Study Guide

embarcadero

GETTING STARTED WITH WINDOWS AND MAC DEVELOPMENT

Using Image and Animation Effects

E-Learning Series Course Book

Lesson 8

Embarcadero Technologies

Americas Headquarters
100 California Street, 12th Floor
San Francisco, California 94111

EMEA Headquarters
York House
18 York Road
Maidenhead, Berkshire
SL6 1SF, United Kingdom

Asia-Pacific Headquarters
L7. 313 La Trobe Street
Melbourne VIC 3000
Australia

Lesson 8 – Using Image and Animation Effects

Version: 0.9
Presented: June 21, 2012
Last Updated: June 25, 2012
Prepared by: David Intersimone "David I", Embarcadero Technologies

davidi@embarcadero.com
http://blogs.embarcadero.com/davidi/

Contents

Introduction

FireMonkey includes more than 60 image and animation effects that you can use in your Windows, mac and iOS applications. The Fire Monkey built-in ImageFX engine provides easy-to-use, shader-powered, GPU image processing without the need for complex programming.

The ImageFX engine can be used for 2D image transformations, real-time effects, UI effects, and more. Like Adobe's Photoshop and CoreImage, ImageFX supports multiple filters simultaneously leveraging a single dynamic GPU instruction pipeline for near real-time pixel-perfect performance.

All of FireMonkey's HD and 3D objects can be animated using timeline-based animations to create motion, transitions and effects. You can hook up animations to user controls and input methods such as mouse, touch and motion controllers for highly interactive applications, kiosks, and demonstrations.

Using FireMonkey image and animation effects requires that your computer has a Graphics Processing Unit (GPU). The GPU must support Pixel Shader 2.0, part of DirectX 9.0, which first appeared from various vendors in these products:

- ATI (now AMD) Radeon 9500-X600 series, introduced in 2002.
- NVIDIA GeForce FX (or GeForce 5) series, introduced in 2003.
- Intel GMA 900, introduced in 2004.

When run in a virtual machine, the host's GPU must be accessible. Such support is found in the following virtualization products:

- VMware Workstation version 7 or higher
- VMware Player version 3 or higher
- VMware Fusion for the Mac version 3 or higher

In lesson 8 you'll learn how to enhance your Windows, Mac and iOS HD and 3D user interfaces using FireMonkey's image and animation effects.

Pixel Shader – Cg and HLSL

The Pixel Shader Language specification (NVidia calls their language Cg – C for Graphics, Microsoft calls there HLSL – High Level Shader Language) started as a joint collaboration project between NVidia and Microsoft.

- NVidia - http://en.wikipedia.org/wiki/Cg_(programming_language)
- Microsoft - http://en.wikipedia.org/wiki/High_Level_Shader_Language

Most of FireMonkey's image effects use implementations of the Pixel Shader language for Microsoft DirectX and OpenGL. What does Shader Language look like? Take a look at the implementation section of any of the Filter source code files included in RAD Studio. Looking at **FMX.FilterCatBlur.pas**, which contains the declarations and implementations for the Blur, Gaussian Blur, Directional Blur and Zoom Blur, you'll see implementation code that looks like the following:

```
constructor TGaussianBlurFilter.Create;
const

DX: array [0..1611] of byte = (
  $00, $02, $FF, $FF, $FE, $FF, $2B, $00, $43, $54,
  $41, $42, $1C, $00, $00, $00, $83, $00, $00, $00,
  $00, $02, $FF, $FF, $02, $00, $00, $00, $1C, $00,
  $00, $00, $00, $01, $00, $20, $7C, $00, $00, $00,
  $44, $00, $00, $00, $02, $00, $00, $00, $01, $00,
  $02, $00, $4C, $00, $00, $00, $00, $00, $00, $00,
…);

GLSL: PAnsiChar =
  'vec4 _TMP1;'+
  'vec2 _TMP2;'+
  'vec2 _TMP6;'+
  'varying vec4 TEX0;'+
  'uniform sampler2D texture0;'+
  'uniform vec4 PSParam0;'+
  'void main()'+
  '{'+
  '    vec4 _Color;'+
  '    vec2 _newCoord;'+
  '    _newCoord = TEX0.xy;'+
  '    _newCoord.x = TEX0.x + -
         8.00000000E+000/PSParam0.x;'+
  '    _TMP2 = min(vec2( 1.00000000E+000, 1.00000000E+000),
         _newCoord);'+
  '    _TMP6 = max(vec2( 0.00000000E+000, 0.00000000E+000),
         _TMP2);'+
…);
```

The hexadecimal bytes are the compiled shader language for DirectX on Windows (Microsoft doesn't allow us to ship their HLSL compiler so we have to pre-compile the code). When you use a FireMonkey image effect, the Shader Language code is added to your application executable and sent to the GPU along with the address of the image to be processed.

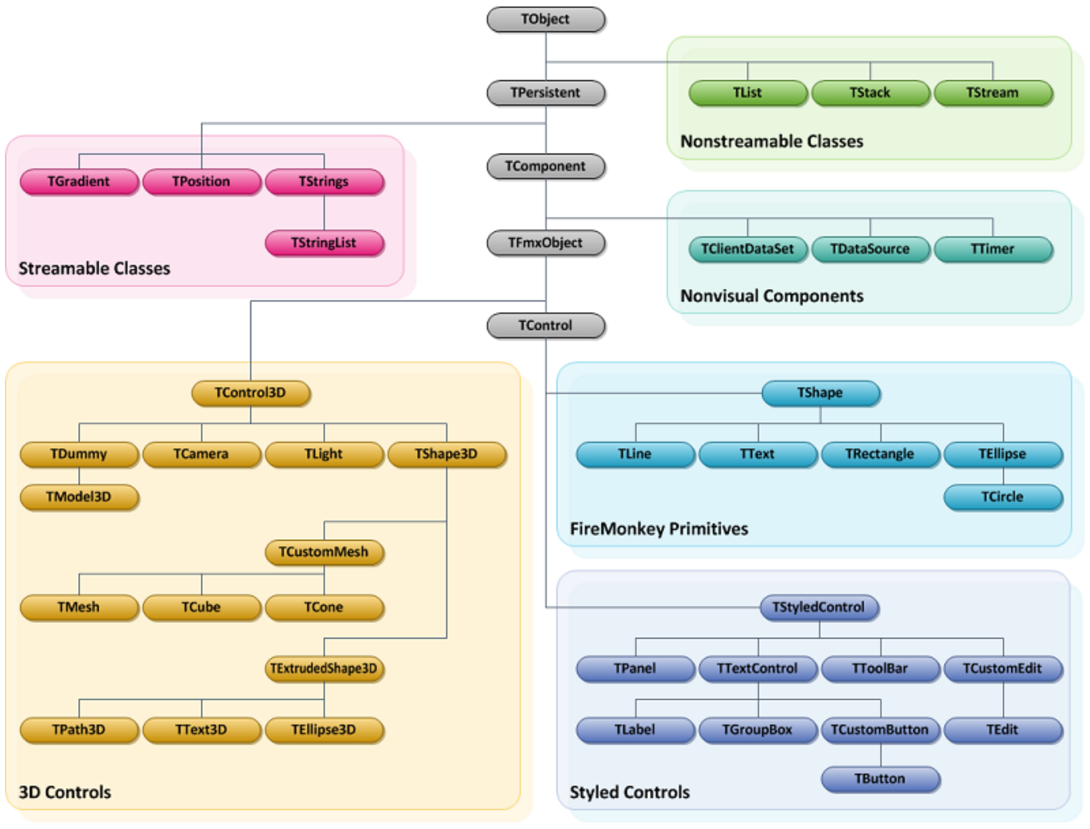
To learn more about FireMonkey's support for Pixel Shader language, check out André Miertschink's Australian Delphi User Group article "How to Create Your Own FireMonkey Effect" at http://members.adug.org.au/2011/12/15/how-to-create-your-own-firemonkeyimage-filtereffect-to-use-with-firemonkey/.

FireMonkey Business Application Platform (FMX)

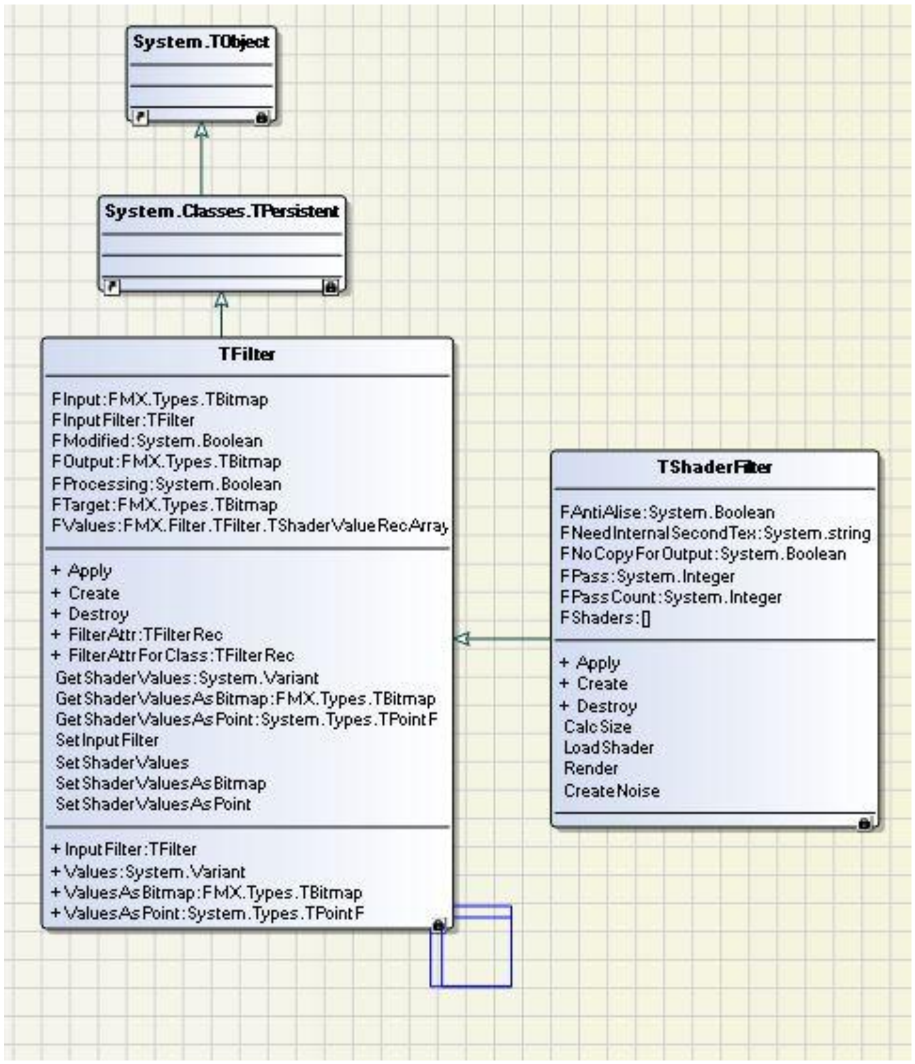
FMX is the unit scope that contains the units and unit scopes of the Fire Monkey application platform. FireMonkey leverages the graphics processing unit (GPU) in modern desktop and mobile devices to create visually engaging applications on multiple platforms, targeting the entire range from the personal to the enterprise. Major features of Fire Monkey include:

- Cross-platform abstraction layer for OS features like windows, menus, timers, and dialogs
- 2D and 3D graphics
- Powerful vector engine (like Adobe Flash or Microsoft WPF)
- Fast real-time anti-aliased vector graphics; resolution independent, with alpha blending and gradients
- WYSIWYG designer and property editors
- Advanced GUI engine - window, button, textbox, numberbox, memo, anglebox, list box, and more
- Advanced skinning engine based on vector graphics styles with sample style themes
- Shape primitives for 2D graphics along with a built-in set of brushes, pens, geometries, and transforms
- **Advanced animations calculated in background thread; easy to use and accurate, with minimal CPU usage and automatic frame rate correction**
- **Bitmap effects rendered by the GPU, including blurring, color filtering, shadows and transitions**
- Flexible layouts and compositing of shapes and other controls
- Layered forms, Unicode-enabled
- JPEG, PNG, TIFF, and GIF format read/write support
- Multi-language engine, editor and examples

The following figure shows the relationship of some key classes that make up the FireMonkey hierarchy. You can also download a FireMonkey architecture schematic poster (PDF file) at http://www.embarcadero-info.com/firemonkey/firemonkey_chart_poster.pdf.



