Sin Function

Delphi command
Sin Function

Declaration
```
function Sin(X: Extended): Extended;
```

Description
The Sin function returns the sine of the argument.

X is a real-type expression. Sin returns the sine of the angle X in radians.

Example
```
var
  R: Extended;
  S: string;
begin
  R := Sin(Pi);
  Str(R:5:3, S);
  Canvas.TextOut(10, 10, 'The Sin of Pi is ' + S);
end;
```

Size Property (Data Access)

Delphi command
`Size Property`

Applies to
TBCDField, TBlobField, TBytesField, TGraphicField, TIntegerField, TMemoField, TStringField, TTimeField, TVarBytesField components

Declaration
```
property Size: Integer;
```

Description
For a TStringField, Size is the number of bytes reserved for the field in the dataset. For a TBCDField, it is the number of digits following the decimal point. For a TBlobField, TBytesField, TVarBytesField, TMemoField, or TGraphicField, it is the size of the field as stored in the table.

Size Property (Font)

Delphi command
`Size Property`

Applies to
TFont component

Declaration
```
property Size: Integer;
```

Description
The Size property is the size of the font in points. It is the height of the font plus the font's excluding internal leading. If you are concerned with the height of the font on the screen--the number of pixels the font needs--use the Height property instead. If you want to specify a font's height using pixels, use the Size property.

Delphi calculates Size using this formula:

Font.Size = -Font.Height * Font.PixelsPerInch / 72

Therefore, whenever you enter a point size in the Size property, you'll notice the Height property changes to a negative value. Conversely, if you enter a positive Height value, the Size property value changes to a negative value.

Example
This examples uses a button on a form. When the user clicks the button, the size of the font used by the button changes to 24 points.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Font.Size := 24;
end;
```

SizeMode Property

Delphi command
`SizeMode Property`

Applies to
TOleContainer component

Declaration
```
property SizeMode: TSizeMode;
```

Description
Determines how the OLE object will be sized within the container. See TSizeMode for possible values.

SmallChange Property

Delphi command
`SmallChange Property`

Applies to
TScrollBar component

Declaration
```
property SmallChange: TScrollBarInc;
```

Description
The SmallChange property determines how far the thumb tab moves when the user clicks the arrows at the end of the scroll bar to scroll or uses the arrow keys on the keyboard. The default value is 1.

For example, if SmallChange is 1000, each time the user clicks an arrow on the scroll bar, the thumb tab moves 1000 positions. The number of positions is determined by the difference between the Max property value and the Min property value. If the Max property is 30000 and the Min property is 0, the user would need to click an arrow on the scroll bar 30 times to move the thumb tab from one end of the scroll bar to the other.

Example
This code determines that when the user clicks an arrow on the scroll bar, the thumb tab moves 10 positions on the scroll bar:

```
ScrollBar1.SmallChange := 10;
```

Snapshot object

Delphi command

`TQuery Component`

Description

See Data Access Page Components

Snapshot-Type Reordset

Delphi command
`TQuery Component`

Description
See Data Access Page Components

Sorted Property

Delphi command
`Sorted Property`

Applies to
TComboBox, TDBComboBox, TDBListBox, TListBox components

Declaration
```
property Sorted: Boolean;
```

Description
The Sorted property indicates whether the items in a list box or combo box are arranged alphabetically. To sort the items, set the Sorted value to True. If Sorted is False, the items are unsorted.

If you add or insert items when Sorted is True, Delphi automatically places them in alphabetical order.

Example
This example uses an edit box, a list box, and two buttons on a form. The buttons are named Add and Sort. When the user clicks the Add button, the text in the edit box is added to the list in the list box. When the user clicks the Sort button, the list in the list box is sorted and remains sorted, even if additional strings are added:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  Listbox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;
procedure TForm1.AddClick(Sender: TObject);
begin
  ListBox1.Items.Add(Edit1.Text);
end;
procedure TForm1.SortClick(Sender: TObject);
begin
  ListBox1.Sorted := True;
end;
```

Souce Property (Data Access)

Delphi command

Sender Parameter of Exception Handler

Example

The following code defines the default exception handling of the application, assuming AppException is declared a method of TForm1.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnException := AppException;
end;
procedure TForm1.AppException(Sender: TObject; E: Exception);
begin
  Application.ShowException(E);
end;
```

Source Property

Delphi command

`Sender Parameter of Exception Handler`

Example

The following code defines the default exception handling of the application, assuming AppException is declared a method of TForm1.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnException := AppException;
end;
procedure TForm1.AppException(Sender: TObject; E: Exception);
begin
  Application.ShowException(E);
end;
```

SourceDoc Property

Delphi command
`SourceDoc Property`

Applies to
TOleContainer component

Declaration
```
property SourceDoc: string;
```

Description
Runtime and readonly. Returns the name of the source document for a linked OLE object. An OLE object must already be loaded in the container before accessing the SourceDoc property. If the OLE object isn't linked, SourceDoc returns an empty string.

SourceField, SourceTable Properties

Delphi command

```
FieldName, DataSet.TableName Properties
```

Declaration

```
property FieldName: string;
property DataSet: TDataSet;
```

Description

FieldName is the name of the physical column in the underlying dataset to which a TField component is bound. FieldName is used as a default column heading by the data grid when the DisplayLabel property is null. For calculated fields, supply a FieldName when you define the field. For non-calculated fields, an exception occurs if a FieldName is not a column name in the physical table.

DataSet identifies the dataset to which a TField component belongs.

SourceTableName Property

Delphi command
`TableName Property`

Applies to
TTable component

Declaration
```
property TableName: TFileName;
```

Description
The TableName property is the name of the database table to which the TTable is linked.

Space Function

Delphi command
`Format Function`

Declaration
```
function Format(const Format: string; const Args: array of const): string;
```

Description
This function formats the series of arguments in the open array Args. Formatting is controlled by the Object Pascal format string Format; the results are returned in the function result as a Pascal string.

For information on the format strings, see Format Strings in Delphi.hlp

Sqr Function

Delphi command
Sqrt Function

Declaration
```
function Sqrt(X: Extended): Extended;
```

Description
The Sqrt function returns the square root of the argument.

X is a floating-point expression. The result is the square root of X.

Example
```
var
  S, Temp: string;
begin
   Str(Sqr(5.0):3:1, Temp);
   S := '5 squared is ' + Temp + #13#10;
   Str(Sqrt(2.0):5:4, Temp);
   S := S + 'The square root of 2 is ' + Temp;
   MessageDlg(S, mtInformation, [mbOk], 0);
end;
```

Stop Statement

Delphi command
`DebugBreak (API)`

Description
See Win32.hlp

Str Function

Delphi command
`Str Procedure`

Declaration
```
procedure Str(X [: Width [: Decimals ]]; var S);
```

Description
The Str procedure converts X to a string representation according to the Width and Decimals formatting parameters. The effect is like a call to Write except the resulting string is stored in S instead of being written to a text file.

X is an integer-type or real-type expression. Width and Decimals are integer-type expressions. S is a string-type variable or a zero-based character array variable if extended syntax is enabled.

Example
```
function MakeItAString(I: Longint): string;
{ Convert any integer type to a string }
var
  S: string[11];
begin
  Str(I, S);
  MakeItAString:= S;
end;
begin
  Canvas.TextOut(10, 10, MakeItAString(-5322));
end;
```

StrComp Function

Delphi command

`StrComp Function`

Declaration

```
function StrComp(Str1, Str2 : PChar): Integer;
```

Description

The StrComp function compares Str1 to Str2.

Return value	Condition
<0	if Str1< Str2
=0	if Str1= Str2
>0	if Str1 > Str2

Example

```
uses SysUtils;
const
  S1: PChar = 'Wacky';
  S2: PChar = 'Code';
var
  C: Integer;
  Result: string;
begin
  C := StrComp(S1, S2);
  if C < 0 then Result := ' is less than ' else
    if C > 0 then Result := ' is greater than ' else
      Result := ' is equal to ';
  Canvas.TextOut(10, 10, StrPas(S1) + Result + StrPas(S2));
end;
```

StrConv Function

Delphi command

`StringToWideChar, WideCharToString Functions`

Declaration

```
function StringToWideChar(const Source: string; Dest: PWideChar; DestSize:
Integer): PWideChar;
```

function WideCharLenToString(Source: PWideChar; SourceLen: Integer): string;

Description

The StringToWideChar function converts the string given by Source from ANSI to UNICODE and stores the result in the buffer given by Dest and DestSize. Following the call, the Dest buffer contains at most DestSize - 1 characters from the source string, terminated by a NULL wide character. The function result value is Dest.

