Properties:

RulerVisible	**Boolean**
Use the RulerVisible property to control the visibility of the rulers at runtime or design time. If RulerVisible is True, the rulers appear. If RulerVisible is False, the rulers are not visible.	
Defined in PCPanel unit	

PanelVisible	**Boolean**	
Use the PanelVisible property to control the visibility of the bottom panel at runtime or design time. If PanelVisible is True, the panel appears. If PanelVisible is False, the panel is not visible.		
Defined in PCPanel unit		

RulerSystem	**TRulerSystem = (rsMetric, rsWhitworth)**
Use the RulerSystem property to define the unit system of the rulers. If RulerSystem is rsMetric, the ruler is drawn in metric unit system. If RulerSystem is rsWhitworth, the ruler is drawn in inch unit system. *But note that, this doesn't effect the dimension unit used in the methods or commands. It is always in dmm (deci milimeter) (10 dmm = 1 mm)*	
Defined in PCPanel unit	

VerticalZero	**TVertZero = (vzBottom,vzTop)**
Use the VerticalZero property to define where the vertical ruler start. This also defines in which direction the *y* value increases or decreases. If VerticalZero is vzBottom, the ruler starts at the bottom of the page and increases to the top. If VerticalZero is vzTop, the ruler starts at the top of the page and increases to the bottom. All methods and commands are affected from this property. Ex: A drawing which is drawn in vzBottom will be vertically mirrored when reopened in a vzTop control. So this property should be fixed for an application.	
Defined in PCPanel unit	

HorizontalZero	**THorzZero = (vzLeft,vzRight)**
Use the HorizontalZero property to define where the horizontal ruler start. This also defines in which direction the *x* value increases or decreases. If HorizontalZero is vzLeft, the ruler starts at the left of the page and increases to the right. If HorizontalZero is vzRight, the ruler starts at the right of the page and increases to the left. All methods and commands are affected from this property. Ex: A drawing which is drawn in vzRight will be horizontally mirrored when reopened in a vzLeft control. So this property should be fixed for an application.	
Defined in PCPanel unit	

AutoRefresh	**Boolean**

Use the AutoRefresh property wheather the CadControl will refresh when figures are added through method calls. If Autorefresh is true then in each figure insertion PowerCad will refresh the drawing, but if it is false then PowerCad will refresh when the Refresh method is called. It is mostly used to prevent screen flicker when adding too much figures.

```
//Example:
PowerCad1.AutoRefresh := False;
For i := 1 to 1000 do
  Powercad1.Line(0,i,10,i,200,1,1,clBlack,0);
Powercad1.AutoRefresh := True;
PowerCad1.Refresh;
```

Defined in PCPanel unit

ToolIdx	TPCTool = (toSelect,toZoom,toInsertBlock, toInsertCurrentBlock,toFigure,toOperation)

Use the ToolIdx property to define the behaviour of the mouse commands.

toSelect: the mouse click will select the clicked figure.
Ex : PowerCad1.ToolIdx := toSelect;

toZoom: the mouse clik will zoom the drawing.
Ex : PowerCad1.ToolIdx := toZoom;

toInsertBlock: the mouse click will insert the block that is selected from the coming open dialog.
Ex : PowerCad1.ToolIdx := toInsertBlock;

toInsertCurrentBlock: the mouse click will insert the block that is assigned to the **CurrentBlock** property. (The path of the block file should be assigned to the CurrentBlock property)
Ex :

```
PowerCad1.ToolIdx := toInsertCurrentBlock;
Powercad1.CurrentBlock := 'c:\Blocks\Vane.pwb';
```

toFigure: the mouse clicks draw the figure that is assigned to the **CurrentFigure** property. (The class name should be assigned to the CurrentFigure property)
Ex :

```
PowerCad1.ToolIdx := toFigure;
Powercad1.CurrentFigure := 'TLine';
```

toOperation: the mose clicks execute the operation (like rotate, mirror, etc.) that is assigned to the CurentFigure. (The class name should be assigned to the CurrentFigure property)

Ex :

```
PowerCad1.ToolIdx := toOperation;
Powercad1.CurrentFigure := 'TRotate';
```

Defined in PCDrawBox unit

DotsPerMilOrig	Extended
Use the DotsPerMilOrig to define the pixel count of one mm when the drawing is in original size (scale = 100). The value is 4 by default. **Ex:** PowerCad1.DotsPerMilOrig := 8;	
Defined in PCDrawBox unit	

Surface	TPaintBoxExt
Use the Surface property to access the paintbox class which the drawing is painted on. **Ex:** **with** PowerCad1 **do** Form1.Canvas.CopyRect(DestRect,Surface.Canvas,SourceRect);	
Defined in PCDrawBox unit	

Scale	Integer
Use the Scale property to define the zoom factor of the drawing. The scale property is based on percent calculation, for actual size use 100, for half size use 50 and for double size use 200, etc. **Ex:** PowerCad1.Scale := 75;	
Defined in PCDrawBox unit	

GuidesVisible	Boolean
Use the **GuidesVisible** property to show or hide the guide lines of the drawing.The guides are not printed. **Ex:** PowerCad1.GuidesVisible := True;	
Defined in PCDrawBox unit	

BackGround	TColor
Use the **BackGround** property to define the background color of the drawing page. The background color is not printed. **Ex:** PowerCad1.BackGround := clWhite;	
Defined in PCDrawBox unit	

Grids	Boolean
Use the **Grids** property to show or hide the grids of the drawing page. The grids are not printed. **Ex:** PowerCad1.BackGround := clWhite;	
Defined in PCDrawBox unit	

GridColor	TColor
Use the **GridColor** property to define the grid color of the drawing page. **Ex:** PowerCad1.GridColor := clSilver;	
Defined in PCDrawBox unit	

GuideColor	TColor
Use the **GuideColor** property to define the color of the guide lines. **Ex:** PowerCad1.GuideColor := clGreen;	
Defined in PCDrawBox unit	

GridStep	Integer
Use the **GridStep** property to define the space between the grids. The GridStep value is defined in dmm unit. **Ex:** PowerCad1.GridStep := 50; // 5 mm	
Defined in PCDrawBox unit	

WorkHeight	Integer
Use the **WorkHeight** property to define the height of the drawing page. The WorkHeight value is defined in dmm unit. **Ex:** PowerCad1.WorkHeight := 2000; //20 cm	
Defined in PCDrawBox unit	

WorkWidth	Integer
Use the **WorkWidth** property to define the width of the drawing page. The WorkWidth value is defined in dmm unit. **Ex:** PowerCad1.WorkWidth := 2000; //20 cm	
Defined in PCDrawBox unit	

PageLayout	TPageLayOut = (plA0, plA1, plA2, plA3, plA4, plA5, plA6, plB4, plB5, plTabloid, plLetter, plCustom);
Use the **PageLayout** property to define the layout of the drawing page in technical standarts. The page layout also defines the WorkWidth and WorkHeight properties. **Ex:** PowerCad1.PageLayout := plA4;	
Defined in PCDrawBox unit	

PageOrient	TPageOrient = (poLandscape , poPortrait);
Use the **PageOrient** property to define the orientation of the drawing page. **Ex:** PowerCad1.PageLayout := poLandscape;	
Defined in PCDrawBox unit	

GuideTrace	TGuideTraces = (gtNone, gtNinty, gtThirty, gtFortyFive, gtSixty);
Use the **GuideTrace** property to define the angle of the cursor guides. To hide the CursorGuides use gtNone value. **Ex:** PowerCad1.GuideTrace := gtNinty;	
Defined in PCDrawBox unit	

SnapToGuides	Boolean
Use the **SnapToGuides** property to snap the cursor to guide lines. **Ex:** PowerCad1.SnapToGuides := True;	
Defined in PCDrawBox unit	

SnapToGrids	Boolean
Use the **SnapToGrids** property to snap the cursor to grid lines. **Ex:** PowerCad1.SnapToGrids := True;	
Defined in PCDrawBox unit	

SnapToNearPoint	Boolean
Use the **SnapToNearPoint** property to snap the cursor to nearest figure point. When this property is true, PowerCad also fires the OnFigureSnap event to provide user custom snapping. **Ex:** PowerCad1.SnapToNearPoint := True;	
Defined in PCDrawBox unit	

Scrollbars	Boolean
Use the **Scrollbars** property to show or hide the scrollbars of the drawing editor. **Ex:** PowerCad1.Scrollbars := True;	
Defined in PCDrawBox unit	

RealScale	Boolean

Use the **RealScale** property to define the interpretion of the diemnsions given in the command bar. When RealScale is true, the coordinates given in the command bar are considered as the rel size of figure in cm, so they are converted to drawing page coordinates before creating the figure by using the mapscale.
 	
Ex: PowerCad1.RealScale := True;
PowerCad1.MapScale := 50;
PowerCad1.ExecuteCommand('Line 0,0,50,0');
// The Command means: Draw a horizontal line from 0,0 in 50
// cm length in real sizes. The map scale is 50 so 50 cm
// will be shown as 1 cm on the paper. And 1 cm is 100 dmm.
// So the line coordiantes on the paper will be 0,0,100,0	

Defined in PCDrawing unit	

DEngine	TPCDrawEngine
Use the **DEngine** property to access the DrawEngine class of the PowerCad. You can make custom calls to this class to make drawings without creating figures. **Ex:**PowerCad1.DEngine.drawtext(0,0,0,'Test',xFont, 50, 0.8);	
Defined in PCDrawing unit	

Layers	TList
Use the **Layers** property to access the list of the layer classes. **Ex:** MyLayer := TLayer(PowerCad1.Layers[0]);	
Defined in PCDrawing unit	

Figures	TList

Use the **Figures** property to access the list of the figure classes.
 	
Ex:
For a := 0 to Powercad1.Figures.Count-1 **do**
begin
  MyFigure := TFigure(Powercad1.Figures[a]);
  if myFigure **is** TBlock **then** inc(cnt);
end;
ShowMessage(' There are ' + inttostr(cnt) + ' blocks');	

Defined in PCDrawing unit	

FPPage	TObjectInspector
Use the **FPPage** property to access the Property Page (Object Inspector) of the PowerCad. **Ex:** Powercad1.FPPage.Show;	
Defined in PCDrawing unit	

CurrentFigure	String
Use the **CurrentFigure** property to define the figure which will be drawn when the toolidx is toFigure, or to define which operation will be held when the tooidx is toOperation. **Ex:** PowerCad1.ToolIdx := toFigure; Powercad1.CurrentFigure := 'TLine'; **// or** PowerCad1.ToolIdx := toOperation; Powercad1.CurrentFigure := 'TRotate';	
Defined in PCDrawing unit	

CurrentBlock	String

Use the **CurrentBlock** property to define the block which will be inserted when the toolidx is toInsertCurrentBlock. So in this way you can assign a specific block to a toolbar button or a menu command.

```
procedure TForm1.InsertVaneClick(Sender: TObject);
begin
  PowerCad1.ToolIdx := toInsertCurrentBlock;
  PowerCad1.CurrentBlock := 'c:\blocks\vane.pwb';
end;
```

Defined in PCDrawing unit

Selection	TList

Use the **Selection** property to access the list of the figures that are selected.

```
Ex:
For a := 0 to Powercad1.Selection.Count-1 do
begin
  MyFigure := TFigure(Powercad1.Selection[a]);
  if myFigure is TLine then inc(cnt);
end;
ShowMessage('There are ' + inttostr(cnt) + ' lines');
```

Defined in PCDrawing unit

ActiveLayer	TList

Use the **ActiveLayer** property to define the layer to which the drawing commands will be applied.

Ex: PowerCad1.Activelayer := 0; // Base layer

Defined in PCDrawing unit

DefaultPenColor	TColor

Use the **DefaultPenColor** property to define the color of the default pen. This default value is used when a figure is drawn by mouse clicks or macros.

Ex: PowerCad1.DefaultPenColor := clBlack;

Defined in PCDrawing unit

DefaultPenWidth	Integer

Use the **DefaultPenWidth** property to define the width of the default pen. This default value is used when a figure is drawn by mouse clicks or macros.

Ex: PowerCad1.DefaultPenWidth := 1;

Defined in PCDrawing unit

DefaultPenStyle	TPenStyle
Use the **DefaultPenStyle** property to define the style of the default pen. This default value is used when a figure is drawn by mouse clicks or macros. **Ex:** PowerCad1.DefaultPenWidth := 1;	
Defined in PCDrawing unit	

DefaultRowStyle	TRowStyle
Use the **DefaultRowStyle** property to define the row style of the default pen when drawing lines or polylines. This default value is used when a line or polyline is drawn by mouse clicks or macros. **Ex:** PowerCad1.DefaultRowStyle := rsNone;	
Defined in PCDrawing unit	

DefaultBrushColor	TColor
Use the **DefaultBrushColor** property to define the color of the default brush. This default value is used when a figure is drawn by mouse clicks or macros. **Ex:** PowerCad1.DefaultBrushColor := clWhite;	
Defined in PCDrawing unit	

DefaultBrushStyle	TBrushStyle
Use the **DefaultBrushStyle** property to define the style of the default brush. This default value is used when a figure is drawn by mouse clicks or macros. **Ex:** PowerCad1.DefaultBrushStyle := bsSolid;	
Defined in PCDrawing unit	

BlockDirectory	String
Set the **BlockDirectory** property with the path of your blocks folder. **Ex:** PowerCad1.BlockDirectory := 'c:\MyBlocks\';	
Defined in PCDrawing unit	

PluginDirectory	String
Set the **PluginDirectory** property with the path of your plugins folder. **Ex:** PowerCad1.PluginDirectory := 'c:\MyPlugins\';	
Defined in PCDrawing unit	

LayerCount	Integer
Use this ReadOnly **LayerCount** property to het the number of the layers in the drawing. **Ex:** cnt **:=** PowerCad1.LayerCount;	
Defined in PCDrawing unit	

MapScale	Integer
Use the **MapScale** property to define the mapping Scale of your drawing. The mapping sclae of a drawing is the ration of the figure sizes (on paper) to the real world sizes. The mapsclae value is used in Dimensioning Lines. **Ex:** // If your mapping sclae is 1/50 PowerCad1.MapScale := 50;	
Defined in PCDrawing unit	

AutSelect	Boolean
Use the **AutoSlect** property to define the selection of a figure drawen by mouse clicks. if it is true, then the figure will be slected just after it is drawen. **Ex:** PowerCad1.AutoSelect := true;	
Defined in PCDrawing unit	

Font	TFont
Use the **Font** property to access the default font of the control. **Ex:** PowerCad1.Font.Name:= 'Arial';	
Defined in PCDrawing unit	

MouseCommands	Boolean
Use the **MouseCommands** property to define the behaviour of PowerCad to mouse clicks. If it is set to false, the mouse clicks will not ne interpretted as commands. If you want to handle mouse clicks by your own, the set this property to false. **Ex:** PowerCad1.MouseCommands := False;	
Defined in PowerCad unit	

KeyCommands	Boolean
Use the **KeyCommands** property to define the behaviour of PowerCad to key strokes. If it is set to false, the key strokes will not ne interpretted as commands. If you want to handle key strokes by your own, the set this property to false. By default this property is true, and the key strokes are checked if they mean for a specific action before they are fired as key event. **Ex:** PowerCad1.KeyCommands := False;	
Defined in PowerCad unit	

Methods

SetZoomHints

Procedure SetZoomHints(Hintlist: TStringList);

Use the **SetZoomHints** method to define the hints (tooltips) of the zoom buttons in the buttom panel.