The following UML diagrams show parts of the ImageFX, TFilter, TShaderFilter and Effects classes:


System.TObject
System.Classes.TPersistent
TFilter
FInput:FMX.Types.TBitmap
FInputFilter:TFilter
FModified:System.Boolean
FOutput:FMX.Types.TBitmap
FProcessing:System.Boolean
FTarget:FMX.Types.TBitmap
FValues:FMX.Filter.TFilter.TShaderValueRecArray
+ Apply
+ Create
+ Destroy
+ FilterAttr:TFilterRec
+ FilterAttrForClass:TFilterRec
GetShaderValues:System.Variant
GetShaderValuesAsBitmap:FMX.Types.TBitmap
GetShaderValuesAsPoint:System.Types.TPointF
SetInputFilter
SetShaderValues
SetShaderValuesAsBitmap
SetShaderValuesAsPoint
+ InputFilter:TFilter
+ Values:System.Variant
+ ValuesAsBitmap:FMX.Types.TBitmap
+ ValuesAsPoint:System.Types.TPointF
TShaderFilter
FAntiAlise:System.Boolean
FNeedInternalSecondTex:System.string
FNoCopyForOutput:System.Boolean
FPass:System.Integer
FPassCount:System.Integer
FShaders:[]
+ Apply
+ Create
+ Destroy
CalcSize
LoadShader
Render
CreateNoise

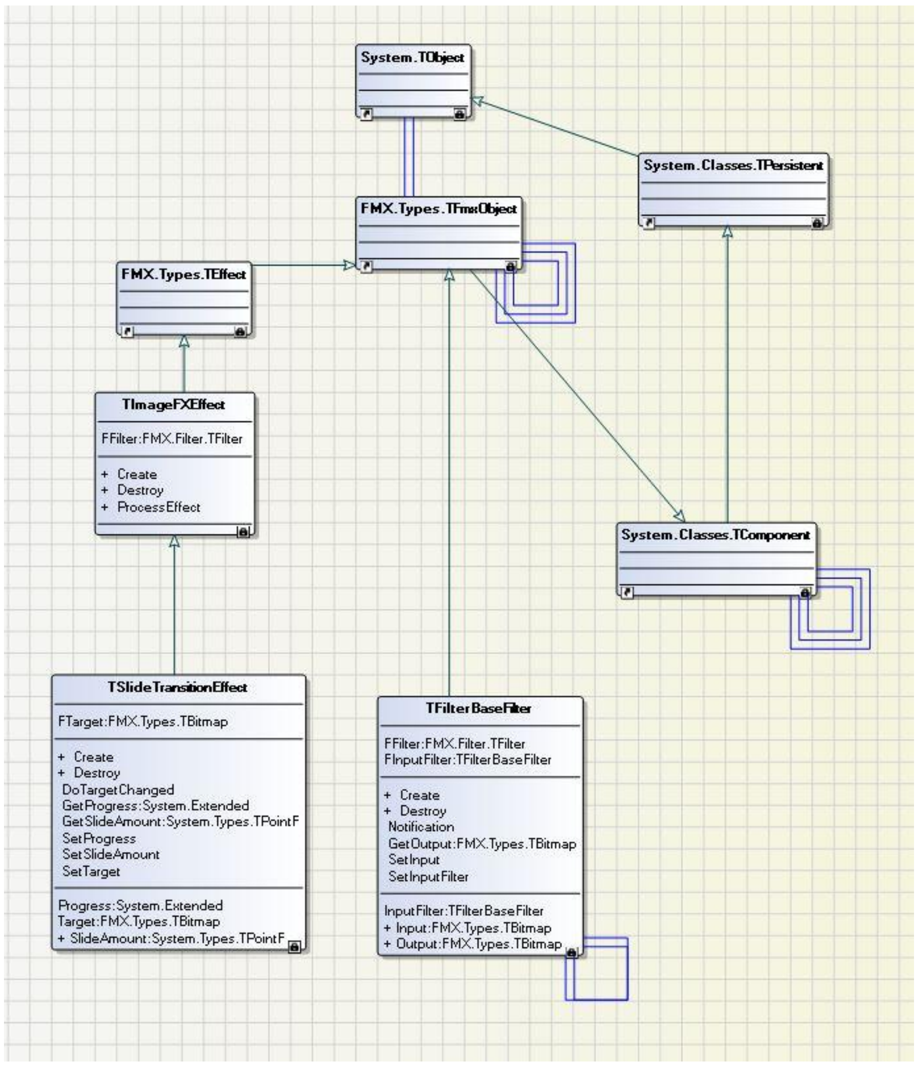

System.TObject
System.Classes.TPersistent
FMX.Types.TFmxObject
FMX.Types.TEffect
TImageFXEffect
FFilter:FMX.Filter.TFilter
+ Create
+ Destroy
+ ProcessEffect
System.Classes.TComponent
TSlideTransitionEffect
FTarget:FMX.Types.TBitmap
+ Create
+ Destroy
DoTargetChanged
GetProgress:System.Extended
GetSlideAmount:System.Types.TPointF
SetProgress
SetSlideAmount
SetTarget
Progress:System.Extended
Target:FMX.Types.TBitmap
+ SlideAmount:System.Types.TPointF
TFilterBaseFilter
FFilter:FMX.Filter.TFilter
FInputFilter:TFilterBaseFilter
+ Create
+ Destroy
Notification
GetOutput:FMX.Types.TBitmap
SetInput
SetInputFilter
InputFilter:TFilterBaseFilter
+ Input:FMX.Types.TBitmap
+ Output:FMX.Types.TBitmap

Image Effects

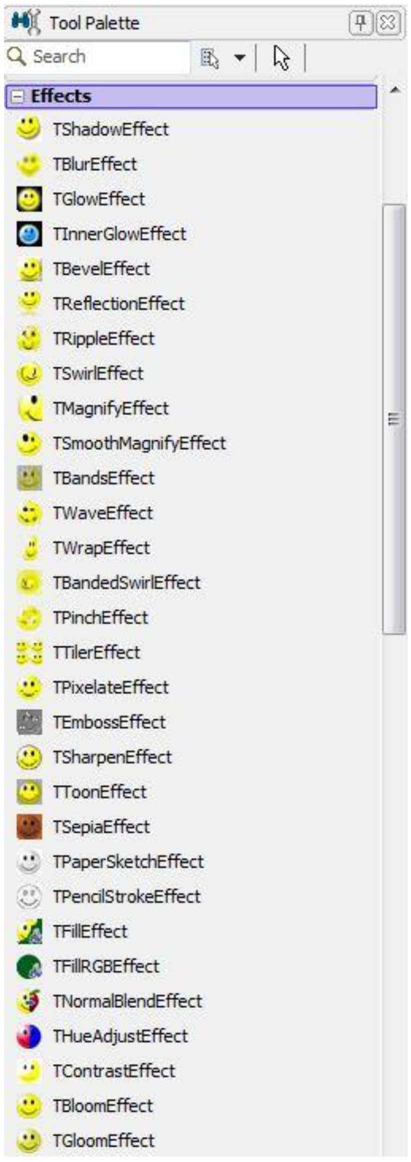
The FireMonkey built-in ImageFX engine provides more than 60 GPU-powered effects. These effects are non-visual components that can be found in the Effects category on the Tool Palette. All the provided effects can be simply enabled or disabled by setting the Enabled flag from the Form Designer, or programmatically.

Almost all the effects have specific properties that you can customize depending on the application. For example, all transition effects have the Progress property, which is used to set the amount of progress (in percentages, %) through the transition from the first texture to the second texture. The specified properties can be found in the Object Inspector when the effect is selected in the Structure View. All numeric properties of any effect can be animated to provide a gradual evolution in time. Image effects can also be triggered.

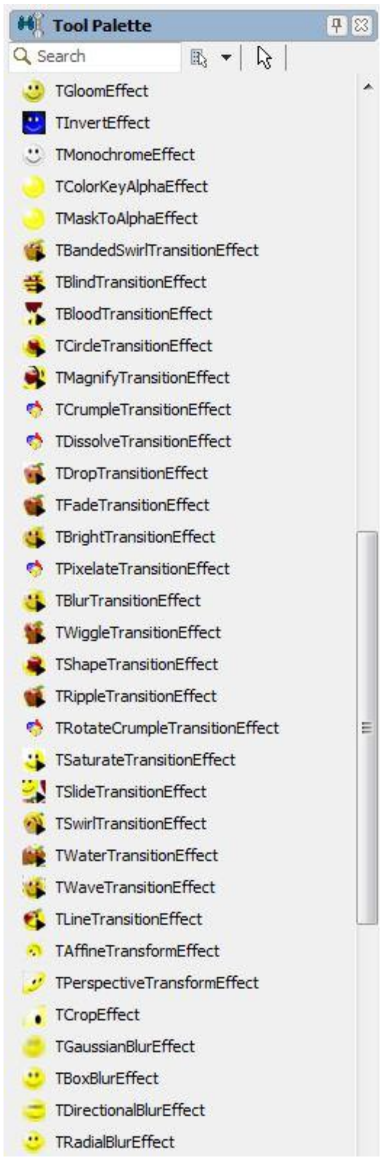
FireMonkey effects are built using pixel shader filters. The shaders modify pixels, either individually or in concert with others, to achieve various visual effects. These effects are not limited to bitmap image data; effects can be applied to the pixels of any 2D control in the user interface. Effects can be used at run time or at design time to change the look of the application's user interface. The FireMonkey effects do not disable any controls or functionalities when they are applied.

Image effects do not work with 3D controls; however they do work with TViewport3D, a 2D control that displays 3D content.

The following bitmaps contain all of the image and transition effects that are available in the XE2 Tool Palette:



TShadowEffect
TBlurEffect
TGlowEffect
TInnerGlowEffect
TBevelEffect
TReflectionEffect
TRippleEffect
TSwirlEffect
TMagnifyEffect
TSmoothMagnifyEffect
TBandsEffect
TWaveEffect
TWrapEffect
TBandedSwirlEffect
TPinchEffect
TTilerEffect
TPixelateEffect
TEmbossEffect
TSharpenEffect
TToonEffect
TSepiaEffect
TPaperSketchEffect
TPencilStrokeEffect
TFillEffect
TFillRGBEffect
TNormalBlendEffect
THueAdjustEffect
TContrastEffect
TBloomEffect
TGloomEffect



TGloomEffect
TInvertEffect
TMonochromeEffect
TColorKeyAlphaEffect
TMaskToAlphaEffect
TBandedSwirlTransitionEffect
TBlindTransitionEffect
TBloodTransitionEffect
TCircleTransitionEffect
TMagnifyTransitionEffect
TCrumpleTransitionEffect
TDissolveTransitionEffect
TDropTransitionEffect
TFadeTransitionEffect
TBrightTransitionEffect
TPixelateTransitionEffect
TBlurTransitionEffect
TWiggleTransitionEffect
TShapeTransitionEffect
TRippleTransitionEffect
TRotateCrumpleTransitionEffect
TSaturateTransitionEffect
TSlideTransitionEffect
TSwirlTransitionEffect
TWaterTransitionEffect
TWaveTransitionEffect
TLineTransitionEffect
TAffineTransformEffect
TPerspectiveTransformEffect
TCropEffect
TGaussianBlurEffect
TBoxBlurEffect
TDirectionalBlurEffect
TRadialBlurEffect

TEffect, TImageFXEffect, TFilter and TShaderFilter

TEffect is the base class for most of the FireMonkey graphic effects classes. Most of the classes in FMX.Effects are descendants (either direct or indirect) of TEffect. FMX.Filter.Effects. TImageFXEffect, a descendant of TEffect, is the base class for some of the filter effects in FMX.Filter.Effects. TFilter is an abstract base class for every filter.

TEffect descendants are non-visual components that can be found in the Effects category on the Tool Palette. You can apply an effect to any FireMonkey visual component. To use a descendant of TEffect at design time, make sure that the effect is a child of the component; for example, a button on a form. To add an effect, drop an e ffect component from the Tool Palette onto the form, and then, in the Structure View, move the effect component so that it is a child of the button control in the hierarchy.

To set an action that will trigger the effect when applied to the visual component, use the Trigger property in the Object Inspector. Each of the TEffect subclasses have additional properties that you can configure. In FireMonkey, non-visual components are not visible at design time, but they can be seen in the Structure View.

TShaderFilter is a special filter class for the GPU shaders. TShaderFilter is the base class for many of the filters used in the image effects.

Types of Image Effects

The Image Effects fall into five difference categories: effects that modify pixels individually, effects that consider a pixel's neighboring pixels, the additive effects, effects that modify the image as a whole and the transition effects.

Effects That Modify Pixels Individually

These effects typically apply changes to color. Each pixel's color can be considered on its own.