Stretch Property

Delphi command
`Stretch Property`

Applies to
TImage, TDBImage components

Declaration
```
property Stretch: Boolean;
```

Description
Setting the Stretch property to True permits bitmaps and metafiles to assume the size and shape of the image control. When the image control is resized, the image resizes also. Stretch will also resize an image to fit into a smaller TImage control. The Stretch property has no affect on icons.

If you prefer to have the image control resize to fit the native size of the image, set the AutoSize property to True.

Example
This example uses an image component on a form. When the form is created, the specified image is loaded and stretched to fit the boundaries of the image component.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Stretch := True;
  Image1.Picture.LoadFromFile('C:\DELPHI\DEMOS\GRAPHEX\PASTE.BMP');
end;
```

StrikeThrough Property

Delphi command

`Font.Style Property`

Declaration

```
property Style: TFontStyles;
```

Description

The Style property determines whether the font is normal, italic, underlined, bold, and so on. With fsStrikeout, the font is displayed with a horizontal line through it.

Example

The following code boldfaces the font used in the memo..

```
Memo1.Font.Style := [fsBold];
```

String Function

Delphi command
`FillChar Procedure`

Declaration
```
procedure FillChar(var X; Count: Integer; value);
```

Description
The FillChar procedure fills Count number of contiguous bytes with a specified value (can be type Byte or Char).

This function does not perform any range checking.

Example
```
 var
  S: array[0..79] of char;
begin
  { Set to all spaces }
  FillChar(S, SizeOf(S), ' ');
end;
```

Style Property

Delphi command

```
Style Property
```

Declaration

```
TComboBoxStyle = (csDropDown, csSimple, csDropDownList, csOwnerDrawFixed,
csOwnerDrawVariable);
```

Description

The TComboBoxStyle type defines the styles of combo boxes. TComboBoxStyle is the type of the combo box control's Style property.

Example

This example uses a combo box and a check box on a form. If the user checks the check box, the combo box becomes a drop-down list. When the user unchecks the check box, the combo box becomes a simple combo box:

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  if CheckBox1.Checked then
    ComboBox1.Style := csDropDownList
  else
    ComboBox1.Style := csSimple;
end;
```

Sub Statement

Delphi command

Procedure Reserved Word

Description

Procedures let you nest additional blocks in the main program block. Each procedure Declaration has a heading followed by a block of statements.

The procedure heading specifies the identifier for the procedure and the formal parameters (if any).

A procedure is activated by a procedure statement, which states the procedure's identifiers and actual parameters, if any.

The procedure heading is followed by:

A Declaration part that declares local objects

The statements between begin and end, which specify what is to be executed when the procedure is called.

Example

```
{ Procedure Declaration }
procedure NumString(N: Integer; var S: string);
var
  V: Integer;
begin
  V := Abs(N);
  S := '';
  repeat
    S := Chr(N mod 10 + Ord('0')) + S;
    N := N div 10;
  until N = 0;
  if N < 0 then
    S := '-' + S;
end;
```

Sum Function (SQL)

Delphi command
`SUM Function (SQL)`

Description:
SQL Syntax

Example
Select Sum(Total) from Orders

System Property

Delphi command
`FileType Property`

Declaration
```
property FileType: TFileType;
```

Description
The FileType property determines which files are displayed in the file list box based on the attributes of the files. When ftSystem is True, the list box can display files with the system attribute.

Example
This example uses a file list box on a form. When the application runs, only read-only files, directories, volume IDs, and files with no attributes appear in the list box.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileType := [ftReadOnly, ftDirectory, ftVolumeID, ftNormal];
end;
```

TabIndex Property

Delphi command
`TabOrder Property`

Declaration
```
property TabOrder: TTabOrder;
```

Description
The TabOrder property indicates the position of the control in its parent's tab order, the order in which controls receive the focus when the user presses the Tab key.

Initially, the tab order is always the order in which the components were added to the form, but you can change this by changing the TabOrder property. The value of the TabOrder property is unique for each component on the form. The first component added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. These values determine where a control is in the tab order.

Example
This example ensures that the check box on the form is the first in the tab order, and therefore, the active control whenever the form appears, no matter how many other controls are on the form:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 CheckBox1.TabStop := True;
 CheckBox1.TabOrder := 0;
end;
```

TabStop Property

Delphi command
`TabStop Property`

Declaration
```
property TabStop: Boolean;
```

Description
The TabStop property determines if the user can tab to a control. If TabStop is True, the control is in the tab order. If TabStop is False, the control is not in the tab order; therefore, the user can't press the Tab key to move to the control. The default value for most controls is True.

Example
This code removes ListBox1 from the tab order, so that the user can't use the Tab key to get to the list box:

```
ListBox1.TabStop := False;
```

Table Object

Delphi command
`TTable Component`

Description
See Data Access Page Components

Table Property

Delphi command

```
Mastersource, MasterFields (TTable); DataSource (TQuery)
```

Declaration

```
property MasterSource: TDataSource;
```

Description

When linking a detail table to a master table, use the MasterSource property to specify the TDataSource from which the TTable will get data for the master table.

Example

Suppose you have a master table named Customer that contains a CustNo field, and you also have a detail table named Orders that also has a CustNo field. To display only those records in Orders that have the same CustNo value as the current record in Customer, write this code:

```
Orders.MasterSource := 'CustSource';
Orders.MasterFields := 'CustNo';
```

If you want to display only the records in the detail table that match more than one field value in the master table, specify each field and separate them with a semicolon.

```
Orders.MasterFields := 'CustNo;SaleDate';
```

Table-type Recordset

Delphi command
`TTable Component`

Description
See Data Access Page Components

Tag Property

Delphi command
`Tag Property`

Declaration
```
property Tag: Longint;
```

Description
The Tag property is available to store an integer value as part of a component. While the Tag property has no meaning to Delphi, your application can use the property to store a value for its special needs.

Example
The following code assumes that the OnClick event handlers of more than one button point to the TForm1.Button1Click method. When a button is clicked, the procedure checks to see if the value of the Tag of the clicked button is 42. If so, the caption of that button is changed.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (Sender as TButton).Tag = 42 then
    (Sender as TButton).Caption := 'A-Ha!';
end;
```

Tan Function

Delphi command
`Tan Function`

Declaration
```
function Tan(X: Extended): Extended;
```

Description
The Tan function returns the tangent of X. Tan(X) = Sin(X) / Cos(X).

TaskVisible Property

Delphi command
`Use ShowWindow.`

Declaration
See Win32.HLP

Terminate Event

Delphi command
OnDestroy Event

Declaration
```
property OnDestroy: TNotifyEvent;
```

Description
The OnDestroy event occurs when a form is about to be destroyed. A form is destroyed by the Destroy, Free, or Release methods, or when the main form of the application is closed.

Example
The following code explicitly allocates memory for a pointer in the OnCreate event of Form1, then releases the memory in the OnDestroy event. Assume that MyPtr is a Pointer type field of TForm1.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  New(MyPtr);
end;
procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(MyPtr);
end;
```

Text Property

Delphi command

`Text Property`

Description

The Text property specifies a text string to appear in a component or object.

Example

The following code stores the value of the first item of a combo box in the Text property in the OnCreate event handler of the form containing the combo box. The first item will be displayed in the combo box at run time.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Text := ComboBox1.Items[0];
end;
```

TextBox Control

Delphi command
`TEdit Component`

Description
See Data Controls Page Components

TextHeight Method

Delphi command
`TextHeight Method`

Applies to
TCanvas object

Declaration
```
function TextHeight(const Text: string): Integer;
```

Description
TextHeight returns the height in pixels of the string passed in Text when rendered in the current font. You can use TextHeight to specify whether the entire string will appear in a given space.

Example
This example displays the height of a text string in the current font of the canvas in an edit box on the form:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  L: LongInt;
begin
  L := Canvas.TextHeight('Object Pascal is the best');
  Edit1.Text := IntToStr(L) + ' pixels in height';
end;
```

TextWidth Method

Delphi command
`TextWidth Method`

Applies to
TCanvas object

Declaration
```
function TextWidth(const Text: string): Integer;
```

Description
The TextWidth method returns the width in pixels of the string passed in Text when rendered in the current font. You can use TextWidth to determine whether a given string will fit in a particular space.

