

Delphi XE2 之 FireMonkey 入门学习笔记



声明： 本文档收集于网络，仅用于学习交流使用，版权归原作者万一所有，感谢他的辛勤付出。原作者： 万一

本文档是原作者学习 Delphi XE2 过程中记录下的学习笔记，笔记中有大量详细的示例代码，对初学及有使用经验的 Delphi 开发者都有较高的参考意义。为便于阅读，现将全部笔记整理成册，与众多 Delphi 开发者分享。书中所有代码你可以直接到网站 <http://www.delphiXEbbs.com> 参阅查询下载。

Delphi XE2 之 FireMonkey 入门(1)

Delphi XE2 的 FireMonkey 是跨平台的，暂时只准备看看它在 Windows 下(我是 32 位 Win7)的应用情况。

很新的东西，相信有了它，以后的界面将会更灵活、漂亮，也会淘汰掉诸多皮肤、透明、图像等第三方组件，但不知和 Win8 的 WinRT 有无关联。

为了它，Delphi XE2 的诸多单元加了前缀，诸如：

Bde、Data、Datasnap、FMX、IB、Macapi、Posix、Soap、System、System.Bindings、System.Generics、System.Internal、System.Mac、System.Win、Vcl、Winapi、Xml...

这样也好，单元类别一目了然。和 FireMonkey 相关的单元有：

FMX.Ani.pas
FMX.ASE.Importer.pas
FMX.ASE.Lexer.pas
FMX.ASE.Model.pas
FMX.Canvas.D2D.pas
FMX.Canvas.GDIP.pas
FMX.Canvas.Mac.pas
FMX.Colors.pas
FMX.Consts.pas
FMX.Context.DX9.pas
FMX.Context.Mac.pas
FMX.Controls.pas
FMX.DAE.Importer.pas
FMX.DAE.Model.pas
FMX.DAE.Schema.pas

FMX.Dialogs.pas
FMX.Edit.pas
FMX.Effects.pas
FMX.ExtCtrls.pas
FMX.Filter.Effects.pas
FMX.Filter.pas
FMX.FilterCatBlur.pas
FMX.FilterCatColor.pas
FMX.FilterCatColorAdjust.pas
FMX.FilterCatComposite.pas
FMX.FilterCatDistortion.pas
FMX.FilterCatGenerator.pas
FMX.FilterCatGeometry.pas
FMX.FilterCatStyle.pas
FMX.FilterCatTiles.pas
FMX.FilterCatTransition.pas
FMX.Forms.pas
FMX.Grid.pas
FMX.Import.pas
FMX.Layers3D.pas
FMX.Layouts.pas
FMX.ListBox.pas
FMX.Memo.pas
FMX.Menus.pas
FMX.OBJ.Importer.pas
FMX.OBJ.Model.pas
FMX.Objects.pas
FMX.Objects3D.pas
FMX.Platform.Mac.pas
FMX.Platform.pas
FMX.Platform.Win.pas
FMX.Printer.Mac.pas
FMX.Printer.pas
FMX.Printer.Win.pas

FMX.TabControl.pas
FMX.TreeView.pas
FMX.Types.pas
FMX.Types3D.pas
FMX.Video.Mac.pas
FMX.Video.pas
FMX.Video.Win.pas

FireMonkey 自成体系, 有自己的 TApplication、TControl 等和 VCL 对应的大多数构件, 但基本还是根植于早期的 TComponent, 应该属于 VCL 的深度扩展.

在 Windows 下创建 FireMonkey 工程可建立 FireMonkey HD Application(2D) 或 FireMonkey 3D Application,
然后可随意添加 FireMonkey HD Form 或 FireMonkey 3D Form. 其窗体文件的后缀是 fmx 了.

在新 FireMonkey HD Application 下简单测试:

```
unit Unit1;
```

```
interface
```

```
uses
```

```
    System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,  
    FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs;
```

```
type
```

```
    TForm1 = class(TForm)
```

```
        Button1: TButton; //现在的 TButton 来自 FMX.Controls 单元
```

```
        procedure Button1Click(Sender: TObject);
```

```
    private
```

```
{ Private declarations }  
  
public  
    { Public declarations }  
  
end;  
  
var  
    Form1: TForm1;  
  
implementation  
  
{$R *.fmx}  
  
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    ShowMessage('Hello FireMonkey!'); //现在 ShowMessage 方法来自 FMX.Dialogs 单元; 底层实现不同但用法一样, 太好了  
  
end;  
  
end.
```

Delphi XE2 之 FireMonkey 入门(2)

FireMonkey 的控件都是自己绘制的(而不是基于系统组件), 我想它们应该是基于一些基本图形; 就从基本图形开始吧.

FMX.Objects 单元给出的类:

TShape //基本图形的基类

TLine

TRectangle

TRoundRect
TCalloutRectangle
TEllipse
TCircle
TPie
TArc
TPath
TText

TImage
TPaintBox
TSelection
TSelectionPoint

添加一个 TRectangle, 先纵观一下它的可用功能:

```
{ TRectangle }  
public  
    constructor Create(AOwner: TComponent); override;  
published  
    property Fill;  
    property Stroke;  
    property StrokeCap;  
    property StrokeDash;  
    property StrokeJoin;  
    property StrokeThickness;  
    property XRadius: Single read FXRadius write SetXRadius;  
    property YRadius: Single read FYRadius write SetYRadius;  
    property Corners: TCorners read FCorners write SetCorners stored  
IsCornersStored;  
    property CornerType: TCornerType read FCornerType write SetCorner  
rType default TCornerType.ctRound;
```

```
property Sides: TSides read FSides write SetSides stored IsSides
Stored;
```

```
{ TRectangle 的父类 TShape }
```

```
protected
```

```
    procedure FillChanged(Sender: TObject); virtual;
```

```
    procedure StrokeChanged(Sender: TObject); virtual;
```

```
    function GetShapeRect: TRectF;
```

```
    procedure Painting; override;
```

```
    procedure AfterPaint; override;
```

```
public
```

```
    constructor Create(AOwner: TComponent); override;
```

```
    destructor Destroy; override;
```

```
    property Fill: TBrush read FFill write SetFill;
```

```
    property Stroke: TBrush read FStroke write SetStroke;
```

```
    property StrokeThickness: Single read FStrokeThickness write Set
StrokeThickness stored IsStrokeThicknessStored;
```

```
    property StrokeCap: TStrokeCap read FStrokeCap write SetStrokeCa
p default TStrokeCap.scFlat;
```

```
    property StrokeDash: TStrokeDash read FStrokeDash write SetStrok
eDash default TStrokeDash.sdSolid;
```

```
    property StrokeJoin: TStrokeJoin read FStrokeJoin write SetStrok
eJoin default TStrokeJoin.sjMiter;
```

```
    property ShapeRect: TRectF read GetShapeRect;
```

```
{ TShape 的父类 TControl (来自 FMX.Types) }
```

```
public
```

```
    constructor Create(AOwner: TComponent); override;
```

```
    destructor Destroy; override;
```

```
    procedure AddObject(AObject: TFmxObject); override;
```

```
    procedure RemoveObject(AObject: TFmxObject); override;
```

```
    procedure SetNewScene(AScene: IScene); virtual;
```

```
    procedure SetBounds(X, Y, AWidth, AHeight: Single); virtual;
```

```
{ matrix }  
function AbsoluteToLocal(P: TPointF): TPointF; virtual;  
function LocalToAbsolute(P: TPointF): TPointF; virtual;  
function AbsoluteToLocalVector(P: TVector): TVector; virtual;  
function LocalToAbsoluteVector(P: TVector): TVector; virtual;  
function PointInObject(X, Y: Single): Boolean; virtual;  
{ optimizations }  
procedure RecalcUpdateRect; virtual;  
procedure RecalcNeedAlign; virtual;  
procedure RecalcOpacity; virtual;  
procedure RecalcAbsolute; virtual;  
procedure RecalcEnabled; virtual;  
procedure RecalcHasEffect; virtual;  
{ drag and drop }  
function MakeScreenshot: TBitmap;  
{ caret }  
procedure ShowCaretProc;  
procedure SetCaretPos(const APoint: TPointF);  
procedure SetCaretSize(const ASize: TPointF);  
procedure SetCaretColor(const AColor: TAlphaColor);  
procedure HideCaret;  
{ align }  
procedure BeginUpdate; virtual;  
procedure EndUpdate; virtual;  
procedure Realign; virtual;  
{ painting }  
procedure ApplyEffect;  
procedure Painting; virtual;  
procedure DoPaint; virtual;  
procedure AfterPaint; virtual;  
{ effects }  
procedure UpdateEffects;  
{ }  
procedure SetFocus;
```



```
procedure PaintTo(const ACanvas: TCanvas; const ARect: TRectF; c
onst AParent: TFmxObject = nil);
procedure Repaint;
procedure InvalidateRect(ARect: TRectF);
procedure Lock;
property AbsoluteMatrix: TMatrix read GetAbsoluteMatrix;
property AbsoluteOpacity: Single read GetAbsoluteOpacity;
property AbsoluteWidth: Single read GetAbsoluteWidth;
property AbsoluteHeight: Single read GetAbsoluteHeight;
property AbsoluteScale: TPointF read GetAbsoluteScale;
property AbsoluteEnabled: Boolean read GetAbsoluteEnabled;
property HasEffect: Boolean read GetAbsoluteHasEffect;
property HasDisablePaintEffect: Boolean read GetAbsoluteHasDisa
blePaintEffect;
property HasAfterPaintEffect: Boolean read GetAbsoluteHasAfterP
aintEffect;
property ChildrenRect: TRectF read GetChildrenRect;
property InvertAbsoluteMatrix: TMatrix read GetInvertAbsoluteMa
trix;
property InPaintTo: Boolean read FInPaintTo;
property LocalRect: TRectF read GetLocalRect;
property AbsoluteRect: TRectF read GetAbsoluteRect;
property UpdateRect: TRectF read GetUpdateRect;
property BoundsRect: TRectF read GetBoundsRect write SetBoundsRe
ct;
property ParentedRect: TRectF read GetParentedRect;
property ParentedVisible: Boolean read GetParentedVisible;
property ClipRect: TRectF read GetClipRect;
property Canvas: TCanvas read GetCanvas;
property Scene: IScene read FScene;
property AutoCapture: Boolean read FAutoCapture write FAutoCaptu
re default False;
property CanFocus: Boolean read FCanFocus write FCanFocus defaul
t False;
```

```
property DisableFocusEffect: Boolean read FDisableFocusEffect write FDisableFocusEffect default False;

property DisableDefaultAlign: Boolean read FDisableDefaultAlign write FDisableDefaultAlign;

property TabOrder: TTabOrder read GetTabOrder write SetTabOrder default -1;

published

    { triggers }

    property IsMouseOver: Boolean read FIsMouseOver;
    property IsDragOver: Boolean read FIsDragOver;
    property IsFocused: Boolean read FIsFocused;
    property IsVisible: Boolean read FVisible;

    { props }

    property Align: TAlignLayout read FAlign write SetAlign default TAlignLayout.alNone;

    property Cursor: TCursor read GetCursor write SetCursor default crDefault;

    property DragMode: TDragMode read GetDragMode write SetDragMode default TDragMode.dmManual;

    property EnableDragHighlight: Boolean read FEnableDragHighlight write FEnableDragHighlight default True;

    property Enabled: Boolean read FEnabled write SetEnabled default True;

    property Position: TPosition read FPosition write SetPosition;
    property RotationAngle: Single read FRotationAngle write SetRotationAngle;

    property RotationCenter: TPosition read FRotationCenter write FRotationCenter;

    property Locked: Boolean read FLocked write SetLocked default False;

    property Width: Single read FWidth write SetWidth;
    property Height: Single read FHeight write SetHeight;
    property Margins: TBounds read FMargins write FMargins;
    property Padding: TBounds read FPadding write FPadding;
```

```
property Opacity: Single read FOpacity write SetOpacity stored I
sOpacityStored;

property ClipChildren: Boolean read FClipChildren write SetClipC
hildren default False;

property ClipParent: Boolean read FClipParent write FClipParent
default False;

property HitTest: Boolean read FHitTest write SetHitTest default
True;

property CanClip: Boolean read FCanClip write FCanClip default T
rue;

property PopupMenu: TCustomPopupMenu read FPopupMenu write SetPo
pupMenu;

property Scale: TPosition read FScale write FScale;

property Visible: Boolean read FVisible write SetVisible default
True;

property DesignVisible: Boolean read FDesignVisible write SetDes
ignVisible default True;

property OnDragEnter: TDragEnterEvent read FOnDragEnter write FO
nDragEnter;

property OnDragLeave: TNotifyEvent read FOnDragLeave write FOnDr
agLeave;

property OnDragOver: TDragOverEvent read FOnDragOver write FOnDr
agOver;

property OnDragDrop: TDragDropEvent read FOnDragDrop write FOnDr
agDrop;

property OnDragEnd: TNotifyEvent read FOnDragEnd write FOnDragEn
d;

property OnKeyDown: TKeyEvent read FOnKeyDown write FOnKeyDown;
property OnKeyUp: TKeyEvent read FOnKeyUp write FOnKeyUp;
property OnClick: TNotifyEvent read FOnClick write FOnClick;
property OnDbClick: TNotifyEvent read FOnDbClick write FOnDbCl
ick;

property OnCanFocus: TCanFocusEvent read FOnCanFocus write FOnCa
nFocus;
```

```

property OnEnter: TNotifyEvent read FOnEnter write FOnEnter;
property OnExit: TNotifyEvent read FOnExit write FOnExit;
property OnMouseDown: TMouseEvent read FOnMouseDown write FOnMouseDown;
property OnMouseMove: TMouseMoveEvent read FOnMouseMove write FOnMouseMove;
property OnMouseUp: TMouseEvent read FOnMouseUp write FOnMouseUp;
property OnMouseWheel: TMouseWheelEvent read FOnMouseWheel write FOnMouseWheel;
property OnMouseEnter: TNotifyEvent read FOnMouseEnter write FOnMouseEnter;
property OnMouseLeave: TNotifyEvent read FOnMouseLeave write FOnMouseLeave;
property OnPainting: TOnPaintEvent read FOnPainting write FOnPainting;
property OnPaint: TOnPaintEvent read FOnPaint write FOnPaint;
property OnResize: TNotifyEvent read FOnResize write FOnResize;
property OnApplyStyleLookup: TNotifyEvent read FOnApplyStyleLookup write FOnApplyStyleLookup;

{ TControl 的父类 TFmxObject (来自 FMX.Types) }
public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure Release(Delay: Single = 0.1);
    { check for support interface }
    function IsIControl: Boolean;
    function AsIControl: IControl;
    procedure SetRoot(ARoot: IRoot);
    { design }
    procedure SetDesign(Value: Boolean; SetChildren: Boolean = True);
    function ItemClass: string; virtual;

```

```

{ clone }
function Clone(const AOwner: TComponent): TFmxObject;
procedure CloneChildFromStream(AStream: TStream);
{ childs }
procedure AddObject(AObject: TFmxObject); virtual;
procedure InsertObject(Index: Integer; AObject: TFmxObject); virtual;

procedure RemoveObject(AObject: TFmxObject); overload; virtual;
procedure RemoveObject(Index: Integer); overload; virtual;
procedure Exchange(AObject1, AObject2: TFmxObject); virtual;
procedure DeleteChildren; virtual;
procedure BringToFront;
procedure SendToBack;
procedure AddObjectsToList(const AList: TList);
procedure AddControlsToList(const AList: TList);
procedure Sort(Compare: TFmxObjectSortCompare); virtual;
{ notify }
procedure AddFreeNotify(const AObject: IFreeNotification);
procedure RemoveFreeNotify(const AObject: IFreeNotification);
{ tab }
procedure GetTabOrderList(const List: TList; AChildren: Boolean);
{ i/o }
procedure LoadFromStream(const AStream: TStream);
procedure SaveToStream(const Stream: TStream);
procedure LoadFromBinStream(const AStream: TStream);
procedure SaveToBinStream(const AStream: TStream);
{ resource }
function FindStyleResource(const AStyleLookup: string): TFmxObject; virtual;
procedure UpdateStyle; virtual;
{ animations }
procedure StartAnimation(const AName: string); virtual;
procedure StopAnimation(const AName: string); virtual;