- TInvertEffect – inverts the color of the textures of visible objects
- TColorKeyAlphaEffect - makes pixels of a particular color transparent. Set the ColorKey property to specify the key of the color to become transparent. The tolerance between colors can be changed through the Tolerance property. If Tolerance is 0, no color becomes transparent. As Tolerance is increased, the number of colors that become transparent increases too.
- TMaskToAlphaEffect – converts a color or grayscale image that is masked by alpha.
- TMonochromeEffect - changes the texture of visible objects to a monochrome texture.
- TBloomEffect - intensifies bright regions for the textures of visible objects. The brightness and saturation of the base image and the bloomed regions can be set independently th rough the TBloomEffect properties.
- TGloomEffect – intensifies the dark regions for the textures of visible objects. The brightness and saturation of the base image and the gloomed regions can be set independently through the TGloomEffect properties.
- TContrastEffect – changes the brightness and contrast for the textures of visible objects. To modify the brightness, use the Brightness property. To increase the contrast, use the Contract

property. Default values are used for the Brightness and Contrast properties (Brightness=0, Contrast=1.5).

- THueAdjustEffect – changes the overall hue (the variety or tint of a color) for the textures of visible objects. To mody the hue, use the Hue property.
- TFillRGBEffect – fills the non-transparent pixels of a visible object's texture with a solid color. The color to fill the texture's pixels can be set using the Color property.
- TFillEffect – fills the texture of a visible object with a solid color. The color to fill the texture can be set using the Color property.

Effects that Consider a Pixel's Neighbors

This type of effect is applied using algorithms that use a pixel's neighbors to define the new value of the pixel. These include the Blur and Distortion effects.

- TBlurEffect - blurs the texture of visible objects. Beside properties provided by TEffect, TBlurEffect provides a property, called Softness, which you can use to change the blur softness.
- TDirectionalBlurEffect - blurs, using a directional blur algorithm, the texture of visible objects. The direction of the blur can be changed using the Angle property, and the blur amount of the effect can be changed using the BlurAmount property.
- TBoxBlurEffect - blurs the texture of visible objects using a Box blur algorithm. You can change the blur amount of the effect using the BlurAmount property.
- TGaussianBlurEffect - blurs the texture of visible objects using a Gaussian blur algorithm. You can change the blur amount of the effect using the BlurAmount property.
- TRadialBlurEffect - blurring effect using a radial blur algorithm on the texture of visible objects. The TRadialBlurEffect produces ripples and the center of the ripples can be set using the Center property. You can change the blur amount of the effect using the BlurAmount property.
- TBandedSwirlEffect - swirls the bands of the texture of visual objects in spirals. The center of the swirl is specified through the Center property. The swirl aspect can be customized through the AspectRatio property. The amount of spiral winding can be set through the Strength property. The number of the bands in the swirl can be set through the Bands property.
- TBandsEffect - creates bands of bright regions from the texture of visual objects. The density of the bands can be set through the BandDensity property. The intensity of the bands can be set through the BandIntensity property. If BandIntensity is set to 0, TBandsEffect has no visual effect. If BandDensity is set to 1, the image brightness gradually increases from the left edge to the right edge, until it reaches the BandIntensity value.
- TMagnifyEffect - magnifies a circular region of the texture of visual objects. TMagnifyEffect imitates the effect of a magnifying glass. The center of the circular region is specified through the Center property. The aspect of the circular region can be customized through the Radius and AspectRatio properties. The magnification factor can be set through the Magnification property. TMagnifyEffect applies the same magnification over the entire surface of the circular region.
- TPinchEffect - pinches a circular region of the texture of visual objects. The center of the circular region is specified through the Center property. The circular region's aspect can be customized through the AspectRatio property. The amount of winding of the pinched area can be set through the Strength property. The circular region's radius is specified through the Radius property.

- TRippleEffect - superimposes rippling waves upon the texture of visual objects. The center of the ripples is specified through the Center property. The ripples' aspect can be customized through the Amplitude, AspectRatio, and Phase properties. The amount of ripples is set through the Frequency property.
- TSmooth MagnifyEffect - smoothly magnifies a circular region of the texture of visual objects. The center of the circular region is specified through the Center property. The magnified circular area is composed of two concentric zones: a) The inner circular area, where a simple TMagnifyEffect effect is applied. The InnerRadius property specifies the radius of the centered circular area. In this area, a single magnifying factor is applied. b) The outer circular area. The OuterRadius property specifies the radius of this area. In this area, the magnifying factor gradually increases until it reaches the magnifying factor of the inner circular area, from the outside radius to the inner radius. If the OuterRadius value is smaller than or equal to the InnerRadius value, then TSmoothMagnifyEffect has the same effect as TMagnifyEffect. The aspect of the circular region can be customized through the AspectRatio property. The magnification factor can be set through the Magnification property.
- TSwirlEffect - swirls the texture of visual objects in a spiral. The center of the swirl is specified through the Center property. The swirl aspect can be customized through the AspectRatio property. The amount of spiral winding can be set through the Strength property.
- TWaveEffect - applies a wave pattern to the texture of visual objects. The amount of waves can be changed by changing the WaveSize property. The wave aspect can be modified by changing the Time property. Animating Time simulates the changes waves go through in time.
- TWrapEffect - wraps the texture of visual objects, following two curves. TWrapEffect uses Bezier curves. A Bezier curve is defined by four points. The TWrapEffect properties define, for each of the curves, the end and start points, and two control points. The wrapping is applied by curving the image, starting from the left and right edges. The LeftControl1, LeftControl2, LeftEnd, and LeftStart properties specify the points that define the curve used at the left side of the texture. The RightControl1, RightControl2, RightEnd, and RightStart properties specify the points that define the curve used at the right side of the texture.

Additive Effects

Additive Effects affect the images by adding new elements to the original image. The elements can be added to the edges of the image or to the entire image.

- TGlowEffect - creates a glow effect around a visible object. Beside the properties provided by TEffect, TGlowEffect provides three specific properties: GlowColor, Opacity, and Softness.
- TInnerGlowEffect - creates a glow effect around a visible object. Beside the properties provided by TEffect, TGlowEffect provides three specific properties: GlowColor, Opacity, and Softness.
- TReflectionEffect - creates a reflection effect below a visible object. Beside the properties provided by TEffect, TReflectionEffect provides three specific properties: Length, Offset, and Opacity.
- TShadowEffect - creates a shadow effect for visible objects. Beside the properties provided by TEffect, TShadowEffect provides five specific properties: Direction, Distance, Opacity, ShadowColor, and Softness.
- TEmbossEffect - creates an effect that embosses the texture of visible objects. TEmbossEffect finds the contrast lines and adds shadows to them in order to depress or raise the image relative

to its background. The embossing amplitude and width can be set through the Amount and Width properties.

- TPaperSketchEffect - creates an effect that sketches the texture of visual objects. The size of the brush with which TPaperSketchEffect draws the sketch can be set through the BrushSize property.
- TPencilStrokeEffect - creates an effect that sketches the texture of visual objects, so that they appear pencil drawn. The size of the stroke with which TPencilStrokeEffect draws the sketch can be set through the BrushSize property.
- TPixelateEffect - creating an effect that pixelates the texture of visible objects. TPixelateEffect reduces the texture details. The amount of details can be changed through the BlockCount property.
- TSepiaEffect - creates a sepia (dark brown-grey) effect. TSepiaEffect affects the texture of visual objects. The intensity of the sepia color applied over the texture can be set through the Amount property.
- TSharpenEffect - creates an effect that sharpens the texture of visible objects. TSharpenEffect increases the difference in intensity between the texture's pixels. The sharpening amount can be set through the Amount property.
- TToonEffect - creates an effect that applies cartoon-like shading to the texture of visible objects. TToonEffect cartoons the object texture by decreasing the levels of color used in the texture. The numbers of the levels can be set through the Levels property.

Effects that Modify the Image as a Whole

These effects apply geometric changes over the input image.

- TAffineTransformEffect - creates an effect that applies an affine transformation (preserves the straight lines and the ratios of distances between points lying on the straight lines. It does not necessarily preserve the angles or the lengths of the straight lines) to the texture of visible objects. TAffineTransformEffect offers the possibility to rotate and scale the texture of the object to which the effect is applied. The changes are applied only to the object's texture, and not to the entire object. The object's dimensions and position are not affected. To apply a rotation transformation, change the Center and Rotation properties. To scale the object's texture, set the Scale property.
- TCropEffect - crops a rectangle area from the texture of visible objects. The rectangle area to be cropped and displayed as the object's texture is specified by the LeftTop and RightBottom properties. The cropped area is repositioned in the upper-left corner of the object, if it is necessary, and scaled to fit the object boundaries.
- TNormalBlendEffect - creates a normal blending of two images. TNormalBlendEffect affects the textures of visual objects. The normal blending is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TNormalBlendEffect has no visual effect. If the Target's image has no transparent areas, the object's image is completely covered by Target.
- TPerspectiveTransformEffect - creates an effect that applies a perspective transformation to the texture of visible objects. The perspective transformation can be set and customized by using the BottomLeft, BottomRight, TopLeft, and TopRight properties. Each property specifies a corner point of the final transformation.

- TTilerEffect - creates an effect that tiles the texture of visual objects across multiple rows and columns. The number of rows and columns can be set through the HorizontalTileCount and VerticalTileCount properties. The vertical and horizontal offsets of each tile can be set through the VerticalOffset and HorizontalOffset properties. The size of a tile is determined by the number of columns and rows in which the original size of the texture is split. The texture is resized to fit the dimensions of a tile, without preserving the ratio.

Transition Effects

FireMonkey includes over twenty image transition effects, in which source pixels are progressively transformed into a target bitmap image, from simple fades to fancy banded swirls. The progress of the transformation is deterministic and can be set to an arbitrary percentage. This percentage can be animated to transition over time by using the Progress property.

- TBandedSwirlTransitionEffect - makes a transition between the texture of visible objects and another texture, swirling the texture of visible objects. The transition is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TBandedSwirlTransitionEffect also uses the texture of the object to which the effect is applied as the second texture of the transition. TBandedSwirlTransitionEffect swirls the texture of the object, forming spirals. The frequency with which spirals are formed can be set through the Frequency property and the amount of twisting in the spirals can be set through the Strength property. The center of the swirling can be set through the Center property.
- TBlindTransitionEffect - creates a blinds effect that makes a transition between the texture of visible objects and another texture. The transition is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TBlindTransitionEffect has no visual effect. TBlindTransitionEffect divides the texture of the object into strips, forming blinds. The blinds' number can be set through the NumberOfBlinds property.
- TBloodTransitionEffect - applies a transition between the texture of visible objects and another texture, using a dripping motion. The transition is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TBloodTransitionEffect also uses, as the second texture of the transition, the texture of the object to which the effect is applied. The dripping can be customized by changing RandomSeed.
- TBlurTransitionEffect - makes a blur transition between the texture of visible objects and another texture. The transition is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TBlurTransitionEffect also uses the texture of the object to which the effect is applied as the second texture of the transition.
- TBrightTransitionEffect - makes a transition between the texture of visible objects and another texture by brightening the two textures. The transition is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TBrightTransitionEffect also uses the texture of the object to which the effect is applied as the second texture of the transition.

- TCircleTransitionEffect - applies a transition between the texture of visible objects and another texture, using a circle mask. The transition is made between the texture of object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TCircleTransitionEffect has no visual effect. TCircleTransitionEffect blinds the object's texture in a circular area. The circular's area size is specified by the Size property and its center is specified by Center. The circular's area is correlated to the object size and influenced by the Progress property. If the object has a rectangular shape, the blinded area is an ellipse. The circle fuzziness can be changed through the FuzzyAmount property.
- TCrumpleTransitionEffect - makes a transition between the texture of visible objects and another texture by crumpling the two textures. The transition is made between th e texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TCrumpleTransitionEffect also uses the texture of the object to which the effect is applied as the second texture of the tra nsition. The applied distortions can be changed through the RandomSeed property.
- TDissolveTransitionEffect - makes a transition between the texture of visible objects and another texture, by dissolving random areas. The transition is made between the text ure of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TDissolveTransitionEffect has no visual effect. The dissolved areas can be changed through the RandomSeed property.
- TDropTransitionEffect - makes a transition between the texture of visible objects and another texture, by randomly dropping down the pixels columns of the first texture. The transition is made between the texture of the object to which the e ffect is applied and the bit map specified by the Target property. If Target is not specified, TDrop TransitionEffect also uses the texture of the object to which the effect is applied as the second texture of the transition. Change RandomSeed property to change the seed that determine the dripping.
- TFadeTransitionEffect - makes a transition between the texture of visible objects and another texture by fading the two textures. The transition is made between the texture of the object to which the effect is applied and the bitmap specifi ed by the Target property. If Target is not specified, TFadeTransitionEffect has no visual effect.
- TLineTransitionEffect - makes a transition between the texture of visible objects and another texture, using a line to tie the textures. The transition is m ade between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TLineTransitionEffect has no visual effect. The line that delimits the two textures is defined through the Origin, Offset, and Nor mal properties. To set the fuzziness amount of the line, set the FuzzyAmou nt property. The progress of the transition between the two textures can be changed by using the Pro gress property. When Progress is set to 0%, the line passes through the Origin point. If Progress is set to 100%, the line passes through the Offset point. The second point that defines the line is calculated depending on Normal. Normal determines the line orientation. If the X coordinate of Normal is 0, then the li ne is parallel with the X axis. If the Y coordinate of Normal is 0, then the line is parallel with the Y axis.
- TMagnifyTransitionEffect - makes a transition between the texture of visible objects and another texture, using a smooth magnify (smoothly magnifies a circular region) effect. The transition is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TMagnifyTransitionEffect also uses the texture of the object to which the effect is applied as the second texture of the transition. TMagnify TransitionEffect applies a smooth magnify effect on the object's texture,

and inserts the Target in the middle of the magnified area. The Target is also distorted when applying the effect. The center of the magnified area is specified by Center.