Parameter Definitions:

HintList : The StringList that includes the Hint strings. The count should be equal to 5.

Ex:

```
Hlist := TStringlist.Create;
HList.Add('Zoom In');
HList.Add('Zoom Out');
HList.Add('Zoom Rect');
HList.Add('Actual Size');
HList.Add('FitToPage');
PowerCad1.SetZoomHints(HList);
HList.Free;
```

Defined in PCPanel unit

ZoomArea

Procedure ZoomArea(ZoomRect:TRect);

Use the **ZoomArea** method to zoom the screen to a specific location.

Parameter Definitions:

ZoomRect: The Trect that includes the screen locations to zoomed. The coordinates should be in dmm unit.

Ex:

```
ZRect := Rect(100,200,300,400);
PowerCad1.ZoomArea(ZRect);
```

Defined in PCDrawBox unit

FitToWindow

Procedure FitToWindow;

Use the **FitToWindow** method to fit entire page to the screen. The scale property value will be calculated automatically.

Ex:

```
PowerCad1.FitToWindow;
```

Defined in PCDrawBox unit

ConvertXY

Procedure ConvertXY(var X,Y: integer);

Use the **ConvertXY** method to convert a drawing point coordinate to pixel coordinate of the Surface paintbox. Normally you will not need to use this method.

Parameter Definitions:

X: The x coordinate of the point to be converted to pixel
Y: The y coordinate of the point to be converted to pixel

Ex:

```
x1 := 100; y1 := 100;
x2 := 200; y2 := 200;
PowerCad1.ConvertXY(x1,y1);
PowerCad1.ConvertXY(x2,y2);
PowerCad1.Surface.Canvas.MoveTo(x1,y1);
PowerCad1.Surface.Canvas.LineTo(x2,y2);
```

Defined in PCDrawBox unit

ConvertDim

Procedure ConvertDim(var Dim: integer);

Use the **ConvertDim** method to convert a drawing dimension (length,radius,etc.) to pixel length in the Surface paintbox. Normally you will not need to use this method.

Parameter Definitions:

Dim : The value to be converted to pixel count

Ex:

```
x1 := 100; y1 := 100;
r := 100;
PowerCad1.ConvertXY(x1,y1);
PowerCad1.ConvertDim(r);
PowerCad1.Surface.Canvas.MoveTo(x1,y1);
PowerCad1.Surface.Canvas.LineTo(x1+r,y1+r);
```

Defined in PCDrawBox unit

DeConvertXY

Procedure DeConvertXY(var X,Y: integer);

Use the **DeConvertXY** method to convert a pixel coordinate of Surface paintbox to drawing drawing point coordinate in dmm units. Normally you will not need to use this method.

Parameter Definitions:

X: The x coordinate of the point to be converted to dmm
Y: The y coordinate of the point to be converted to dmm

Ex:

```
x1 := 0; y1 := 0;
x2 := Powercad1.Surface.Width;
y2 := Powercad1.Surface.Height;

PowerCad1.DeConvertXY(x1,y1);
PowerCad1.DeConvertXY(x2,y2);
PowerCad1.Rectangle(0,x1,y1,x2,y2,1,0,255,0,0,true);
```

Defined in PCDrawBox unit

DeConvertDim

Procedure DeConvertDim(var Dim: integer);

Use the **DeConvertDim** method to convert a pixel length of Surface paintbox to drawing dimension in dmm unit. Normally you will not need to use this method.

Parameter Definitions:

Dim : The value to be converted to dmm

Ex:

```
x1 := 25; y1 := 25;
r := 50;
PowerCad1.ConvertXY(x1,y1);
PowerCad1.ConvertDim(r);
PowerCad1.Circle(0,x1,y1,r,1,0,255,0,0,true);
```

Defined in PCDrawBox unit

DeConvertDim

Procedure DeConvertDim(var Dim: integer);

Use the **DeConvertDim** method to convert a pixel length of Surface paintbox to drawing dimension in dmm unit. Normally you will not need to use this method.

Parameter Definitions:

Dim : The value to be converted to dmm

Ex:

```
x1 := 25; y1 := 25;
r := 50;
PowerCad1.ConvertXY(x1,y1);
PowerCad1.ConvertDim(r);
PowerCad1.Circle(0,x1,y1,r,1,0,255,0,0,true);
```

Defined in PCDrawBox unit

NewLayer

Function NewLayer(LayerName:string):Integer;

Use **NewLayer** method to create a new layer.

Parameter Definitions:

LayerName: The name of the layer to be created.

ReturnHandle : The Index of the layer in the Layers list.

Ex:

```
Index := PowerCad1.NewLayer('Mylayer');
Layer := TLayer(PowerCad1.Layers[Index]);
```

Defined in PCDrawing unit

GetLayerNbr

Function GetLayerNbr(LayerName: string):Integer;

Use **GetLayerNbr** method to get the list index of a layer by using its name.

Parameter Definitions:

LayerName: The name of the layer..

ReturnHandle : The list index of the layer.

Ex:

```
lNbr :=  PowerCad1.GetLayerNbr('Mylayer');
Powercad1.HideLayer(lNbr);
```

Defined in PCDrawing unit

DeleteLayer

Function DeleteLayer(LayerName: string): boolean;

Use **DeleteLayer** method to delete layer from the list. Deleting a layer will also delete the figures that belong to the deleted layer.

Parameter Definitions:

LayerName: The name of the layer to be deleted.

ReturnHandle : Returns **true** if deletion is succeded.

Ex:

```
if PowerCad1.DeleteLayer('Mylayer') then
  ShowMessage('Layer is deleted');
```

Defined in PCDrawing unit

DeleteLayerWithNbr

Function DeleteLayerWithNbr(LayerNbr:Integer): boolean;

Use **DeleteLayerWith** method to delete layer from the list with the layer index. Deleting a layer will also delete the figures that belong to the deleted layer.

Parameter Definitions:

LayerNBr: The list index of the layer to be deleted.

ReturnHandle : Returns **true** if deletion is succeded.

Ex:

```
if PowerCad1.DeleteLayerWithNbr(1) then
  ShowMessage('Layer is deleted');
```

Defined in PCDrawing unit

DeleteAllUserLayers

Procedure DeleteAllUserLayers;

Use **DeleteAllUserLayers** method to delete the layers that are created by your application. All layers that are not base layer are user layers.

Ex:

```
PowerCad1.DeleteAllUserLayers;
```

Defined in PCDrawing unit

ShowLayer

Procedure ShowLayer(LayerNbr:Integer);

Use **ShowLayer** method to make a layer visible.

Parameter Definitions:

LayerNBr: The list index of the layer to be shown.

Ex:

```
PowerCad1.ShowLayer(1);
```

Defined in PCDrawing unit

HideLayer

Procedure HideLayer(LayerNbr:Integer);

Use **HideLayer** method to make a layer invisible.

Parameter Definitions:

LayerNBr: The list index of the layer to be hidden.

Ex:

```
PowerCad1.HideLayer(1);
```

Defined in PCDrawing unit

FlueLayer

Procedure FlueLayer(LayerNbr:Integer);

Use **FlueLayer** method to make a layer grayed. A grayed layer is drawn in gray color, so that other layers are more clear to view.

Parameter Definitions:

LayerNBr: The list index of the layer to be hidden.

Ex:

```
PowerCad1.FlueLayer(1);
```

Defined in PCDrawing unit

ExFlueLayer

Procedure ExFlueLayer(LayerNbr:Integer);

Use **ExFlueLayer** method to make all layers grayed but not one. A grayed layer is drawn in gray color, so that other layers are more clear to view. Use ExFlueLayer when you want focus on one layer and flue all others.

Parameter Definitions:

LayerNBr: The list index of the layer to be exflued.

Ex:

```
PowerCad1.ExFlueLayer(1);
```

Defined in PCDrawing unit

ExHideLayer

Procedure ExHideLayer(LayerNbr:Integer);

Use **ExHideLayer** method to make all layers invisible but not one. Use ExHideLayer when you want focus on one layer and hide all others.

Parameter Definitions:

LayerNBr: The list index of the layer to be exflued.

Ex:

```
PowerCad1.ExHideLayer(1);
```

Defined in PCDrawing unit

HideAllLayers

Procedure HideAllLayers;

Use **HideAllLayers** method to make all layers invisible.

Ex:

```
PowerCad1.HideAllLayers;
```

Defined in PCDrawing unit

HideAllLayers

Procedure ShowAllLayers;

Use **HideAllLayers** method to make all layers visible.

Ex:

```
PowerCad1.ShowAllLayers;
```

Defined in PCDrawing unit

MergeAllLayers

Procedure MergeAllLayers;

Use **MergeAllLayers** method to merge all layers in the Base Layer.

Ex:

```
PowerCad1.MergeAllLayers;
```

Defined in PCDrawing unit

MergeVisibleLayers

Procedure MergeVisibleLayers;

Use **MergeVisibleLayers** method to merge all visible layers in the first visible layer.

Ex:

```
PowerCad1.MergeVisibleLayers;
```

Defined in PCDrawing unit

GetLayerInfo

Function GetLayerInfo(LayerNbr:Integer): TLayerInfo;

Use **GetLayerInfo** method to get all information about a layer.

Parameter Definitions:

LayerNBr: The list index of the layer .

ReturnHandle : Returns a comapct information in TLayerInfo type.

```
TLayerInfo = record
  Name : String; // Name of the layer
  Visible : Boolean; // True if layer is visible
  Flue  : Boolean; // True if layer is grayed
End;
```

Ex:

```
LInfo := PowerCad1.GetLayerInfo(4);
Mes := LInfo.Name + ' Layer';
if LInfo.Visible then
  Mes := Mes + ' is visible and'
else
  Mes := Mes + ' is invisible and';
if LInfo.Flue then
  Mes := Mes + ' is grayed'
else
  Mes := Mes + ' is not grayed';
ShowMessage(Mes);
// Vanes Layer is visible and not grayed.
```

Defined in PCDrawing unit

Undo

```
Procedure Undo;
```

Use **Undo** method to return to the state just before the last action. Powercad allows unlimited undos.

Ex:

```
PowerCad1.Undo;
```

Defined in PCDrawing unit

Redo

```
Procedure Redo;
```

Use **Redo** method to redo the action which is undone by an unco call.

Ex:

```
PowerCad1.Redo;
```

Defined in PCDrawing unit

ClearUndoList

```
Procedure ClearUndoList;
```

Use **ClearUndoList** method to clear the list that the actions are recorded to be undone. If you clear the undo list you can not undo the actions that are before this method call. Normally, PowerCad clears undo list only when saving the drawing.

Ex:

```
PowerCad1.ClearUndoList;
```

Defined in PCDrawing unit

DrawFigures

```
Procedure DrawFigures;
```

Use **DrawFigures** method to draw the figures within the inner DrawEngine of PowerCad. This method is usefull when you want to draw figures to a different canvas with different conversion settings.

Ex:

```
  PowerCad1.DEngine.Canvas := Form1.Canvas;
  PowerCad1.DEngine.ConvertPoint := CPointProc;
  PowerCad1.DEngine.ConvertLen := CDimProc;
  PowerCad1.DEngine.DeConvertPoint := DeCPointProc;
  PowerCad1.DEngine.DeConvertLen := DeCDimProc;
  Powercad1.DrawFigures;
```

Defined in PCDrawing unit

DrawSelectedFigures

Procedure DrawSelectedFigures;

Use **DrawSelectedFigures** method to draw the selected figures within the inner DrawEngine of PowerCad. This method is usefull when you want to draw selected figures to a different canvas with different conversion settings.

Ex:

```
PowerCad1.DEngine.Canvas := Form1.Canvas;
PowerCad1.DEngine.ConvertPoint := CPointProc;
PowerCad1.DEngine.ConvertLen := CDimProc;
PowerCad1.DEngine.DeConvertPoint := DeCPointProc;
PowerCad1.DEngine.DeConvertLen := DeCDimProc;
Powercad1.DrawSelectedFigures;
```

Defined in PCDrawing unit

SelectAll

Procedure SelectAll(LayerNbr:Integer);

Use **SelectAll** method to select all the figures in the given layer. If you give the baselayer in the parameter, figures in all layers will be selected.

Parameter Definitions:

LayerNBr: The list index of the layer. Give 0 for all layers.

Ex:

```
PowerCad1.SelectAll(0);
```

Defined in PCDrawing unit

DeSelectAll

Procedure DeSelectAll(LayerNbr:Integer);

Use **DeSelectAll** method to deselect all the figures in the given layer. If you give the baselayer in the parameter, figures in all layers will be deselected.