```

```
procedure StartTriggerAnimation(AInstance: TFmxObject; const AT  
rigger: string); virtual;  
    procedure StartTriggerAnimationWait(AInstance: TFmxObject; cons  
t ATrigger: string); virtual;  
    procedure StopTriggerAnimation(AInstance: TFmxObject); virtual;  
    procedure ApplyTriggerEffect(AInstance: TFmxObject; const ATrig  
ger: string); virtual;  
    { animation property }  
    procedure AnimateFloat(const APropertyName: string; const NewVal  
ue: Single; Duration: ...);  
    procedure AnimateColor(const APropertyName: string; NewValue: TA  
lphaColor; Duration: ...);  
    procedure AnimateFloatDelay(const APropertyName: string; const N  
ewValue: Single; Duration: ...);  
    procedure AnimateFloatWait(const APropertyName: string; const Ne  
wValue: Single; Duration: ...);  
    procedure StopPropertyAnimation(const APropertyName: string);  
    { }  
    property Root: IRoot read FRoot;  
    property Stored: Boolean read FStored write SetStored;  
    { }  
    property TagObject: TObject read FTagObject write FTagObject;  
    property TagFloat: Single read FTagFloat write FTagFloat;  
    property TagString: string read FTagString write FTagString;  
    { children }  
    property ChildrenCount: Integer read GetChildrenCount;  
    property Children[Index: Integer]: TFmxObject read GetChild;  
    { binding }  
    function FindBinding(const ABinding: string): TFmxObject;  
    property Data: Variant read GetData write SetData;  
    property Binding[const Index: string]: Variant read GetBinding w  
rite SetBinding;  
    property Parent: TFmxObject read FParent write SetParent;  
    property Index: Integer read GetIndex write SetIndex;
```

```

published
    property BindingName: string read FBindingName write SetBindingName;
    property StyleName: string read FStyleName write SetStyleName;

{ TEmxObject 的父类 TComponent (来自 System.Classes), 到这里不新鲜了 }

public
    constructor Create(AOwner: TComponent); virtual;
    destructor Destroy; override;
    procedure BeforeDestruction; override;
    procedure DestroyComponents;
    procedure Destroying;
    function ExecuteAction(Action: TBasicAction): Boolean; dynamic;
    function FindComponent(const AName: string): TComponent;
    procedure FreeNotification(AComponent: TComponent);
    procedure RemoveFreeNotification(AComponent: TComponent);
    procedure FreeOnRelease;
    function GetEnumerator: TComponentEnumerator;
    function GetParentComponent: TComponent; dynamic;
    function GetNamePath: string; override;
    function HasParent: Boolean; dynamic;
    procedure InsertComponent(AComponent: TComponent);
    procedure RemoveComponent(AComponent: TComponent);
    procedure SetSubComponent(IsSubComponent: Boolean);
    function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HRESULT; override;
    function UpdateAction(Action: TBasicAction): Boolean; virtual;
    function IsImplementorOf(const I: IInterface): Boolean;
    function ReferenceInterface(const I: IInterface; Operation: TOperation): Boolean;

    property ComObject: IUnknown read GetComObject;
    property Components[Index: Integer]: TComponent read GetComponent;

    property ComponentCount: Integer read GetComponentCount;

```

```

property ComponentIndex: Integer read GetComponentIndex write SetComponentIndex;

property ComponentState: TComponentState read FComponentState;
property ComponentStyle: TComponentStyle read FComponentStyle;
property DesignInfo: Longint read FDesignInfo write FDesignInfo;
property Owner: TComponent read FOwner;
property VCLComObject: Pointer read FVCLComObject write FVCLComObject;

property Observers: TObservers read GetObservers;
published
property Name: TComponentName read FName write SetName stored False;
property Tag: NativeInt read FTag write FTag default 0;

```

{ TComponent 的父类 TPersistent (来自 System.Classes) }

public

```

destructor Destroy; override;
procedure Assign(Source: TPersistent); virtual;
function GetNamePath: string; dynamic;

```

{ TPersistent 的父类 TObject (来自 System) }

public

```

constructor Create;
procedure Free;
class function InitInstance(Instance: Pointer): TObject;
procedure CleanupInstance;
function ClassType: TClass; inline;
class function ClassName: string;
class function ClassNameIs(const Name: string): Boolean;
class function ClassParent: TClass;
class function ClassInfo: Pointer; inline;
class function InstanceSize: Longint; inline;
class function InheritsFrom(AClass: TClass): Boolean;

```

```
class function MethodAddress(const Name: ShortString): Pointer;  
overload;  
class function MethodAddress(const Name: string): Pointer; overl  
oad;  
class function MethodName(Address: Pointer): string;  
class function QualifiedClassName: string;  
function FieldAddress(const Name: ShortString): Pointer; overloa  
d;  
function FieldAddress(const Name: string): Pointer; overload;  
function GetInterface(const IID: TGUID; out Obj): Boolean;  
class function GetInterfaceEntry(const IID: TGUID): PInterfaceEn  
try;  
class function GetInterfaceTable: PInterfaceTable;  
class function UnitName: string;  
class function UnitScope: string;  
function Equals(Obj: TObject): Boolean; virtual;  
function GetHashCode: Integer; virtual;  
function ToString: string; virtual;  
function SafeCallException(ExceptObject: TObject; ExceptAddr: P  
ointer): HResult; virtual;  
procedure AfterConstruction; virtual;  
procedure BeforeDestruction; virtual;  
procedure Dispatch(var Message); virtual;  
procedure DefaultHandler(var Message); virtual;  
class function NewInstance: TObject; virtual;  
procedure FreeInstance; virtual;  
destructor Destroy; virtual;
```

Delphi XE2 之 FireMonkey 入门(3) - 关于 TPosition

把 FireMonkey 简称为 FM 吧。FM 的窗体继续使用 Left、Top 属性, 但更多控件不是了。

//FM 控件的位置控制不再是 Left、Top, 取而代之的是 Position 属性

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    Rectangle1.Position.X := Rectangle1.Position.X + 10;  
    Rectangle1.Position.Y := Rectangle1.Position.Y + 10;  
end;
```

//TPosition 是类

```
procedure TForm1.Button2Click(Sender: TObject);  
var  
    posObj: TPosition;  
begin  
    posObj := TPosition.Create(TPointF.Create(10, 10));  
    Rectangle1.Position.Assign(posObj);  
    posObj.Free;  
end;
```

//TPosition.Point 是 TPointF 类型的结构

```
procedure TForm1.Button3Click(Sender: TObject);  
begin  
    Rectangle1.Position.Point := TPointF.Create(50, 50);  
end;
```

//TPointF 结构也拥有许多方便的方法和运算符重载

```
procedure TForm1.Button4Click(Sender: TObject);  
var  
    ptf: TPointF;  
begin  
    ptf.X := ClientWidth / 2;  
    ptf.Y := ClientHeight / 2;  
    ptf.Offset(-Rectangle1.Width / 2, -Rectangle1.Height / 2);
```

```
Rectangle1.Position.Point := ptf;
end;

//TVector 是包含三个元素的结构体, TPosition 可以直接使用其前两个数据

procedure TForm1.Button5Click(Sender: TObject);
var
    vector: TVector;
begin
    vector.X := ClientWidth - Rectangle1.Width;
    vector.Y := ClientHeight - Rectangle1.Height;
    // vector.W := 0.0;
    Rectangle1.Position.Vector := vector;
end;
```

Delphi XE2 之 FireMonkey 入门(4) - 控件天生可做容器

- 1、新建 FM(HD) 工程, 先添加 TLine(默认名称是 Line1);
- 2、在 Line1 选择状态下添加 Button1;
- 3、取消选择后添加 Button2

此时, Button1.Parent 是 Line1; Button2.Parent 是窗体.