- TPixelateTransitionEffect - makes a transition between the texture of visible objects and another texture by applying a pixelating effect over the two textures. The transition is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TPixelateTransitionEffect also uses the texture of the object to which the effect is applied as the second texture of the transition.
- TRippleTransitionEffect - makes a transition between the texture of visible objects and another texture, imitating water ripples. The transition is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TRippleTransitionEffect also uses the texture of the object to which the effect is applied as the second texture of the transition. The ripples are centered and applied over both textures.
- TRotateCrumpleTransitionEffect - makes a transition between the texture of visible objects and another texture by crumpling the two textures. The transition is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TRotateCrumpleTransitionEffect also uses the texture of the object to which the effect is applied as the second texture of the transition. The crumpling is applied by rotating the textures, clockwise, from the margins. The rotation center is in the center of the textures. The distortions that form the crumpling can be changed through the RandomSeed property.
- TSaturateTransitionEffect - makes a transition between the texture of visible objects and another texture by saturating the first texture. The transition is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TSaturateTransitionEffect also uses the texture of the object to which the effect is applied as the second texture of the transition. The transition is made after the first texture is saturated.
- TShapeTransitionEffect - makes a transition between the texture of visible objects and another texture by wiping an irregular shape. The transition is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TShapeTransitionEffect has no visual effect. The wiped irregular shape is positioned in the middle of the textures.
- TSlideTransitionEffect - makes a transition between the texture of visible objects and another texture by sliding the textures. The transition is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TSlideTransitionEffect also uses the texture of the object to which the effect is applied as the second texture of the transition. The maximum distance of the slide, on the X and Y axes, can be set using the point specified through the SlideAmount property.
- TSwirlTransitionEffect - makes a transition between the texture of visible objects and another texture by swirling the first texture. The transition is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TSwirlTransitionEffect also uses the texture of the object to which the effect is applied as the second texture of the transition. TSwirlTransitionEffect swirls the texture of the object, producing spirals. The amount of twisting of the spirals can be set through the Strength property. The swirling center is in the center of the texture.
- TWaterTransitionEffect - makes a transition between the texture of visible objects and another texture, using a troubled water effect. The transition is made between the texture of the object

to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TWaterTransitionEffect also uses the texture of the object to which the effect is applied as the second texture of the transition. TWaterTransitionEffect applies the troubled water effect over the second texture, and then overlaps it over the first texture (the texture of the object to which the effect is applied). To change the water troubling, set the RandomSeed property.

- TWaveTransitionEffect - makes a transition between the texture of visible objects and another texture, using vertical waves. The transition is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TWaveTransitionEffect also uses the texture of the object to which the effect is applied as the second texture of the transition. The wave effect is applied to the first texture.
- TWiggleTransitionEffect - applies a transition between the texture of visible objects and another texture, by wiggling the two textures. The transition is made between the texture of the object to which the effect is applied and the bitmap specified by the Target property. If Target is not specified, TWiggleTransitionEffect also uses the texture of the object to which the effect is applied as the second texture of the transition.

You can see some of these effects in action (using animated GIFs) on the Embarcadero DocWiki at http://docwiki.embarcadero.com/RADStudio/en/FireMonkey_Image_Effects

Setting Effect Triggers

Image effects can be triggered from property changes, setting the Enabled flag in response. Triggers do not work with every arbitrary property, but only with particular properties that are checked during the component's internal event processing. All built-in triggers check boolean properties, and by convention the names of these properties start with "Is".

TControl provides four triggers for every control and shape:

- IsDragOver
- IsFocused
- IsMouseOver (requires HitTest set to True)
- IsVisible

Other built-in triggers include:

- IsActive (TCustomForm)
- IsChecked (TMenuItem)
- IsOpen (TEffect)
- IsPressed (FMX.Controls.TCustomButton)
- IsSelected (TMenuItem, TTabItem, TListBoxItem, TTreeViewItem)

A slightly different set of triggers is provided for animation effects (see the Animations section below). FMX.Types.TFmxObject defines the procedures StartTriggerAnimation, StartTriggerAnimationWait, and StopTriggerAnimation in addition to ApplyTriggerEffect.

Effects can be set to apply when any of these properties change as the result of code or user action. When the trigger condition no longer holds, the effect is removed. Trigger conditions are limited to equality checks, and if the trigger contains more than one condition, they must all evaluate to true for the trigger to fire.

A trigger is expressed as a string containing one or more trigger conditions, separated by semicolons. Each trigger condition consists of the property name, an equal sign, and the trigger value. All the built-in triggers are boolean, so their value must be either True or False. For example:

```
IsMouseOver=true
```

Trigger conditions are stored in the Trigger property.

Using Image Effects

FireMonkey provides many different types of built-in image effects that modify an image either individually or in concert with others to achieve various visual effects.

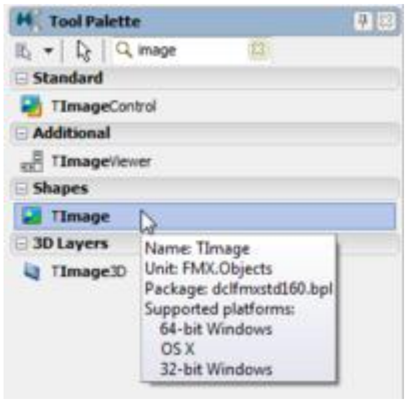
Use the following steps to create a FireMonkey application that uses several basic image effects.

Step 1: Apply an Effect to a Picture

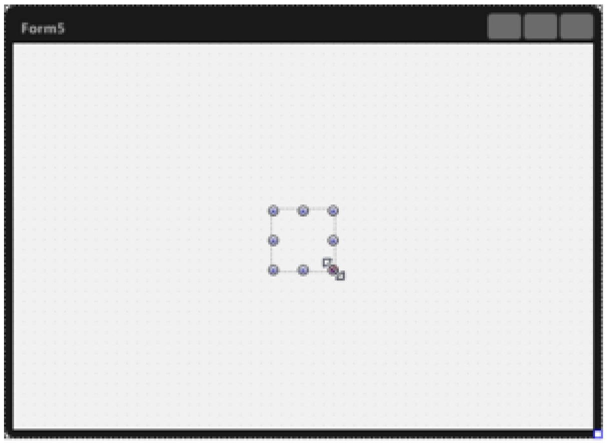
In FireMonkey, applying an image effect to a picture is a straightforward process. Simply create a component that holds a picture, and then apply one of the image effect components.

Create a new FireMonkey application (File > New > FireMonkey HD Application).

Place a TImage component on the form. To do so, type "image" in the search box on the Tool Palette, and then double-click the TImage component:

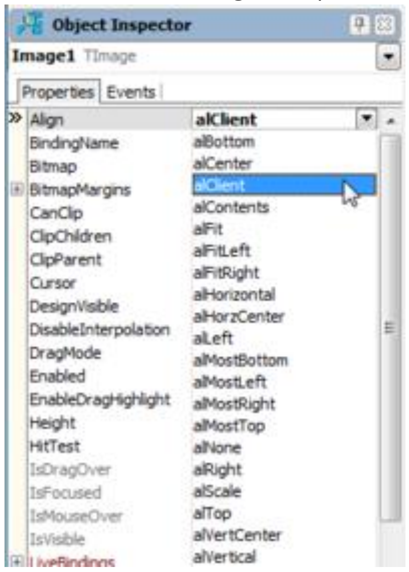


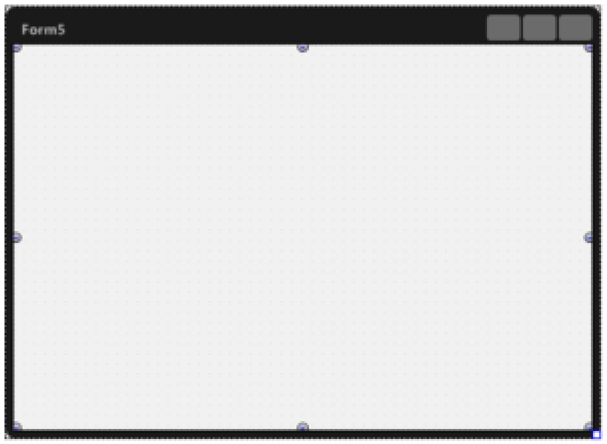
Selecting a TImage component on the Tool Palette



Placing a TImage component on the Form Designer

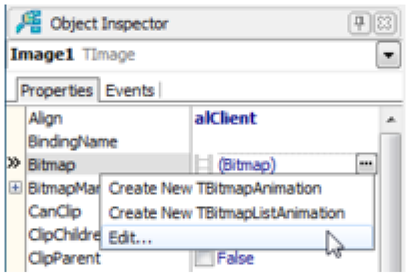
You can see that the TImage component is not placed at the center of the Form Designer. As shown in the image, you need to make the size of the image area as big as possible. To do so, select the TImage component on the Form Designer, and then change the Align property to alClient in the Object Inspector to make the size of the TImage component be the same as the client area of the form.

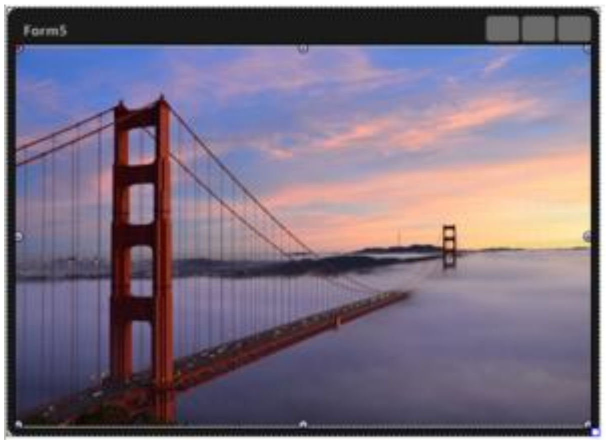




Changing the Align property to alClient

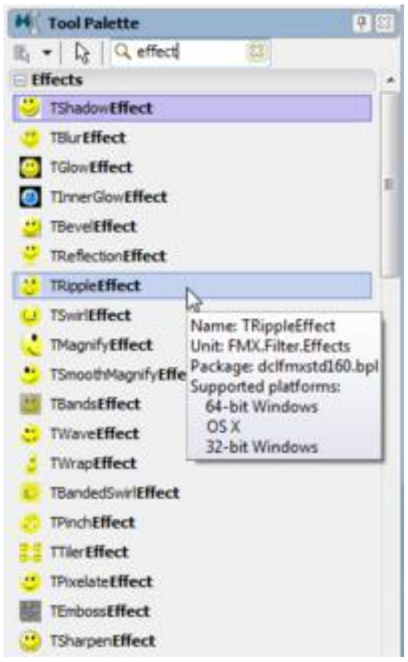
Select the picture to which you want to apply the image effect. The TImage component holds the picture in its Bitmap property. Select the Bitmap property on the Object Inspector, and use the Edit... menu to select a picture.



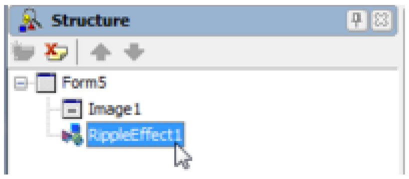


Selecting a picture on a TImage component

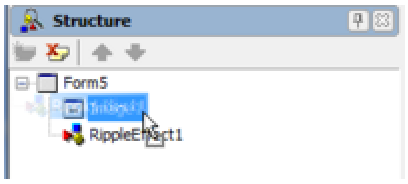
Now you can select an Image Effect Component. Go to the Tool Palette, type “effect” in the search b ox, and select TRippleEffect. In the Tool Palette you can find many effects available. You can read detailed explanations of these effects at our API Reference (FMX. Filter.Effects).