Parameter Definitions:

LayerNBr: The list index of the layer. Give 0 for all layers.

Ex:

```
PowerCad1.DeSelectAll(0);
```

Defined in PCDrawing unit

InvertSelection

Procedure InvertSelection;

Use **InvertSelection** method to select the unselected figures and to deselect the selected figures.

Ex:

```
PowerCad1.InvertSelection;
```

Defined in PCDrawing unit

GroupSelection

Function GroupSelection:TFigHandle;

Use **GroupSelection** method to make a TFigureGrp object from the selected figures. Use the return handle to access the new group.

Parameter Definitions:

Return Value: The handle of the created TFigureGrp

Ex:

```
Handle := PowerCad1.GroupSelection;
TFigureGrp(Handle).Combined := True;
```

Defined in PCDrawing unit

UnGroupSelection

Procedure UnGroupSelection;

Use **UnGroupSelection** method to ungroup the figure groups that are in the selection. This method ungroups all groups that are selected.

Ex:

```
PowerCad1.UnGroupSelection;
```

Defined in PCDrawing unit

OrderSelection

Procedure OrderSelection(Dest: TOrderStyle);

Use **OrderSelection** method to change the z-order of the selected figures.

Parameter Definitions:

Dest: The order destination parameter. It can have following values of TOrderStyle.
  osBack : Sends the selected figures to the most back.
  osFront : Sends the selected figures to the most front.
  osBWard : Sends the selected figures to one step back.
  osFWard : Sends the selected figures to one step front.

Ex:
```
  PowerCad1.OrderSelection(osBack);
```

Defined in PCDrawing unit

RemoveSelection

Procedure RemoveSelection;

Use **RemoveSelection** method to delete the selected figures.

Ex:
```
  PowerCad1.RemoveSelection;
```

Defined in PCDrawing unit

RotateSelection

Procedure RotateSelection(Angle:Integer;rPoint: TPoint);

Use **RotateSelection** method to rotate the selected figures arround a center point with a given angle.

Parameter Definitions:

Angle : The rotation angle, the unit is 1/10 of degree.
        Ex: For 10 degrees use 100.
rPoint: The rotation center point.

Ex:
```
  PowerCad1.RotateSelection(100,Point(450,235));
```

Defined in PCDrawing unit

MirrorSelection

Procedure MirrorSelection(Point1,Point2: TPoint;Dupl:Boolean);

Use **MirrorSelection** method to reflect the selected figures across a given axis.

Parameter Definitions:

Point1,
Point2 : The points that define the reflection axis.
Dupl : When set to true, the selected objects are first duplicated and then the duplication is reflected, so that the original figures are saved.

Ex:

```
p1 := Point(100,100); p2 := Point(100,200);
PowerCad1.MirrorSelection(p1,p2,True);
```

Defined in PCDrawing unit

InvertArcsOfSelection

Procedure InvertArcsOfSelection;

Use **InvertArcsOfSelection** method to invert the direction of the selected arcs.

Ex:

```
PowerCad1.InvertArcsOfSelection;
```

Defined in PCDrawing unit

ArrangeArcStyleOfSelection

Procedure ArrangeArcStyleOfSelection(Value: Integer);

Use **ArrangeArcStyleOfSelection** method to arraneg the style of the selected arcs.

Parameter Definitions:

Value : The style that will be applied to the arcs. It can have following values.
0: Normal Arc (Open)
1: Pie Arc (Close)
2: Chord Arc (Close)

Ex:

```
PowerCad1.ArrangeArcStyleOfSelection(0);
```

Defined in PCDrawing unit

CloseSelectedPolyLine

Procedure CloseSelectedPolyline;

Use **CloseSelectedPolyLine** method to close the selected polyline. Only closed figures can have a region to be filled. A closed polyline will have additional segment from last point to the first point.

Ex:

```
PowerCad1. CloseSelectedPolyline;
```

Defined in PCDrawing unit

OpenSelectedPolyLine

Procedure OpenSelectedPolyline;

Use **OpenSelectedPolyLine** method to open the selected polyline. Open figures can't have a region to be filled.

Ex:

```
PowerCad1.OpenSelectedPolyline;
```

Defined in PCDrawing unit

ConvertToBezier

Procedure ConvertToBezier;

Use **ConvertToBezier** method to make curve all segments of the selected polyline(s).

Ex:

```
PowerCad1.ConvertToBezier;
```

Defined in PCDrawing unit

ConvertToPolyline

Procedure ConvertToPolyline;

Use **ConvertToPolyline** method to make line all segments of the selected polyline(s).

Ex:

```
PowerCad1.ConvertToPolyline;
```

Defined in PCDrawing unit

FlipImagesOfSelection

Procedure FlipImagesOfSelection(FlipMode:TFlipMode);

Use **FlipImagesOfSelection** method to flip the bitmap objects that are selected in a drawing.

Parameter Definitions:

FlipMode : Use this parameter to define the orientation of flipping.
fmVert : Use this to flip in vertical direction.
fmHorz : Use this to flip in horizontal direction.

Ex:

```
PowerCad1.FlipImagesOfSelection(fmVert);
```

Defined in PCDrawing unit

SetTransparentOfSelection

Procedure SetTransparentOfSelection (Transparent:Boolean);

Use **SetTransparentOfSelection** method to set the transparency of the selected bitmap objects.

Parameter Definitions:

Transparent : Set to **true** to make transparent, **false** to make opaque.

Ex:

```
PowerCad1.SetTransparentOfSelection(True);
```

Defined in PCDrawing unit

ScaleSelection

```
Procedure ScaleSelection(PercentX,PercentY:Integer;
                         rPoint:TPoint);
```

Use **ScaleSelection** method to redimension the selected figures. The figures' point coordinates will be recalculated according to the percent values and scaling center.

Parameter Definitions:

PercentX : The scale percantage in horizontal direction. Use 100 fro no sclaing, 50 for half size, 200 for souble size, etc.
PercentY : The scale percantage in vartical direction. Use 100 fro no sclaing, 50 for half size, 200 for souble size, etc.
rPoint : The scale center point. The new coordinates will be relocated according to the distance to this point. The percantage will be applied to this distance.

Ex:

```
PowerCad1.ScaleSelection(100,200,Point(0,0));
```

Defined in PCDrawing unit

ModifySelection

```
Procedure ModifySelection(mm: TModifyMode; Value: Integer);
```

Use **ModifySelection** method to modify the pen and brush attributes of the selected figures.

Parameter Definitions:

mm : Use this parameter to define which attribute of the figure will be modified. It can have following values.
mmPenColor: The pen color will be modified
mmPenWidth: The pen width will be modified
mmPenStyle: The pen style will be modified
mmRowStyle: The row style will be modified
mmBrushColor: The brush color will be modified
mmBrushStyle: The brush style will be modified
value : The new value of the attribute.

Remark: *Other values of TModifyMode are ignored in this method.*

Ex:

```
PowerCad1.ModifySelection(mmPenStyle,ord(psDash));
```

Defined in PCDrawing unit

ModifyTextAndFont

```
Procedure ModifyTextAndFont(mm: TModifyMode; ValueI: Integer;
          ValueS: String; ValueSt: TFontStyles);
```

Use **ModifyTextAndFont** method to modify the text and font attributes of the selected figures.

Parameter Definitions:

mm : Use this parameter to define which attribute of the figure will be modified. It can have following values.
mmText: The text will be modified
mmFontName: The font name will be modified
mmFontSize: The font size will be modified
mmFontCs: The font charset will be modified
mmFontColor: The font color will be modified
mmFontStyle: The font style will be modified

valueI : The new value of the attribute when it is Integer. Use this parameter when *mm* is set to mmFontSize, mmFontCs or mmFontColor.
valueS : The new value of the attribute when it is String. Use this parameter when *mm* is set to mmText, mmFontName.
valueSt: The new value of the attribute when it is fontstyle. Use this parameter when *mm* is set to mmFontStyle.

Remark: *Other values of TModifyMode are ignored in this method.*

Ex:

```
  PowerCad1.modifyTextAndFont(mmText, 'New Text');
```

Defined in PCDrawing unit

GetSelectionRect

```
Function GetSelectionRect:TRect;
```

Use **GetSelectionRect** method to get the bounding rectangle coordinates of the selected figures.

Parameter Definitions:

Retrun Value: The coordinates of the bounding rectnagle of all selected figures together in dmm unit.

Ex:

```
  xRect := PowerCad1.GetSelectionRect;
  with xrect do
    Powercad1.DEngine.DrawRect(left,top,right,bottom,clRed,1,
                               ord(psDash),0,ord(bsClear));
```

Defined in PCDrawing unit

AlignSelection

Prcedure AlignSelection(HorzAlign:THorzAligns;
VertAlign:TVertAligns);

Use **AlignSelection** method to align the selected figures in horizontal direction and vertical axis.

Parameter Definitions:

HorzAlign: The align type in horizontal axis. It can have following values.
haNoChange: No aligning in horizontal axe axis
haTop: Aligns the tops of the figures.
haBottom: Aligns the bottoms of the figures.
haCenter: Aligns the vertical centers on horizontal axis.
haDistHorz: Distrubutes the distances between figures equally in horizontal axis.
VertAlign: The align type in vertical axis. It can have following values.
vaNoChange: No aligning in vertical axis.
vaLeft: Aligns the right sides of the figures.
vaRight: Aligns the right sides of the figures.
vaCenter: Aligns the horizontal centers on vertical axis.
vaDistVert: Distrubutes the distances between figures equally in vertical axis.

Ex:

```
PowerCad1.AlignSelection(haTop,vaLeft);
```

Defined in PCDrawing unit

Refresh

Prcedure Refresh;

Use **Refresh** method to redraw the drawing. This method will not work if the AutoRefresh is set to false. *See ManualRefresh.*

Ex:

```
PowerCad1.Refresh;
```

Defined in PCDrawing unit

ManualRefresh

Prcedure ManualRefresh;

Use **ManualRefresh** method to redraw the drawing. This method will refresh the drawing regardless of AutoRefresh property.

Ex:

```
PowerCad1.ManualRefresh;
```

Defined in PCDrawing unit

CheckByPoint

Function CheckByPoint(LayerNbr,x,y: integer):TFigure;

Use **CheckByPoint** method to learn if a point is in/on a figure. Use this method in hit test.

Parameter Definitions:

LayerNbr : The list index of the layer that the check will be done. Use 0 for all layers.
x,y : The coordinates of the test point to be checked.

Return Value: Returns the class reference of the hit figure. If the point stands for no figure then the result is nil.

Ex:

```
xFig := PowerCad1.CheckByPoint(0,x,y);
if (xFig <> nil) then xFig.Color := clRed;
```

Defined in PCDrawing unit

SelectByPoint

Function SelectByPoint(LayerNbr,x,y: integer;
ShiftPressed:Boolean): Boolean;

Use **SelectByPoint** method to select a figure by the given point. The figure is selected if the point is in/on the figure..

Parameter Definitions:

LayerNbr : The list index of the layer that the selection will be done. Use 0 for all layers.
x,y : The coordinates of the selecting point.
ShiftPressed : if this parameter is true, then the selected figure is added to the selection. Otherwise already selected figures are deselected and then the new selection is made.

Return Value: Returns **true** if any figure can be selected.

Ex:

```
PowerCad1.SelectByPoint(0,x,y,false);
```

Defined in PCDrawing unit

SelectWithInArea

Function SelectWithInArea(LayerNbr: integer; Area: TRect;
ShiftPressed: Boolean): Boolean;

Use **SelectWithInArea** method to select figure or figures that are with in the borders of the given rectangular area.

Parameter Definitions:

LayerNbr : The list index of the layer that the selection will be done. Use 0 for all layers.
Area : The Selection Area in TRect type. The Rect values should be in dmm.
ShiftPressed : if this parameter is true, then the selected figures are added to the selection. Otherwise already selected figures are deselected and then the new selection is made.

Return Value: Returns **true** if any figure can be selected.

Ex:

```
// This code below, selects all the figures that are in the
   limits of the drawing page.
ww :=  Powercad1.WorkWidth;
wh :=  Powercad1.WorkHeight;
PowerCad1.SelectWithInArea(0,Rect(0,0,ww,wh),false);
```

Defined in PCDrawing unit

MoveSelection

Procedure MoveSelection(deltaX,deltaY: integer);

Use **MoveSelection** method to move the the selected figures in horizontal and/or vertical directions.

Parameter Definitions:

deltaX : The movement distance in horizontal direction. Should be in dmm unit.
deltaY : The movement distance in vertical direction. Should be in dmm unit.

Ex:

```
PowerCad1.MoveSelection(100,120);
```

Defined in PCDrawing unit

DuplicateSelection

Procedure DuplicateSelection(deltaX,deltaY: integer);

Use **DuplicateSelection** method to clone the selected figures, and then move the new figures.

Parameter Definitions:

deltaX : The movement distance after cloning in horizontal direction. Should be in dmm unit.
deltaY : The movement distance after cloning in vertical direction. Should be in dmm unit.

Ex:

```
PowerCad1.DuplicateSelection(100,120);
```

Defined in PCDrawing unit

ArrayRectSelection

Procedure ArrayRectSelection(distanceX,distanceY: integer;
col,row: integer);

Use **ArrayRectSelection** method to clone the selected figures, as an rectangular array.

Parameter Definitions:

distanceX : The distance between the clones in horizontal direction. Should be in dmm unit.
distanceY : The distance between the clones in vertical direction. Should be in dmm unit.
col : The number of columns of the clone array
row : The number of rows of the clone array

Remark : The selected figure(s) will be cloned (col*row)-1 times.

Ex:

```
PowerCad1.ArrayRectSelection(50,50,4,3);
```

Defined in PCDrawing unit

ArrayPolarSelection

Procedure ArrayPolarSelection(cPoint:Tpoint; angle:integer);

Use **ArrayPolarSelection** method to clone the selected figures, as a circular (polar) array around a center point.