```
unit Unit1;
```

```
interface
```

```
uses
```

```
    System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,
    FMX.Types, FMX.Controls, FMX.Forms, FMX.Objects;
```

type

```
TForm1 = class (TForm)
    Line1: TLine;
    Button1: TButton;
    Button2: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
end;
```

var

```
Form1: TForm1;
```

implementation

```
{ $R *.fmx }
```

```
procedure TForm1.FormCreate(Sender: TObject);
```

begin

```
    Button2.OnClick := Button1.OnClick; { TLine }
```

end;

```
procedure TForm1.Button1Click(Sender: TObject);
```

begin

```
    Caption := TButton(Sender).Parent.ClassName; { TForm1; 现在的窗体没  
有 Text 属性了 }
```

```
    Line1.Position.X := Line1.Position.X + 10; { Btuuon1 会随同移动 }
```

end;

end.

Delphi XE2 之 FireMonkey 入门(5) - TAlphaColor

不是 TColor, 是 TAlphaColor 了.

TAlphaColor = type Cardinal; 还是一个整数.

四个字节分别是: AA RR GG BB(透明度、红、绿、蓝); 这和 TColor 的颜色序相反, 并增加了透明度.

在 HD 窗体上添加一个 TRectangle 和三个 TButton, 测试:

//下面四种赋值方法相同

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    Rectangle1.Fill.Color := $FFFF0000;  
    Rectangle1.Fill.Color := claRed;  
    Rectangle1.Fill.Color := TAlphaColors.Red;  
    Rectangle1.Fill.Color := TAlphaColorRec.Red;  
end;
```

//通过函数构建 TAlphaColor

```
procedure TForm1.Button2Click(Sender: TObject);  
begin  
    Rectangle1.Fill.Color := MakeColor(0, 0, 255);  
  
    //另有 AppendColor()、SubtractColor() 等颜色加减或转换函数  
end;
```

//通过 TAlphaColorRec 结构调整颜色获取或设置颜色分量

```
procedure TForm1.Button3Click(Sender: TObject);
```

```
var
    C: TAlphaColor;
    A,R,G,B: Byte;
begin
    C := MakeColor($82, $00, $4B, $FF);
    Rectangle1.Fill.Color := C;

    A := TAlphaColorRec(C).A;
    R := TAlphaColorRec(C).R;
    G := TAlphaColorRec(C).G;
    B := TAlphaColorRec(C).B;
    ShowMessageFmt('%x %x %x %x', [A, R, G, B]);

    TAlphaColorRec(C).A := 127; //调下透明度
    Rectangle1.Fill.Color := C;
    A := TAlphaColorRec(C).A;
    ShowMessageFmt('%x %x %x %x', [A, R, G, B]);
end;
```

调整颜色的控件也很方便, 现在也能方便地调整: 色调、饱和度、亮度.

添加 TRectangle、TColorComboBox、TColorPanel、TColorPicker、TComboColorBox、TColorQuad 各一个, 测试:

```
//在 TColorComboBox 的 OnChange 中修改颜色
procedure TForm1.ColorComboBox1Change(Sender: TObject);
begin
    Rectangle1.Fill.Color := ColorComboBox1.Color;
end;

//在 TColorPanel 的 OnChange 中修改颜色
```

```
procedure TForm1.ColorPanel1Change(Sender: TObject);  
begin  
    Rectangle1.Fill.Color := ColorPanel1.Color;  
end;  
  
//在 TColorPicker 的 OnChange 中修改颜色  
procedure TForm1.ColorPicker1Click(Sender: TObject);  
begin  
    Rectangle1.Fill.Color := ColorPicker1.Color;  
end;  
  
//在 TComboColorBox 的 OnChange 中修改颜色  
procedure TForm1.ComboColorBox1Change(Sender: TObject);  
begin  
    Rectangle1.Fill.Color := ComboColorBox1.Color;  
end;  
  
//通过 TColorQuad 调整 HSL(色调、饱和度、亮度)  
procedure TForm1.ColorQuad1Change(Sender: TObject);  
var  
    C: TAlphaColor;  
begin  
    C := Rectangle1.Fill.Color;  
    Rectangle1.Fill.Color := ChangeHSL(C, ColorQuad1.Hue, ColorQuad1.  
Sat, ColorQuad1.Lum);  
    Caption := Format('%f, %f, %f', [ColorQuad1.Hue, ColorQuad1.Sat, C  
olorQuad1.Lum]);  
end;
```

在 System.UIConsts 单元也有 StringToAlphaColor()、AlphaColorToIdent()、IdentToAlphaColor() 等相关函数。

Delphi XE2 之 FireMonkey 入门(6) - TLine、TEllipse、TCircle、TPie、TArc、TRectangle、TRoundRect、TCalloutRectangle

它们都是继承自 TShape 类, 共同拥有如下属性:

Fill	: TBrush;	//填充
Stroke	: TBrush;	//边线(画笔)
StrokeThickness	: Single;	//厚度(边线宽度)
StrokeCap	: TStrokeCap;	//线帽样式, TStrokeCap (枚举) 类型
StrokeDash	: TStrokeDash;	//虚线样式, TStrokeDash (枚举) 类型
StrokeJoin	: TStrokeJoin;	//拐点结合样式, TStrokeJoin (枚举) 类型
ShapeRect	: TRectF;	//可填充范围的矩形(相对于当前图形)

TLine 用不着 Fill, 但增加了 LineType 属性(TLineType 枚举类型);

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Line1.LineType := TLineType.ltDiagonal; //斜线
    Line2.LineType := TLineType.ltTop;      //横线
    Line3.LineType := TLineType.ltLeft;     //竖线
end;
```

TEllipse 和 TCircle 没有新属性, 应该也用不着 StrokeCap、StrokeJoin.

```
procedure TForm1.Button1Click(Sender: TObject);
```

begin

```
Ellipse1.StrokeDash := TStrokeDash.sdDot; //虚线样式
```

```
Circle1.Fill.Kind := TBrushKind.bkNone; //取消填充
```

end;

TArc 和 TPie 增加了 StartAngle、EndAngle 属性。

TRectangle 增加了控制圆角的 XRadius、YRadius 属性、控制边线的 Sides 属性、控制四个角的 Corners、CornerType 属性;

TRoundRect 只加了 Corners 属性; 看来要做更随意的圆角矩形得用 TRectangle 而不是 TRoundRect.

procedure TForm1.Button1Click(Sender: TObject);

begin

```
Rectangle1.Position.X := 50;
```

```
Rectangle1.Position.Y := 10;
```

```
Rectangle1.Width := 100;
```

```
Rectangle1.Height := 120;
```

```
Rectangle1.StrokeThickness := 16;
```

```
Rectangle1.Fill.Color := $80FF0000;
```

```
Rectangle1.Stroke.Color := $800000FF;
```

```
Rectangle1.XRadius := 8;
```

```
Rectangle1.YRadius := 8;
```

end;

procedure TForm1.Button2Click(Sender: TObject);

begin

```
Rectangle2.Position.X := 350;
Rectangle2.Position.Y := 10;
Rectangle2.Width := 100;
Rectangle2.Height := 120;
Rectangle2.StrokeThickness := 4;
Rectangle2.Fill.Color := claRed;
Rectangle2.Stroke.Color := claBlack;

Rectangle2.CornerType := TCornerType.ctBevel;
Rectangle2.Corners := [TCorner.crBottomLeft, TCorner.crBottomRight];
Rectangle2.Sides := [TSide.sdBottom, TSide.sdRight];

//关于 Corners 和 Sides 还有两个非常方便的常量: AllCorners、AllSides
end;
```

TCalloutRectangle 很有意思, 矩形外带一个三角, 应该是用于图形化的注释.

给它增加的 CalloutWidth、CalloutLength、CalloutPosition、CalloutOffset 四个属性都是用于控制三角的.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    CalloutRectangle1.Width := 200;
    CalloutRectangle1.Height := 150;
    CalloutRectangle1.CalloutPosition := TCalloutPosition.cpBottom;
    CalloutRectangle1.CalloutWidth := CalloutRectangle1.Width / 4;
    CalloutRectangle1.CalloutLength := CalloutRectangle1.Height / 2;
    CalloutRectangle1.CalloutOffset := -CalloutRectangle1.Width / 2;
end;
```

这其中需要进一步学习的是 Fill、Stroke 属性, 它们都是 TBrush 类型, 会涉及到 TBitmap、TBitmapObject、TCanvas 等等.