Example
This example determines the width of a specified string, and if the string is too wide to display in an edit box, the edit box is widened to accommodate the string. The string displays in the edit box.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  T: Longint;
  S: string;
begin
  S := 'Object Pascal is the language for me';
  T := Canvas.TextWidth(S);
  if T > Edit1.Width then
    Edit1.Width := T + 10;
  Edit1.Text := S;
end;
```

Time Function

Delphi command
`Time Function`

Declaration
```
function Time: TDateTime;
```

Description
The Time function returns the current time.

Example
This example uses a label and a button on a form. When the user clicks the button, the current time displays in the caption of the label:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := 'The time is  ' + TimeToStr(Time);
end;
```

TimeSerial Function

Delphi command
EncodeTime Function

Declaration
```
function EncodeTime(Hour, Min, Sec, MSec: Word): TDateTime;
```

Description
The EncodeTime function returns a TDateTime type from the values specified as the Hour, Min, Sec, and MSec parameters.

If the value of the Time24Hour typed constant is False, valid Hour values are 0 through 12. If the value of Time24Hour is True, valid Hour values are 0 through 23.

Valid Min and Sec values are 0 through 59. Valid MSec values are 0 through 999.

If the specified values are not within range, an EConvertError exception is raised. The resulting value is a number between 0 (inclusive) and 1 (not inclusive) that indicates the fractional part of a day given by the specified time. The value 0 corresponds to midnight, 0.5 corresponds to noon, 0.75 corresponds to 6:00 pm, etc.

Example
```
procedure TForm1.Button1Click(Sender: TObject);
var
  MyTime: TDateTime;
begin
  MyTime := EncodeTime(0, 45, 45, 7);
  Label1.Caption := TimeToStr(MyTime);
end;
```

TimeValue Function

Delphi command
`StrToTime Function`

Declaration
```
function StrToTime(const S: string): TDateTime;
```

Description
The StrToTime function converts the given string to a time value. The string must consist of two or three numbers, separated by the character defined by the TimeSeparator global variable, optionally followed by an AM or PM indicator. The numbers represent hour, minute, and (optionally) second, in that order. If the time is followed by AM or PM, it is assumed to be in 12-hour clock format. If no AM or PM indicator is included, the time is assumed to be in 24-hour clock format. If the given string does not contain a valid time, an EConvertError exception is raised.

Example
This example uses an edit box, a label, and a button on a form. When the user enters a time in the edit box in the HH:MM:SS format, the string entered is converted to a TDateTime value. This value is then converted back to a string value so it can appear as the caption of the label:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ATime: TDateTime;
begin
  ATime := StrToTime(Edit1.Text);
  Label1.Caption := TimeToStr(ATime);
end;
```

Timer Control

Delphi command
`TTimer Component`

Description
See System Page Components

Timer Event

Delphi command
`OnTimer Event`

Declaration
```
property OnTimer: TNotifyEvent;
```

Description
The OnTimer event is used to execute code at regular intervals. Place the code you want to execute within the OnTimer event handler.

The Interval property of a timer component determines how frequently the OnTimer event occurs. Each time the specified interval passes, the OnTimer event occurs.

Example
Here is an example of an OnTimer event handler that moves a ball slowly across the screen:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Interval := 100;
  Shape1.Left := Shape1.Left + 1;
end;
```

Title Property

Delphi command
`Title Property`

Applies to
TApplication component

Declaration
```
property Title: string;
```

Description
The Title property determines the text that appears with an icon representing your application when it is minimized. The default value is the project name (the name of the project file without the .PRJ file extension).

You can set the title at run time, or you can enter the value of the Title property on the Application page of the Options|Project Options dialog box.

Example
```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.Title := 'My Incredible Application';
end;
```

ToPage Property

Delphi command
`ToPage Property`

Applies to
TPrintDialog component

Declaration
```
property ToPage: Integer;
```

Description
The value of the ToPage property determines on which page the print job ends. The default value is 0, which means no ending page is specified.

Example
This example uses a print dialog box on a form. The code sets up the print dialog box so that when it appears, the default values of 1 and 1 are the default starting and ending values for the Pages From and To edit boxes:

```
PrintDialog1.Options := [poPageNums];
PrintDialog1.FromPage := 1;
PrintDialog1.ToPage := 1;
```

Top Property

Delphi command
`Top Property`

Declaration
```
property Top: Integer;
```

Description
The Top property determines the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels. If the control is contained in a TPanel, the Left and Top properties will be relative to the panel. If the control is contained directly by the form, it will be relative to the form. For forms, the value of the Top property is relative to the screen in pixels.

For the Find and Replace dialog boxes, Top is a run-time only property. The default value is -1.

Example
The following code moves a button 10 pixels up each time a user clicks it:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Top := Button1.Top - 10;
end;
```

TopIndex Property

Delphi command
`TopIndex Property`

Declaration
```
property TopIndex: Integer;
```

Description
The TopIndex property is the index number of the item that appears at the top of the list box. You can use the TopIndex property to determine which item is the first item displayed at the top of the list box and to set it to the item of your choosing.

Example
This example uses a list box containing a list of strings, a button, and an edit box on a form. When the user runs the application and clicks the button, the third item in the list becomes the first item, and the index value of that item appears in the edit box. The index value displayed is 2, indicating the third item in the list (the first item in the list has an index value of 0):

```
procedure TForm1.FormCreate(Sender: TObject);
var
  Number: Integer;
begin
  for Number := 1 to 20 do
    ListBox1.Items.Add('Item ' + IntToStr(Number));
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.TopIndex := 2;
  Edit1.Text := IntToStr(ListBox1.TopIndex);
end;
```

TopRow Property

Delphi command
`TopRow Property`

Applies to
TDrawGrid, TStringGrid components

Declaration
```
property TopRow: Longint;
```

Description
Run-time only. The TopRow property determines which row in the grid appears at the top of the grid.

If you have one or more nonscrolling rows in the grid, they remain at the top, regardless of the value of the TopRow property. In this case, the row you specify as the top row will be the first row below the nonscrolling rows.

Example
This code uses a string grid and a button on a form. When the user clicks the button, the last row of the string grid becomes the top row:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  StringGrid1.TopRow := StringGrid1.RowCount;
end;
```

TrackDefault Property

Delphi command
`UpdateFormatSettings Property`

Declaration
```
property UpdateFormatSettings: Boolean;
```

Description
The UpdateFormatSettings property specifies whether the format settings are updated automatically when the user alters the system configuration. Set UpdateFormatSettings to False at application startup to not have format setting s update automatically.

Trim Function

Delphi command
`Trim Function`

Declaration
```
function Trim(const S: string): string;
```

Description
The Trim function trims leading and trailing spaces and control characters from the given string S.

TwipsPerPixelX, TwipsPerPixelY Properties

Delphi command
`Use PixelsPerInch Property`

Applies to
TScreen components

Declaration
```
property PixelsPerInch: Integer;
```

Description
Read and run-time only. Windows PixelsPerInch is a misnomer: it tells you nothing about the pixel resolution or aspect ratio of the video screen. All PixelsPerInch tells you is the relative size of the Windows system font. The value in PixelsPerInch is retrieved from Windows when Delphi loads.

Example
This example adds 30 to the form's PixelsPerInch property if the screen's PixelsPerInch property is greater than 100:

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  Form1.Scaled := True;
  if PixelsPerInch > 100 then
    Form1.PixelsPerInch := Form1.PixelsPerInch + 30;
end;
```

Type Property (Data Access)

Delphi command
`DataType Property`

Applies to
All field components

Declaration
```
property DataType: TFieldType;
```

Description
Run-time and read-only. DataType identifies the data type of the TField. Possible values are those of the TFieldType type.

Type Property (Picture)

Delphi command
`BitMap Property; MetaFile Property, Icon Property (TPicture)`

Applies to
TPicture object

Declaration
```
property Bitmap: TBitmap;
```

Description
The Bitmap property specifies the contents of the TPicture object as a bitmap graphic (.BMP file format). If Bitmap is referenced when the TPicture contains a Metafile or Icon graphic, the graphic won't be converted. Instead, the original contents of the TPicture are discarded and Bitmap returns a new, blank bitmap.

Example
The following code copies the bitmap in Picture1 to the Glyph of BitBtn1.

