PowerCAD 2.6 Release Notes

1 General

The most important change in this release is the ***double precession*** and customisable office like ***GUI***. Since it has been too much time after that last version published, this release is being published without the update of the PowerCAD Developers Handbook. So this document is important to understand the new concepts and the changes in PowerCAD. Please apply to the Handbook when you need information about general concepts of PowerCAD used in this document.

2 How to Install

Before installing the component package first run the demo application (PowerDraw.exe) which is published with this release. So first get a general information about the new release and be sure if you really need the new things or not.

To install the package, first uninstall the old one. To uninstall it successfully without the problems created by Delphi IDE bugs, remove the PCD#.dpl file from the disk (or copy it to a safe place). Open your Delphi, and click **no** when a future load is asked. Now, you can safely install PCD#.dpk by using the package window. Also if you use the PowerCAD installer application , the package will be instaled automatically.

3 What you will SEE and what you will not SEE on the component tab

With this release, the usage of the PowerCAD library is simplified. So now we will have only one component on the PowerCAD tab and that is **Tpowercad**.

> **Where are my Toolbar and Dialog Components?**
>
> Simply, **they have gone**. Because we will not use that stuff any more. Now PowerCAD has a combined interface which can be customised by the developer or the user. **Sorry that the new package doesn't support the toolbar and dialog components. If you want to get the benefits of the new release you should change your old projects.**

4 Double precession

Almost every number which defines a coordinate, delta or angle value in Powercad is now double now. The metric unit is based on **mm**, so our ancient home-made unit the famous **dmm** is thrown to the trash. (My math teacher warned me not to make-up nonsense units in math ☺) . And the polar (angle) unit is based on Radian. *180 degrees is equal to pi (~3.14) radians.*

And the class interface for the figure insertion is all changed. Now the procedures waits double values for the coordinates and dimesions. *In the old interface if you send a 12 value , now you should send 1,2.* **The old dmm interface is not supported.**

Example:

```
//old interface
Function Circle(LayerNbr: Integer; cx,cy,radius,w,s,c,brs,brc:integer;
select: boolean):TFigHandle;
Powercad1.Circle(0,100,125,200,1,1,clRed,0,0,False);

// new interface
Function Circle(LayerNbr: Integer; cx,cy,radius: Double;
w,s,c,brs,brc:integer;select: boolean):TFigHandle;
Powercad1.Circle(0,10,12.5,20,1,1,clRed,0,0,False);
```

What about the old files????

Don't worry. Of course they are supported. In Tekhnelogos CM policy (configuration management), the end users' material should never be damaged by the updates. So simply open them and se that they are opened with no problem.