Delphi XE2 之 FireMonkey 入门(7) - TText 与 TFont

TText 也是从 TShape(TControl -> TShape)继承;

而与之类似的 TLabel 的继承序列是 TControl -> TStyledControl -> TTextControl -> TLabel.

TText 的主要成员:

{ 属性 }

```
Text          : string;      // 文本内容

Font          : TFont;       // 字体

Fill          : TBrush;      // 文本画刷

HorzTextAlign : TTextAlign;  // 横向对齐

VertTextAlign : TTextAlign;  // 纵向对齐

AutoSize      : Boolean;     // 改变控件大小以适合文本

Stretch       : Boolean;     // 拉伸文本以适合控件

WordWrap      : Boolean;     // 是否换行
```

{ 方法 }

```
Realign; // 重新对齐
```

TFont(来自 FMX.Types) 的主要成员:

{ 属性 }

Family : TFontName; //名称

Size : Single; //大小

Style : TFontStyles; //样式

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
```

```
    Text1.Align := TAlignLayout.alClient;
```

```
    Text1.Text := ' Delphi XE2 ';
```

```
    Text1.Font.Family := '微软雅黑';
```

```
    Text1.Font.Size := 32;
```

```
    Text1.Font.Style := [TFontStyle.fsBold, TFontStyle.fsUnderline];
```

```
    Text1.Fill.Color := claRed;
```

```
    Text1.Stretch := True;
```

```
end;
```

Delphi XE2 之 FireMonkey 入门(8) - TImage

TImage 主要成员:

{ 属性 }

```
Bitmap          : TBitmap;           //图像
BitmapMargins   : TBounds;           //边缘空白
WrapMode        : TImageWrapMode;    //枚举; iwOriginal、iwFit、iwStretch、iwTile(原始、适合、拉伸、平铺)
DisableInterpolation : Boolean;       //是否使用(像素)插入算法
```

另有 TImageViewer、TImageControl 和 TImage 类似.

TImageViewer 继承自 TScrollBox, 可自动加滚动条;

TImageControl 继承自 TStyledControl, 可设置样式、且可在选择时呈现焦点.

添加 TImage、TImageViewer、TImageControl、TOpenDialog 和若干 TButton 后测试:

```
//Bitmap 属性测试

procedure TForm1.Button1Click(Sender: TObject);
begin
    if OpenFileDialog1.Execute then
    begin
        Image1.Bitmap.LoadFromFile(OpenDialog1.FileName);
        ImageControl1.Bitmap.LoadFromFile(OpenDialog1.FileName);
        ImageViewer1.Bitmap.LoadFromFile(OpenDialog1.FileName);
    end;
end;
```

```
//WrapMode 属性测试

procedure TForm1.Button2Click(Sender: TObject);
begin
```

```
Image1.WrapMode := TImageWrapMode(Tag); //iwOriginal, iwFit, iwStretch, iwTile
Tag := Tag + 1;
if Tag = 4 then Tag := 0;
end;

//BitmapMargins 属性测试

procedure TForm1.Button3Click(Sender: TObject);
begin
  with Image1.BitmapMargins do
  begin
    Left := Left + 10;
  end;

  Image1.Bitmap.BitmapChanged; //刷新
end;

//DisableInterpolation 属性测试

procedure TForm1.Button4Click(Sender: TObject);
begin
  Image1.DisableInterpolation := not Image1.DisableInterpolation;
  Image1.Bitmap.BitmapChanged;
end;
```

Delphi XE2 之 FireMonkey 入门(9) - TBitmap

TBitmap 主要成员:

{ 方法 }

SetSize();	//设置大小
Clear();	//取消，就是用指定颜色覆盖
ClearRect();	//覆盖指定矩形范围，默认覆盖为透明色
BitmapChanged();	//刷新改变
IsEmpty();	//是否为空
UpdateHandles();	//将 Handles 数组中的对象标记为需要更新
AddFreeNotify();	//将指定对象添加到可以释放的列表；这一般是针对添加到 Handles 中的对象
RemoveFreeNotify();	//释放指定对象；这一般是针对添加到 Handles 中的对象
Rotate();	//旋转角度
FlipHorizontal();	//水平翻转
FlipVertical();	//垂直翻转
InvertAlpha();	//翻转透明度，只适于透明图片
FillColor();	//填充遮罩色，用于透明图片
CreateMask();	//建立蒙版；是从 TBitmap 中把各像素的透明度提取为一个数组，返回数组指针
ApplyMask();	//添加蒙版，其参数应该是有 CreateMask() 建立的
CreateThumbnail();	//建立略缩图
LoadFromFile();	//从文件载入
LoadFromStream();	//从流中载入
LoadThumbnailFromFile();	//从文件载入为略缩图
SaveToFile();	//保存到文件

```
SaveToStream();           // 保存到流

HandleRemove();           // 从 Handles 数组移除对象，并没有释放

HandleAdd();              // 添加对象到 Handles 数组

HandleExists();           // 判断指定对象是否已添加到 Handles 数组

{ 属性 }

Width                    : Integer;           // 宽

Height                   : Integer;           // 高

Handles[AItem: Pointer]  : Pointer;           // 访问 Handles
中的对象，索引是指针；写入前需要先 HandleAdd();

HandlesNeedUpdate[AItem: Pointer] : Boolean;           // 判断 Handles
中的指定对象是否需要更新

Canvas                   : TCanvas;           // 获取绘图表面

Pixels[X, Y: Integer]    : TAlphaColor;       // 获取或设置指定
位置的颜色值

ScanLine[Y: Integer]     : PAlphaColorArray; // 获取一条横线上
的像素数组的指针

StartLine                : PAlphaColorArray; // 获取像素数组的指
针

ResourceBitmap           : TBitmap;           // 该属性应该是内部
使用的

StyleLookup              : string;           // 这应该是和控件的
造型相关的，暂不知 TBitmap 要它干嘛

{ 事件 }
```



```
OnChange: TNotifyEvent; //
```

Create()、SetSize()、Clear()、IsEmpty():

```
procedure TForm1.Button1Click(Sender: TObject);  
var  
    bit: TBitmap;  
begin  
    bit := TBitmap.Create(0, 0);  
    ShowMessage(BoolToStr(bit.IsEmpty, True));  
    bit.SetSize(100, 100);  
    ShowMessage(BoolToStr(bit.IsEmpty, True));  
    bit.Clear(claRed);  
    Image1.Bitmap.Assign(bit);  
    bit.Free;  
end;
```

Rotate()、FlipHorizontal()、FlipVertical():

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    if OpenDialog1.Execute then  
        begin  
            Image1.Bitmap.LoadFromFile(OpenDialog1.FileName);  
            Image1.WrapMode := TImageWrapMode.iwFit;  
        end;  
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);  
begin
```

```
ShowMessage('Rotate');
Image1.Bitmap.Rotate(60);

ShowMessage('FlipHorizontal');
Image1.Bitmap.FlipHorizontal;

ShowMessage('FlipVertical');
Image1.Bitmap.FlipVertical;
end;
```

FillColor():

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    if OpenDialog1.Execute then
    begin
        Image1.Bitmap.LoadFromFile(OpenDialog1.FileName); //要个透明图片
        Image1.WrapMode := TImageWrapMode.iwFit;
    end;
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Image1.Bitmap.FillColor($800000FF);
    // Image1.Bitmap.FillColor($FF0000FF);
end;
```

CreateMask()、ApplyMask():

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  if OpenDialog1.Execute then
  begin
    Image1.Bitmap.LoadFromFile(OpenDialog1.FileName); //要个透明图片
    Image1.WrapMode := TImageWrapMode.iwFit;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  bit: TBitmap;
  bitAs: PByteArray;
begin
  bit := TBitmap.Create(0, 0);

  bit.Assign(Image1.Bitmap); //从 Image1 复制 TBitmap

  bitAs := bit.CreateMask;
  Image1.Bitmap.Clear(cbaBlue);
  Image1.Bitmap.ApplyMask(bitAs);

  bit.Free;
end;
```