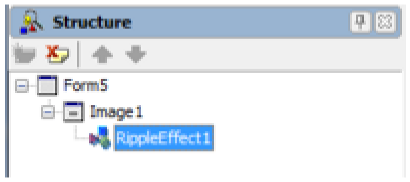


Now the RippleEffect component is displayed on the Structure Pane.



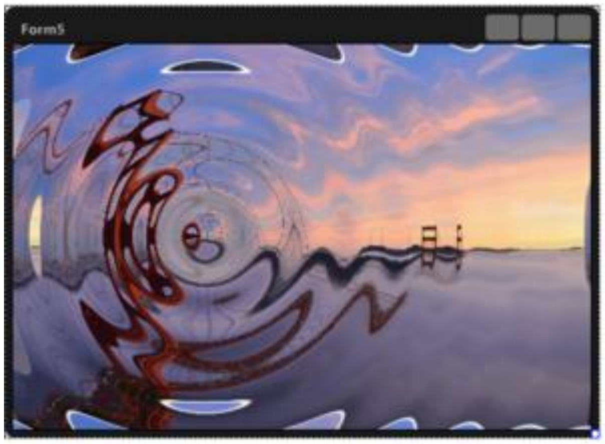
To apply an effect, an effect component has to be defined as a child of another component. In this case, RippleEffect1 should be defined as a child of Image1. To do so, drag RippleEffect1 and drop it to Image1 component on the Structure Pane.





Making the RippleEffect component a child of the Image1 component

Now you can see that RippleEffect is already in works on the Form Designer.



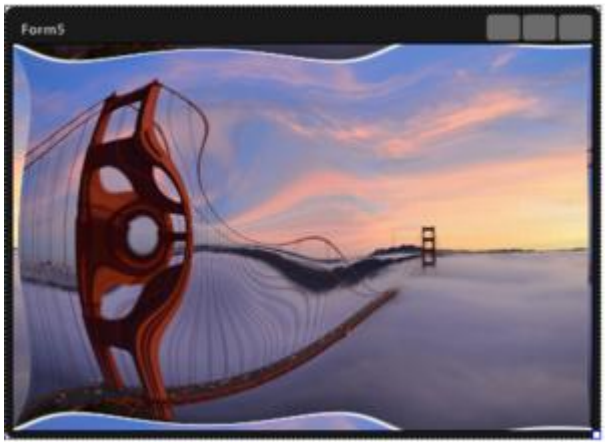
Applying a ripple effect to a picture on the Form Designer

You can also change how the RippleEffect applies to the image by changing some its properties:

- Frequency - Specifies the frequency of the ripples. Frequency is a System.Single value that takes values in the range from 0.00 through 100.00. If Frequency is not explicitly set, it is considered equal to 70.
- Amplitude – Specifies the amplitude of the ripples. Amplitude is a System.Single value that takes values in the range from 0.00 through 1.00. If Amplitude is not explicitly set, it is considered equal to 0.1.
- AspectRatio - Specifies the ratio between the width and height of the ripples. AspectRatio is a System.Single value that takes values in the range from 0.50 through 2.00. If AspectRatio is not explicitly set, it is considered equal to 1.50.
- Phase - Specifies the phase of the ripples. Phase is a System.Single value that takes values in the range from -20.00 through 20.00. If Phase is not explicitly set, it is considered equal to 0.00.
- Center - Center is a TPointF value. If Center is not explicitly set, it is considered equal to (150,150). Assign Center a TPointF value with the X and Y coordinates or use the PointF function as follows:

```
// Delphi code:
RippleEffect1.Center := PointF(0, 0);
// C++ code:
RippleEffect1->Center = PointF(250,150);
```

For example, changing the **Frequency** property to *20* changes the effect as displayed in the following image.



Step 2: Using a Trigger to Enable an Effect.

Using the above example, for the RippleEffect set the **Enabled** property to *False*.

Change the **Trigger** property to *IsMouseOver=True*

Run the application and move the mouse over the bitmap, the RippleEffect will take place.

The Shader Filter Sample Project

You can find the ShaderFilters sample project at: Start > Programs > Embarcadero RAD Studio XE2 > Samples and then navigate to FireMonkey\ShaderFilters

Subversion Repository for the ShaderFilters sample project:
https://radstudiodemos.svn.sourceforge.net/svnroot/radstudiodemos/branches/RadStudio_XE2/FireMonkey/ShaderFilters/.

The ShaderFilters sample project uses the following components:

- Three TImage objects
- Three TListBox objects
- Two TSplitter objects
- Two TPanel objects
- A TAniIndicator
- Two TButton objects
- A TLabel
- A TStyleBook
- Two TLayout objects

The first TListBox displays the main categories of effects. When you select a category of effects from the first TListBox, a list with several specific effects is displayed in the second TListBox. When you select a specific item from the list of effects, the respective effect is applied to the image.

The second TListBox is subordinated to the first one. The second TListBox displays the effects subordinated to the category selected from the first TListBox.

The selected effect is applied to the Source and Target images and a preview of the result is displayed on the Destination area.

Some of the effects have additional characteristics (BlurAmount, BrushSize, and so on). When you select one of these effects, the options are displayed in the third TListBox. Use the TScrollBar to change the characteristics of the effect.

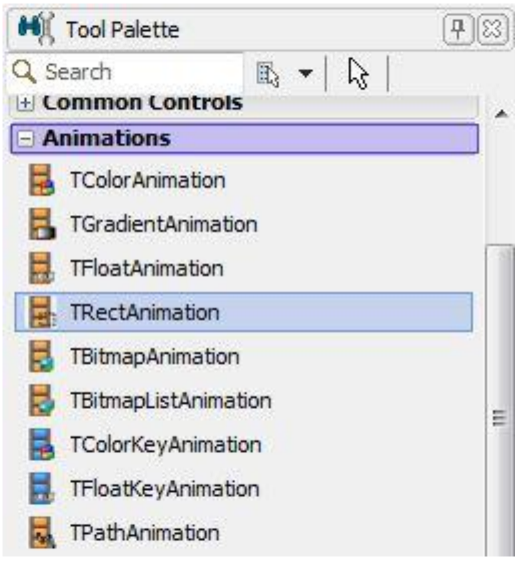
The effects from the Transition main category have an additional button labeled as Play/Pause. Click this button to preview the picture transition.

When you click the Benchmark button, the label next to the button displays the amount of time that passes until the effect is applied to the picture.

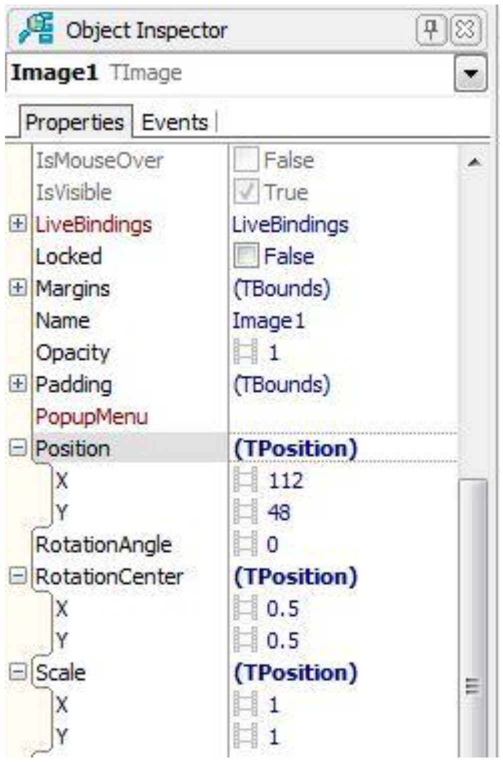
Animations

Animations modify property values over time. They can be started automatically or manually, both with an optional delay. After the animation has run its course over the defined time period, it can stop, start over, or do the same but in reverse.

Non-visual animation components are available for Color, ColorKey, Gradient, Float, FloatKey, Rect, Bitmap, BitmapList, and Path properties.



In the Object Inspector you can see the properties of a component can be animated if you see a film icon in front of a property value.



IsMouseOver	False
IsVisible	✓ True
⊞ LiveBindings	LiveBindings
Locked	False
⊞ Margins	(TBounds)
Name	Image1
Opacity	1
⊞ Padding	(TBounds)
PopupMenu	
⊟ Position	(TPosition)
X	112
Y	48
RotationAngle	0
⊟ RotationCenter	(TPosition)
X	0.5
Y	0.5
⊟ Scale	(TPosition)
X	1
Y	1

You can also animate property values in your code.

Types of Animations

Animations fall into three categories:.

- Interpolations from a start value to an end value:
  - TFloatAnimation changes any numeric property like position (X, Y, and Z axes must be done individually), rotation, and opacity.
  - TRectAnimation changes the location of the four edges of a TBounds property.
  - TColorAnimation changes any string or integer property that contains a color, including those of type TAlphaColor (which is actually a Cardinal number), by modifying the red, green, blue, and alpha values of the color.
  - TGradientAnimation changes a gradient (type TGradient) by modifying the colors of each point that defines the gradient.
  - TBitmapAnimation transitions from a starting bitmap image to another by drawing the final image (type TBitmap) with increasing opacity, causing it to fade into view.

- Interpolating through a series of values, not just two: from the first to the second, from the second to the third, and so on:

  - TFloatKeyAnimation transitions through a list of numbers.
  - TColorKeyAnimation transitions through a list of colors.
  - TPathAnimation modifies an object's 2D Position to follow a Path, optionally rotating it along the way.

- Stepping through a list without interpolation:

  - TBitmapListAnimation works like a timed slideshow, with all images combined horizontally into a single bitmap. With a fast frame rate (short duration and/or many images), it looks like a movie.

Creating Animations

Animations attach as children of the object being animated, just like any other subcomponent. Then the **PropertyName** property is set to a dotted property path, as used with System.TypInfo functions like GetPropInfo; for example, "Opacity" and "Position.Y".

In the Object Inspector, commonly animated properties are indicated with a film strip icon. Choosing to create an animation from the property value's drop-down menu will automatically set the PropertyName. Animation objects added from the Component Palette must set the PropertyName manually.

TFmxObject provides a few convenience methods to create numeric and color animations from code, with the PropertyName and ending value as the required arguments. These animations always start immediately from the current value, are non-looping, and free themselves when finished. The numeric animations can also start with a delay or wait on the main thread until finished.

AnimationType and Interpolation Properties

AnimationType and Interpolation are two important animation properties.

AnimationType determines how the interpolation of an animation is applied. Use TAnimationType to specify how the value of a property changes from its starting value to its ending value (StopValue). Imagine the graph of the quadratic function: y = x**2. The slope of the graph is zero at x = 0. That means that y changes very slowly near x = 0. As x increases, the slope gets steeper and steeper, meaning that y is changing faster and faster. x represents time in the animation, and y is the value of the property being animated.

AnimationType can have one of the following values:

- atIn - The curve that applies to the TInterpolationType for this animation starts at the starting value of the property animated.
- atOut - The curve that applies to the TInterpolationType for this animation starts at the ending value of the property animated and proceeds backwards to the starting value.

- atInOut - The curve that applies to the TInterpolationType for this animation starts at both the starting value and the ending value of the property animated and meets at the center point.

The **Interpolation** property for an animation determines how the rate at which the current value (or StartValue) of a property is changed to the destination value (StopValue) over time. On a graph plotting the value of a property the animation is applied to (using the TAnimation **PropertyName** property) one endpoint is represented by the start value at t=0. The other endpoint is the stop value at t=Duration seconds. Many different paths can connect these two points. The only requirement is that time keeps moving forward! TInterpolationType provides a variety of paths to choose from.

Interpolation can have one of the following values:

- itLinear - A linear interpolation. The property value this animation applies to changes constantly over time.
- itQuadratic - A quadratic function is applied to the path between the start and stop points. The slope of the path is zero at the start point and increases constantly over time. A scalar is applied to the function to make the endpoint fall on the path.
- itCubic - The interpolation is of the form y = x**3. The slope of the path is zero at the start point and increases much faster than the quadratic function over the path.
- itQuartic - The interpolation is of the form y = x**4. The slope of the path is zero at the start point and increases much faster than the quadratic function over the path.
- itQuintic - The interpolation is of the form y = x**5. The slope of the path is zero at the start point and increases much faster than the quadratic function over the path.
- itSinusoidal - The interpolation is of the form y = sin(x). The slope of the path is zero at the start point and places the first inflexion of the sin curve (x=pi) at the stop point.
- itExponential - The interpolation is of the form y = e**x. The slope of the path is one at the start point and increase much faster than the quadratic function over the path.
- itCircular - The path between the start and stop point for this interpolation is a quarter of a circle. The slope of the path is zero at the start point and verticle at the stop point.
- itElastic - The path does not follow a geometric interpolation. The value (y coordinate) may decrease, moving back toward the Start Value, but time (x value) must always move in a positive direction.
- itBack - The path does not follow a geometric interpolation. The value (y coordinate) may decrease, moving back toward the Start Value, but time (x value) must always move in a positive direction.
- itBounce - The path depicts a bouncing ball. The path is made up of circular curves with curvature away from the straight line that connects the start and stop points. These curves are connected by sharp points.