Parameter Definitions:

cPoint : The center point of the circlar array.
angle : the angle between tow clones. It should be given as 1/10 degrees. (for 10 degrees use 100)

Remark : Each clones is also rotated around its own center according to its angle.
The number of clones is equal to (3600 **div** angle)-1

Ex:

```
PowerCad1.ArrayPolarSelection(Point(50,50),450);
```

Defined in PCDrawing unit

MakeSelectionBlock

Procedure MakeSelectionBlock(FileName:**string**);

Use **MakeSelectionBlock** method to create a block file from the selected figures.

Parameter Definitions:

FileName : The file name of the block with its path.

Remark : To create a block, only one figure should be selected. To make a block from more than one figure, you should first group the selection.

Ex:

```
PowerCad1.MakeSelectionBlock('c:\Blocks\TestBlock.pwb')
```

Defined in PCDrawing unit

BoundLineToFigures

Procedure BoundLineToFigures(BLine,jf1,jf2:TFigure);

Use **BoundLineToFigures** method to join the ends of a Line or a Polyline to other figures. Bounded (Joined) Line-Ends move when the join figure moves.

Parameter Definitions:

BLine : The figure class reference or figure handle of the Line/PolyLine that will be bound.

jF1 : The figure class reference or figure handle of any figure that the starting point of the line/polyline will be joined to. Set this to **nil** if you don't want to join the start point.

jF2 : The figure class reference or figure handle of any figure that the ending point of the line/polyline will be joined to. Set this to **nil** if you don't want to join the ending point.

Ex:

```
PowerCad1.BoundLineToigures (myLine,myRect,myCircle);
```

Defined in PCDrawing unit

BoundLinePoint

Procedure BoundLinePoint(BLine:TFigure; SeqNbr:integer;
bPoint:Tpoint);

Use **BoundLinePoint** method to join one end of a Line or a Polyline to the figure in/on which the bPoint is.PowerCad first checks if any figure is selectable by bPoint, then bounds the end of a line to that figure from the bPoint.

Parameter Definitions:

BLine : The figure class reference or figure handle of the Line/PolyLine that will be bound.

SeqNbr : The indication for the bounding point of Line/PolyLine. Use 1 for the start point, 2 for the ending point.

bPoint : The hit test point that will defined the join figure and defines the bounding point of two figures.

Ex:

```
// This code takes the center of a circle as bounding point.
// Sets the ending point of a line to the bounding point.
// Then bounds the figures from this point.
bPoint := myCircle.FigurePoints[1];
MyLine.FigurePoints[2] := bPoint;
PowerCad1.BoundLinePoint (myLine,2,bPoint);
```

Defined in PCDrawing unit

BoundSelectedLine

Procedure BoundSelectedLine;

Use **BoundSelectedLine** method to join a line or polyline with any figure that the line ends or in/on. Powercad checks if the line ends of the selected line/polyline can hit on any figure and then bounds the hitting end to the hit figure.

Ex:

```
// To use this method first move the line as its end can hit
// any figure. The call this method. See that the line will
// be joined with the hit figure.
PowerCad1.BoundSelectedLine;
```

Defined in PCDrawing unit

UnboundLine

Procedure UnBoundLine;

Use **UnBoundLine** method to break the joins of any selected line/polyline if its bound to any figure.

Ex:

```
PowerCad1.UnBoundLine;
```

Defined in PCDrawing unit

MakeSelectedLinesPolyline

Function MakeSelectedLinesPolyline:TFigHandle;

Use **MakeSelectedLinesPolyline** method to create a polyline from the selected lines. The line ends form the knots of the polyline.

Parameter Definitions:

Return Value : The figure handle of the created polyline is returned.

Ex:

```
Handle := PowerCad1.MakeSelectedLinesPolyLine;
TPolyLine(Handle).Closed := True;
```

Defined in PCDrawing unit

ClipSelBitmapToSelFigure

Procedure ClipSelBitmapToSelFigure;

Use **ClipSelBitmapToSelFigure** method to clip bitmaps in closed regions. To make this, before calling this method, a bitmap and a close figure (circle,rectangle,ellipse or closed polyline) should be selected. The bitmap is drawn in the bounds of the closed figure.

Ex:

```
PowerCad1.ClipSelBitmapToSelFigure;
```

Defined in PCDrawing unit

UnClipSelBitmap

Procedure UnClipSelBitmap;

Use **UnClipSelBitmap** method to cancel the clipping of a clipped bitmap.

Ex:

```
PowerCad1.UnClipSelBitmap;
```

Defined in PCDrawing unit

SaveToFile

Procedure SaveToFile(LayerNbr: Integer; FileName:String);

Use **SaveToFile** method to save the drawing to a file in PowerCad's own file format. This method can save either a layer or all layers to the file.

Parameter Definitions:

LayerNbr : The list index of the layer to be saved. Use 0 for all drawing to be saved.

FileName : The name of the file with its path that the drawing will be saved to. If the file already exists, it is overwritten with out any prompt.

Ex:

```
if SaveDialog1.Execute then
  PowerCad1.SaveToFile(0,SaveDialog1.FileName);
```

Defined in PCDrawing unit

LoadFromFile

```
Procedure LoadFromFile(FileName:String);
```

Use **LoadFromFile** method to load a drawing from a file. The file should be saved with the SaveToFile method of PowerCad.

Parameter Definitions:

FileName : The name of the file with its path that the drawing will be loaded from.

Ex:

```
  if OpenDialog1.Execute then
    PowerCad1.LoadFromFile(OpenDialog1.FileName);
```

Defined in PCDrawing unit

SaveToStream

```
Procedure SaveToStream(Stream:TStream);
```

Use **SaveToStream** method to save the drawing to a Delphi TStream. The drawing is written to the stream from the current stream position. The stream can have some application specific data before or after the PowerCad drawing data.

Parameter Definitions:

Stream : The TStream object in which the drawing is stored.

Ex:

```
  if SaveDialog1.execute then
  begin
    fStream := TFileStream.Create(fmWrite,
                                  SaveDialog1.FileName);
    ApplicationData1.SaveToStream(fStream);
    PowerCad1.SaveToStream(fStream);
    ApplicationData2.SaveToStream(fStream);
    fStream.free;
  end;
```

Defined in PCDrawing unit

LoadFromStream

```
Procedure LoadFromStream(Stream:TStream);
```

Use **LoadFromStream** method to load a drawing from a Delphi TStream. The stream position should be in the location where the drawing is started to be written. If the stream is only for the Powercad drawing, then the position should be 0.

Parameter Definitions:

Stream : The TStream object in which the drawing is stored.

Ex:

```
  if OpenDialog1.execute then
  begin
    fStream := TFileStream.Create(fmOpenRead,
                                  OpenDialog1.FileName);
    ApplicationData1.LoadFromStream(fStream);
    PowerCad1.LoadFromStream(fStream);
    ApplicationData2. LoadFromStream(fStream);
    fStream.free;
  end;
```

Defined in PCDrawing unit

InsertBlockWithFileName

```
Function InsertBlockWithFileName(LayerNbr:Integer;
                                 FileName:string;
                                 x,y:Integer):TFigHandle;
```

Use **InsertBlockWithFileName** method to insert a block object from a file.

Parameter Definitions:

LayerNbr : The list index of the layer to which the block will be inserted.
FileName : The file name of the block with its path information.
x,y : The coordinates that the inserted block will be located on.

Return Value: The figure handle of the inserted block.

Ex:

```
  if OpenDialog1.execute then
  begin
    handle := PowerCad1.InsertBlockWithFileName(
                  Powercad1.ActivaLayer,
                  OpenDialog1.FileName,0,0);
    TBlock(Handle).Color := clRed;
  end;
```

Defined in PCDrawing unit

InsertBlockFromStream

```
Function InsertBlockFromStream(LayerNbr:Integer;
                               Stream:TStream;
                               x,y:Integer):TFigHandle;
```

Use **InsertBlockFromStream** method to insert a block object from a Delphi Stream. The stream position should be in the location where the block is started to be written.

Parameter Definitions:

LayerNbr : The list index of the layer to which the block will be inserted.
Stream : The Tstream object.
x,y : The coordinates that the inserted block will be located on.

Return Value: The figure handle of the inserted block.

Ex:

```
handle := PowerCad1.InsertBlockFromStream(
                  Powercad1.ActivaLayer,
                  blkStream,0,0);
TBlock(Handle).Color := clRed;
```

Defined in PCDrawing unit

ExportAsWmf

```
Procedure ExportAsWmf(FileName:String);
```

Use **ExportAsWmf** method to export the drawing as Windows Enhanced Metafile.

Parameter Definitions:

FileName : The name of the windows meta file that the drawing will be exported. If the file already exists, then the existing file will be overwritten.

Ex:

```
if SaveDialog1.Execute then
  PowerCad1.ExportAsWmf(SaveDialog1.Filename);
```

Defined in PCDrawing unit

SaveAsBitmap

Procedure SaveAsBitmap(FileName:String);

Use **SaveAsBitmap** method to export the drawing as Windows Bitmap.

Parameter Definitions:

FileName : The name of the windows bitmap that the drawing will be saved. If the file already exists, then the existing file will be overwritten.

Ex:

```
  if SaveDialog1.Execute then
    PowerCad1.SaveAsBitmap(SaveDialog1.Filename);
```

Defined in PCDrawing unit

SelectionAsWmf

Function SelectionAsWmf:TMetafile;

Use **SelectionAsWmf** method to get the selected figures to be drawn as metafile.

Parameter Definitions:

Return Value : The metafile object that the selction is drawn. This metafile is created by the function itself, but it should be freed by the function user after it is used.

Ex:

```
  mf := Powercad1.SelectionAsWmf;
  Form1.Canvas.Draw(0,0,mf);
  mf.free;
```

Defined in PCDrawing unit

DrawToDc

Procedure DrawToDc(dc,x,y,DScale:Integer);

Use **DrawToDc** method to draw the drawing to a different device context.

Parameter Definitions:

dc : The handle (HDC) of the device context that the drawing will be drawn.
x,y : The coordinates of location where the drawing will be drawn.
DScale : The scale value of the drawing. Use 100 for original size, 50 for half size, 200 for double. If the scale is 100; a 10 mm (100 dmm) long line will be drawn as 10*DotsPerMilOrig pixels. The DotsPerMilorig is 4 by default, you can also update it to change the scale.

Ex:

```
Powercad1.DrawToDc(Image1.Canvas.Handle,0,0,100);
```

Defined in PCDrawing unit

StretchToDc

Procedure StretchToDc(dc,aLeft,aTop,aright,aBottom:Integer);

Use **StretchToDc** method to draw the drawing to a different device context by fitting it to a given rectangular area.

Parameter Definitions:

dc : The handle (HDC) of the device context that the drawing will be drawn.

aLeft,atop,
aRight,aBottom : The coordinates of rectangle where the drawing will be drawn.

Ex:

```
Powercad1.StretchToDc(Image1.Canvas.Handle,
                      0,0, Image1.Width,Image1.Height);
```

Defined in PCDrawing unit

Print

Procedure Print(TitleInStatusBox:String);

Use **Print** method to print the drawing to any printer. This method uses the currently available printer settings.

Parameter Definitions:

TitleInStatusBox : This string will be shown in the printer status dialog indicating the job title.

Ex:

```
Powercad1.Print('PowerCad Drawing');
```

Defined in PCDrawing unit

PrintByTiling

Procedure PrintByTiling(TitleInStatusBox:String);

Use **PrintByTiling** method to print the drawing to multiple pages. This method uses the currently available printer settings.

Parameter Definitions:

TitleInStatusBox : This string will be shown in the printer status dialog indicating the job title.
prWmm : The width of the drawing portion that will be printed in one page.
prHmm : The height of the drawing portion that will be printed in one page.

Ex:

```
Powercad1.PrintbyTiling('PowerCad Drawing', 19,29);
```

Defined in PCDrawing unit

ImportDxf

Procedure ImportDxf(FileName:String;
Layered,IncVertex:Boolean);

Use **ImportDxf** method to import a AutoCad DXf file to your drawing. The DXf file is imported by using the freeware code of *John Biddiscombe*.

Parameter Definitions:

FileName : The FileName of the dxf file to be imported.
Layered : Set this true if you import the dxf objects in their own layers. When it is false, the objects will be all in one layer called DXFLayer.
IncVertes: Set this true if you want to import vertex objects.

Ex:

```
PowerCad1.ImportDXF('c:\graphs\House.dxf',False,False);
```

Defined in PCDrawing unit

Clear

Procedure Clear(LayerNbr:Integer);

Use **Clear** method to remove all objects of one layer or all drawing.

Parameter Definitions:

LayerNbr : The list index of the layer to be cleared. Use 0 for all drawing.

Ex:

```
PowerCad1.Clear(0);
```

Defined in PCDrawing unit

HitTestModPoint

Function HitTestModPoint(x,y:Integer):TModPoint;

Use **HitTestModPoint** to learn if any modification point is in the location of x,y.

Parameter Definitions:

x,y : The coordinates of the hit test point in dmm unit.

Return Value : The class reference of the ,mod-point that stands in x,y. Returns **nil** if there is no mod-point in the specified location.

Ex:

```
procedure TForm1.PowerCad1SurfaceTrace(Sender: TObject;
                                       Shift: TShiftState;
                                       X, Y: Integer);
var mp: TModPoint;
begin
  mp := Powercad1.HittestModPoint(x,y);
  if assigned(mp) then
    PowerCad1.Cursor := crCross
  else
    PowerCad1.Cursor = crDefault;
end;
```

Defined in PCDrawing unit

CopyToClipBoard

Procedure CopyToClipBoard;

Use **CopyToClipBoard** method to copy the selected objects to the clipboard. The objects are copied in PowerCad streaming format and Metafile format.

Ex:

```
  PowerCad1.CopyToClipboard;
```

Defined in PCDrawing unit

CutToClipBoard

Procedure CutToClipBoard;

Use **CutToClipBoard** method to cut the selected objects to the clipboard. The objects are copied in PowerCad streaming format and Metafile format.