```
BitBtn1.Glyph := Picture1.Bitmap;
```

Type Statement

Delphi command

```
Type Reserved Word
```

Description

A type Declaration specifies an identifier that denotes a type. A variable's type defines the set of values it can have and the operations that can be performed on it.

TypeName Function

Delphi command
`ClassName Method; Is Operator`

Declaration
```
class function ClassName: ShortString;
```

Description
The ClassName method returns a string containing the name of the actual type of an object. For example, you can assign any type of object to a variable of type TObject. If you then call that variable's ClassName method, it returns the actual type of the assigned object, rather than TObject.

For most purposes, you don't need the name of an object type. If you need to know the type of an object, the is operator or the ClassType method provide more useful information than ClassName.

Example
This example uses a button, a label, a list box, a check box, and an edit box on a form. When the user clicks one of the controls, the name of the control's class appears in the edit box.

```
procedure FindClassName(AControl:TObject);
begin
  Form1.Edit1.Text := AControl.ClassName;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindClassName(Button1);
end;
procedure TForm1.Label1Click(Sender: TObject);
begin
  FindClassName(Label1);
end;
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  FindClassName(CheckBox1);
end;
procedure TForm1.ListBox1Click(Sender: TObject);

begin
  FindClassName(ListBox1);
end;
```

UBound Function

Delphi command
High Function

Declaration
```
function High(X);
```

Description
The High function returns the highest value in the range of the argument.

Example
```
function Sum( var X: array of Double): Double;
var
  I: Word;
  S: Double;
begin
  S := 0;  { Note that open array index range is always zero-based. }
  for I := 0 to High(X) do S := S + X[I];
  Sum := S;
end;
procedure TForm1.Button1Click(Sender: TObject);
var
  List1: array[0..3] of Double;
  List2: array[5..17] of Double;
  X: Word;
  S, TempStr: string;

begin
  for X := Low(List1) to High(List1) do
    List1[X] := X * 3.4;
    for X := Low(List2) to High(List2) do
    List2[X] := X * 0.0123;
  Str(Sum(List1):4:2, S);
  S := 'Sum of List1: ' + S + #13#10;
  S := S + 'Sum of List2: ';
  Str(Sum(List2):4:2, TempStr);
  S := S + TempStr;
  MessageDlg(S, mtInformation, [mbOk], 0);
end;
```

UBound Property

Delphi command

`ControlCount Property; ComponentCount Property`

Declaration

```
property ControlCount: Integer;
```

Description

Run-time and read only. The ControlCount property indicates the number of controls that are children of the control. The children are listed in the Controls property array.

Example

This code uses several controls on a form, including a button and an edit box. When the user clicks the button, the code counts all the components on the form and displays the number in the Edit1 edit box. While the components are being counted, each is evaluated to see if it is a button component. If the component is a button, the code changes the font on the button face.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to ComponentCount -1 do
     if Components[I] is TButton then
       TButton(Components[I]).Font.Name := 'Courier';
  Edit1.Text := IntToStr(ComponentCount) + ' components';
end;
```

UCase Function

Delphi command
`UpperCase Function`

Declaration
```
function UpperCase(const S: string): string;
```

Description
The UpperCase function returns a string containing the same text as S, but with all letters converted to uppercase.

Example
This example uses a list box and a button on a form. Use the Items property editor in the Object Inspector to enter a list of strings in the list box. When you run the application and click the button, the strings in the list box become uppercase.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to ListBox1.Items.Count -1 do
    ListBox1.Items[I] := UpperCase(ListBox1.Items[I]);
end;
```

UNION Operation (SQL)

Delphi command
UNION Clause (SQL)

Description
SQL Syntax

Example
```
Select * from OldOrders
Union
Select * from Orders
where OrdDate < '01/01/96'
```

UPDATE Statement (SQL)

Delphi command
UPDATE Statement (SQL)

Description
SQL Syntax

Example
```
Update Orders Set Status = 'Closed'
where OrderNo = 1100
```

Underline Property

Delphi command

`TFont.Style Property`

Declaration

```
TFontStyle = (fsBold, fsItalic, fsUnderline, fsStrikeOut);
TFontStyles = set of TFontStyle;
```

Description

The TFontStyles type is the set of font styles the Style property of a font object (TFont) can assume.

Example

The following code boldfaces the font used in the memo..

```
Memo1.Font.Style := [fsBold];
```

Unique Property

Delphi command
AddIndex Method

Declaration
```
procedure AddIndex(const Name, Fields: string; Options: TIndexOptions);
```

Description
The AddIndex method creates a new index for the TTable. Name is the name of the new index. Fields is a list of the fields to include in the index. Separate the field names by a semicolon. Options is a set of values from the TIndexOptions type.

Example
```
Table1.AddIndex('NewIndex', 'CustNo;CustName', [ixUnique,
ixCaseInsensitive]);
```

Unload Event

Delphi command
`OnClose Event`

Applies to
TForm component

Declaration
```
TCloseAction = (caNone, caHide, caFree, caMinimize);
TCloseEvent = procedure(Sender: TObject; var Action: TCloseAction) of object;
property OnClose: TCloseEvent;
```

Description
The OnClose event specifies which event handler to call when a form is about to close. The handler specified by OnClose might, for example, test to make sure all fields in a data-entry form have valid contents before allowing the form to close.

A form is closed by the Close method or when the user chooses Close from the form's system menu.

Example
This example displays a message dialog box when the user attempts to close the form. If the user clicks the Yes button, the form closes; otherwise, the form remains open.

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  if MessageDlg('Close application ?', mtConfirmation,
    [mbYes, mbNo], 0) = mrYes then
    Action := caFree
  else
    Action := caNone;
end;
```

Updatable Property

Delphi command
`CanModify Property`

Declaration
```
property CanModify: Boolean;
```

Description
Run-time and read only. Specifies if a field can be modified for any reason, such as during a SetKey operation. CanModify is True if the value of the field can be modified. If the ReadOnly property of the field is True, or the ReadOnly property of the dataset is True, then CanModify is False.

Update Method (Data Access)

Delphi command
`UpdateRecord Method`

Declaration
```
procedure UpdateRecord;
```

Description
The UpdateRecord method notifies each TDataSource component that the current record is about to be posted to the dataset. Each data source in turn notifies all data controls so that they can update the fields of the record from the current values displayed in the controls. UpdateRecord is called automatically by Post, but an application can also use it separately to bring the current record up to date without posting it.

Update Method (OLE Container)

Delphi command
`UpdateObject Method`

Applies to
TOleContainer component

Declaration
```
procedure UpdateObject;
```

Description
Updates the OLE object. Linked OLE objects and embedded objects that contain linked OLE objects get outofdate when the source of the link is updated. UpdateObject rereads the source to ensure that the OLE object has current data. If there is no OLE object loaded into the container, UpdateObject has no effect.

Example
The following example will reread the linked OLE object to ensure that the OLE object has current data.

```
OleContainer1.UpdateObject;
```

UpdateControls Method

Delphi command
`Refresh Method`

Declaration
```
procedure Refresh;
```

Description
The Refresh method erases whatever image is on the screen and then repaints the entire control. Within the implementation of Refresh, the Invalidate and then the Update methods are called.

Example
```
Table1.Refresh;
```

UpdateOptions Property

Delphi command
See UpdateObject Method in Delphi.hlp

Example
The following example will reread the linked OLE object to ensure that the OLE object has current data.

```
OleContainer1.UpdateObject;
```

UseMnemonic Property

Delphi command

Always True by Default

User-Defined Date/Time Formats (Format Function)

Delphi command
`FormatDateTime Function`

Example
The following example assigns 'The meeting is on Wednesday, February 15, 1995 at 10:30 AM' to the string variable S.

```
S := FormatDateTime('"The meeting is on" dddd, mmmm d, yyyy, ' +
  '"at" hh:mm AM/PM', StrToDateTime('2/15/95 10:30am'));
```

User-Defined Numeric Formats (Format Function)

Delphi command
`FormatFloat Function`

Declaration
```
function FormatFloat(const Format: string; Value: Extended): string;
```

Description
FormatFloat formats the floating-point value given by Value using the format string given by Format.