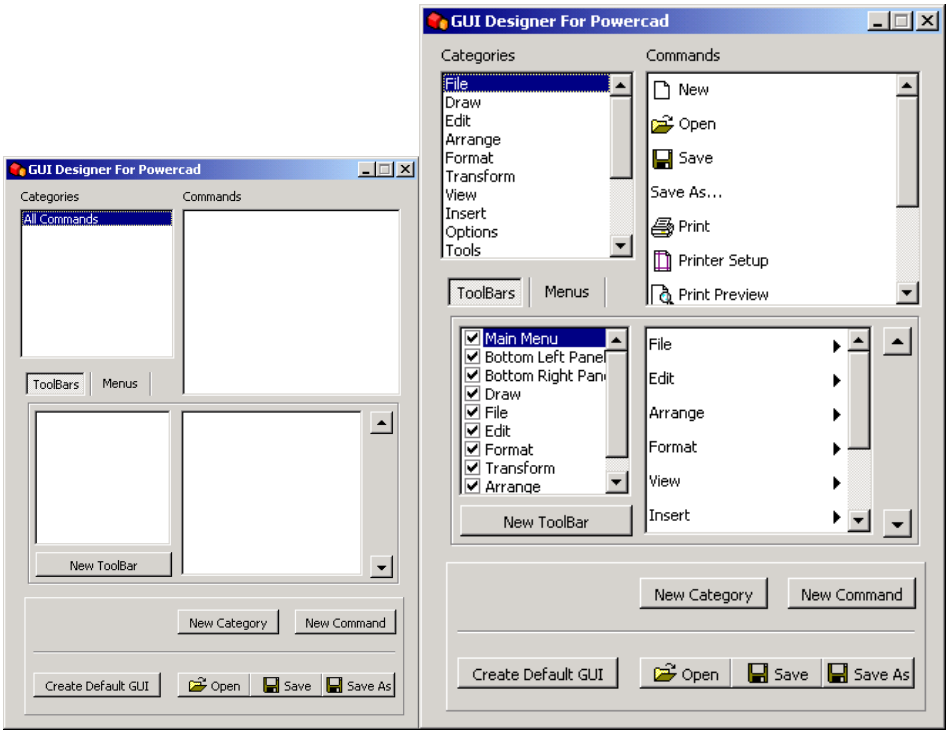
5 PowerCad GUI

The Powercad GUI is now, combination of Powercad commands in menus and toolbars. If your application use the Powercad GUI, it shouldn't use any other menus (main menu) or toolbars. The Powercad Gui can be designed by a separate application (PCGUIDEs.exe), and can be saved to a file. This GUI can be later modified by the user in runtime. You should load the GUI from the file that you have created with PCGUIDes.exe. In addition to the standard interface (commands, menus, toolbars) you can create custom commands, you can add menus or toolbars. You can change the captions and the bitmaps on the designer. More your user can do also most of these customizations if you allow.

Using the GUI.

Before the Delphi Session open the pCGUIDes application.

You will see this editor form . In this form, when you click on the "Create Default GUI" button, the standard interface will be created as it is on the right picture. You can make changes on this standard interface. Before closing the designer save this GUI content to a file. You can use any filename and file extension when saving to disk. Note that, later you can open this file and make any modifications on it and save it again. It is recommended that the GUI file should be delivered in the same location with the application for security.



a. Start a new project in Delphi. Put a Tpowercad on the form and align it to alClient.
b. In the Creation of the form tell Ppowercad to use the GUI file that you have stored. Here we assume that it is in the same location with the executable.

```
procedure TForm1.FormCreate(Sender: TObject);
var fGName:String;
begin
  fGName := extractfilepath(application.ExeName)+'PCGUI.Dat';
  if fileexists(fGname) then Powercad1.GUI.LoadFromFile(fGName);
end;
```

c. Now run the application. You will see a CAD application that you can do everything.

Locating the toolbars

The toolbars are located at the top dock area by the beginning. When you run the applicaton you can relocate them by dragging to the left,right or bottom dock areas. To store these location in the registry you should call the save procedure on the form close and load procedure on the create.

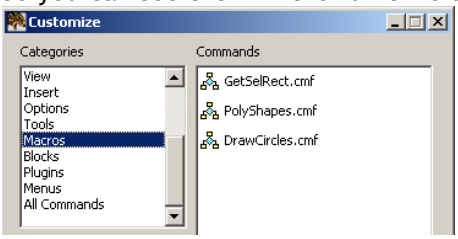
```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Powercad1.Gui.LoadFromRegistry('\Software\Tekhnelogos\Powercad\GUI');
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Powercad1.Gui.SaveToRegistry('\Software\Tekhnelogos\Powercad\GUI');
end;
```