Ex:

```
  PowerCad1.CutToClipboard;
```

Defined in PCDrawing unit

PasteFromClipBoard

Procedure PasteFromClipBoard(LayerNbr: Integer);

Use **PasteFromClipBoard** method to paste the contents of the clipboard to the drawing.Powercad first checks if any data in Powercad Streaming Format is in the clipboard. If PowerCad Data exists then PowerCad imports these objects to the drawing. If no PowerCad data exists in the clipboard, then PowerCad imports the metafile data if exists any.

Parameter Definitions:

LayerNbr : The list index of the layer where the clipboard data will be pasted.

Ex:

```
PowerSad1.PasteFromClipboard(Powercad1.ActiveLayer);
```

Defined in PCDrawing unit

FindFigureByName

Function FindFigureByName(FigName:String):TFigure;

Use **FindFigureByName** to get the object reference of a figure with its figure name.

Parameter Definitions:

FigName : The name of the figure

Return Value : The object reference of the figure.

Ex:

```
var Fig: TFigure;

Fig := PowerCad1.FindFigureByName('Line01');
Fig.Color := clRed;
```

Defined in PCDrawing unit

RefreshPropertyPage

Procedure RefreshPropertyPage;virtual;

Use **RefreshPropertyPage** to refresh the content of the object inspector (Property Page).

Ex:

```
PowerCad1.RefreshPropertyPage
```

Defined in PCDrawing unit

AddCustomProperty

```
Procedure AddCustomProperty(BlockName:String;
                            CustomProp:TProperty);
```

Use **AddCustomProperty** to add custom properties to the property page of a block class. When you add a custom property to a block class, all instances of that block will have the custom property in its property page. By using the OnPropertyChanged event you can get the data that the user has entered to this property.

Parameter Definitions:

BlockName : The name of the block that the property will be added.
CustomProp : The object reference of the custom property.

Ex:

```
  // This code, first creates a string property named specie
  // with a default value 'Bird'. Then this property is
  // registered to the block named 'Dove'.

  var cProp: TProperty;

  cProp := TStringProperty.Create('Specie',false,'Bird');
  PowerCad1.AddCustomProperty('Dove',cProp);
```

Defined in PCDrawing unit

GetCustomPropList

```
Function GetCustomPropList(BlockName:String):TList;
```

Use **GetCustomPropList** to get the TList object that stores the custom properties of a block class. The returning TList is the internally created list, so you shouldn't free it in anyway.

Parameter Definitions:

BlockName : The name of the block that the property list will be gathered.

Return Value: The object reference of the Custom Property List.

Ex:

```
  List := PowerCad1.GetCustomPropList('Dove');
  if assigned(List) then
    ShowMessage('Dove Block has '+ inttostr(List.Count) +
                'custom properties.');
```

Defined in PCDrawing unit

Line

Function Line(LayerNbr,x1,y1,x2,y2,w,s,c,row: integer;
Select:Boolean):TFigHandle;

Use **Line** method to create a Line figure and add it to Powercad.

Parameter Definitions:

LayerNbr : The list index of the layer that the figure will be inserted.

x1,y1,x2,y2 : The figure coordinates in dmm unit.

w : The pen width of the figure.

s : The pen style of the figure.It can have following values.
0: Solid Pen *// ord(psSolid)*
1: Dash Pen *// ord(psDash)*
2: DashDot Pen *// ord(psDashDot)*
3: DashDotDot Pen *// ord(psDashDotDot)*
4: Clear Pen *// ord(psClear)*
5: Inside Frame Pen *// ord(psInsideFrame)*

c : The pen color of the figure.

row : The row style of the figure. It can have following values.
0: No Rows *// ord(rsNone)*
1: Right Solid *// ord(rsRightSolid)*
2: Left Solid *// ord(rsLeftSolid)*
3: Both Solid *// ord(rsBothSolid)*
4: Right Light *// ord(rsRightLight)*
5: Left Light *// ord(rsLeftLight)*
6: Both Light *// ord(rsBothLight)*

Select : Set this true if you want the figure to be selected after it is added to Powercad.

Return Value: The handle of the created figure.

Ex:

```
FigHandle := Powercad1.Line(0,100,100,200,100,1,ord(psSolid),
             clRed,ord(rsNone),true);
TFigure(FigHandle).LockMove := True;
```

Defined in PCDrawing unit

PolyLine

```
Function PolyLine(LayerNbr:Integer; points:array of TPoint;
                  nbrPoint,w,s,c,row,brs,brc:integer;
                  Closed,Select:Boolean):TFigHandle;
```

Use **PolyLine** method to create a PolyLine figure and add it to Powercad.

Parameter Definitions:

LayerNbr : The list index of the layer that the figure will be inserted.
points : The array that is storing the points of the figure in dmm unit. Must be 0 based.
nbrPoint : The number of the points in the array.
w : The pen width of the figure.
s : The pen style of the figure.It can have following values.
- **0**: Solid Pen *// ord(psSolid)*
- **1**: Dash Pen *// ord(psDash)*
- **2**: DashDot Pen *// ord(psDashDot)*
- **3**: DashDotDot Pen *// ord(psDashDotDot)*
- **4**: Clear Pen *// ord(psClear)*
- **5**: Inside Frame Pen *// ord(psInsideFrame)*

c : The pen color of the figure.
row : The row style of the figure. It can have following values.
- **0**: No Rows *// ord(rsNone)*
- **1**: Right Solid *// ord(rsRightSolid)*
- **2**: Left Solid *// ord(rsLeftSolid)*
- **3**: Both Solid *// ord(rsBothSolid)*
- **4**: Right Light *// ord(rsRightLight)*
- **5**: Left Light *// ord(rsLeftLight)*
- **6**: Both Light *// ord(rsBothLight)*

brs : The brush style of the figure. It can have following values.
- **0**: Solid Brush *// ord(bsSolid)*
- **1**: Clear Brush *// ord(bsClear)*
- **2**: Horizontal hatch *// ord(bsHorizontal)*
- **3**: Vertical hatch *// ord(bsVertical)*
- **4**: FDiagonal hatch *// ord(bsFDiagonal)*
- **5**: BDiagonal hatch *// ord(bsBDiagonal)*
- **6**: Cross Hatch *// ord(bsCross)*
- **7**: Diagonal Cross Hatch *// ord(bsDiagCross)*

brc : The brush color of the figure.
Select : Set this true if you want the figure to be selected after it is added to Powercad.
Return Value: The handle of the created figure.

Ex:

```
  Points[0] := Point(100,100);
  Points[1] := Point(200,100);
  Points[2] := Point(200,200);
  FigHandle := Powercad1.PolyLine(0,Points,3,1,ord(psSolid),
               clRed,ord(rsNone),ord(bsSolid),clBlue,true,
               true);
  TPolyLine(FigHandle).ConvertToBezier;
```

Defined in PCDrawing unit

Vertex

Function Vertex(LayerNbr,x,y: integer;
Select:Boolean):TFigHandle;

Use **Vertex** method to create a Vertex figure and add it to Powercad.

Parameter Definitions:

LayerNbr : The list index of the layer that the figure will be inserted.

x,y : The figure coordinates in dmm unit.

Select : Set this true if you want the figure to be selected after it is added to Powercad.

Return Value: The handle of the created figure.

Ex:

```
FigHandle := Powercad1.Vertex(0,100,100,true);
TFigure(FigHandle).LockMove := True;
```

Defined in PCDrawing unit

Rectangle

```
Function Rectangle(LayerNbr,x1,y1,x2,y2,w,s,c,
                   brs,brc:integer;Select:Boolean):TFigHandle;
```

Use **Rectangle** method to create a Rectangle figure and add it to Powercad.

Parameter Definitions:

LayerNbr : The list index of the layer that the figure will be inserted.
x1,y1,x2,y2 : The figure coordinates in dmm unit.
w : The pen width of the figure.
s : The pen style of the figure.It can have following values.
- **0**: Solid Pen *// ord(psSolid)*
- **1**: Dash Pen *// ord(psDash)*
- **2**: DashDot Pen *// ord(psDashDot)*
- **3**: DashDotDot Pen *// ord(psDashDotDot)*
- **4**: Clear Pen *// ord(psClear)*
- **5**: Inside Frame Pen *// ord(psInsideFrame)*

c : The pen color of the figure.
brs : The brush style of the figure. It can have following values.
- **0**: Solid Brush *// ord(bsSolid)*
- **1**: Clear Brush *// ord(bsClear)*
- **2**: Horizontal hatch *// ord(bsHorizontal)*
- **3**: Vertical hatch *// ord(bsVertical)*
- **4**: FDiagonal hatch *// ord(bsFDiagonal)*
- **5**: BDiagonal hatch *// ord(bsBDiagonal)*
- **6**: Cross Hatch *// ord(bsCross)*
- **7**: Diagonal Cross Hatch *// ord(bsDiagCross)*

brc : The brush color of the figure.

Select : Set this true if you want the figure to be selected after it is added to Powercad.

Return Value: The handle of the created figure.

Ex:

```
  FigHandle := Powercad1.Rectangle(0,10,10,70,70,1,
                                   ord(psSolid), clRed,
                                   ord(bsSolid),clBlue,true);
  TFigure(FigHandle).LockMove := True;
```

Defined in PCDrawing unit

Ellipse

```
Function Ellipse(LayerNbr,cx,cy,lenA,lenB,Angle,w,s,c,
                  brs,brc:integer;Select:Boolean):TFigHandle;
```

Use **Ellipse** method to create an Ellipse figure and add it to Powercad.

Parameter Definitions:

LayerNbr : The list index of the layer that the figure will be inserted.
cx,cy : The figure center coordinates in dmm unit.
LenA,Lenb : The horizontal and Vertical Radius of ellipse.
Angle: The angle of the ellipse. It should be in 1/10 degrees. For 10 degrees use a 100.
w : The pen width of the figure.
s : The pen style of the figure.It can have following values.
- **0**: Solid Pen *// ord(psSolid)*
- **1**: Dash Pen *// ord(psDash)*
- **2**: DashDot Pen *// ord(psDashDot)*
- **3**: DashDotDot Pen *// ord(psDashDotDot)*
- **4**: Clear Pen *// ord(psClear)*
- **5**: Inside Frame Pen *// ord(psInsideFrame)*

c : The pen color of the figure.
brs : The brush style of the figure. It can have following values.
- **0**: Solid Brush *// ord(bsSolid)*
- **1**: Clear Brush *// ord(bsClear)*
- **2**: Horizontal hatch *// ord(bsHorizontal)*
- **3**: Vertical hatch *// ord(bsVertical)*
- **4**: FDiagonal hatch *// ord(bsFDiagonal)*
- **5**: BDiagonal hatch *// ord(bsBDiagonal)*
- **6**: Cross Hatch *// ord(bsCross)*
- **7**: Diagonal Cross Hatch *// ord(bsDiagCross)*

brc : The brush color of the figure.

Select : Set this true if you want the figure to be selected after it is added to Powercad.

Return Value: The handle of the created figure.

Ex:

```
  FigHandle := Powercad1.Ellipse(0,10,10,60,120,450,1,
                                 ord(psSolid), clRed,
                                 ord(bsSolid),clBlue,true);
  TFigure(FigHandle).LockMove := True;
```

Defined in PCDrawing unit

Ellipse3p

Function Ellipse3p(LayerNbr,cx,cy,x1,y1,x2,y2,lenA,lenB,Angle,
w,s,c,brs,brc:integer;Select:Boolean):TFigHandle;

Use **Ellipse3p** method to create an Ellipse figure from center point and 2 control points. The control points define the radiuses and angle of the ellipse.

Parameter Definitions:

LayerNbr : The list index of the layer that the figure will be inserted.
cx,cy : The figure center coordinates in dmm unit.
x1,y1,x2,y2 : The figure control points coordinates in dmm unit.
w : The pen width of the figure.
s : The pen style of the figure.It can have following values.
0: Solid Pen *// ord(psSolid)*
1: Dash Pen *// ord(psDash)*
2: DashDot Pen *// ord(psDashDot)*
3: DashDotDot Pen *// ord(psDashDotDot)*
4: Clear Pen *// ord(psClear)*
5: Inside Frame Pen *// ord(psInsideFrame)*
c : The pen color of the figure.
brs : The brush style of the figure. It can have following values.
0: Solid Brush *// ord(bsSolid)*
1: Clear Brush *// ord(bsClear)*
2: Horizontal hatch *// ord(bsHorizontal)*
3: Vertical hatch *// ord(bsVertical)*
4: FDiagonal hatch *// ord(bsFDiagonal)*
5: BDiagonal hatch *// ord(bsBDiagonal)*
6: Cross Hatch *// ord(bsCross)*
7: Diagonal Cross Hatch *// ord(bsDiagCross)*
brc : The brush color of the figure.

Select : Set this true if you want the figure to be selected after it is added to Powercad.

Return Value: The handle of the created figure.

Ex:

```
FigHandle := Powercad1.Ellipse3p(0,100,100,100,60,140,100,1
                                 ord(psSolid), clRed,
                                 ord(bsSolid),clBlue,true);
TFigure(FigHandle).LockMove := True;
```

Defined in PCDrawing unit

Circle

Function Circle(LayerNbr,cx,cy,Radius,w,s,c,
brs,brc:integer;Select:Boolean):TFigHandle;

Use **Circle** method to create a Circle figure and add it to Powercad.

Parameter Definitions:

LayerNbr : The list index of the layer that the figure will be inserted.
cx,cy : The figure center coordinates in dmm unit.
Radius : The Radius of the circle.
w : The pen width of the figure.
s : The pen style of the figure.It can have following values.
0: Solid Pen *// ord(psSolid)*
1: Dash Pen *// ord(psDash)*
2: DashDot Pen *// ord(psDashDot)*
3: DashDotDot Pen *// ord(psDashDotDot)*
4: Clear Pen *// ord(psClear)*
5: Inside Frame Pen *// ord(psInsideFrame)*
c : The pen color of the figure.
brs : The brush style of the figure. It can have following values.
0: Solid Brush *// ord(bsSolid)*
1: Clear Brush *// ord(bsClear)*
2: Horizontal hatch *// ord(bsHorizontal)*
3: Vertical hatch *// ord(bsVertical)*
4: FDiagonal hatch *// ord(bsFDiagonal)*
5: BDiagonal hatch *// ord(bsBDiagonal)*
6: Cross Hatch *// ord(bsCross)*
7: Diagonal Cross Hatch *// ord(bsDiagCross)*
brc : The brush color of the figure.