User-Defined String Formats (Format Function)

Delphi command

```
Format Function
```

Declaration

```
function Format(const Format: string; const Args: array of const): string;
```

Description

This function formats the series of arguments in the open array Args. Formatting is controlled by the Object Pascal format string Format; the results are returned in the function result as a Pascal string.

For information on the format strings, see Format Strings in DELPHI.HLP.

Example

```
Format('%d %d %0:d %d', [10, 20]) = '10 20 10 20'.
```

Val Function

Delphi command
Val Function

Declaration
```
procedure Val(S; var V; var Code: Integer);
```

Description
The Val function converts the string value S to its numeric representation, as if it were read from a text file with Read.

S is a string-type expression; it must be a sequence of characters that form a signed whole number. V is an integer-type or real-type variable. Code is a variable of type Integer.

Example
```
uses Dialogs;
var
  I, Code: Integer;
begin
  { Get text from TEdit control }
  Val(Edit1.Text, I, Code);
  { Error during conversion to integer? }
  if code <> 0 then
    MessageDlg('Error at position: ' + IntToStr(Code), mtWarning, [mbOk], 0);
  else
    Canvas.TextOut(10, 10, 'Value = ' + IntToStr(I));
  Readln;
end;
```

Validate Event

Delphi command
`OnValidate Event`

Declaration
```
TFieldNotifyEvent = procedure(Sender: TField) of object;
property OnValidate: TFieldNotifyEvent;
```

Description
The OnValidate event is activated when a field is modified.

Example
```
Field1.OnValidate := ValidateFieldRange;
```

ValidationRule Property

Delphi command
Use OnValidate Event

Declaration
```
TFieldNotifyEvent = procedure(Sender: TField) of object;
property OnValidate: TFieldNotifyEvent;
```

Description
The OnValidate event is activated when a field is modified.

Example
```
Field1.OnValidate := ValidateFieldRange;
```

ValidationText Property

Delphi command

```
Use OnValidate Event
```

Declaration

```
TFieldNotifyEvent = procedure(Sender: TField) of object;
property OnValidate: TFieldNotifyEvent;
```

Description

The OnValidate event is activated when a field is modified.

Example

```
Field1.OnValidate := ValidateFieldRange;
```

Value Property

Delphi command
`Checked Property`

Declaration
```
property Checked: Boolean;
```

Description
Run-time only. The Checked property determines whether an option is selected.

Example
This example fills in a radio button at run time:

```
RadioButton1.Checked := True;
```

This example uses a main menu component that contains a menu item named SnapToGrid1 on a form. When the user chooses the Snap To Grid command, a check mark appears next to the command. When the user chooses the Snap To Grid command again, the check marks disappears:

```
procedure TForm1.SnapToGrid1Click(Sender: TObject);
begin
  SnapToGrid1.Checked := not SnapToGrid1.Checked;
end;
```

Value Property (Data Access)

Delphi command

```
Value Property
```

Description

Run-time only. Value is the actual data in a TField. Use Value to read data directly from and write data directly to a TField.

Examples

```
StringField1.Value := 'Delphi';

DateField1.Value := StrToDateTime('02/14/95 00:00:00');
```

VarType Functions

Delphi command

```
VarType Function (Only for Variant); Is Operator
```

Declaration

```
function VarType(const V: Variant): Integer;
```

Description

The VarType function returns the type code of the given variant.

Verb Property

Delphi command
`DoVerb Method`

Declaration
```
procedure DoVerb(Verb: Integer);
```

Description
Requests the OLE object to perform some action.

Example
The following code will activate the OLE Object with its primary verb.

```
OleContainer1.DoVerb( ovPrimary );
```

Visible Property

Delphi command
`Visible Property`

Declaration
```
property Visible: Boolean;
```

Description
The Visible property determines whether the component appears onscreen.

Example
The following code shows how to make a button invisible:

```
Button1.Visible := False;
```

VisibleCols Property

Delphi command
`FieldCount Property`

Declaration
```
property FieldCount: Integer;
```

Description
Run-time and read-only. The FieldCount property specifies the number of fields (columns) in a dataset.

Example
This example displays a message box with the names of all fields in the table.

```
procedure TForm1.Button2Click(Sender: TObject);
var
   i: Integer;
   Info: String;
begin
   Info := 'The fields are:'#13#10#13#10;
   for i := 0 to Table1.FieldCount - 1 do
      Info := Info + Table1.Fields[i].FieldName + #13#10;
   ShowMessage(Info);
end;
```

VisibleCount Property

Delphi command
`DropDownCount Property`

Declaration
```
property DropDownCount: Integer;
```

Description
The DropDownCount property determines how long the drop-down list of a combo box is.

Example
The following code assigns three to the DropDownCount property of ComboBox1. To see more than three items in the drop-down list, the user must scroll.

```
ComboBox1.DropDownCount := 3;
```

VisibleRows Property

Delphi command
`TField.Visible Property`

Declaration
```
property Visible: Boolean;
```

Description
The Visible property determines whether the component appears onscreen. If Visible is True, the component appears. If Visible is False, the component is not visible.

Example
The following code prevents the BillDate field from appearing in a DBGrid.

```
Table1.FieldByName( 'BillDate' ).Visible := False;
```

WHERE Clause (SQL)

Delphi command
`WHERE Clause (SQL)`

Description
SQL Syntax

Example

```
SELECT * FROM PARTS
WHERE PART_NO IN (543, 544, 546, 547)
```

Weekday Function

Delphi command
DayOfWeek Function

Declaration
```
function DayOfWeek(Date: TDateTime): Integer;
```

Description
The DayOfWeek function returns the day of the week of the specified date as an integer between 1 and 7. Sunday is the first day of the week and Saturday is the seventh.

Example
This example uses a button, an edit box, and a label on a form. When the user enters a date in the edit box using the Month/Day/Year format, the caption of the label reports the day of the week for the specified date.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ADate: TDateTime;
begin
  ADate := StrToDate(Edit1.Text);
  Label1.Caption := 'Day ' + IntToStr(DayOfWeek(ADate)) + ' of the week';
end;
```

Weight Property

Delphi command

`TFont.Style Property`

Declaration

```
TFontStyle = (fsBold, fsItalic, fsUnderline, fsStrikeOut);
TFontStyles = set of TFontStyle;
```

Description

The TFontStyles type is the set of font styles the Style property of a font object (TFont) can assume.

Example

The following code boldfaces the font used in the memo..

```
Memo1.Font.Style := [fsBold];
```

WhatsThisButton Property (Windows 95)

Delphi command
`BorderIcons Property`

Example
The following code removes a form's Maximize button when the user clicks a button:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  BorderIcons := BorderIcons - [biMaximize];
end;
```

WhatsThisID Property (Windows 95)

Delphi command
`HelpContext Property`

Declaration
```
property HelpContext: THelpContext;
```

Description
The HelpContext property provides a context number for use in calling context-sensitive online Help.

Example
The following code associates a Help file with the application, and makes the screen with a context number of 7 the context-sensitive Help screen for the Edit1 edit box:

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  Application.HelpFile := 'MYHELP.HLP';
  Edit1.HelpContext := 7;
end;
```

While...Wend Statement

Delphi command
`While Statement`

Description
A while statement controls the repeated execution of a singular or compound statement.

The statement after do executes as long as the Boolean expression is True.

The expression is evaluated before the statement is executed, so if the expression is False at the beginning, the statement is not executed.

Example
The following example will run through the records of a table.

