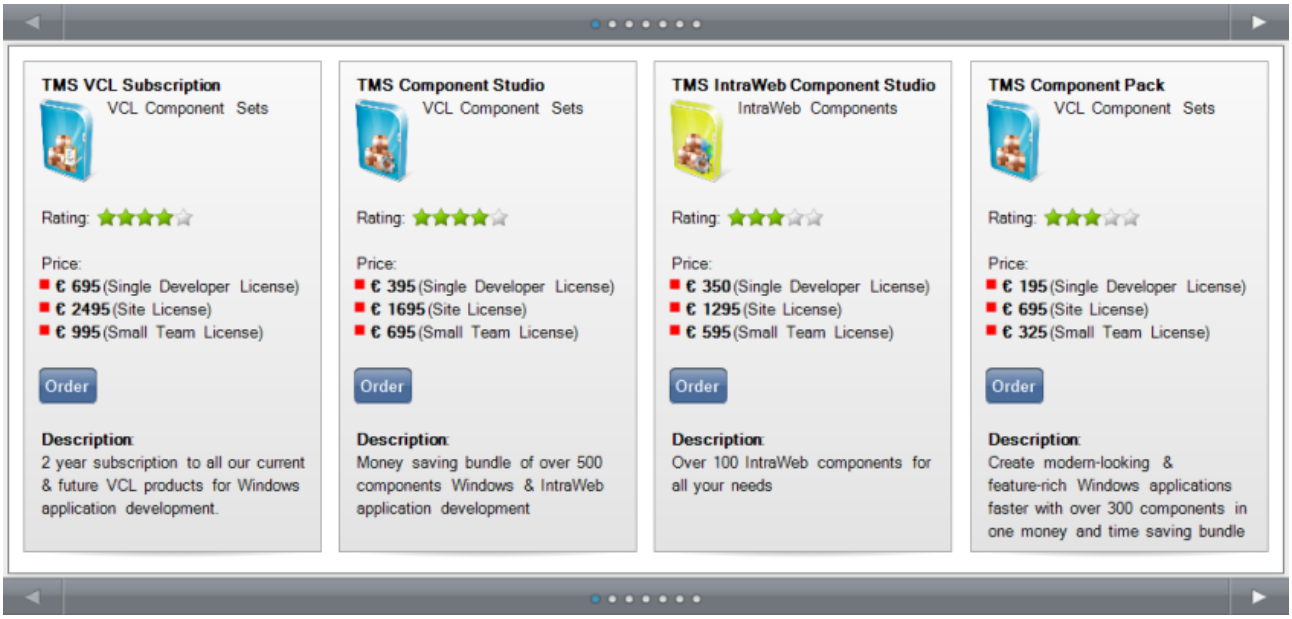
Tiles everywhere : introduction to TAdvSmoothTileList

With the recent demonstration of Windows 8 at E3, Microsoft reassured the Metro, tile based UI is also the way forward in Windows 8. For Microsoft, Windows Phone 7, XBox and Windows 8 will all be based on this Metro tile style UI. At the same time, Apple also uses a tile concept in many places. Its App Store on iPad,iPhone UI for example consists of tiles of information about the apps. At TMS software, we recently developed a new component TAdvSmoothTileList to offer Delphi developers an easy way to add tile based user interface experiences to Delphi applications where it is appropriate. Building a tile based gallery for all kinds of purposes is such perfect example.