Select : Set this true if you want the figure to be selected after it is added to Powercad.

Return Value: The handle of the created figure.

Ex:

```
FigHandle := Powercad1.Circle(0,50,100,60,1,
                              ord(psSolid), clRed,
                              ord(bsSolid),clBlue,true);
TFigure(FigHandle).LockMove := True;
```

Defined in PCDrawing unit

Arc

Function Arc(LayerNbr,cx,cy,Radius,a1,a2,w,s,c,brs,brc,
ArcStyle:integer;Select:Boolean):TFigHandle;

Use **Arc** method to create an Arc figure and add it to Powercad.

Parameter Definitions:

LayerNbr : The list index of the layer that the figure will be inserted.
cx,cy : The figure center coordinates in dmm unit.
Radius : The Radius of the arc.
a1: The starting angle of the arc. It should be in degree unit.
a2: The ending angle of the arc. It should be in degree unit.
w : The pen width of the figure.
s : The pen style of the figure.It can have following values.
0: Solid Pen *// ord(psSolid)*
1: Dash Pen *// ord(psDash)*
2: DashDot Pen *// ord(psDashDot)*
3: DashDotDot Pen *// ord(psDashDotDot)*
4: Clear Pen *// ord(psClear)*
5: Inside Frame Pen *// ord(psInsideFrame)*
c : The pen color of the figure.
brs : The brush style of the figure. It can have following values.
0: Solid Brush *// ord(bsSolid)*
1: Clear Brush *// ord(bsClear)*
2: Horizontal hatch *// ord(bsHorizontal)*
3: Vertical hatch *// ord(bsVertical)*
4: FDiagonal hatch *// ord(bsFDiagonal)*
5: BDiagonal hatch *// ord(bsBDiagonal)*
6: Cross Hatch *// ord(bsCross)*
7: Diagonal Cross Hatch *// ord(bsDiagCross)*
brc : The brush color of the figure.
ArcStyle : The style of the arc. It can have following values.
0: Normal Arc
1: Pie Arc
2: Chord Arc
Select : Set this true if you want the figure to be selected after it is added to Powercad.

Return Value: The handle of the created figure.

Ex:

```
FigHandle := Powercad1.Arc(0,50,100,60,0,270,1,
                           ord(psSolid), clRed,
                           ord(bsSolid),clBlue,true);
TFigure(FigHandle).LockMove := True;
```

Defined in PCDrawing unit

TextOut

```
Function TextOut(LayerNbr,x1,y1,Angle,Height:integer;
                 ratio: double; aText,aFontName:String;
                 FontCharset:Byte;Select:Boolean):TFigHandle;
```

Use **TextOut** method to create a Text figure and add it to Powercad.

Parameter Definitions:

LayerNbr : The list index of the layer that the figure will be inserted.
x1,y1 : The figure center coordinates in dmm unit.
Angle : The angle of the text. It should be in 1/10 degree unit. For 10 degrees use a 100.
Height : The height of the font in dmm unit.
ratio : The ratio of width to height. Use 0.9 for a normal ratio.
aText: The text that will be drawn.
aFontname: The name of the font of the text.
FontCharset: The character set number of the text. Use 0 for default.
Select : Set this true if you want the figure to be selected after it is added to Powercad.

Return Value: The handle of the created figure.

Ex:

```
  FigHandle := Powercad1.TextOut(0,50,100,0,40,0.9,
                                 'Hello World', 'Arial',
                                 0,true);
  TFigure(FigHandle).LockMove := True;
```

Defined in PCDrawing unit

InsertBitmap

```
Function InsertBitmap(LayerNbr,x,y,:integer; FName:String;
                Transparent,Select:Boolean):TFigHandle;
```

Use **InsertBitmap** method to create a Bitmap Object figure from the disk and add it to Powercad.

Parameter Definitions:

LayerNbr : The list index of the layer that the figure will be inserted.
x,y : The figure center coordinates in dmm unit.
FName : The file name of the bitmap with its path information.
FName : Set this true if you want the bitmap drawn transparent.
Select : Set this true if you want the figure to be selected after it is added to Powercad.

Return Value: The handle of the created figure.

Ex:

```
  FigHandle := Powercad1.InsertBitmap(0,50,100,'c:\test.bmp',
                                      true,true);
  TFigure(FigHandle).LockMove := True;
```

Defined in PCDrawing unit

InsertWMF

```
Function InsertWMf(LayerNbr,x,y,:integer; FName:String;
                  Select:Boolean):TFigHandle;
```

Use **InsertWmf** method to create a WMFObject figure from the disk and add it to Powercad.

Parameter Definitions:

LayerNbr : The list index of the layer that the figure will be inserted.
x,y : The figure center coordinates in dmm unit.
FName : The file name of the metafile with its path information.
Select : Set this true if you want the figure to be selected after it is added to Powercad.

Return Value: The handle of the created figure.

Ex:

```
  FigHandle := Powercad1.InsertWmf(0,50,100,'c:\test.wmf',
                                   true);
  TFigure(FigHandle).LockMove := True;
```

Defined in PCDrawing unit

ImportWMF

```
Function ImportWMF(LayerNbr:integer; FName:String;
                   Select:Boolean):TFigHandle;
```

Use **ImportWmf** method to import a Windows metafile as editable PowerCad objects to the drawing from a disk file.

Parameter Definitions:

LayerNbr : The list index of the layer that the figure will be inserted.
FName : The file name of the metafile with its path information.
Select : Set this true if you want the inserted figures to be selected.

Return Value: The handle of the created figure group that includes the WMF figures..

Ex:

```
  FigHandle := Powercad1.ImportWmf(0,'c:\test.wmf',true);
  TFigureGrp(FigHandle).Combined := True;
```

Defined in PCDrawing unit

ImportMetafile

```
Function ImportMetafile(LayerNbr:integer; mf:TMetafile;
                   Select:Boolean):TFigHandle;
```

Use **ImportMetafile** method to import a Windows metafile as editable PowerCad objects to the drawing from an already created TMetafile object.

Parameter Definitions:

LayerNbr : The list index of the layer that the figure will be inserted.
mf: The Tmetafile object that includes the metafile.

Select : Set this true if you want the inserted figures to be selected.

Return Value: The handle of the created figure group that includes the WMF figures..

Ex:

```
  FigHandle := Powercad1.ImportMetafile(0,
                           Image1.Picture.Metafile ,true);
  TFigureGrp(FigHandle).Combined := True;
```

Defined in PCDrawing unit

AddCustomFigure

```
Function AddCustomFigure(LayerNbr:integer;CustomFig:TFigure;
                         Select:Boolean):TFigHandle;
```

Use **AddCustomFigure** method to add a TFigure descendant that is already created.

Parameter Definitions:

LayerNbr : The list index of the layer that the figure will be inserted.
CustomFig: The object reference of the custom figure.
Select : Set this true if you want the inserted figures to be selected.

Return Value: The handle of the added custom figure.

Ex:

```
  cFig := TFrame.Create(10,10,60,70,1,ord(psSolid),clRed,
                        ord(bsSolid),clBlue,
                        PowerCad1.Layers[0],
                        dsNormal,PowerCad1);
  Powercad1.AddCustomFigure(0,cFig,true);
```

Defined in PCDrawing unit

RunMacro

```
Procedure RunMacro(Macro:TStringList);
```

Use **RunMacro** method to execute a PSCL script given as a stringlist.

Parameter Definitions:

Macro : The stringlist that includes the script.

Ex:

```
  Macro := TStringList.Create;
  Macro.LoadFromFile('c:\macros\test.cmf');
  Powercad1.RunMacro(Macro);
  Macro.free;
```

Defined in PCDrawing unit

RunMacroText

Procedure RunMacroText(Macro: String);

Use **RunMacroText** method to execute a PSCL script given as a string. The script lines should be separated with return character.

Parameter Definitions:

Macro : The string that includes the script.

Ex:

```
Macro := 'Begin'+#13+'Refresh;'+#13+'End;';
Powercad1.RunMacroText(Macro);
```

Defined in PCDrawing unit

RunMacroByFileName

Procedure RunMacroByFilename(MacroName: String);

Use **RunMacroByFileName** method to execute a PSCL script given in a text file.

Parameter Definitions:

MacroName : The filename of the macro with its path information.

Ex:

```
Powercad1.RunMacroByFileName('c:\macros\test.cmf');;
```

Defined in PCDrawing unit

AddPSCLConstant

Procedure AddPSCLConstant(ConstName:String; Value:Variant);

Use **AddPSCLConstant** method to add a constant expression to the PSCL language.

Parameter Definitions:

ConstName : The string that will represent the constant value.
Value : The value of the constant.

Ex:

```
PowerCad1.AddPsclConstant('PI', 3.14);
PowerCad1.AddPsclConstant('Err', 'There is an error');
```

Defined in PCDrawing unit

AddPSCLProcedure

```
Procedure AddPSCLProcedure(ProcName:String;
                           ProcAddr:TProcType;
                           const Params: array of Byte);
```

Use **AddPSCLProcedure** method to add a procedure expression to the PSCl language. This procedure expression will represent a real code (function) in the application.

Parameter Definitions:

ProcName : The string that will represent the procedure call.
ProcAddr : The address of the real function that will be execute when it is called within the script. This function should be in a specific prototype.
Params : A Byte Array indicating the parameters of the script procedure.

Ex:

```
// The real procedure
function MyXProc(slf:TObject;var s:array of variant):variant;
Begin
  // Implementation
End;

PowerCad1.AddPsclProc('XProc',MyXProc,[0,0]);
```

Defined in PCDrawing unit

AddPSCLFunction

```
Procedure AddPSCLFunction(ProcName:String;
                          ProcAddr:TProcType;
                          const Params: array of Byte);
```

Use **AddPSCLFunction** method to add a function expression to the PSCL language. This function expression will represent a real code (function) in the application.

Parameter Definitions:

ProcName : The string that will represent the fucntion call.
ProcAddr : The address of the real function that will be executed when it is called within the script. This function should be in a specific prototype.
Params : A Byte Array indicating the parameters of the script function.

Ex:

```
// The real function
function MyXFunc(slf:TObject;var s:array of variant):variant;
Begin
  // Implementation
End;

PowerCad1.AddPsclFunction('XFunc',MyXFunc,[0,0]);
```

Defined in PCDrawing unit

LoadPlugins

```
Procedure LoadPlugins;
```

Use **LoadPlugins** method to load the Plugin DLLs in the plugin directory. You can make any plugin call before you load the plugins. Normally load plugins in the OnCreate event handler of your application's main form.

Ex:

```
  PowerCad1.LoadPlugins;
```

Defined in PCDrawing unit

UnLoadPlugins

Procedure UnLoadPlugins;

Use **UnLoadPlugins** method to unload the Plugin DLLs that are loaded. Make this call when you don't need the plugins anymore. Normally unload the plugins in the OnClose event handler of your applications's main form.

Ex:

```
PowerCad1.UnLoadPlugins;
```

Defined in PCDrawing unit

GetPlugins

Function GetPlugins:TStringlist;

Use **GetPlugins** method to get the list of currently loaded plugins. The list includes the names of the plugins in their list order.

Parameter Definitions:

Return Value : The Stringlist that includes the list of the plugin names. This returning list is created by the function itself, but should be freed by the application after it is used. The list is returned **nil** if there is no plugin loaded.

Ex:

```
plgList := PowerCad1.GetPlugins;
if assigned(plglist) then
begin
  CreatePluginMenuFromList(plgList);
  plgList.Free;
end;
```

Defined in PCDrawing unit

GetPluginVerbs

```
Function GetPluginVerbs(PluginIdx:Integer):String;
```

Use **GetPluginVerbs** method to get the list of the verbs of a currently loaded plugin.

Parameter Definitions:

PluginIdx : The list index of the plugin whose verbs will be returned.
Return Value : A string that includes the names of the verbs in their list order separated by a return character.IIt is returned blank whe there is no verb to list.

Ex:

```
  var VerbList: TStringList;
      Verbs: String;

  verbs := Powercad1.GetPluginVerbs(0);
  if Verbs <> '' then
  begin
    VerbList := TStringlist.Create;
    VerbList.text := Verbs;
    CreateVerbsMenuFromList(VerbList);
    VerbList.free;
  end;
```

Defined in PCDrawing unit

DoPluginVerb

```
Procedure DoPluginVerb(PluginIdx,VerbIdx:Integer);
```

Use **DoPluginVerb** method to execute a verb of a plugin.

Parameter Definitions:

PluginIdx : The list index of the plugin whose verb will be executed.
VerbIdx : The list index of the verb that vill be executed.

Ex:

```
  Powercad1.DoPluginVerb(0,0);
```

Defined in PCDrawing unit

PrintPreview

Procedure PrintPreview;

Use **PrintPreview** method to preview a drawing before printing.

Ex:

```
  Powercad1.PrintPreview;
```

Defined in PCDrawing unit

ShowPropertyWindow

Procedure ShowPropertyWindow;

Use **ShowPropertyWindow** method to show the property page (object inspector).

Ex:

```
  Powercad1.ShowPropertyWindow;
```

Defined in PCDrawing unit

PrintMessage

Procedure PrintMessage(Mes:String);

Use **PrintMessage** method to write a string message in the bottom panel of the PowerCad. It will be cleared in the first refresh. So write the message after you have made your refresh.

Parameter Definitions:

Mes : The message that will be printed.

Ex:

```
  Powercad1.PrintMessage('The figure is rotated');
```

Defined in PCDrawing unit

RegisterFigureClass

Procedure RegisterFigureClass(Fig:TFigureClass);

Use **RegisterFigureClass** method to register the custom figure that you have inherited from TFigure. The custom figures can be drawn in PowerCad after they are registered.

Parameter Definitions:

Fig : The class name of the custom figure.