```
Procedure TForm1.Button1Click( Sender : TObject );
begin
  Table1.First;
  While NOT Table1.EOF do
  begin
    {Do Something Here}
    Table1.Next;
  end;
end;
```

Width Property

Delphi command
`Width Property`

Description
The Width property determines horizontal size

Example
The following code doubles the width of a button:

```
Button1.Width := Button1.Width * 2;
```

WindowList Property

Delphi command
`WindowMenu Property`

Declaration
```
property WindowMenu: TMenuItem;
```

Description
Most Windows MDI applications contain a Window menu that contains menu items such as Cascade, Arrange Icons, Tile, and so on that let the user manage the windows in the application. Usually this menu lists (at the bottom) the child windows that are currently open in the application. When the user selects one of these windows from the menu, the window becomes the active window in the application.

The WindowMenu property determines which menu includes the open child windows (or forms) in your application.

Example
For this code to run, a menu item called MyWindows must exist on an MDI form parent form. This line of code designates the MyWindows menu to be the Window menu, the menu that lists all open child windows in an MDI application:

```
WindowMenu := MyWindows;
```

WindowStateProperty

Delphi command
`WindowState Property`

Declaration
```
property WindowState: TWindowState
```

Description
The WindowState determines the initial state of the form.

Example
The following code responds to the user clicking a button named Shrink by minimizing the form:

```
procedure TForm1.ShrinkClick(Sender: TObject);
begin
  WindowState := wsMinimized;
end;
```

With Statement

Delphi command
`With Statement`

Description
The with statement is a shorthand method for referencing the fields of a record and the fields and methods of an object.

Within a with statement, the fields of one or more record variables can be referenced using only their field identifiers.

Within a with statement, each variable reference is first checked to see if it can be interpreted as a field of the record. If so, it is always interpreted as such, even if a variable with the same name is also accessible.

Example

```
type
   TDate = record
      Day : Integer;
      Month: Integer;
      Year : Integer;
   end;

var
  OrderDate: TDate;

with OrderDate do
  if Month = 12 then
  begin
    Month := 1;
    Year := Year + 1
  end
  else
    Month := Month + 1;
```

WordWrap Property

Delphi command
`WordWrap Property`

Declaration
```
property WordWrap: Boolean;
```

Description
The WordWrap property determines if text in a label, memo, or rich edit control wraps at the right margin so that it fits in the control. You can give the user access to the lines which aren't visible in a memo or rich edit control by setting its ScrollBars property to add horizontal, vertical, or both scrollbars to the memo control. There should be no reason to use a horizontal scroll bar if WordWrap is True.

Example
This example allows text a user enters in the Memo1 control to wrap to the next line, if the control is large enough to hold the text:

```
Memo1.WordWrap := True;
```

Workspace Object, Workspaces Collection

Delphi command

`TSession Component`

Description

The TSession component provides global control over database connections for an application. Delphi automatically creates a default TSession component at runtime for applications which use database controls. This component may be accessed at runtime through the global variable Session.

Write # Statement

Delphi command
`WriteLn`

Declaration
```
procedure Writeln([ var F: Text; ] P1 [, P2, ...,Pn ] );
```

Description
The Writeln procedure is an extension to the Write procedure, as it is defined for text files.

After executing Write, Writeln writes an end-of-line marker (carriage-return/linefeed) to the file. Writeln(F) with no parameters writes an end-of-line marker to the file. (Writeln with no parameter list corresponds to Writeln(Output).)

Example
```
program Project1;

{$AppType ConSole}

uses windows;

var
   s : string;
begin
   Write('Enter a line of text: ');
   Readln(s);
   Writeln('You typed: ',s);
   Writeln('Hit <Enter> to exit');
   Readln;
end.
```

X1, Y1, X2, Y2 Properties

Delphi command
`MoveTo and LineTo Methods`

Applies to
TCanvas object

Declaration
```
procedure LineTo(X, Y: Integer);
procedure MoveTo(X, Y: Integer);
```

Description
The LineTo method draws a line on the canvas from the current drawing position (specified by the PenPos property) to the point specified by X and Y and sets the pen position to (X, Y).

The MoveTo method changes the current drawing position to the coordinates passed in X and Y. The current position is given by the PenPos property.

Example
The following example draws a line from the upper left corner of a form to the point clicked with the mouse.

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  Canvas.MoveTo(0, 0);
  Canvas.LineTo(X, Y);
end;
```

Year Function

Delphi command
`FormatDateTime Function`

Declaration
```
function FormatDateTime(const Format: string; DateTime: TDateTime): string;
```

Description
FormatDateTime formats the date-and-time value given by DateTime using the format given by Format.

yy displays the year as a two-digit number (00-99).

Yyyy displays the year as a four-digit number (0000-9999).

Example
The following example will display the current year on the form's caption bar.

```
Caption := FormatDateTime( 'yyyy', now );
```

Zoom Property

Delphi command
`StretchDraw Method`

Declaration
```
procedure StretchDraw(const Rect: TRect; Graphic: TGraphic);
```

Description
This method draws the graphic specified by the Graphic parameter in the rectangle specified by the Rect parameter. Use this method to stretch or resize a graphic to the size of the rectangle.

Example
The following code stretches the bitmap to fill the client area of Form1.

```
Form1.Canvas.StretchDraw( Form1.ClientRect, TheGraphic );
```

Zorder Method

Delphi command
`BringToFront Method; SendToBack Method`

Declaration
```
procedure BringToFront;
procedure SendToBack;
```

Description
The BringToFront method puts the component or form in front of all other components or forms within its parent component or form. BringToFront is especially useful for making certain that a form is visible. You can also use it to reorder overlapping components within a form.

The SendToBack method puts a windowed component behind all other windowed components within its parent component or form, or it puts a non-windowed component behind all other non-windowed components within its parent component or form.

Example
The following example will Show Form2 and bring it to the top in the Zorder.

```
procedure TForm1.ShowPaletteButtonClick(Sender: TObject);
begin
  if NOT Form2.Visible then Form2.Show;
  Form2.BringToFront;
end;
```

hDC Property

Delphi command

```
TCanvas.Handle Property
property Handle: HDC;        {for TCanvas objects}
```

Description

The Handle property lets you access the Windows GDI object handle, so you can access the GDI object. If you need to use a Windows API function that requires the handle of a pen object, you could pass the handle from the Handle property of a TPen object.

Example

The following example will select the Palette from Image2 for Image1.

```
Image1.Picture.Bitmap.Palette := SelectPalette( Image2.Canvas.Handle,
Image2.Picture.Bitmap.Palette, TRUE );
```

hInstance Property

Delphi command
`HInstance Variable`

Declaration
```
var HInstance: Longint;
```

Description
The HInstance variable contains the instance handle of the application or library as provided by the Windows environment.

Example
The following code will load a bitmap from a resource file linked into the application's EXE and then display it on the form.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Bmp: TBitmap;
begin
  Bmp := TBitmap.Create;
  Bmp.Handle := LoadBitmap(HInstance,'BITMAP_1');
  Canvas.Draw(0, 0, Bmp);
  Bmp.Free;
end;
```

hPal Property

Delphi command
`Palette Property`

Declaration
```
property Palette: HPalette;
```

Description
The Palette property controls a bitmap's color mapping. The Palette of a bitmap contains up to 256 colors that can be used to display the bitmap on screen.

When running in a 256 color video mode, and if the bitmap is drawn by an application running in the foreground, all colors of Palette will be added to the Windows system palette. If the bitmap is drawn by an application running in the background and another application has loaded the system palette with its own colors, the bitmap's colors will be mapped to the system palette. Palette=0 if the bitmap has no palette. You can assign custom palettes created with CreatePalette to this property.

Example
The following code will select the Palette from Image2 for use by Image1.

```
  Image1.Picture.Bitmap.Palette := SelectPalette( Image1.Canvas.Handle,
Image2.Picture.Bitmap.Palette, TRUE );
```

hScrollBar, VScrollBar Controls

Delphi command
`TScrollBar Component`

Description
Standard Page Components

hWnd Property

Delphi command
`Handle Property`

Applies to
All windowed controls

Declaration
```
property Handle: HWND;
```

Description
Read and run-time only. The Handle property gives you access to window handle of the application, the Find and Replace dialog boxes, and all controls in case you need to call a Windows API function that requires a handle.

Example
The following code will send the cursor to the next control on the form.