Having the macros, blocks and plugins in the GUI

The Powercad GUI, can create commands for using the macros,blocks and plugins. To be able to do this it will need the locations of these so, at the creation of the application you should set the paths of these materials.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Powercad1.BlockDirectory  := extractfilepath(application.ExeName)+ 'blocks\';
  PowerCad1.PluginDirectory := extractfilepath(application.ExeName)+ 'plugins\';
  PowerCad1.MacroDirectory  := extractfilepath(application.ExeName)+ 'macros\';
  Powercad1.Gui.LoadFromRegistry('\Software\Tekhnelogos\Powercad\GUI');
end;
```

So you can see them in the run time and the next design time.



To be able to use them, drag and drop to one of the toolbars or menus.

Modifying the GUI

You can remove the existing commands, toolbars (except the manin menu and bottom panels) , menus and categories. You can add new command categories, you can create new toolbars, you can create new menus. You can change the bitmap of the commands or of the instance of a command in a menu or toolbar. You can change the text of the commands or of the instance of a command in a menu or toolbar. Your user can not do all these. He/She can not modify the top elements, means

the categories and commands. He can only modify the toolbars,menus and the instances of commands in these. He can not ad a new command but add a command to toolbar or to a menu. He can not chnage the text of a command , but can change the text of a command instance in a toolbar or menu. He can not change the bitmap of a command but can change the bitmap of a command instance in a menu or a toolbar.

When you create a new menu, you will not see it directly in GUI. Because you should locate this menu to a toolbar (generally yo the main menu). After creating and filling it, find it on the top list box in the menus category. And now drag and drop this menu command to the main menu or to an other toolbar.

All the bitmaps of the commands are stored in a resource file: **PcGUI.res.** You can open this resource file and change the bitmaps. But be carefull about the locations of the bitmaps. Each location belongs to specific command.

All the GUI strings are stored in a pascal file: **GUIStrings.pas.** You can open this and can change the strings. This pascal file is distributed as source in binary packages also.

The modifications on the editor can be done by the right click menu.

Custom Command Handling

If you want to place a custom command in the GUI, just select the category you want it in and click the "New Command" button. Assign a name and a bitmap to this command. Now you can place it in a menu or a toolbar, or you can just leave it without an instance as for your user will place it on the GUI if he/she wants. But what will happen when this command is clicked in the application. For this Powercad has an event onCustomCommand. In this event check the command name and to the proper action.

```
procedure TForm1.PowerCad1CustomCommand(Sender: TObject; ComName: String);
begin
  if ComName = 'About' then frmAbout.Show;
end;
```

6 The PowerCad PopUp Menu

The Popup menu in this release now more powerful. There is a new property called PopStyle. You can set it either to psStandart or psNone. When you set it to psNone, the standard PopUpmenu will not rise. Instead your poup menu will be activated if it exists. If you set the PppStyle property to psStandart then the standard menu items will be created each time according to the clicked object. But if you handle the ***OnUpdatePopUp***event, the menu items will be sent to this event. You can search the list and you can remove any of them or you can change the captions. More important you can add your menus. This event gives you a mnIndex parameter that is the ID (tag) of the last ıtem in the Item list. So when you add a new menu items you should set their tags starting from this index. And when the user clicks on the your custom menu items, the ***OnPopMenuClick*** event will be triggered. In this event you should look at the index and call the proper action.

```
procedure TForm1.PowerCad1UpdatePopUp(Sender: TObject; PopMenu: TPopupMenu;
  var mnIndex: Integer);
var mnItem: TmenuItem;
begin
  popIndex := mnIndex;

  mnItem := TMenuItem.Create(PopMenu);
  mnItem.Tag := popIndex +1;
  mnItem.Caption := 'Clear All';
  Popmenu.Items.Add(mnItem);

  mnItem := TMenuItem.Create(PopMenu);
  mnItem.Tag := popIndex +2;
  mnItem.Caption := 'Set Grid Step';
  Popmenu.Items.Add(mnItem);
```

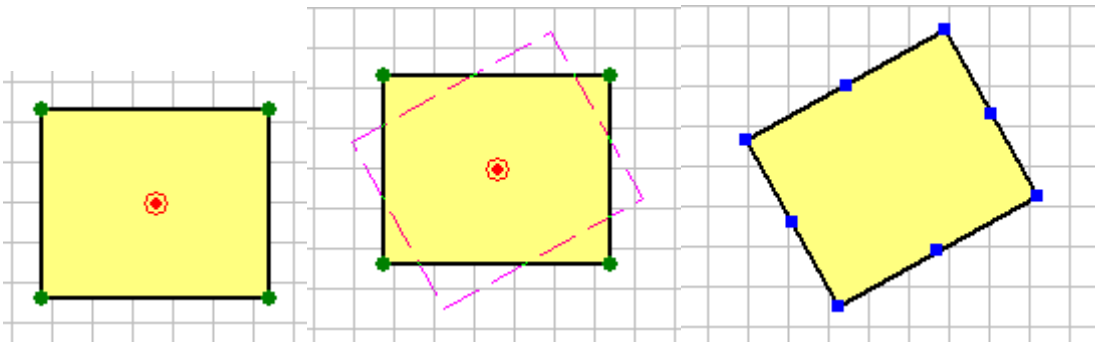
```
  mnIndex := popIndex +3;
end;