StartLine: 尽管该属性是只读的, 但因为是指针, 所以可以直接改写像素.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  if OpenDialog1.Execute then
  begin
    Image1.Bitmap.LoadFromFile(OpenDialog1.FileName); //要个透明图片
    Image1.WrapMode := TImageWrapMode.iwFit;
```

```
end;
end;

// 点击按钮调整图像的透明度

procedure TForm1.Button1Click(Sender: TObject);
var
    bit: TBitmap;
    bts: PAlphaColorArray;
    i: Integer;
    j: Integer;
begin
    bit := TBitmap.Create(0, 0);
    bit.Assign(Image1.Bitmap);
    bts := bit.StartLine;

    for i := 0 to bit.Width - 1 do
        for j := 0 to bit.Height - 1 do
            TAlphaColorRec(bts[i*j]).A := TAlphaColorRec(bts[i*j]).A div 2;
        end;
    end;

    // 现在通过数组指针可以直接索引元素了, 太好

    Image1.Bitmap.Assign(bit);
    bit.Free;
end;
```

Pixels[X, Y: Integer]:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    if OpenDialog1.Execute then
    begin
        Image1.Bitmap.LoadFromFile(OpenDialog1.FileName); // 要个透明图片
    end;
end;
```

```
Image1.WrapMode := TImageWrapMode.iwFit;
Image1.Position.X := 0;
Image1.Position.Y := 0;
Image1.Width := Image1.Bitmap.Width;
Image1.Height := Image1.Bitmap.Height;
end;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
    bit: TBitmap;
    W,H,i,j: Integer;
    R: TRectF;
begin
    W := Trunc(Image1.Width);
    H := Trunc(Image1.Height);
    bit := TBitmap.Create(W div 2, H div 2);

    for i := 0 to W - 1 do
        for j := 0 to H - 1 do
            begin
                if Odd(i) and Odd(j) then //复制单数行以缩小图像一倍
                    bit.Pixels[i div 2, j div 2] := Image1.Bitmap.Pixels[i, j];
                end;
            end;
        end;
    Image1.Width := bit.Width;
    Image1.Height := bit.Height;
    Image1.Bitmap.Assign(bit);
    Realign;
    bit.Free;
end;
```

Handles 相关: TBitmap 内部管理着一个对象数组, 访问索引也是一个指针.

var

bit1,bit2,bit3: TBitmap;

procedure TForm1.FormCreate(Sender: TObject);

begin

bit1 := TBitmap.Create(100, 100);

bit2 := TBitmap.Create(100, 100);

bit3 := TBitmap.Create(100, 100);

bit1.Clear(clearRed);

bit2.Clear(clearGreen);

bit3.Clear(clearBlue);

Image1.Width := 100;

Image1.Height := 100;

Image1.Bitmap.HandleAdd(Button1);

Image1.Bitmap.Handles[Button1] := bit1;

Image1.Bitmap.HandleAdd(Button2);

Image1.Bitmap.Handles[Button2] := bit2;

Image1.Bitmap.HandleAdd(Button3);

Image1.Bitmap.Handles[Button3] := bit3;

Button2.OnClick := Button1.OnClick;

Button3.OnClick := Button1.OnClick;

end;

procedure TForm1.Button1Click(Sender: TObject);

begin

Image1.Bitmap := Image1.Bitmap.Handles[Sender];

end;

通过 FMX.Types 单元中的 GetMeasureBitmap() 函数可以快速获取 1*1 的 TBitmap 对象。

Delphi XE2 之 FireMonkey 入门(10) - 常用结构 TPoint、TPointF、TSmallPoint、TSize、TRect、TRectF 及相关方法

它们都是结构, TPointF、TRectF 属新增, 其它也都有升级; 现在都拥有丰富的方法和方便的运算符重载; 且有一组相关的公共函数。

这组内容重要的是它们都来自 System.Types 单元, 也就是不仅仅在 FM 中可用。

TPoint:

```
Create();           //

{ 运算符重载 }

Equal;              // =
NotEqual;           // <>
Add;                // +
Subtract;           // -

Implicit;           // 可从 TSmallPoint 隐身转换到 TPoint
Explicit;           // 可显示转换到 TSmallPoint

{ 方法 }

Distance();         // 计算两点之间的距离

SetLocation();      // 重定位

Offset();           // 偏移

Add();              // 加
```

```
Subtract();    // 减

IsZero();      // 是否在 [0,0] 点

{数据成员}

X,Y: Longint;

{ 测试 }

procedure TForm1.Button1Click(Sender: TObject);
var
    p1,p2,p3: TPoint;
begin
    p1 := TPoint.Create(11, 22);
    p2.Create(11, 11);
    p3 := p1 + p2;
    ShowMessageFmt('%d,%d', [p3.X, p3.Y]); //22,33

    p3.SetLocation(0, 0);
    p3 := p1.Add(p2);
    ShowMessageFmt('%d,%d', [p3.X, p3.Y]); //22,33
end;
```

TPointF: 比 TPoint 多出三个方法:

```
Ceiling();    //
Truncate();   //
Round();      //

procedure TForm1.Button1Click(Sender: TObject);
var
    pf: TPointF;
    p1,p2,p3: TPoint;
begin
```



```
pf.Create(1.4, 1.6);
p1 := pf.Ceiling;
p2 := pf.Truncate;
p3 := pf.Round;
ShowMessageFmt('%d,%d %d,%d %d,%d', [p1.X, p1.Y, p2.X, p2.Y, p3.X, p3.Y]); //2,2 1,1 1,2
end;
```

TSmallPoint: 用得少, 成员也少.

```
Create();    //
{运算符重载}

Equal;       // =
NotEqual;    // <>
Add;         // +
Subtract;    // -

{方法}

Add();       // 加

Subtract();  // 减

Distance();  // 计算两点间距

IsZero();    // 是否是 [0,0] 点

{数据成员}

X,Y: SmallInt;
```

TSize:

```
Create();    //

{运算符重载}
```

```
Equal;           // =
NotEqual;        // <>
Add;             // +
Subtract;        // -

{ 方法 }

Add();           // 加

Subtract();      // 减

Distance();      // 计算两点间距

IsZero();        // 是否是 [0,0] 点

{ 属性 }

Width;           //
Height;          //

{ 数据成员 }

cx,cy: Single;
```

TRect:

```
Create();        //建立时和可同时规格化矩形; 参见 NormalizeRect() 方法

{ 运算符重载 }

Equal;           // =
NotEqual;        // <>
Add;             // + (并集)
Multiply;        // * (交集)

{ 类方法 }

Empty;           //获取一个空的 TRect 对象; 内联

Intersect;       //获取两个矩形的交集矩形
```

```
Union;           //获取两个矩形的并集矩形

{方法}

NormalizeRect(); //规格化矩形; 当 Top > Bottom 或 Left > Right 时, 会置
换数据使之合理.

IsEmpty();       //是否为空

Contains();       //是否包含指定点或指定矩形

IntersectsWith(); //判断和指定矩形是否交叉

Intersect();      //和指定矩形进行交集运算

Union();          //和指定矩形进行并集运算

Offset();         //偏移

SetLocation();    //设置新原点

Inflate();        //放大矩形(保持中心点)

CenterPoint();    //获取中心点

SplitRect();      //切除, TSplitRectType 枚举(Left、Top、Right、Bottom)
指示要留住的一边

{属性}

Width: Integer;   //有这些属性很方便
Height: Integer;  //
Size: TSize;      //
Location: TPoint; //

{数据结构, 这同前}

(Left, Top, Right, Bottom: Longint) 或 (TopLeft, BottomRight: TPoint)
```

TRectF: 只比 TRect 多出了转换到 TRect 时关于小数取舍的三个方法。

```
Ceiling(); //
Truncate(); //
Round(); //
```