```
PostMessage( Handle, WM_NEXTDLGCTL, 0, 0 );
```

Key Mappings

The following tables contain the shortcut keys and mappings to the Visual Basic keys that produce the same results in the Delphi environment.

Debug window key mappings

Visual Basic	Delphi	To
ENTER	F7	Trace into code.
CTRL+Z	CTRL+Z	Undo the last edit.
CTRL+C	CTRL+C	Copy the selected text to the Clipboard.
CTRL+V	CTRL+V	Paste the contents of the Clipboard.
CTRL+X	CTRL+X	Cut the selected text to the Clipboard.
F5	F9	Run application.
F8	F7	Trace into code.
SHIFT+F8	F8	Step over procedure.
DEL	DEL	Delete the selected text.
CTRL+ENTER	ENTER	Insert carriage return.
ENTER	CTRL F5	Display Watch Window.
F4	F11	Display the Object Inspector.

Code window key combinations

Visual Basic	Delphi	To
F1	F1	Get context-sensitive Help.
F9	F5	Toggle breakpoint.
F5	F9	Run an application.
F8	F7	Trace into procedure.
SHIFT+F8	F8	Step over procedure.
CTRL+BREAK	CTRL F2	Program reset.
PAGE DOWN	PAGE DOWN	Page down.
PAGE UP	PAGE UP	Page up.
CTRL+SHIFT+F2	CTRL Q P	Move back to the last position in your code.
CTRL+HOME	CTRL+HOME	Move to the beginning of the unit.
CTRL+END	CTRL+END	Move to the end of the unit.
CTRL+RIGHT	CTRL+RIGHT	Move one word to the right.
CTRL+LEFT	CTRL+LEFT	Move one word to the left.
END	END	Move to the end of the line.
HOME	HOME	Move to the beginning of the line.
CTRL+C	CTRL+C	Copy the selected text to the Clipboard.
CTRL+X	CTRL+X	Cut the selected text to the Clipboard.
DEL	DEL	Delete the selected text.
CTRL+V	CTRL+V	Paste the contents of the Clipboard.
CTRL+Z	CTRL+Z	Undo the last edit.
CTRL+BACKSPACE	CTRL+BACKSPACE	Delete to the beginning of the word.
F3	F3	Find Next.

Code Window Keyboard Shortcuts

Visual Basic	Delphi	To
F7	F12	View Editor Window.
CTRL+F	CTRL+F	Find.
CTRL+R	CTRL+R	Replace.
F3	F3	Find Next.
CTRL+PAGE DOWN	PAGE DOWN	Move one screen down.
CTRL+PAGE UP	PAGE UP	Move one screen up.
CTRL+SHIFT+F2	CTRL Q P	Go to last position.
CTRL+HOME	CTRL+HOME	Move to beginning of Unit.
CTRL+END	CTRL+END	Move to end of Unit.
CTRL+RIGHT ARROW	CTRL+RIGHT ARROW	Move one word right.
CTRL+LEFT ARROW	CTRL+LEFT ARROW	Move one word left.
END	END	Move to end of line.
HOME	HOME	Move to beginning of line.
CTRL+N	ENTER	Insert new line.
CTRL+BACKSPACE	SHIFT+CTRL+Z	Redo.
CTRL+Z	CTRL+Z	Undo.
CTRL+Y	CTRL+Y	Delete current line.
CTRL+DELETE	CTRL+RIGHT ARROW DEL	Delete to end of word.
TAB	TAB	Indent.
SHIFT+TAB	BACKSPACE	Outdent.

Project Manager Keys

Visual Basic	Delphi	To
SHIFT+ENTER	ENTER	Open the selected unit from list.
ENTER	SHIFT+ENTER	Open the selected form from list.
HOME	HOME	Select the first object in the list.
END	END	Select the last object in the list.

Form Window Keys

Visual Basic	Delphi	To
SHIFT+CTRL+alpha	F11 TAB alpha	Select a property in the Properties list of the Object Inspector.
alpha	alpha	Enter a value in the Object Inspector for the selected property.
F7	F12	Open the Editor Window.
CTRL+C	CTRL+C	Copy the selected components to the Clipboard.
CTRL+X	CTRL+X	Cut the slected components to the Clipboard.
DEL	DEL	Delete the selected components.
CTRL+V	CTRL+V	Paste the Clipboard contents on the form.
CTRL+Z	CTRL+Z	Undo a deletion of components.
TAB	TAB	Cycle forward through components in tab order.
SHIFT+TAB	SHIFT+TAB	Cycle backward through components in tab order.
CTRL+CLICK	SHIFT+CLICK	Add or remove a components from the selection.
CLICK+DRAG	CLICK+DRAG	Select multiple components.
F4	F11	Display the Object Inspector.

Object Inspector Keys (Properties List focus)

Use these key combinations when the Properties List has the focus in the Object Inspector

Visual Basic	Delphi	To

PAGE DOWN	PAGE DOWN	Move down one screen of the Property list.
PAGE UP	PAGE UP	Move up one screen of the Property list.
DOWN ARROW	DOWN ARROW	Move down through each property.
UP ARROW	UP ARROW	Move up through each property.
ALT+F6	F11	Switch from the Object Inspector to the form.
TAB	TAB	Move the insertion point to the property's settings box.
SHIFT+CTRL+alpha	N/A	Move to the next property in the list that begins with the alpha character.
Double-Click	Double-Click	Cycle through settings of enumerated properties.

Object Inspector Keys (Settings Box focus)

Use these key combinations when the Settings Box for a property has the focus in the Object Inspector

Visual Basic	**Delphi**	**To**
CTRL+Z	CTRL+Z	Undo the last edit.
CTRL+C	CTRL+C	Copy the selcted text to the Clipboard.
CTRL+X	CTRL+X	Cut the selected text to the Clipboard.
DEL	DEL	Delete the selected text.
CTRL+V	CTRL+V	Paste the Clipboard contents at the insertion point.
ENTER	ENTER	Commit the change of the property.
TAB	TAB	Move the focus to the Property list.
ESC	ESC	Cancel the property change.

Components Pages Reference

Components are the building blocks of every Delphi application, and the basis of the Delphi visual component library. Each page tab in the Component palette displays a group of icons representing the components used to design your application interface.

Components can be either visual or non-visual. Each component has specific attributes that enable you to control your application. These attributes are Properties, Events, and Methods.

Choose one of the following topics for information about its components and their functions.

Visual Basic Controls and Delphi Components

The following is a list of Visual Basic controls and their corresponding Delphi components.

To see a description of the Delphi component, click on the appropriate component page.

VB control	Delphi component	Delphi Component Page
Image	TImage	Additional
Label	TLabel	Standard
TextBox	TEdit	Standard
Frame	TGroupBox	Standard
CommandButton	TButton	Standard
CheckBox	TCheckBox	Standard
OptionButton	TRadioButton	Standard
ComboBox	TComboBox	Standard
ListBox	TListBox	Standard
HScrollBar	TScrollBar	Standard
VScrollBar	TScrollBar	Standard
Timer	TTimer	System
DriveListBox	TDriveComboBox	System
DirListBox	TDirectoryListBox	System
FileListBox	TFileListBox	System
Shape	TShape	Additional
OleControl	TOleContainer	System
Grid	TStringGrid	Additional
CommonDialog	TOpenDialog	Dialog
	TSaveDialog	Dialog
	TFontDialog	Dialog
	TColorDialog	Dialog
	TPrintDialog	Dialog
	TPrinterSetupDialog	Dialog
	TFindDialog	Dialog
	TReplaceDialog	Dialog
Gauge	TGauge	Samples
Graph	TChart	OCX
MMControl	TMediaPlayer	Additional
MaskEdBox	TMaskEdit	Additional
Outline	TOutline	Additional
SpinButton	TSpinButton	Samples
SSCommand	TBitBtn	Additional

Data access and display controls can be found on the Data Access and Data Controls component pages.

Standard Page Components

For complete information about using Delphi components, see the appropriate TComponent name in Delphi.hlp

The components on the Standard page of the Component palette make the standard Windows control elements available to your Delphi applications.

TMainMenu	Creates menus for your form
TPopUpMenu	Create popup menus for your form
TLabel	Displays text the user cannot select or manipulate, such as title text
TEdit	Displays an editing area where the user can enter or modify a single line of data
TMemo	Displays an editing area where the user can enter or modify multiple lines of data
TButton	Creates a pushbutton control that users choose to initiate action
TCheckBox	Presents a control that a user can toggle between Yes/No or True/False. You can use checkboxes to display selections that are not mutually exclusive. Users can select more than one checkbox in a group.
TRadioButton	Presents a control that a user can toggle between Yes/No or True/False. You can use RadioButtons to display selections that are mutually exclusive. Users can not select more than one RadioGroup in a group.
TListBox	Displays a scrolling list of choices.
TComboBox	Displays a list of choices in a combined edit and list box. Users can edit data in the edit box, or select data in the list box.
TScrollBar	Provides a way to change which portion of a list or form is visible, or to move through a range by increments.
TGroupBox	Provides a container to group related options on a form
TRadioGroup	Creates a group box that contains RadioButtons on a form
TPanel	Creates a panel that can contain other components on a form. You can use the Tpanel to create ToolBars or Status lines.

Additional Page Components

For complete information about using Delphi components, locate the appropriate TComponent name in Delphi.hlp

The components on the Additional page of the Component palette make specialized Windows control elements available to your Delphi applications.

TBitButton	Creates a button component that can display a bitmap
TSpeedButton	Creates a button that can display a glyph, but not a caption. SpeedButtons are often grouped in a panel to create a speedbar.
TMaskEdit	Similar to an edit box, but provides a way to specify particular formats for data entry or display.
TStringGrid	Creates a grid that you can use to display string data in columns or rows.
TDrawGrid	Creates a grid that you can use to display data in columns or rows.
TImage	Displays a bitmap, icon or metafile.
TShape	Draws geometric shapes, including an ellipse or circle, rectangle or square, rounded rectangle or rounded square.
TBevel	Creates lines or boxes with a three dimensional or chiseled appearance.
TScrollBox	Creates a resizeable container that automatically displays scrollbars if necessary.

Win95 Page Components

For complete information about using Delphi components, locate the appropriate TComponent name in Delphi.hlp