procedure TForm1.PowerCad1PopMenuClick(Sender: TObject;MenuIndex: Integer);
begin
  case idx of
    popIndex +1: Powercad1.Clear(0);
    popIndex +1: PowerCad1.ExecuteTbCommand(cGridStep);
  end;
end;
```

6 The PowerCad Event Engines - Will be documented later

Any action in Powercad that should be reported to the developer is now is an internal event, and when you register a function to this event it is called when the even is raised. This is different then the standard Delphi events , because these events can call limited number of functions. These events will be documented later.

6 Quick Rotate

On the Popupmenu, click the rotate item and the selected figure's modification points will change. When you move the green points, the figure will be rotated, and when you move the center red point, the rotation center will be changed for the next rotate.

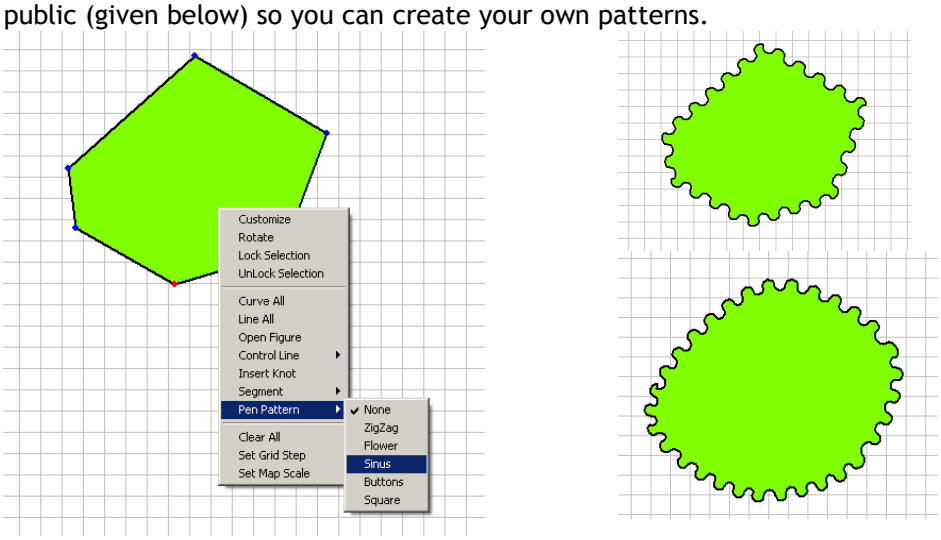


7 Polyline Issues

There are two main update in the polyline feature. The vector pen patterns and knot insertion.

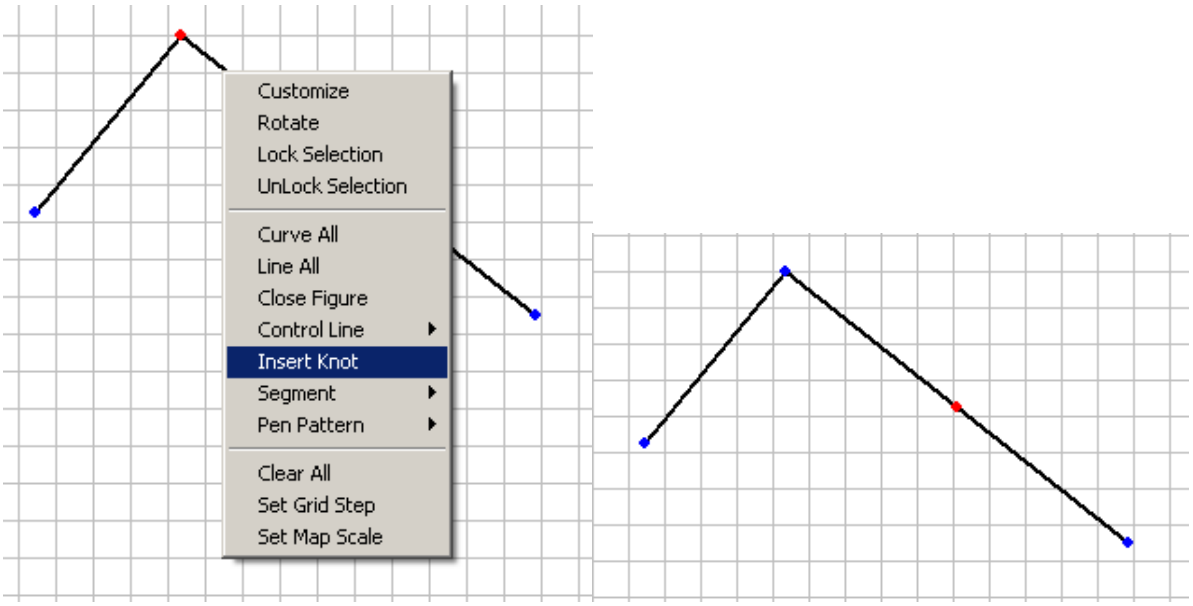
Pen patterns: Now powercad supports vectoral defined patterns to draw through the path of the polyline. A pattern is a vector set which tells the poyline pen what to draw through the path. The patterns are so powerful and can be used in an application in any way, they can be stored to files and they can be created in runtime or design time.

At the moment only polyline figure can use the pen patterns but later all figures will support patterns. To show what a pattern is, we have introduced built in patterns and their sources are public (given below) so you can create your own patterns.



Above, a polyline is assigned a sinus pattern. Below the same pattern is assigned to a bezier form polyline.

Knot Insertion: You can insert knots to polylines using the right-click menu. The knot is inserted in the middle of the segment.



Source Code for Pen Pattern Creation