Ex:

```
Powercad1.RegisterFigureClass(TFrame);
```

Defined in PCDrawing unit

ExecuteCommand

Procedure ExecuteCommand(Command:String);

Use **ExecuteCommand** method to execute a cad command written in a string. The commandbar also uses this method to execute the commands written in itself.

Parameter Definitions:

Command : The string representation of the command to be executed.

Ex:

```
Powercad1.ExecuteCommand('Line 0 0 100 100');
```

Defined in PCDrawing unit

ExecuteTBCommand

Procedure ExecuteTBCommand(CommandID:Integer);

Use **ExecuteTBCommand** method to execute toolbar commands. A toolbar command is an ID that represents a specific action in PowerCad that don't need a parameter for starting.

Parameter Definitions:

CommandID : The id of the command to be executed. For a list of TB commands see the PCTypesUtils unit.

Ex:

```
Powercad1.ExecuteTBCommand(cNew);
```

Defined in PCDrawing unit

CountBlock

```
Function CountBlock(BlockName:String):integer;
```

Use **CountBlock** method to get the number of a specific block in the drawing.

Parameter Definitions:

BlockName : The name of the block that will be counted.

Ex:

```
  Powercad1.CountBlock('Vane');
```

Defined in PCDrawing unit

GetSelectionHandles

```
Function GetSelectionHandles(var Handles: Array of
                             TfigureHandle):integer;
```

Use **GetSelectionHandles** method to get the handles of the figures that are selected. This method is designed for the DLL version so you should prefer the Selection property instead of using this method.

Parameter Definitions:

Handles : The array that is including the selection handles.

Return Value: The count of the selected figures.

Ex:

```
  var Handles: array [0...254] of Integer;
  cnt := Powercad1.GetSelectionHandles(Handles);
  for i := 0 to cnt-1 do
  begin
    TFigure(Handles[i]).Color := clRed;
  end;
```

Defined in PCDrawing unit

GetBlockFileNames

Procedure GetBlockFileNames(List: TStringList);

Use **GetBlockFileNames** method to get the file names of the blocks that are in the block directory of your application. The file names includes the path information also.

Parameter Definitions:

List : The stringlist that the block file names will be written to. It should be created and freed by the application.

Ex:

```
List := TSTringlist.Create;
Powercad1.GetBlokFileNames(List);
for a := 0 to List.count -1 do
begin
  Listbox1.Add(List[i]);
end;
```

Defined in PCDrawing unit

GetVersion

Function GetVersion:Integer;

Use **GetVersion** method to get major and minor version of the current code.

Parameter Definitions:

Return Value : The first number of the returned integer is the major version and the second number is the minor version. For instance; if the majaor version is 2 and minor version is 1 then the returned integer is 21.

Ex:

```
Ver := PowerCad1.GetVersion;
```

Defined in PCDrawing unit

GetBuildNumber

Function GetBuildNumber:Integer;

Use **GetBuildNumber** method to get the build number of the current version.

Parameter Definitions:

Return Value : The build number.

Ex:

```
  bn := PowerCad1.GetBuildNumber;
```

Defined in PCDrawing unit

GetSlcPenStyle

Function GetSlcPenStyle:Integer;

Use **GetSlcPenStyle** method to get the pen style of the currently selected figure or figures.

Parameter Definitions:

Return Value : The pen style of the currently selected figures. If the selection count is bigger than 1 and if the pen styles of the selected figures are different from each other then returning value is –1.

Ex:

```
  val := PowerCad1.GetSlcPenStyle;
  if val <> -1 then
    PowerCad1.PrintMessage('PenStyle:'+ inttostr(val));
```

Defined in PCDrawing unit

GetSlcPenWidth

```
Function GetSlcPenWidth:Integer;
```

Use **GetSlcPenWidth** method to get the pen width of the currently selected figure or figures.

Parameter Definitions:

Return Value : The pen width of the currently selected figures. If the selection count is bigger than 1 and if the pen widths of the selected figures are different from each other then returning value is –1.

Ex:

```
  val := PowerCad1.GetSlcPenWidth;
  if val <> -1 then
    PowerCad1.PrintMessage('PenWidth:'+ inttostr(val));
```

Defined in PCDrawing unit

GetSlcPenColor

```
Function GetSlcPenColor:Integer;
```

Use **GetSlcPenColor** method to get the pen color of the currently selected figure or figures.

Parameter Definitions:

Return Value : The pen color of the currently selected figures. If the selection count is bigger than 1 and if the pen colors of the selected figures are different from each other then returning value is –1.

Ex:

```
  val := PowerCad1.GetSlcPenColor;
  if val <> -1 then
    PowerCad1.PrintMessage('PenColor:'+ inttostr(val));
```

Defined in PCDrawing unit

GetSlcRowStyle

```
Function GetSlcRowStyle:Integer;
```

Use **GetSlcRowStyle** method to get the row style of the currently selected figure or figures.

Parameter Definitions:

Return Value : The row style of the currently selected figures. If the selection count is bigger than 1 and if the row styles of the selected figures are different from each other then returning value is –1.

Ex:

```
  val := PowerCad1.GetSlcRowStyle;
  if val <> -1 then
    PowerCad1.PrintMessage('RowStyle:'+ inttostr(val));
```

Defined in PCDrawing unit

GetSlcBrushColor

```
Function GetSlcBrushColor:Integer;
```

Use **GetSlcBrushColor** method to get the brush color of the currently selected figure or figures.

Parameter Definitions:

Return Value : The brush color of the currently selected figures. If the selection count is bigger than 1 and if the brush colors of the selected figures are different from each other then returning value is –1.

Ex:

```
  val := PowerCad1.GetSlcBrushColor;
  if val <> -1 then
    PowerCad1.PrintMessage('BrushColor:'+ inttostr(val));
```

Defined in PCDrawing unit

GetSlcBrushStyle

Function GetSlcBrushStyle:Integer;

Use **GetSlcBrushStyle** method to get the brush style of the currently selected figure or figures.

Parameter Definitions:

Return Value : The brush style of the currently selected figures. If the selection count is bigger than 1 and if the brush styles of the selected figures are different from each other then returning value is –1.

Ex:

```
  val := PowerCad1.GetSlcBrushStyle;
  if val <> -1 then
    PowerCad1.PrintMessage('BrushStyle:'+ inttostr(val));
```

Defined in PCDrawing unit

GetSlcFont

Function GetSlcFont:TFont;

Use **GetSlcFont** method to get the font of the currently selected figure or figures.

Parameter Definitions:

Return Value : The font of the currently selected figures. If the selection count is bigger than 1 and if the fonts of the selected figures are different from each other then returning value is **nil**. The returning font information only is for the fontname,fontstyles and font charset.

Ex:

```
  fnt := PowerCad1.GetSlcDont;
  if fnt <> nil then
    PowerCad1.PrintMessage('FontName:'+ fnt.Name);
```

Defined in PCDrawing unit

SetOptionHints

Procedure SetOptionHints(HintList:TStringList);

Use **SetOptionHints** method to set the hints of the option buttons in the bottom panel.

Parameter Definitions:

HintLits : The StringList that includes the hints of the buttons. They number of the items in the stringlist should be 6. The hintlist should be created and freed by the application.

Ex:

```
HList := Tstringlist.Create;
HList.Add('Ruler Visible');
HList.Add('Grids Visible');
HList.Add('Guides Visible');
HList.Add('Snap To Grids');
HList.Add('Snap To Guides');
HList.Add('Grids Visible');
HList.Add('Snap To Guides');
PowerCad1.SetOptionHints(hList);
hList.free;
```

Defined in PCDrawing unit

RegisterDlg

Procedure RegisterDlg(Value:Pointer);

Use **RegisterDlg** method to register the custom dialogs that are inherited from TDlgBase. If a dialog is registered its Syncronize method is called from PowerCad when there is a change. Normally you shouldn’t use this method, because a dialog is registered by the TDlgbase code when the CadControl property is assigned.

Parameter Definitions:

Value : The object reference of the Dialog.

Ex:

```
PowerCad1.RegisterDlg(MyCustomDlg);
```

Defined in PCDrawing unit

UnRegisterDlg

Procedure UnRegisterDlg(Value:Pointer);

Use **UnRegisterDlg** method to unregister the custom dialogs that are inherited from TDlgBase. Normally you shouldn't use this method, because a dialog is unregistered by the TDlgbase code when the CadControl property is assigned **nil**.

Parameter Definitions:

Value : The object reference of the Dialog.

Ex:

```
PowerCad1.UnRegisterDlg(MyCustomDlg);
```

Defined in PCDrawing unit

RegisterBar

Procedure RegisterBar(Value:Pointer);

Use **RegisterBar** method to register the custom toolbars that are inherited from TBarBase. If a toolbar is registered its Syncronize method is called from PowerCad when there is a change. Normally you shouldn't use this method, because a toolbar is registered by the TBarBase code when the CadControl property is assigned.

Parameter Definitions:

Value : The object reference of the Toolbar.

Ex:

```
PowerCad1.RegisterBar(MyCustomBar);
```

Defined in PCDrawing unit

UnRegisterBar

Procedure UnRegisterBar(Value:Pointer);

Use **UnRegisterBar** method to unregister the custom toolbars that are inherited from TBarBase. Normally you shouldn't use this method, because a toolbar is unregistered by the TBarBase code when the CadControl property is assigned **nil**.

Parameter Definitions:

Value : The object reference of the Toolbar.

Ex:

```
PowerCad1.UnRegisterBar(MyCustomBar);
```

Defined in PCDrawing unit

Events:

OnZoomIn

```
type TNotifyEvent = procedure(Sender: TObject) of object;
Property OnZoomIn: TNotifyEvent;
```

Occures when the user presses the ZoomIn button of the bottom panel. Use **OnZoomIn** event handler to implement any special processing that should occur after the **ZoomIn** is executed.

Sender is the cad control that raises this event.

Ex:

```
procedure TForm1.PowerCad1ZoomIn(Sender: TObject);
begin
 TPowerCad(Sender).PrintMessage('Zoom In Executed');
end;
```

Defined in PCPanel unit

OnZoomOut

```
type TNotifyEvent = procedure(Sender: TObject) of object;
Property OnZoomOut: TNotifyEvent;
```

Occures when the user presses the ZoomOut button of the bottom panel. Use **OnZoomOut** event handler to implement any special processing that should occur after the **ZoomOut** is executed.

Sender is the cad control that raises this event.

Ex:

```
procedure TForm1.PowerCad1ZoomOut(Sender: TObject);
begin
 TPowerCad(Sender).PrintMessage('Zoom Out Executed');
end;
```

Defined in PCPanel unit

OnZoomArea

```
type TNotifyEvent = procedure(Sender: TObject) of object;
Property OnZoomArea: TNotifyEvent;
```

Occures when the user presses the ZoomArea button of the bottom panel. Use **OnZoomArea** event handler to implement any special processing that should occur after the **ZoomArea** button is clicked. Normally ZoomArea is executed by the zoom tool after the user selects the area, so the event is raised before the ZoomArea is executed.

Sender is the cad control that raises this event.

Ex:

```
procedure TForm1.PowerCad1ZoomArea(Sender: TObject);
begin
 TPowerCad(Sender).PrintMessage('Zoom Area Tool is Active');
end;
```

Defined in PCPanel unit

OnZoomFitToWindow

```
type TNotifyEvent = procedure(Sender: TObject) of object;
Property OnZoomFitToWindow: TNotifyEvent;
```

Occures when the user presses the FitToWindow button of the bottom panel. Use **OnZoomFitToWindow** event handler to implement any special processing that should occur after the **FitToWindow** zooming is executed.

Sender is the cad control that raises this event.

Ex:

```
procedure TForm1.PowerCad1FitToWindow(Sender: TObject);
begin
 TPowerCad(Sender).PrintMessage('The page is fit To Window');
end;
```

Defined in PCPanel unit

OnScale

```
type TNotifyEvent = procedure(Sender: TObject) of object;
Property OnScale: TNotifyEvent;
```

Occurs when the drawing's scale property is changed. Use **OnScale** event handler to implement any special processing that should occur as a result of changing the view scale of the page.

Sender is the cad control that raises this event.

Ex:

```
procedure TForm1.PowerCad1Scale(Sender: TObject);
var CadControl: TPowercad;
begin
  CadControl := Sender as TPowercad;
  with CadControl do
    PrintMessage('CurrentScale='+InttoStr(Scale));
end;
```

Defined in PCDrawBox unit

OnSurfaceTrace

```
type TMouseTraceEvent = procedure (Sender: TObject;
                                   Shift:TShiftState;
                                   X,Y: Integer) of object;
Property OnSurfaceTrace: TMouseTraceEvent;
```

Occurs when the user moves the mouse pointer while the mouse pointer is over the PowerCad drawing surface. This event is a simulation of OnMouseMove event. The difference is that this event occurs on the drawing area of the control and the raising coordinates are in dmm unit. Use **OnSurfaceTrace** event handler to respond when the mouse pointer moves after the cad control has captured the mouse.

Sender is the cad control that raises this event. *Shift* is used to determine that state of shift keys, *x* and *y* is the current coordinates of the mouse cursor in dmm unit.

Ex:

```
procedure TForm1.PowerCad1SurfaceTrace(Sender: TObject; Shift:
                                   TShiftState;X, Y: Integer);
var CadControl: TPowercad;
    fig: TFigure;
begin
  CadControl := Sender as TPowercad;
  fig := CadControl.CheckByPoint(CadControl.ActiveLayer,x,y);
  if assigned(fig) then
    CadControl.PrintMessage('Mouse is over '+ fig.name)
  else
    CadControl.PrintMessage('');
end;
```

Defined in PCDrawBox unit

OnSurfacePush

```
type TMousePushEvent = procedure (Sender: TObject;
                                  Button: TMouseButton;
                                  Shift:TShiftState;
                                  X,Y: Integer) of object;
Property OnSurfacePush: TMousePushEvent;
```

Occurs when the user presses a mouse button with the mouse pointer over the PowerCad drawing surface. This event is a simulation of OnMouseDown event. The difference is that this event occurs on the drawing area of the control and the raising coordinates are in dmm unit. Use **OnSurfacePush** event handler to implement any special processing that should occur as a result of pressing a mouse button.