Tilelist sample with HTML formatted text

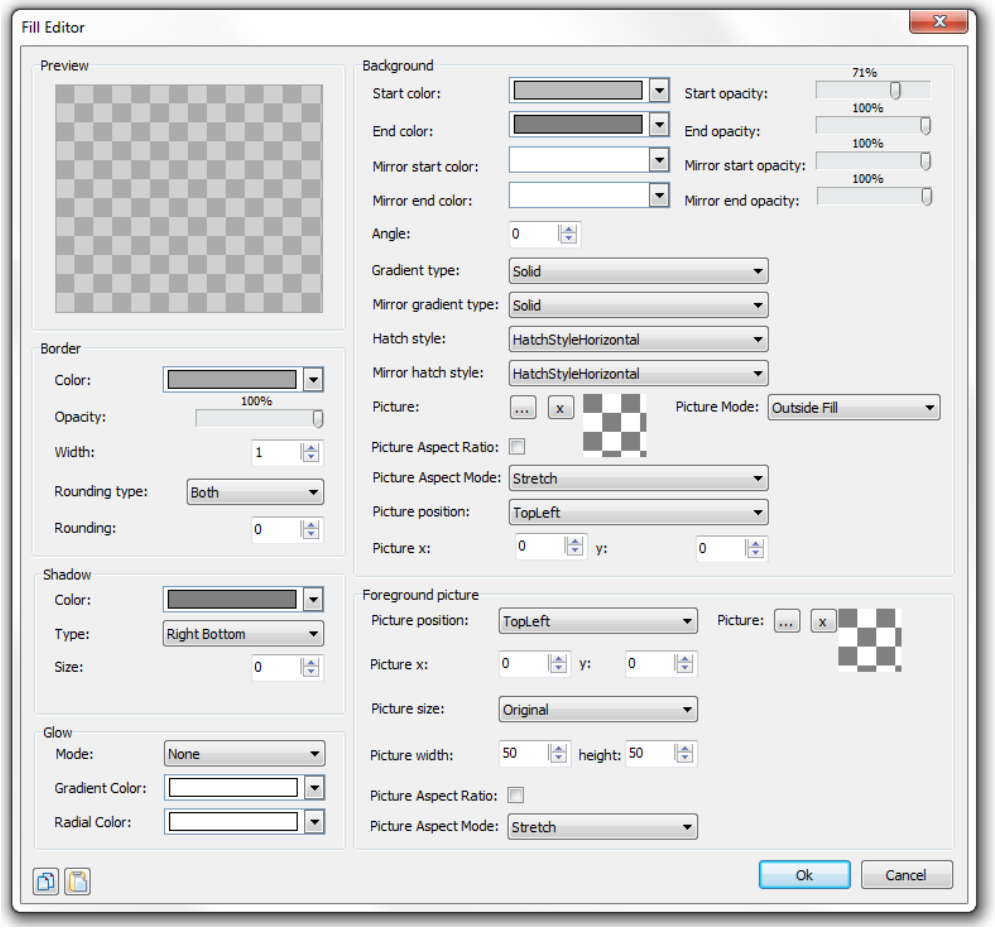
Feature overview TadvSmoothTileList

The tile list is a control displaying tiles with a configurable number of rows and columns. When there are more tiles than can fit on a page, the control will automatically perform paging. A tile can have one of the following states: normal, hover, disabled, selected, maximized.
An optional header and footer can be used to show pages, to navigate between pages, to show a header and/or footer text. The TAdvSmoothTileList is fully touch enabled. Swipes left or right allow to scroll through pages. The TAdvSmoothTileList can be put in edit mode by a 2 seconds click on a tile. In this mode, a delete button appears top left for each tile. A 2 seconds click on a selected tile allows to reorganize the tile in the list by drag & drop and animated reorganization happens. As with all TMS smooth controls, predefined built-in styles are available to switch the UI colors of the control to multiple Office 2003,2007,2010 color styles, a Windows XP, Vista or 7 color style or a terminal style.
One of the most interesting concepts for the TAdvSmoothTileList is hierarchical tiles and custom visualizers. Each tile in the tile list can have sub tiles. When a tile is double-clicked, the tile list shows the subtiles if it has any defined, alternatively, the tile will be displayed in maximized mode. You can have as many levels of sub tiles. A tile can have a different content for its normal state and its maximized state. The default built-in visualizer shows the content of a tile as text. It is possible though to create and assign custom visualizers for an item to display an item in its normal state and maximized state. Currently, the TAdvSmoothTileList

comes with 2 visualizers: one to display tiles of images and one to render HTML formatted content of a tile.

Getting started with TAdvSmoothTileList

Using the TAdvSmoothTileList is simple and straightforward. At its heart is a collection of tiles: TAdvSmoothTileList.Tiles. The appearance of a tile is controlled by TAdvSmoothTileList.TileAppearance. This basically has settings for the appearance of a tile in its small view for normal mode, hover mode, selected mode and disabled mode. See TAdvSmoothTileList.TileAppearance.SmallViewFill* properties.