Several properties of the TAnimation and TFloatAnimation can affect the path specified by the interpolation. The descriptions above are for:

- StartFromCurrent = True
- AnimationType = atIn

Setting the AnimationType to atOut causes everything said above about the start point to apply to the stop point. See documentation for these animation properties for their effect.

With an AnimationType property value of atIn and a TInterpolationType of itQuadratic, the value of the property that this animation is applied to (PropertyName) changes slowly near the starting point (equivalent to the quadradic function at x = 0).

With a TAnimationType of atOut, change is slow near the endpoint. For a TAnimationType of atInOut, change is slow at both ends. The curve is mirrored about the center point and meets in the middle.

With an **Interpolation** property of *itLinear*, the value of the property changes linearly over time, and the path between our start and stop points is a straight line.

Starting and Stopping Animations

If an animation's **Enabled** property is set to *True* in the Form Designer, it will automatically Start after the application starts execution. Setting **Enabled** to *True* in code will also Start the animation; setting it to *False* will Stop it. Conversely, calling the Start() and Stop () methods will set **Enabled** to match.

In TFloatAnimation, StartFromCurrent will automatically overwrite the StartValue with the current property value when the animation is started (either by calling Start or if it is Enabled). There is no point setting StartValue if StartFromCurrent is True.

This is especially relevant when manually looping or reusing animation objects. If the previous run used StartFromCurrent, and the next run uses a StartValue, then StartFromCurrent must be set to False.

To stop the animation:

- Setting Pause to True will allow the animation to resume from that point, by setting it back to False.
- Calling Stop will skip to the end of the animation. The property is set to the final value (StopValue if Inverse is False, StartValue if Inverse is True), and OnFinish fires.
- StopAtCurrent does not set the property to the final value, but still fires OnFinish.

Animation Triggers

In addition to automatically starting with Enabled and manually calling Start and Stop, animations can be triggered from property changes. Triggers don't work with every arbitrary property, but only with particular properties that are checked during the component's internal event processing. All built-in triggers check boolean properties, and by convention the names of these properties start with "Is". TControl and TControl3D provide four triggers for every control and shape:

- IsMouseOver
- IsDragOver
- IsFocused
- IsVisible

Other built-in triggers include:

- IsPressed (TCustomButton, TRadioButton, TCheckBox)
- IsChecked (TRadioButton, TCheckBox, TMenuItem)
- IsSelected (TMenuItem, TTabItem, TListBoxItem, TTreeViewItem)
- IsExpanded (TTreeViewItem, TExpander)
- IsActive (TCustomForm)

A slightly different set of triggers is provided for other non-animation image effects.

Animations can be set to Start when any of these properties change as the result of code or user action. When the trigger condition no longer holds, the animation will Stop. Trigger conditions are limited to equality checks, and if the trigger contains more than one condition, they must all evaluate to true for the trigger to fire. A trigger is expressed as a string containing one or more trigger conditions, separated by semicolons. Each trigger condition consists of the property name, an equal sign, and the trigger value. All the built-in triggers are boolean, so their value must be either "true" or " false". For example:

```
IsMouseOver=true
```

Trigger conditions are stored in two properties, **Trigger** and **TriggerInverse**. As their names suggest, the former will simply Start the animation as defined, while the latter will set the animation's Inverse flag first. Because of the way animations run in reverse, and the way animations will immediately "stop at the finish" when the condition no longer holds, instead of a single animation with opposite trigger conditions, sometimes two separate animations defined as opposites are required, each with one of the opposite triggers.

Inverting and Looping Animations

Inverse works by "running time backwards"; it does not flip the start and stop values. Therefore, with both Inverse and StartFromCurrent True, the property will first jump to StopValue, and then animate back to the value at the time the animation started: in effect, "stop at current" (like the boolean StartAtCurrent, not the procedure StopAtCurrent).

An animation can loop repeatedly by setting the **Loop** property to *True*, either in the same direction over and over or back and forth like a pendulum if the **AutoReverse** property is set to *True*.

Controlling Animations using Code

You can also use program code to control animations. In addition to the animations information already presented, here are a some of the animation properties, methods and events that you can use in your programs.

Properties:

- Enabled – set to true to start an animation.

- Pause – if set to True it pauses the animation until set to False.
- Delay – number of seconds to wait before starting the animation.
- StartFromCurrent – if set to true, starts the animation for a property at its current value.
- Running – Indicates whether the animation is currently changing the controlled property. Running is a read-only Boolean value indicating whether the animation is still running. Considering that an animation processes for Duration seconds, returning from the Start method or the procedures calling Start is not an indication that the animation has completed. The OnFinish event handler can also be used to determine when the animation has finished.

Methods:

- Start – initializes the processing of the animation. If you stop the animation before it completes, the animation resumes at the current value if the StartFromCurrent property is true, otherwise it starts over at the StartValue.
- Stop – terminates the processing of an animation. The animation property value is moved to the StopValue property.
- StopAtCurrent – stops at the current property value. Does not change the current value for the property.
- StartTrigger - sets a trigger for a given animation. Besides user applications, StartTrigger is called by StartTriggerAnimation, which allows you to dynamically set the same trigger for all the child animations and for child animations of all the children of the TFmxObject.
- ProcessTick(Time,DeltaTime) - Calls ProcessAnimation a number of times for a given time period. Time is the current time. DeltaTime is the time period for this step.
- NormalizedTime - Returns the percentage of the completion of the animation. Given the current time, NormalizedTime returns a number in the range from 0 through 1, indicating how far the controlled property value has changed from the StartValue to the StopValue. For an Interpolation of itLinear, NormalizedTime is calculated as CurrentTime/Duration. NormalizedTime gets much more complicated for the other Interpolation settings. Using NormalizedTime, the current value for any Interpolation can then be calculated as Result = Start + (Stop - Start) * NormalizedTime; This is the calculation done for float animation or color animation, although color values are not linear.

Events:

- OnProcess - event handler called during the processing of an animation. OnProcess gets called repeatedly while the value controlled by the animation is changing. If the animation changes the appearance of the parent object, the parent object is automatically repainted. If processing inside of OnProcess causes painting outside of the parent's bounding box, call the form's Invalidate method. OnProcess is the correct place to monitor and respond to changes in an animation. Use OnProcess instead of the parent's OnPaint event.
- OnFinish – event handler is called (if one is defined) after the animation has stopped. An animation continues for a Duration number of seconds after it starts, long after the procedure calling Start completes. Place any processing needed after the animation completes in the OnFinish event handler. OnFinish does not get called if the Loop property is True, unless the Stop method is called. OnFinish is the correct place to monitor and respond to the completion of an animation.

In addition to adding animation components to your form, you can use code to create animations for controls and their properties.

AnimateFloat

For a TFloatAnimation you can use the following in your code:

AnimateFloat - Creates and runs a TFloatAnimation for an object's property.

```
// Delphi
procedure AnimateFloat(const APropertyName: string; const
NewValue: Single; Duration: Single = 0.2;  AType:
TAnimationType = TAnimationType.atIn; AInterpolation:
TInterpolationType = TInterpolationType.itLinear);

// C++
void __fastcall AnimateFloat(const System::UnicodeString
APropertyName, const float NewValue, float Duration =
2.000000E-01, TAnimationType AType = (TAnimationType)(0x0),
TInterpolationType AInterpolation =
(TInterpolationType)(0x0));
```

- Creates a TFloatAnimation and makes its parent this TFmxObject (self).
- Sets the PropertyName property of this float animation to be the string provided in the APropertyName parameter. This string must be the name of one of the parent's properties that is of type Float. Use dot notation to reference nested properties. Examples: 'Opacity' and 'Position.X'
- Sets the StopValue property of this float animation to be the short integer provided by the NewValue parameter.
- Sets the Duration property of this float animation to be the integer provided in the Duration parameter. Duration is the number of seconds to transition from the current value of the property named in the PropertyName property to the value of the StopValue property of this float animation.
- Sets the AnimationType property of this float animation to be the TAnimationType provided in the AType parameter.
- Sets the Interpolation property of this float animation to be the TInterpolationType provided in the AInterpolation parameter.
- Sets the OnFinish event of this float animation to be the DoAniFinished method of this TFmxObject.
- Sets the StartFromCurrent property of this float animation to be True.
- Calls the Start to start the animation. The property will be animated from its current value to the StopValue over the Duration time.
- Note: If this object is not Visible at the moment this method is called, no animation runs: the animation is created, but does not execute.

AnimateFloatDelay – Creates a TFloatAnimation for an object's property and delays the start of the execution.

```
// Delphi
procedure AnimateFloatDelay(const APropertyName: string;
const NewValue: Single; Duration: Single = 0.2;  Delay:
Single = 0.0; AType: TAnimationType = TAnimationType.atIn;
AInterpolation: TInterpolationType =
TInterpolationType.itLinear);

// C++
void __fastcall AnimateFloat(const System::UnicodeString
APropertyName, const float NewValue, float Duration =
2.000000E-01, float Delay = 0.0, TAnimationType AType =
(TAnimationType)(0x0), TInterpolationType AInterpolation =
(TInterpolationType)(0x0));
```

- Creates a TFloatAnimation for this object (self).
- AnimateFloatDelay creates a TFloatAnimation and makes its parent this TFmxObject (self).
- Sets the AnimationType property of this float animation to be the TAnimationType provided in the AType parameter.
- Sets the Interpolation property of this float animation to be the TInterpolationType provided in the AInterpolation parameter.
- Sets the Delay property of this float animation to be the real number provided in the Delay parameter.
- Sets the Duration property of this float animation to be the integer provided in the Duration parameter.
- Sets the PropertyName property of this float animation to be the string provided in the APropertyName parameter.
- Sets the StartFromCurrent property of this float animation to be True.
- Sets the StopValue property of this float animation to be the real number provided by the NewValue parameter.
- Calls the Start to start the animation.

AnimateFloatWait – creates a TFloatAnimation and does not return back to your program until the animation is finished.

```
// Delphi
procedure AnimateFloatWait(const APropertyName: string;
const NewValue: Single; Duration: Single = 0.2;  AType:
TAnimationType = TAnimationType.atIn; AInterpolation:
TInterpolationType = TInterpolationType.itLinear);

// C++
void __fastcall AnimateFloatWait(const
System::UnicodeString APropertyName, const float NewValue,
float Duration = 2.000000E-01, TAnimationType AType =
(TAnimationType)(0x0), TInterpolationType AInterpolation =
(TInterpolationType)(0x0));
```

- Creates a TFloatAnimation, makes its parent this TFmxObject (self), and does not return until the animation is finished.

- AnimateFloatWait does the following:
- Sets the AnimationType property of this float animation to be the TAnimationType provided in the AType parameter.
- Sets the Interpolation property of this float animation to be the TInterpolationType provided in the AInterpolation parameter.
- Sets the Duration event of this float animation to be the integer provided in the Duration parameter.
- Sets the PropertyName property of this float animation to be the string provided in the APropertyName parameter.
- Sets the StartFromCurrent property of this float animation to be True.
- Sets the StopValue property of this float animation to be the real number provided by the NewValue parameter.
- Calls Start to start the animation.

The Animation Multi-Platform sample Delphi program shows how to use these methods in action for Windows, Mac and iOS. You can find the Animation-Multi-platform sample project at: Start > Programs > Embarcadero RAD Studio XE2 > Samples and then navigate to FireMonkey\Animation-Multi-platform. The sample is also available in the RAD Studio SourceForge Subversion Repository for Delphi: https://radstudiodemos.svn.sourceforge.net/svnroot/radstudiodemos/branches/RadStudio_XE2/FireMonkey/Animation-Multi-platform/

AnimateColor

Creates a TColorAnimation for this object (self).

```
// Delphi
procedure AnimateColor(const APropertyName: string;
  NewValue: TAlphaColor;
  Duration: Single = 0.2;
  AType: TAnimationType = TAnimationType.atIn;
  AInterpolation: TInterpolationType =
    TInterpolationType.itLinear);

//C++
void __fastcall AnimateColor(const System::UnicodeString
    APropertyName,
  System::Uitypes::TAlphaColor NewValue,
  float Duration = 2.000000E-01, TAnimationType AType =
    (TAnimationType)(0x0),
  TInterpolationType AInterpolation = (TInterpolationType)(0x0));
```

- AnimateColor creates a TColorAnimation and makes its parent this TFmxObject (self).
- Sets the AnimationType property of this color animation to be the TAnimationType provided in the AType parameter.
- Sets the Interpolation property of this color animation to be the TInterpolationType provided in the AInterpolation parameter.
- Sets the OnFinish event of this color animation to be the DoAniFinished method of this TFmxObject.