Sender is the cad control that raises this event. *Shift* is used to determine that state of shift keys, *button* is used to determine which button is pressed, *x* and *y* is the current coordinates of the mouse cursor in dmm unit.

Ex:

```
procedure TForm1.PowerCad1SurfacePush(Sender: TObject;
      Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var CadControl: TPowercad;
begin
  CadControl := Sender as TPowercad;
  if (CadControl.toolIdx = toSelect)
      and (button = mbRight) then
    CadControl.Vertex(CadControl.ActiveLayer,x,y,false);
end;
```

Defined in PCDrawBox unit

OnSurfacePull

```
type TMousePullEvent = procedure (Sender: TObject;
                                  Button: TMouseButton;
                                  Shift:TShiftState;
                                  X,Y: Integer) of object;
Property OnSurfacePull: TMousePullEvent;
```

Occurs when the user releases a mouse button that was pressed with the mouse pointer over the PowerCad drawing surface. This event is a simulation of OnMouseUp event. The difference is that this event occurs on the drawing area of the control and the raising coordinates are in dmm unit. Use **OnSurfacePull** event handler to implement any special processing that should occur as a result of releasing a mouse button.

Sender is the cad control that raises this event. *Shift* is used to determine that state of shift keys, *button* is used to determine which button is released, *x* and *y* is the current coordinates of the mouse cursor in dmm unit.

Ex:

```
procedure TForm1.PowerCad1SurfacePull(Sender: TObject;
      Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var CadControl: TPowercad;
begin
  CadControl := Sender as TPowercad;
  if (CadControl.toolIdx = toSelect)
      and (button = mbRight) then
    CadControl.Vertex(CadControl.ActiveLayer,x,y,false);
end;
```

Defined in PCDrawBox unit

OnSurfaceLeave

```
type TNotifyEvent = procedure(Sender: TObject) of object;
Property OnSurfaceLeave: TNotifyEvent;
```

Occurs when the mouse pointer leaves the PowerCad drawing surface. Use **OnSurfaceLeave** event handler to implement any special processing that should occur as a result of leaving the drawing surface.

Sender is the cad control that raises this event.

Ex:

```
procedure TForm1.PowerCad1SurfaceLeave(Sender: TObject);
var CadControl: TPowercad;
begin
  CadControl := Sender as TPowercad;
  CadControl.PrintMessage('Mouse Leaved the surface');
end;
```

Defined in PCDrawBox unit

OnSurfaceClick

```
type TNotifyEvent = procedure(Sender: TObject) of object;
Property OnSurfaceClick: TNotifyEvent;
```

Occurs when the user clicks a mouse button over the PowerCad drawing surface. This event is a simulation of OnMouseClick event. The difference is that this event occurs on the drawing area of the control. Use **OnSurfaceClick** event handler to implement any special processing that should occur as a result of clicking the drawing surface.

Sender is the cad control that raises this event.

Ex:

```
procedure TForm1.PowerCad1SurfaceClick(Sender: TObject);
var CadControl: TPowercad;
begin
  CadControl := Sender as TPowercad;
  CadControl.PrintMessage('Surface is clicked');
end;
```

Defined in PCDrawBox unit

OnSurfaceDblClick

```
type TNotifyEvent = procedure(Sender: TObject) of object;
Property OnSurfaceDblClick: TNotifyEvent;
```

Occurs when the user double clicks a mouse button over the PowerCad drawing surface. This event is a simulation of OnMouseDblClick event. The difference is that this event occurs on the drawing area of the control. Use **OnSurfacedblClick** event handler to implement any special processing that should occur as a result of double clicking the drawing surface.

Sender is the cad control that raises this event.

Ex:

```
procedure TForm1.PowerCad1SurfaceDblClick(Sender: TObject);
var CadControl: TPowercad;
begin
  CadControl := Sender as TPowercad;
  CadControl.PrintMessage('Surface is double clicked');
end;
```

Defined in PCDrawBox unit

OnSurfaceDragDrop

```
type TDropDragEvent = procedure (Sender, Source: TObject;
                                 X, Y: Integer) of object;
Property OnSurfaceDragDrop: TDropDragEvent;
```

Occurs when the user drops an object being dragged over the Powercad Drawing Surface. This event is a simulation of OnDragDrop event. The difference is that this event occurs on the drawing area of the control and the raised coordinates are in dmm unit. Use **OnSurfaceDragDrop** event handler to implement any special processing that should occur as a result of dropping the mouse over the drawing surface.

Sender is the cad control that raises this event. *Source* is the dropped object. *X* and *Y* are drop coordinates in dmm unit.

Ex:

```
procedure TForm1.PowerCad1SurfaceDragDrop(Sender, Source:
          TObject; X,Y: Integer);
var xLabel: TLabel;
    CadControl: TPowercad;
begin
  CadControl := Sender as TPowercad;
  if Source is TLabel then
  begin
    xLabel := Source as TLabel;
    CadControl.TextOut(CadControl.ActiveLayer,x,y,0,50,0.9,
                       xLabel.Caption,xlabel.Font.name,0,
                       False);
  end;
end;
```

Defined in PCDrawBox unit

OnSurfaceDragOver

```
type TDragTraceEvent = procedure(Sender, Source: TObject;
                                 X, Y: Integer;
                                 State:TDragState;
                                 var Accept: Boolean) of object;

Property OnSurfaceDragOver: TDragTraceEvent;
```

Occurs when the user drags an object over the Powercad Drawing Surface. This event is a simulation of OnDragOver event. The difference is that this event occurs on the drawing area of the control and the raised coordinates are in dmm unit. Use **OnSurfaceDragOver** event handler to implement any special processing that should occur as a result of dragging the mouse over the drawing surface.

Sender is the cad control that raises this event. *Source* is the dropped object. *X* and *Y* are drop coordinates in dmm unit.The *state* parameter describes how the mouse is moving in relation to the surface that it is passing over. If the dragged object can be dropped on the surface set the *Accept* parameter true otherwise set to false.

Ex:

```
procedure TForm1.PowerCad1SurfaceDragOver(Sender, Source:
          TObject; X,Y: Integer; State:TDragState;
          var Accept: Boolean);
begin
  Accept := False;
  if Source is TLabel then Accept := True;
end;
```

Defined in PCDrawBox unit

OnSurfaceDragEnd

```
type TDragEndEvent = procedure (Sender, Target: TObject;
                                X, Y:Integer) of object;

Property OnSurfaceDragEnd: TDragEndEvent;
```

Occurs when the dragging of the CadControl ends, either by dropping the object or by canceling the dragging. This event is a simulation of OnEndDrag event. The difference is that this event occurs on the drawing area of the control and the raised coordinates are in dmm unit. Use **OnSurfaceDragEnd** event handler to implement any special processing that should occur when the dragging stops.

Sender is the cad control that raises this event. *Target* is the object where the dragging is ended. *X* and *Y* are drop coordinates in dmm unit.

Ex:

```
procedure TForm1.PowerCad1SurfaceDragEnd(Sender, Target:
          TObject; X,Y: Integer);
var CadControl: TPowercad;
begin
  CadControl := Sender as TPowercad;
  if assigned(Target) then
    CadControl.PrintMessage('PowerCad is dropped on a' +
                            Target.ClassName);
end;
```

Defined in PCDrawBox unit

OnSurfaceDragStart

```
type TDragStartEvent = procedure (Sender: TObject;
                       var DragObject : TDragObject) of object;

Property OnSurfaceDragStart: TDragStartEvent;
```

Occurs when the user begins to drag CadControl Drawing Surface. This event is a simulation of OnStartDrag event. The difference is that this event occurs on the drawing area of the control. Use **OnSurfaceDragStart** event handler to implement any special processing that should occur when the dragging starts.

Sender is the cad control that raises this event. *DragObject* can be created in event handler to make a more visual dragging.

Ex:

```
procedure TForm1.PowerCad1SurfaceDragStart(Sender:TObject;
                var DragObject : TDragObject);
var CadControl: TPowercad;
begin
  CadControl := Sender as TPowercad;
  DragObject := nil;
  CadControl.PrintMessage('Drag Started');
end;
```

Defined in PCDrawBox unit

OnSurfacePaint

```
type TNotifyEvent = procedure(Sender: TObject) of object;
Property OnSurfacePaint: TNotifyEvent;
```

Occurs when the Surface paintbox is repainted. Use **OnSurfacePaint** event handler to implement any special drawing that should occur each time the paintbox is painted.

Sender is the cad control that raises this event.

Ex:

```
procedure TForm1.PowerCad1SurfacePainted(Sender:TObject);
var CadControl: TPowercad;
begin
  CadControl := Sender as TPowercad;
  CadControl.Surface.Canvas.Draw(0,0,Form1.Icon);
end;
```

Defined in PCDrawing unit

OnSelectionChange

```
type TNotifyEvent = procedure(Sender: TObject) of object;
Property OnSelection: TNotifyEvent;
```

Occurs when the selection changes in the Cad Control . Use **OnSelectionChange** event handler to implement any special drawing that should occur each time the selection is changed.

Sender is the cad control that raises this event.

Ex:

```
procedure TForm1.PowerCad1SelectionChange(Sender:TObject);
var CadControl: TPowercad;
begin
  CadControl := Sender as TPowercad;
  CadControl.PrintMesage('Selected Figures:'+
                       inttostr(CadControl.Selection.Count));
end;
```

Defined in PCDrawing unit

OnObjectInserted

```
type TNotifyEvent = procedure(Sender: TObject) of object;
Property OnObjectInserted: TNotifyEvent;
```

Occurs when any new figure is created/inserted. Use **OnObjectInserted** event handler to implement any special drawing that should occur each time an insertion is made.

Sender is the cad control that raises this event.

Ex:

```
procedure TForm1.PowerCad1ObjectInserted(Sender:TObject);
var CadControl: TPowercad;
begin
  CadControl := Sender as TPowercad;
  CadControl.PrintMesage('Figure Count:'+
                       inttostr(CadControl.Figures.Count));
end;
```

Defined in PCDrawing unit

OnFigureMoved

```
type TMoveEvent = procedure (Sender: TFigure;
                             dx,dy: integer)of Object;
Property OnFigureMoved:TMoveEvent;
```

Occures when any figure is moved in the drawing surface. If multiple figures are moved together, this event occurs for each figure separately. Use **OnFigureMoved** event handler to implement any special proceesing that should occur when a figure is moved.

Sender is the figure that is moved. *dx* and *dy* is delta distances of the movement in dmm unit.

Ex:

```
procedure TForm1.PowerCad1FigureMoved(Sender:Tfigure; dx,dy:
integer);
begin
  PowerCad1.PrintMesage(Sender.name + ' is moved');
end;
```

Defined in PCDrawing unit

OnBeforeDelete

```
type TNotifyEvent = procedure(Sender: TObject) of object;
Property OnBeforeDelete: TNotifyEvent;
```

Occures before any figure is deleted. If multiple figures are deleted together, this event occurs for each figure separately. Use **OnBeforeDelete** event handler to implement any special processing that should occur before a figure is deleted.

Sender is the figure that is deleted.

Ex:

```
procedure TForm1.PowerCad1BeforeDelete(Sender:TObject);
var Fig: TFigure;
begin
  Fig := Sender as Tfigure;
  PowerCad1.PrintMesage(Fig.name + ' is deleted');
end;
```

Defined in PCDrawing unit

OnPropertyChanged

```
type TPropChangeEvent = procedure (Sender:TFigure;
                    PropName:String; Data:PPropData) of Object;
Property OnPropertyChanged: TPropChangeEvent;
```

Occures when a figure property is changed from the object inspector.Use **OnPropertyChanged** event handler to implement any special processing that should occure when a figure property is changed.

Sender is the figure whose property is changed. *PropName* is the name of the changed property. *Data* is the new value of the property.

Ex:

```
procedure TForm1.PowerCad1PropertyChanged(Sender:TFigure;
                                PropName:String; Data:PPropData)
begin
  PowerCad1.PrintMesage(PropName + ' of '+Sender.name+
                        ' is changed');
end;
```

Defined in PCDrawing unit

OnSnapToFigure

```
type TSnapEvent = Function(Sender: TObject;
     SnapFigure: TFigure; var x,y: integer):Boolean of Object;
Property OnSnapToFigure: TSnapEvent;
```

Occures when the mouse cursor enters in the snap distance of a figure. If the *SnapToNearPoint* property is true then Powercad fires this event as well as making its own figure snap calculations.Use **OnSnapToFigure** event handler to implement your own figure snapping calculations when the mouse cursor approaches to your figure.

Sender is the PowerCad Control that fires the event. *SnapFigure* is the figure that the cursor will be snapped. *x* and *y* are the current coordinates in dmm unit.

Make your own calculations in the event handler and set the new value of the x and y. If you change the x and y, means if you make any snapping, set the return value (the result variable) *true* else set the return value *false*.

Ex:

```
procedure TForm1.PowerCad1SnapToFigure(Sender:TObject;
SnapFigure: TFigure; var x,y: integer);
var myRect:TRectangle;
    p1,p2,p3,p4: TPoint;
    tp,bp,cp: TPoint;
begin
  result := false;
  if SnapFigure is TRectangle then
  begin
    myRect := Trectangle(SnapFigure);
    // Calculate rectangle center
    p1 := myRect.FigurePoints[1];
    p2 := myRect.FigurePoints[2];
    p3 := myRect.FigurePoints[3];
    p4 := myRect.FigurePoints[4];
    tp := Point((p1.x+p2.x) div 2, (p1.y+p2.y) div 2);
    bp := Point((p3.x+p4.x) div 2, (p3.y+p4.y) div 2);
    cp := Point((tp.x+bp.x) div 2, (tp.y+bp.y) div 2);
    if (abs(cp.x-x) < 50) and  (abs(cp.y-y) < 50) then
    begin
      x := cp.x;
      y := cp.y;
      Result := True;
    end;
  end;
end;
```

Defined in PCDrawing unit