The components on the Win95 page of the Component palette provide access to Win95 user interface common controls available to your Delphi applications.

TTabControl	TTabControl component is a tab set which functions similarly to a TTabSet and has the appearance of notebook dividers.
TPageControl	TPageControl component is a page set which is used to make a multiple page dialog box. It displays multiple overlapping pages
TTreeView	Displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk. The control includes buttons that allow items to be expanded or collapsed.
TListView	Displays a list of items in a variety of ways. The ViewStyle property determines whether items are displayed in columns with column headers and sub-items, or vertically or horizontally, with small or large icons.
TImageList	TImageList component is a container for a group of graphic images.
THeaderControl	Contains multiple, movable headers and is similar to a THeader component.
TRichEdit	TRichEdit component is a Rich Text Format memo control. As such it includes the properties of a memo such as HideSelection, HideScrollBars, Lines, ScrollBars, WantReturn, WantTabs, and WordWrap..
TStatusBar	The TStatusBar component is a window, always horizontal and aligned along the bottom edge of an application frame, that displays status information about the application.
TTrackBar	A TTrackBar component is an optionally-ticked bar control that contains a slider which marks a current Position.
TProgressBar	Tracks the progress of a procedure within an application. As the procedure progresses, the rectangular TProgressBar gradually fills from left to right with the system highlight color.
TUpDown	The TUpDown component consists of a pair of Up and Down arrow buttons. Clicking on these buttons increments and decrements a numeric value held in the Position property.
THotKey	The THotKey component is used to set a shortcut property at run time. The user can enter a key combination, typically consisting of a modifier key (such as Ctrl, Alt, or Shift) and an accompanying key (such as a character key, an arrow key, a function key, etc.), into the hot-key control.

Dialogs Page Components

For complete information about using Delphi components, locate the appropriate TComponent name in Delphi.hlp

The components on the Dialogs page of the Component Palette make the Windows common dialog boxes available to your Delphi applications. The common dialog boxes provide a consistent interface for file operations such as opening, saving, and printing files. The following illustration shows the Dialogs components.

TOpenDialog	Makes an Open dialog box available to your application.
TSaveDialog	Makes a Save dialog box available to your application.
TFontDialog	Makes a Font dialog box available to your application.
TColorDialog	Makes a Color dialog box available to your application.
TPrintDialog	Displays a Print dialog box that permits the user to select which printer to print to, which pages to print, how many copies to print, and if the print job should be collated.
TPrinterSetupDialog	Displays a Printer Setup dialog box in your application. Users can use the dialog box to setup their printer before printing a job.
TFindDialog	Provides a Find dialog box to your application.
TReplaceDialog	Provides a Replace dialog box your application can use. TReplaceDialog contains all the capabilities of the TFindDialog component, but it also allows the user to replace found text with a replacement string.

Data Access Page Components

For complete information about using Delphi components, locate the appropriate TComponent name in Delphi.hlp

The components on the Data Access page of the Component palette make specialized database access elements available to your Delphi applications. The following illustration shows the Data Access components.

TDatasource	Acts as a conduit between ttable, tquery, tstoredproc component and data aware components such as dbgrid.
TTable	Provides live access to database tables through the Borland Database Engine. TTable is the interface between the Borland Database Engine and TDataSource components.
TQuery	Enables Delphi applications to issue SQL statements to a database engine--either the BDE or an SQL server. TQuery provides the interface between an SQL server (or the BDE) and TDataSource components.
TStoredProc	Enables Delphi applications to execute server stored procedures.
TDatabase	While not required for database access, provides additional control over factors that are important for client/server applications.
TSession	Provides global control over database connections for an application. Delphi automatically creates a default TSession component at runtime for applications which use database controls.
TBatchMove	Enables you to perform operations on groups of records or entire tables.
TUpdateSQL	Provides a way to use Delphi's cached updates support with read-only datasets. For example, you could use a TUpdateSQL component with a "canned" query to provide a way of updating the underlying datasets, essentially giving you the ability to post updates to a read-only dataset.
TReport	Provides an interface to Borland's ReportSmith application. Once you place the TReport component on a form, you can double-click it to begin running ReportSmith.

Data Controls Page Components

For complete information about using Delphi components, locate the appropriate TComponent name in Delphi.hlp

The components on the Data Controls page of the Component palette make specialized database control elements available to your Delphi applications. The following illustration shows the Data Controls components.

TDBGrid	Accesses the data in a database table or query and displays it in a grid. Your application can use the data grid to insert, delete, or edit data in the database, or simply to display it.
TDBNavigator	(a database navigator) moves through the data in a database table or query, and performs operations on the data, such as inserting a blank record or posting a record.
TDBText	Displays text on a form in a data-aware control.
TDBEdit	Data-aware edit box with all the capabilities of an ordinary edit box (a TEdit component).
TDBMemo	Displays text for the user and permits the user display and enter data into a field much like a TDBEdit component. The TDBMemo component permits multiple lines to be entered or displayed, including text BLOBs (binary large objects).
TDBImage	Displays a graphic image from a BLOB (binary large object) stored in a field of the current record of a dataset.
TDBListBox	Data-aware list box. It allows the user to change the value of the field of the current record in a dataset by selecting an item from a list.
TDBComboBox	Data-aware combo box control. It allows the user to change the value of the field of the current record in a dataset either by selecting an item from a list or by typing in the edit box part of the control.
TDBCheckBox	Presents an option to the user; the user can check it to select the option, or uncheck it to deselect the option. A database check box (TDBCheckBox) is much like an ordinary check box (TCheckBox), except that it is aware of the data in a particular field of a dataset.
TDBRadioGroup	Displays a group of data-aware radio buttons. Only one of the radio buttons can be selected at a time, so the radio buttons present a set of mutually exclusive choices.
TDBLookupList	Provides the user with a convenient list of lookup items for filling in fields that require data from another dataset.
TDBLookupComboBox	Provides the user with a convenient drop-down list of lookup items for filling in fields that require data from another dataset.
TDBCtrlGrid	Displays multiple records from a data source. Unlike the TDBGrid component, which displays each record in a single row, you control the layout and appearance of each record in a TDBCtrlGrid.

System Page Components

For complete information about using Delphi components, locate the appropriate TComponent name in Delphi.hlp

The components on the System page of the Component palette make specialized system control elements available to your Delphi applications. The following illustrations show the System components.

TTimer	Causes an OnTimer event to occur whenever a specified period of time passes.
TPaintBox	Provides a way for your application to draw on the form in a specified rectangular area, preventing drawing outside of the boundaries of the paint box.
TFileListBox	Lists all the files in the current directory. To display files in a different directory, change the value of the Directory property.
TDirectoryListBox	A list box that is aware of the directory structure of the current drive.
TDriveComboBox	A combo box that displays all the drives available when the application runs.
TFilterComboBox	A combo box that is used to present the user with a choice of file filters.
TMediaPlayer	Controls devices that provide a Media Control Interface (MCI) driver. The component is a set of buttons (Play, Stop, Eject, and so on) that controls a multimedia device such as a CD-ROM drive, a MIDI sequencer, or a VCR.
TOleContainer	Embeds or links OLE objects in your Delphi 2.0 application.
TDdeClientConv	Establishes a DDE conversation with a DDE server application.
TDDEClientItem	Defines the item of a DDE conversation.
TDDEServerConv	Establishes a DDE conversation with a DDE client application.
TDDEServerItem	Defines the item of a DDE conversation