- Sets the Duration property of this color animation to be the integer provided in the Duration parameter. Duration is the number of seconds to transition from the current color of the property named in the PropertyName property to the color of the Stop Value.
- Sets the PropertyName property of this color animation to be the string provided in the APropertyName parameter. This string is the name of a property of type TColor associated with the parent to animate. For instance, if the parent is a TRectangle, PropertyName could be set to "Fill.Color" or "Stroke.Color". The value of this property will change from the current value stored (if the StartFromCurrent property is "True ") in that property to the value stored in the color animation StopValue property.
- Sets the StartFromCurrent property of this color animation to be True. This causes the animation of the color to start with the color value that is currently stored in the Color property associated with the parent. This could be a TRectangle.Fill.Color.
- Sets the StopValue property of this color animation to be the TColor provided by the New Value parameter. The color animation transitions from the current color value to this StopValue.
- Calls the Start to start the animation.

Custom Animations

Custom animation components can be also created by sub-classing TAnimation and implementing the ProcessAnimation method. ProcessAnimation moves the value of the controlled property by one increment.

ProcessAnimation is a protected method used in the implementation of a descendant class of a TAnimation. ProcessAnimation should be called by the thread configured to process an animation for each increment of time. The increment of time is platform-specific; a specific number of time increments make up a delta time. The ProcessTick method takes a delta time as a parameter and calls ProcessAnimation the required number of times.

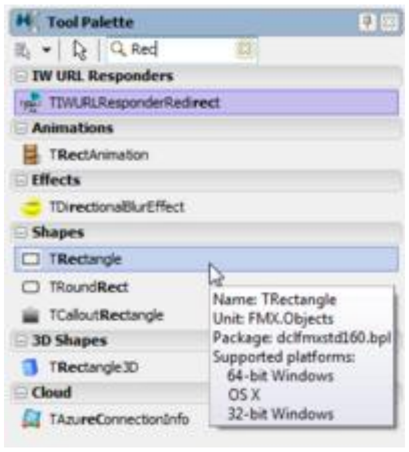
Using Animations

FireMonkey provides many different types of built-in animations that modify the value of the selected property over time. Use the following steps to build a project that uses several basic animation effects.

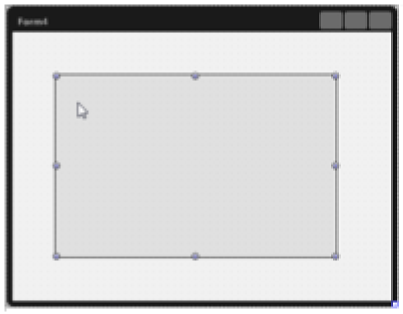
Step 1: Using TFloatAnimation to Change a Floating Property Value

In FireMonkey, any property that uses floating numbers can be modified using TFloatAnimation. So, let's change some values using animation effects.

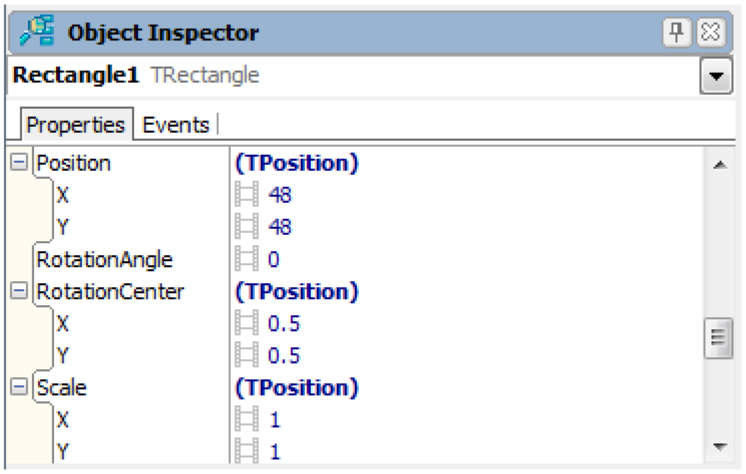
- Create a new FireMonkey HD Application using either **File > New > FireMonkey HD Application – Delphi** or **File > New > FireMonkey HD Application – C++Builder**.
- Place a TRectangle component on the Form Designer. To do so, type rec in the search box on the Tool Palette, and then double-click the TRectanglecomponent.



Name: TRectangle
Unit: FMX.Objects
Package: dclfmxstd160.bpl
Supported platforms:
64-bit Windows
OS X
32-bit Windows



- After you place a **TRectangle** component on the Form Designer, resize the rectangle to look something like the picture above. In the Object Inspector you will see several properties that have a film icon ( ). The film icon indicates that these component properties can be animated.

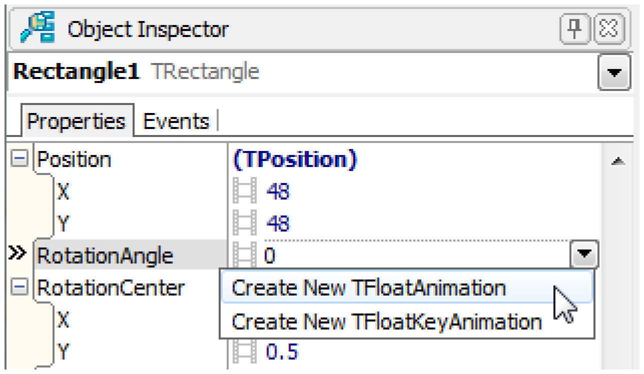


Rectangle1 TRectangle

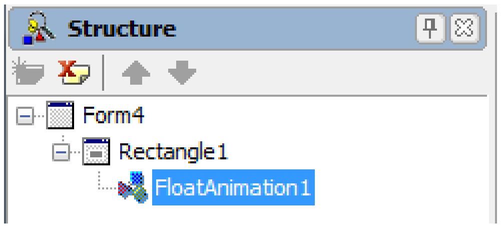
Position	(TPosition)
X	48
Y	48
RotationAngle	0
RotationCenter	(TPosition)
X	0.5
Y	0.5
Scale	(TPosition)
X	1
Y	1

- The following are typical properties (depending on the component) that you can change through **TFloatingAnimation**:
  - Height
  - Position.X
  - Position.Y
  - RotationAngle
  - RotationCenter.X
  - RotationCenter.Y
  - Scale.X
  - Scale.Y
  - StrokeThickness
  - XRadious
  - YRadious
  - Width

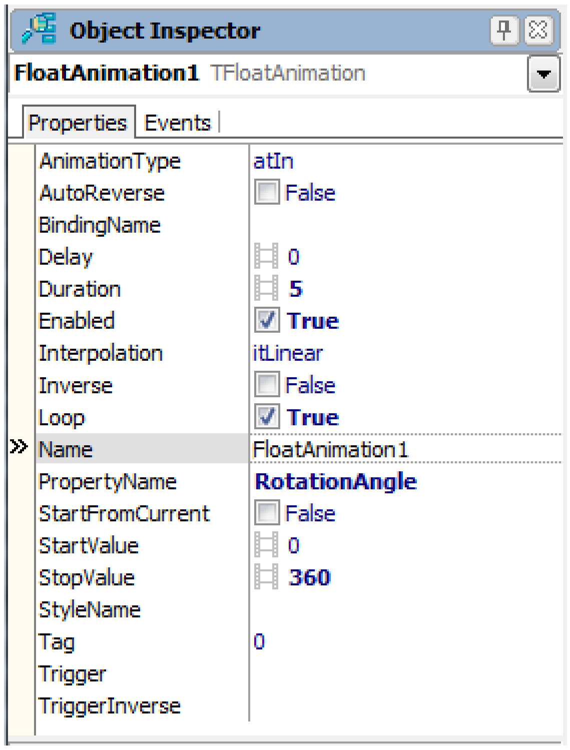
- To modify the value of a property, click the drop-down menu for the **RotationAngle** property, and then select **Create New TFloatAnimation**.



- Now the new TFloatAnimation component is created. You will see that, in the Structure pane, FloatAnimation1 is defined as a child of Rectangle1. Note: Please keep in mind that the effect of the animation-effect components applies to the parent component.



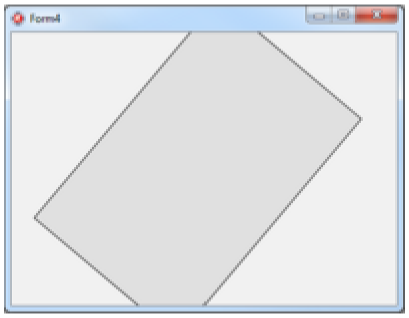
- The Object Inspector shows the properties for the FloatAnimation1 component. Change the following properties:
    - Duration = 5 - The amount of time (in seconds) to animate from the start value to the stop value.
    - Enabled True - The animation starts when the application starts
    - Loop True - Repeats the animation indefinitely.
    - StopValue = 360 - Terminates the animation of this property when it reaches this value.
    - Another important property, which is defined automatically, is PropertyName. In this case, this property is set to RotationAngle; therefore this animation affects the value of the RotationAngle property of its parent component.



FloatAnimation1 TFloatAnimation

AnimationType	atIn
AutoReverse	☐ False
BindingName	
Delay	0
Duration	5
Enabled	✓ True
Interpolation	itLinear
Inverse	☐ False
Loop	✓ True
Name	FloatAnimation1
PropertyName	RotationAngle
StartFromCurrent	☐ False
StartValue	0
StopValue	360
StyleName	
Tag	0
Trigger	
TriggerInverse	

- Execute the application. Now the rectangle component rotates on the form:

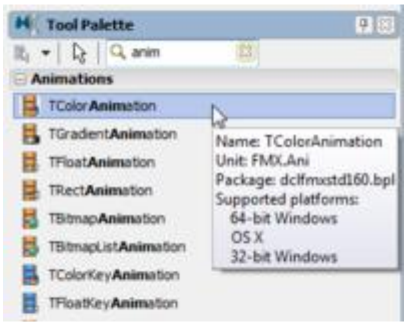


- You can animate any other numeric properties using the same steps.

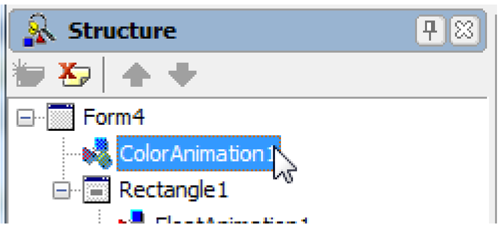
Step 2: Changing the Color by Adding TColorAnimation

Next, you will apply a color animation to change the color of the rectangle, in addition to the rotation described in Step 1.

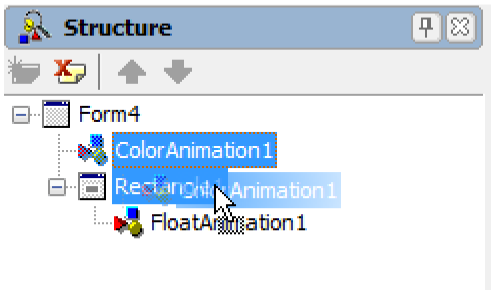
- Select the TColorAnimation component from the Tool Palette. To do so, type anim in the search box on the Tool Palette, and then double-click TColorAnimation:



Name: TColorAnimation
Unit: FMX.Ani
Package: dclfmxstd160.bpl
Supported platforms:
64-bit Windows
OS X
32-bit Windows

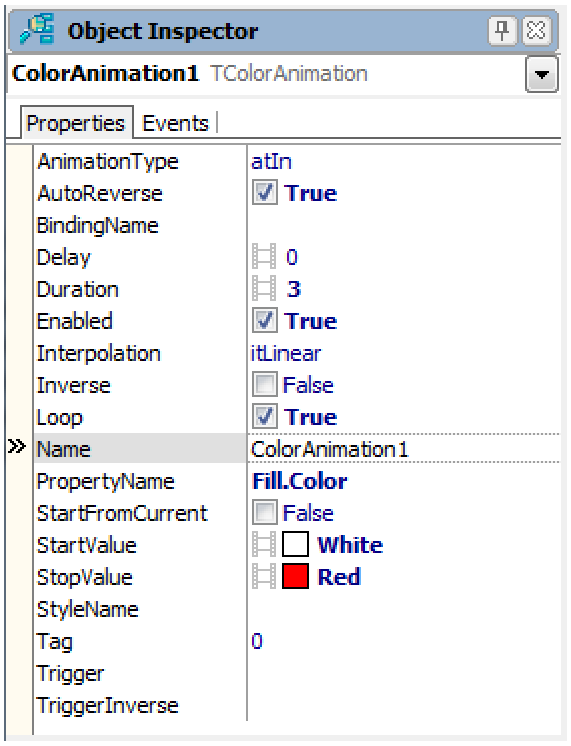


- TColorAnimation1 is defined as a child of the Form while FloatAnimation1 is defined as a child of Rectangle1 (as discussed at Step 1). Drag-and-drop ColorAnimation1 onto Rectangle1. Now ColorAnimation1 is defined as a child of Rectangle1, and therefore ColorAnimation1 affects Rectangle1:



- The Object Inspector shows the properties for the **TColorAnimation1** component. Change the following properties:

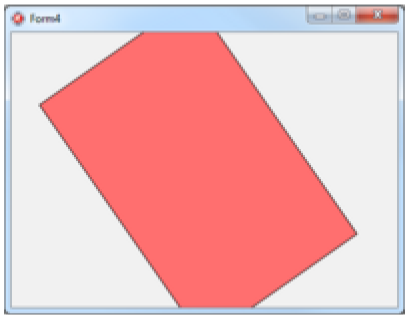
  - PropertyName = Fill.Color - Name of the property to animate.
  - Enabled = True - The animation starts when the application starts.
  - Duration = 3 - The amount of time (in seconds) to animate from the start value to the stop value.
  - Loop = True - Repeats the animation indefinitely.
  - AutoReverse = True- Animates backward after animating forward.
  - StartValue = White
  - StopValue = Red - Terminates the animation of this property when it reaches this value.