Design-time fill editor

For the maximized state, the appearance is defined by TAdvSmoothTileList.TileAppearance.LargeViewFill.
To add tiles to the list, add code similar to:

```
var
  i: integer;
  t: TAdvSmoothTile;
begin
  for i := 0 to 20 do
  begin
    t := AdvSmoothTileList1.Tiles.Add;
    t.Content.Text := 'Item '+ inttostr(i);
    t.ContentMaximized.Text := 'This is the content for the item in maximized
state for item '+ inttostr(i);
  end;
```

```
end;
```

To create a hierarchical tile list, following code can be used:

```
var
  i,j: integer;
  t,st: TAdvSmoothTile;
begin

  AdvSmoothTileList1.Tiles.Clear;

  for i := 0 to 4 do
  begin
    t := AdvSmoothTileList1.Tiles.Add;
    t.Content.Text := 'Category '+ inttostr(i);

    for j := 0 to 20 do
    begin
      st := t.SubTiles.Add;
      st.Content.Text := 'Item '+inttostr(j)+' in category ' + inttostr(i);
      st.ContentMaximized.Text := 'Maximized item here';
      st.ContentMaximized.TextPosition := tpTopLeft;
    end;
  end;
end;
```

Note that by double-clicking on a tile that has subtiles, the sub tiles will be displayed in the list. Press ESC to navigate back to the parent tiles or this can be done programmatically as well with the method : TAdvSmoothTileList.GoBack.

Using the visualizers to display images and HTML formatted text

TAdvSmoothTileList has two properties : TAdvSmoothTileList.Visualizer and TAdvSmoothTileList.VisualizerMaximized. This allows to assign an instance of a class TAdvSmoothTileListVisualizer to the tile list and this will handle the drawing of tiles. There are two visualizers that can be set. One for the tile in normal state and one for the tile in maximized state. In this sample, we're also going to introduce the TGDIPPictureContainer. The TGDIPPictureContainer can be considered as a TImageList on steroids. It offers a plethora on supported image formats: .BMP, .GIF, .JPG, .ICO, .PNG, .WMF. It can mix all these formats and it can also have mixed sizes. Each picture in the TGDIPPictureContainer can be given a name and in controls like TAdvSmoothTileList supporting the TGDIPPictureContainer, the images can be simply referenced with their name. Obvious advantage is that we can easily reuse any kind of image resource we need in the application and as such reduce EXE size. In this code snippet, we first fill the TGDIPPictureContainer and assign it to the TAdvSmoothTileList. Then tiles are created with the picture from the TGDIPPictureContainer set via the Tile.Content.ImageName and in maximized state, the tile content set as HTML formatted text referencing again the same picture but this time via the HTML <IMG> tag:

```
const
  pictures: array[0..4] of string = ('Aerodynamik
1.ico','Autobahn.ico','Computer World.ico','Techno Pop.ico','Man-Machine.ico');

var
  i: integer;
  t: TAdvSmoothTile;
  p: TPictureItem;

begin
```

```
  AdvSmoothTileList1.Tiles.Clear;

  // assign a different visualizer for small view and for maximized view
  AdvSmoothTileList1.Visualizer :=
TAdvSmoothTileListImageVisualizer.Create(self);
  AdvSmoothTileList1.VisualizerMaximized :=
TAdvSmoothTileListHTMLVisualizer.Create(self);

  // fill the picture container
  GDIPPictureContainer1.Items.Clear;
  for i := 0 to 4 do
  begin
    p := GDIPPictureContainer1.Items.Add;
    p.Picture.LoadFromFile('.\'+pictures[i]);
    p.Name := pictures[i];
  end;

  // assign the picturecontainer to the TAdvSmoothTileList
  AdvSmoothTileList1.PictureContainer := GDIPPictureContainer1;

  // create tiles, with image for small view and HTML formatted text for
maximized view
  for i := 0 to 4 do
  begin
    t := AdvSmoothTileList1.Tiles.Add;
    t.Content.Text := 'Main '+ inttostr(i);
    t.Content.ImageName := pictures[i];
    t.Content.ImageStretch := true;
    t.ContentMaximized.TextPosition := tpTopLeft;
    t.ContentMaximized.Text := '<b>Title '+ChangeFileExt(pictures[i],'')+
'</b><br><br>Some information and a <a
href="http://www.tmssoftware.com/">link</a><br>'+
       'Image <img src="'+pictures[i]+'" width="64" height="64">';
  end;
end;
```