相关的公共函数与过程:

```
{ function }

EqualRect();           //判断 TRect 或 TRectF 是否相等

Rect();               //构建 TRect

RectF();              //构建 TRectF

NormalizeRectF();      //从 TPointF 数组规格化出一个 TRectF

NormalizeRect();       //规格化 TRectF

RectWidth();          //获取矩形宽度

RectHeight();         //获取矩形高度

RectCenter();         //让矩形在另一指定矩形中居中

Bounds();             //根据原点、宽、高构建矩形

Point();              //构建 TPoint

PointF();             //构建 TPointF

MinPoint();           //比对两个点返回小的 (优先判断了 Y)

ScalePoint();         //按指定比例移动点

SmallPoint();         //构建 TSmallPoint

PtInRect();           //判断点是否在指定矩形中
```

```
PtInCircle(); //判断点是否在指定了中心与半径的圆中

IntersectRect(); //判断两矩形是否相交

UnionRect(); //结合两矩形

IsRectEmpty(); //判断矩形是否为空

OffsetRect(); //偏移矩形

CenterPoint(); //获取矩形中心点

SplitRect(); //矩形切除

CenteredRect(); //算出两个矩形的中间过渡矩形；应该是用于动画的

IntersectRectF(); //输出两矩形的交集

UnionRectF(); //输出两矩形的并集

{ procedure }

MultiplyRect(); //按比放缩矩形

InflateRect(); //按量放缩矩形
```

Delphi XE2 之 FireMonkey 入门(11) - 控件居中、旋转、透明

RotationAngle、RotationCenter、Opacity 属性继承自 TControl(FMX.Types), 这些新属性成了控件的基本功能.

先在 HD 窗体上添加 TRectangle 和两个按钮...

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Rectangle1.Width := 100;
```

```
Rectangle1.Height := 100;
Rectangle1.Fill.Color := claYellow;
Rectangle1.Stroke.Color := claRed;

{居中; 喜欢这种带枚举名的赋值方式}

Rectangle1.Align := TAlignLayout.alCenter;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    {旋转中心点: [0.5, 0.5] 是默认值, 表示控件的中心点; [0, 0] 和 [1, 1] 分别表示控件的左上角和右下角}

    Rectangle1.RotationCenter.X := 0.5;
    Rectangle1.RotationCenter.Y := 0.5; //

    {旋转角度}

    Rectangle1.RotationAngle := 45;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    {透明度: 0.0 - 1.0}

    Rectangle1.Opacity := 0.5;
end;
```

Delphi XE2 之 FireMonkey 入门(12) - 动画(上)

在 HD 窗体上添加一个 TAniIndicator, 修改其 Enabled 属性为 True, 动画完成了.

这是最简单的动画相关的控件了, 只有两个值得注意的属性:

```
Enabled: Boolean;           //
Style: TAniIndicatorStyle; //TAniIndicatorStyle = (aiLinear, aiCircular);
```

{ 例 }

```
AniIndicator1.Style := TAniIndicatorStyle.aiCircular;
```

它是怎么动起来的? 追源码, 发现它有一个 `FAni: TFloatAnimation`; 内部变量.

再就追出 `TFloatAnimation` 的父类 `TAnimation`; `TAnimation` 在 `FMX.Types` 单元, 看来是核心成员了.

`TAnimation` 的子类们都在 `FMX.Ani` 单元:

```
TFloatAnimation           //
TFloatKeyAnimation        //
TColorAnimation           //
TColorKeyAnimation        //
TGradientAnimation        //
TPathAnimation            //
TRectAnimation            //
TBitmapAnimation          //
TBitmapListAnimation      //
TFloatKeyAnimation        //
TColorKeyAnimation        //
```

早在 `TFmxObject`(`FMX` 们的祖先)就有了一些动画相关的方法:

```
StartAnimation();          //
StopAnimation();           //
StartTriggerAnimation();    //
StartTriggerAnimationWait(); //
StopTriggerAnimation();     //
```

```
AnimateFloat();           //
AnimateColor();           //
AnimateFloatDelay();      //
AnimateFloatWait();       //
StopPropertyAnimation();  //
```

另在 **FMX.Types** 单元还有一些动画插入算法的一些公用函数(应该主要是内部使用):

```
InterpolateSingle();      //
InterpolateRotation();    //
InterpolateColor();       //
InterpolateLinear();      //
InterpolateSine();        //
InterpolateQuint();       //
InterpolateQuart();       //
InterpolateQuad();        //
InterpolateExpo();        //
InterpolateElastic();     //
InterpolateCubic();       //
InterpolateCirc();        //
InterpolateBounce();      //
InterpolateBack();        //
```

很多动画应该在设计时就可以方便完成, 在选择某些属性值时可直接添加动画, 如:

//Bitmap 属性:

```
Create New TBitmapAnimation
Create New TBitmapListAnimation
```

//Color 属性:

```
Create New TColorAnimation
```


Create New TColorKeyAnimation

//Gradient 属性:

Create New TGradientAnimation

//Width、Height、X、Y、StrokeThickness、XRadius、YRadius、Opacity、RotationAngle 等属性:

Create New TFloatAnimation

Create New TFloatKeyAnimation

先尝试一个让控件转起来的动画吧:

添加一个 TRectangle, 从其 RotationAngle 属性 Create New TFloatAnimation (需要删除时, 选定后按 Delete),

然后调整自动建立的 FloatAnimation1 的属性值:

//一般在设计时取值即可, 下面是运行时的代码:

procedure TForm1.FormCreate(Sender: TObject);

begin

FloatAnimation1.Enabled := True;

FloatAnimation1.Loop := True;

FloatAnimation1.Duration := 2.5; //一个动画周期的长度(秒)

FloatAnimation1.StartValue := 0; //起点角度

FloatAnimation1.StopValue := 360; //终点角度

end;

在设计时制作上面动画的另一方法:

- 1、添加 TRectangle(Rectangle1);
 - 2、选定 Rectangle1 后添加 TFloatAnimation(FloatAnimation1);
 - 3、修改 FloatAnimation1 的属性 PropertyName 值为 RotationAngle;
 - 4、如上设置 FloatAnimation1 的其它属性.
-

完全在运行时实现上面动画的代码:

```
uses FMX.Objects, FMX.Anim; //添加, 但不要重复添加
```

```
var
```

```
    rect: TRectangle;
```

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
```

```
    rect := TRectangle.Create(Self);
```

```
    rect.Parent := Self;
```

```
    rect.Align := TAlignLayout.alCenter;
```

```
    with TFloatAnimation.Create(Self) do
```

```
    begin
```

```
        Parent := rect;
```

```
        PropertyName := 'RotationAngle';
```

```
        Enabled := True;
```

```
        Loop := True;
```

```
        Duration := 2.5;
```

```
        StartValue := 0;
```

```
        StopValue := 360;
```

```
    end;
```

```
end;
```

Delphi XE2 之 FireMonkey 入门(13) - 动画(下)

TAnimation 类的主要成员:

protected

```
function NormalizedTime: Single;      //
```

```
procedure ProcessAnimation; virtual;  //其子类们主要通过覆盖此方法来实  
现不同的动画
```

```
procedure Loaded; override;          //
```

public

```
procedure Start; virtual;             //播放
```

```
procedure Stop; virtual;              //停止
```

```
procedure StopAtCurrent; virtual;     //停止在当前帧; 和 Pause 属性不  
同的是它会触动 onFinish 事件
```

```
procedure StartTrigger(...); virtual; //如果不是覆盖, 一般应使用 Trig  
ger、TriggerInverse 属性而不是该方法
```

```
procedure ProcessTick(...);           //内部使用的动画执行方法, 主要由  
它来调用 ProcessAnimation 过程.
```

```
property Running: Boolean ...;        //是否运行中; 只读
```

```
property Pause: Boolean ...;          //暂停
```

published

```
property AnimationType: TAnimationType ...;  //动画类型; 它好像只  
影响到插入(Interpolation)算法
```

```
property AutoReverse: Boolean ...;          //自动逆向(起点->终点  
->起点)
```

```

property Enabled: Boolean ...;           //是否可用

property Delay: Single ...;              //延迟多少秒再开始动画

property Duration: Single ...;          //动画长度(秒); 其子