ColorAnimation1 TColorAnimation

AnimationType	atIn
AutoReverse	✓ True
BindingName	
Delay	0
Duration	3
Enabled	✓ True
Interpolation	itLinear
Inverse	False
Loop	✓ True
Name	ColorAnimation1
PropertyName	Fill.Color
StartFromCurrent	False
StartValue	⬜ White
StopValue	🟥 Red
StyleName	
Tag	0
Trigger	
TriggerInverse	

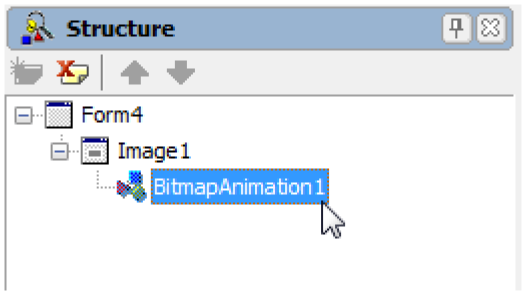
- Execute the application. Now the rectangle component rotates on the form and has its fill.color property change from white to red:



Step 3: Changing an Image by Using TBitmapAnimation

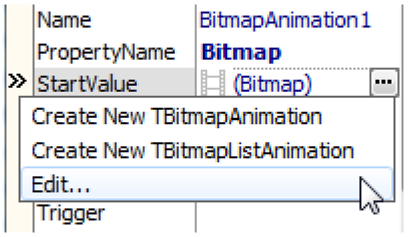
The last step uses a bitmap animation with an image component.

- Create a new FireMonkey application, as in Step 1 of this tutorial.
- Place a TImage component (Image1) on the Form Designer.
- Place a TBitmapAnimation component on the Form Designer. Using the Structure View, set this component (BitmapAnimation1) as a child of Image1:



- Set the properties of **BitmapAnimation1** on the Object Inspector as follows:
    - PropertyName= Bitmap - Name of the property to animate.
    - Enabled = True - The animation starts when the application starts.
    - Duration = 10 - The amount of time (in seconds) to animate from the start value to the stop value.
    - Loop = True = Repeats the animation indefinitely.
    - AutoReverse = True - Animates backward after animating forward.

- Set the StartValue and StopValue Bitmap properties. These properties hold the picture as initial image and final image. Click the "Edit..." property editor from the menu, and then select your favorite pictures using the Bitmap Editor.



Name	BitmapAnimation1
PropertyName	**Bitmap**
StartValue	(Bitmap)
Trigger	

- Execute the application. Now the two pictures you chose are animated over time.

Additional Animation Examples for Delphi and C++

There are several additional animation code examples for Delphi and C++ in the Embarcadero DocWiki.

The FMXObjectAnimateFloat example shows how to call the FMX.Types.TFmxObject method FMX.Types.AnimateFloat. Note that once called, you do not have access to the FMX.Ani.TFloatAnimation instance that FMX.Types.AnimateFloat creates.

- http://docwiki.embarcadero.com/CodeExamples/en/FMXTFmxObjectAnimateFloat_(Delphi)
- http://docwiki.embarcadero.com/CodeExamples/en/FMXTFmxObjectAnimateFloat_(C%2B%2B)

The FMXObjectAnimateColor example shows how to call the TFmxObject method AnimateColor. Note that once called, you do not have access to the TColorAnimation instance that AnimateColor creates.

- http://docwiki.embarcadero.com/CodeExamples/en/FMXTFmxObjectAnimateColor_(Delphi)

- http://docwiki.embarcadero.com/CodeExamples/XE2/en/FMXTFmxObjectAnimateColor_(C%2B%2B)

The AttachTAnimation example demonstrates how the AnimationType and Interpolation properties affect the rate at which the value of a property changes under their control. This is illustrated by controlling the X and Y of a TRectangle with TFloatAnimation instances.

- http://docwiki.embarcadero.com/CodeExamples/en/AttachTAnimation_(Delphi)
- http://docwiki.embarcadero.com/CodeExamples/XE2/en/FMXAttachTAnimation_(C%2B%2B)

The FMXTimerAnimation examples show how to move an image on a form, using three different techniques. The first example uses a TTimer object. The last two projects use animations (TFloatAnimation and TPathAnimation, respectively). All three have the same result: when the user presses the button, the image moves in a diamond shape around the button.

- http://docwiki.embarcadero.com/CodeExamples/en/FMXTimerAnimation_(Delphi)
- http://docwiki.embarcadero.com/CodeExamples/en/FMXTimerAnimation_(C%2B%2B)

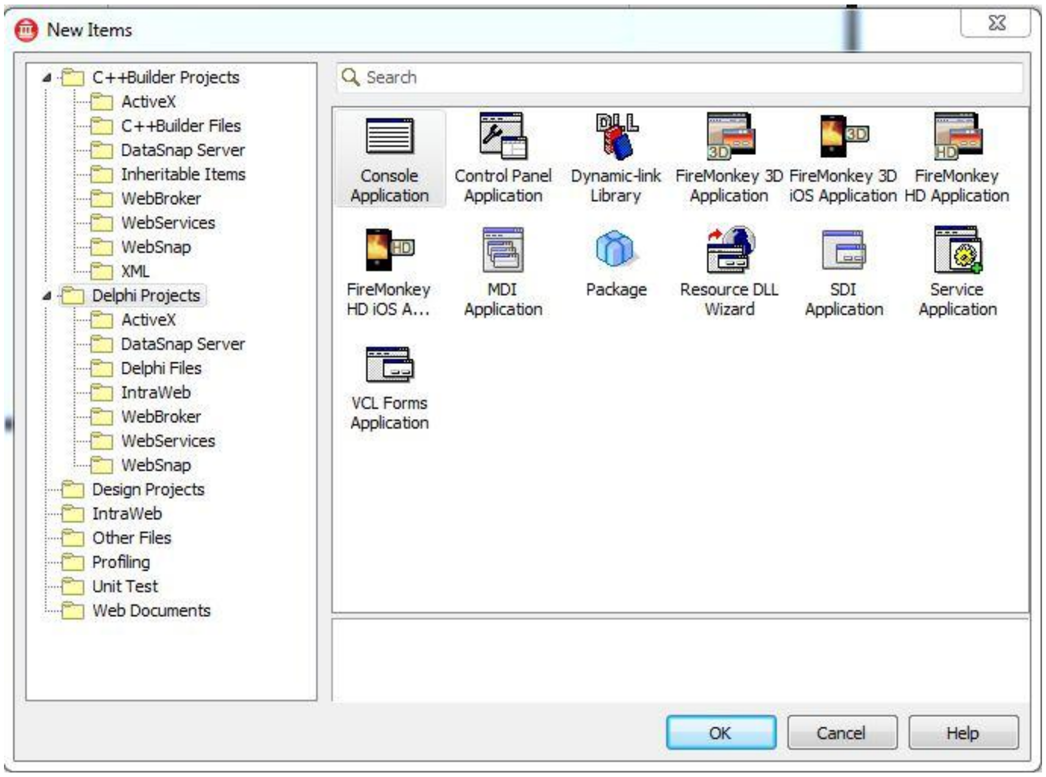
AnimationDemo HD and 3D Samples

Example projects are included with RAD Studio that illustrates how to create animations for HD and 3D Windows and Mac applications.

http://docwiki.embarcadero.com/CodeExamples/en/FMX.AnimationDemoHD_Sample
http://docwiki.embarcadero.com/CodeExamples/en/FMX.AnimationDemo3D_Sample

Image Effects and Animations for iOS

All of the information and examples covered in this lesson will also work using FireMonkey for iOS. Start your project with **File > New > Other… > Delphi Projects > FireMonkey HD iOS Application** or **File > New > Other… > Delphi Projects > FireMonkey 3D iOS Application**.



Follow the same steps outlined above to use Image Effects and Animations in your iOS applications.

Summary, Looking Forward, To Do Items, Resources, Q&A and the Quiz

In Lesson 8 you learned how to add dazzling image and animation effects to your Windows, Mac and iOS (Delphi XE2 only) applications. You learned about the 50+ effects included with FireMonkey. You learned how to use these effects to enhance the user experience in your applications.

In Lesson 9, you'll learn how to put it all together, everything you've learned over the first 8 lessons, to build multi-client, multi-platform and mult-tier applications.

In the meantime, here are some things to do, articles to read and videos to watch to enhance what you learned in Lesson 8 and to prepare you for lesson 8.

To Do Items

Explore all of the image and animation effects components that are included with RAD Studio. Open, explore and run the Animation HD and 3D examples programs that are included with RAD Studio.

Links to Additional Resources

- Getting Started Course landing page - http://www.embarcadero.com/firemonkey/firemonkey-e-learning-series
- FireMonkey Application Platform - http://docwiki.embarcadero.com/RADStudio/en/FireMonkey_Application_Platform
- How to Create Your Own FireMonkey Image Effect by André Miertschink - http://members.adug.org.au/2011/12/15/how-to-create-your-own-firemonkeyimage-filtereffect-to-use-with-firemonkey/
- Rich HD and 3D Business Applications with FireMonkey (YouTube video) - http://www.youtube.com/watch?v=80hfge8NwCE
- Introduction to FireMonkey Effects Delphi and C++ (YouTube video) - http://edn.embarcadero.com/article/42104

Delphi:

- RAD Studio Delphi sample programs on SourceForge - http://radstudiodemos.svn.sourceforge.net/viewvc/radstudiodemos/branches/RadStudio_XE2/FireMonkey/

C++:

- RAD Studio C++ sample programs on SourceForge - http://radstudiodemos.svn.sourceforge.net/viewvc/radstudiodemos/branches/RadStudio_XE2/CPP/FireMonkey/

Pixel Shader:

- Writing HLSL Shaders in Direct3D 9 – http://msdn.microsoft.com/en-us/library/windows/desktop/bb944006(v=VS.85).aspx
- DirectX Pixel Shader 2 – http://msdn.microsoft.com/en-us/library/windows/desktop/bb219843(v=vs.85).aspx
- Microsoft DirectX SDK – http://go.microsoft.com/fwlink/?LinkId=150942
- NVIDIA's Cg Toolkit – http://developer.nvidia.com/cg-toolkit
- The Open GL ES Shading Language – http://www.khronos.org/registry/gles/specs/2.0/GLSL_ES_Specification_1.0.17.pdf

Q&A:

Here are some of the answers for the questions I've received (so far) for this lesson. I will continue to update this Course Book during and after course.

Q:
A:

If you have any additional questions – send me an email - davidi@embarcadero.com

Self Check Quiz

1. Which of the following image effects is not included with FireMonkey?

a) TSepiaEffect
b) TToon Effect
c) TDissolveTransitionEffect
d) TSeasonalEffect
e) TFillEffect

2. Which of the following animation effects is not included with FireMonkey?

a) TFloatAnimation
b) TFilmAnimation
c) TColorAnimation
d) TPathAnimation

3. Image and Animation effects can be used on computers that don't have a GPU?

a) True
b) False

Answers to the Self Check Quiz:

1d, 2b, 3b