Tilelist with image tile visualizer

Creating a tile list with custom extended tile item class

The TAdvSmoothTileList component has been designed with in mind to create descendent classes with extended tile classes that can in turn have extended or derived content classes. To create a new component descending from TAdvSmoothTileList with custom tile content in a custom tile class, first of all, create a class that descends from TAdvSmoothTileContent:

```
TAdvSmoothTileContentEx = class(TAdvSmoothTileContent)
private
  FExtra: string;
published
  property Extra: string read FExtra write FExtra;
end;
```

Next, a class descending from TAdvSmoothTile must be created that will override the creation of the Content and ContentMaximized properties and return instances of the descending new class:

```
TAdvSmoothTileEx = class(TAdvSmoothTile)
private
  FExtra: string;
public
  procedure Assign(Source: TPersistent); override;
  function CreateContent: TAdvSmoothTileContent; override;
  function CreateContentMaximized: TAdvSmoothTileContent; override;
published
property Extra: string read FExtra write FExtra;
end;
```

At the same time, the TAdvSmoothTileEx class, descending from TAdvSmoothTile can introduce new properties, methods, events.
The overrides of the virtual functions CreateContent and CreateContentMaximized should return instances of the classes descending from TAdvSmoothTileContent:

```
function TAdvSmoothTileEx.CreateContent: TAdvSmoothTileContent;
begin
  Result := TAdvSmoothTileContentEx.Create(Self);
end;
```

Next in the hierarchy is the tiles collection: TAdvSmoothTiles. For this collection, it is needed to override the function CreateItemClass and return the new class descending from TAdvSmoothTile:

```
TAdvSmoothTilesEx = class(TAdvSmoothTiles)
public
  function CreateItemClass: TCollectionItemClass; override;
end;

function TAdvSmoothTilesEx.CreateItemClass: TCollectionItemClass;
begin
  Result := TAdvSmoothTileEx;
end;
```

Finally, on the highest level, the class descending from TAdvSmoothTileList should override the virtual function CreateTiles and return an instance of the new collection class descending from TAdvSmoothTiles:

```
TAdvSmoothTileListEx = class(TAdvSmoothTileList)
public
  function CreateTiles: TAdvSmoothTiles; override;
end;

function TAdvSmoothTileListEx.CreateTiles: TAdvSmoothTiles;
begin
  Result := TAdvSmoothTilesEx.Create(Self);
end;
```

With this code in place, a new TAdvSmoothTileListEx class is designed that introduces a tile collection that can be an extension of the default TAdvSmoothTile collection and that can have extended Content and ContentMaximized classes. The full sample source code can be found in AdvSmoothTilesEx.pas and shows how easy it is to extend the component.

Creating a custom visualizer

It is simple to create a custom class that will perform the rendering of a tile in either the small view state or the maximized state. In its minimum form, this is a class that overrides one method and performs the drawing in this method. The definition of this basic class becomes:

```
  TMyCustomVisualizer = class(TAdvSmoothTileListVisualizer)
    function DrawText(g: TGPGraphics; R: TGPRectF; Tile: TAdvSmoothTile; Text:
String): TGPRectF; override;
  end;
```

This method DrawText() is called whenever the tile needs to be drawn. The background of the tile is drawn for us before text is drawn. The state of the tile can be retrieved by the property Tile.TileState.