```
PVector:= TVector.Create(0);
PVector.AddLineSegment(DoublePoint(2,2));
PVector.AddLineSegment(DoublePoint(4,0));
BPattern := Tpattern.Create(PVector,4,0);
BPattern.PatName := 'ZigZag';
Powercad1.NewPattern(BPattern);

PVector:= TVector.Create(0);
PVector.AddBezierSegment(DoublePoint(3,0),DoublePoint(0,2),DoublePoint(3,2));
BPattern := Tpattern.Create(PVector,3,0);
BPattern.PatName := 'Flower';
Powercad1.NewPattern(BPattern);

PVector:= TVector.Create(0);
PVector.AddBezierSegment(DoublePoint(3,0),DoublePoint(0,2),DoublePoint(3,2));
PVector.AddBezierSegment(DoublePoint(6,0),DoublePoint(3,-2),DoublePoint(6,-2));
BPattern := Tpattern.Create(PVector,6,0);
BPattern.PatName := 'Sinus';
Powercad1.NewPattern (BPattern);

PVector:= TVector.Create(0);
cObject := TVectorObject.CreateCircleObject(DoublePoint(1.5,0),1.5);
PVector.AddVectorObject(cObject);
BPattern := Tpattern.Create(PVector,3,0.5);
BPattern.PatName := 'Buttons';
Powercad1.NewPattern (BPattern);

PVector:= TVector.Create(0);
SetLength(dp,4);
dp[0] := doublePoint(0,0);
dp[1] := doublePoint(0,3);
dp[2] := doublePoint(3,3);
dp[3] := doublePoint(3,0);
cObject := TVectorObject.CreatePolygonObject(4,dp);
PVector.AddVectorObject(cObject);
BPattern := Tpattern.Create(PVector,3,0.5);
BPattern.PatName := 'Square';
Powercad1.NewPattern (BPattern);
```

Source Code for Pen Pattern Assigment

```
// To assign a pattern to a polyline
MyPolyline.AssignPenPattern(Powercad1.Penpatterrn[2]);

// To remove pattern from a polyline
MyPolyline.AssignPenPattern(nil);
```

8 New Text Features

The text object can be resized by using the modification points. A new modification point is added to use for character spacing.