The implementation of the DrawText becomes for this sample:

```
{ TMyCustomVisualizer }

function TMyCustomVisualizer.DrawText(g: TGPGraphics; R: TGPRectF;
  Tile: TAdvSmoothTile; Text: String): TGPRectF;
var
  rct: TRect;
  gp: TGDIPProgress;
begin
  rct := Rect(round(R.X), round(R.Y), round(R.X + R.Width), round(R.Y +
R.Height));
  g.DrawText(Tile.Content.Text, Length(Tile.Content.Text), rct,
Tile.TileList.TileAppearance.SmallViewFont,0, clNone);

  gp := TGDIPProgress.Create;

  // change progressbar color depending on state of the tile
 case Tile.TileState of
  tsSelected, tsMaximized:   gp.ProgressFill.Color := clRed;
  tsDisabled:   gp.ProgressFill.Color := clGray;
  else
    gp.ProgressFill.Color := clLime;
  end;

  OffsetRect(rct, 0, 50);
  InflateRect(rct, -10, 0);
```

```
  rct.Bottom := rct.Top + 20;

  gp.Draw(g, rct, 0,100, Tile.Tag, pbdHorizontal);
  gp.Free;
end;
```

Here we simply reuse the TGDIPProgress class from the GDIPFill unit to draw a progress bar. We could create a custom tile class with the techniques as described in the previous paragraph but to make the sample as simple as possible, the value of the progress bar is set via Tile.Tag.

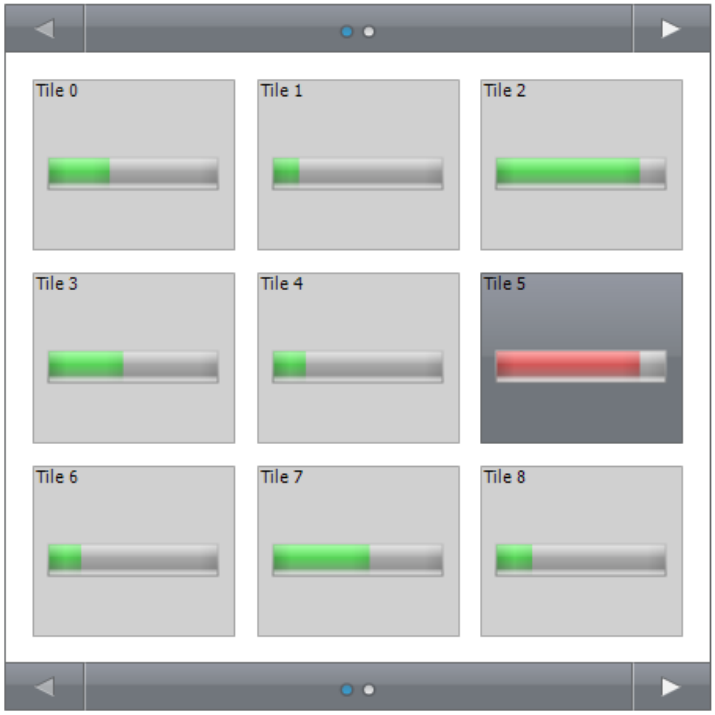
Filling the tile list is done via the code:

```
var
  i: integer;
  t: TAdvSmoothTile;
begin
  AdvSmoothTileList4.Visualizer := TMyCustomVisualizer.Create(Self);
  // same visualizer is used for the maximized state
  AdvSmoothTileList4.VisualizerMaximized := AdvSmoothTileList4.Visualizer;

  AdvSmoothTileList4.Tiles.Clear;

  for i := 0 to 10 do
  begin
    t := AdvSmoothTileList4.Tiles.Add;
    t.Content.Text := 'Tile '+ inttostr(i);
    t.Tag := Random(100);
  end;
end;
```

By overriding the TAdvSmoothTileListVisualizer DoMouseDown method, we could handle clicks on specific areas of the tile in a custom way as well.



Tilelist with custom tile visualizer

Conclusion

We hope this whitepaper illustrated the flexibility and extensibility of our new TAdvSmoothTileList component. We hope it can become a valuable component in your toolbox to create modern user interfaces and we look forward to receive information about the apps you created with this control. TAdvSmoothTileList is part of the TMS Smooth Controls Pack: http://www.tmssoftware.com/site/advsmoothcontrols.asp and TMS Component Pack: http://www.tmssoftware.com/site/tmspack.asp
The sample project used for this whitepaper can be downloaded from http://www.tmssoftware.com/download/AdvSmoothTileListWhitepaper.zip