```

类一般会默认为 0.2

```

property Interpolation: TInterpolationType ...; //动画插入类型; 通过
此选项可实现像反弹等多种动画效果(很好玩)

```

```

property Inverse: Boolean ...;           //逆向动画(终点->起
点)

```

```

property Loop: Boolean ...;             //循环播放

property Trigger: TTrigger ...;         //指定可触发动画的事件;

```

其值是个字符串(见下表)

```

property TriggerInverse: TTrigger ...;   //指定可触发逆向动画
的事件; 其值是个字符串(见下表)

```

```

property OnProcess: TNotifyEvent ...;    //每个动画帧触发的事
件

```

```

property OnFinish: TNotifyEvent ...;     //停止时触发的事件

end;

```

{ Trigger、TriggerInverse 属性的可选值: }

```

'IsMouseOver=true'
'IsMouseOver=false'
'IsFocused=true'
'IsFocused=false'
'IsVisible=true'
'IsVisible=false'
'IsDragOver=true'

```

```
'IsDragOver=false'  
'IsOpen=true'  
'IsOpen=false'
```

TFloatAnimation 用于尺寸变化的动画;

TColorAnimation 用于颜色变化的动画;

TGradientAnimation 用于颜色梯度动画;

TRectAnimation 用于边界(Padding、Margins)动画; 它们的扩展属性是一样的(但参数类型不一样):

StartValue //起点值

StopValue //终点值

StartFromCurrent //是否从当前帧开始动画

PropertyName //动画要控制的属性; 其值是个字符串(见下表)

{ PropertyName 常用取值: }

```
'Width'  
'Height'  
'StrokeThickness'  
'Position.X'  
'Position.Y'  
'Scale.X'  
'Scale.Y'  
'RotationCenter.X'  
'RotationCenter.Y'  
'RotationAngle'  
'Opacity'  
'Margins.Left'  
'Margins.Top'  
'Margins.Right'  
'Margins.Bottom'
```

```
'Padding.Left'  
'Padding.Top'  
'Padding.Rigth'  
'Padding.Bottom'  
  
'Fill.Color'  
'Stroke.Color'  
  
'Fill.Gradient'  
'Stroke.Gradient'  
  
'Margins'  
'Padding'
```

TFloatKeyAnimation、**TColorKeyAnimation** 可通过其 **Keys** 属性定义多个关键帧(前面几种都只有两个关键帧), 其属性扩展:

```
Keys           //TKeys 类型的集合, 元素类型是 TKey; 主要使用 TKey.ID(关键帧序号)、TKey.Key(参数值) 两个属性  
  
PropertyName   //  
StartFromCurrent //
```

TBitmapAnimation 用于两张图片的切换动画; 它的扩展属性有:

```
StartValue     //起点图片  
  
StopValue      //终点图片  
  
PropertyName   //只能是 'Bitmap'
```

TBitmapListAnimation 只需要一张图片, 根据需要的动画帧数(AnimationCount)把图片均分, 然后让切分后的各部分连成动画; 它的扩展属性有:

AnimationCount //动画帧数

AnimationBitmap //图片

PropertyName //

TPathAnimation 可以让对象绕一个指定的路径运动; 它的扩展属性有:

Path: TPathData; //路径数据; 一般通过 TPathData.Data 读写数据, 数据(字符串)使用了 SVG 中 Path 的格式标准

Rotate: Boolean; //是否旋转(自转)

路径动画简单示例: 先在 HD 窗体上放置一个 TRectangle, 选定后给它添加一个 TPathAnimation; 测试代码:

const

```
    strPath = 'M 3.84500002861023,3.47300004959106 ' +  
              'C 4.83799982070923,24.6110000610352 26.2040004730225,34.974998  
4741211 42.875,30.8320007324219 ' +  
              'C 69.8730010986328,24.121000289917 82.1620025634766,-9.7609996  
7956543 74.6009979248047,-37.4169998168945 ' +  
              'C 63.992000579834,-76.2160034179688 23.7210006713867,-93.31400  
29907227 -9.33600044250488,-80.4909973144531 ' +  
              'C -52.2709999084473,-63.8380012512207 -70.8339996337891,-8.086  
00044250488 -55.2200012207031,37.6059989929199 ' +  
              'C -35.9000015258789,94.1429977416992 24.3759994506836,118.2809  
9822998 73.6240005493164,96.3960037231445 ' +
```

```
'C 132.567993164063,70.2020034790039 157.505996704102,-7.920000
07629395 133.664993286133,-71.5500030517578 ' +
'C 105.81600189209,-145.880996704102 25.3040008544922,-177.1139
98413086 -40.0859985351563,-146.054992675781 ' +
'C -95.6880035400391,-119.646003723145 -129.975997924805,-55.07
50007629395 -127.064002990723,11.4790000915527';
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    PathAnimation1.Path.Data := strPath;
    PathAnimation1.Duration := 8;
    PathAnimation1.AutoReverse := True;
    PathAnimation1.Loop := True;
    PathAnimation1.Rotate := True;
    PathAnimation1.Enabled := True;
end;
```

{ 如果要把路径显示出来, 还需要使用 TPath, 它是专用于呈现路径数据的. }

另外: 动画应该可以叠加和嵌套的, 暂不再深究.

还有, 真的在实用时, 使用这些类恐怕不如直接使用 FMXObject 中的方法来得便宜, 如:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Button1.AnimateFloat('Position.X', Button1.Position.X*2, 1.5);
end;
```

相关单元:

```
FMX.Filter
FMX.FilterCatBlur
FMX.FilterCatGeometry
FMX.FilterCatTransition
FMX_FilterCatColor
FMX_FilterCatColorAdjust
FMX_FilterCatComposite
FMX_FilterCatGenerator
FMX_FilterCatStyle
FMX_FilterCatTiles
FMX.FilterCatDistortion
```

FM 提供了 **10** 个类别的滤镜:

{ 分类名称	实现单元 }
Blur	//FMX.FilterCatBlur
Geometry	//FMX.FilterCatGeometry
Transition	//FMX.FilterCatTransition
Color	//FMX_FilterCatColor
Color Adjust	//FMX_FilterCatColorAdjust
Composite	//FMX_FilterCatComposite
Generator	//FMX_FilterCatGenerator
Style	//FMX_FilterCatStyle
Tiles	//FMX_FilterCatTiles
Distortion	//FMX.FilterCatDistortion

每个分类中包括若干个滤镜:

```
{ Blur }
```

```
    olorKeyAlpha
    askToAlpha
{ Color Adjust }
    HueAdjust
    ontrast
    loom
    loom
{ Composite }
    NormalBlend
{ Generator }
    Fill
    illRGB
{ Style }
    Pixelate
    mboss
    harpen
    oon
    epia
    aperSketch
    encilStroke
{ Tiles }
    Tiler
{ Distortion }
    Ripple
    wirl
    agnify
    moothMagnify
    ands
    ave
    rap
    andedSwirl
    inch
```

滤镜参数:

滤镜类名: T GaussianBlurFilter

滤镜名称: GaussianBlur

滤镜描述: An effect that GaussianBlurs.

参数名称: BlurAmount

参数描述: The GaussianBlur factor.

参数类型: vtFloat

当前值: 1

最小值: .01

默认值: 1

最大值: 10

//-----

滤镜类名: TBlurFilter

滤镜名称: BoxBlur

滤镜描述: An effect that blurs.

参数名称: BlurAmount

参数描述: The blur factor.

参数类型: vtFloat

当前值: 1

最小值: .01

默认值: 1

最大值: 10

//-----

滤镜类名: TDirectionalBlurFilter

滤镜名称: DirectionalBlur

滤镜描述: An effect that blurs **in** a single direction.

参数名称: Angle

参数描述: The direction **of** the blur (**in** degrees).

参数类型: vtFloat

当前值: 0

最小值: 0

默认值: 0

最大值: 360

参数名称: BlurAmount

参数描述: The scale **of** the blur (**as** a fraction **of** the input size).

参数类型: vtFloat

当前值: 1

最小值: .01

默认值: 1

最大值: 10

//-----

滤镜类名: TZoomBlurFilter

滤镜名称: RadialBlur

滤镜描述: An effect that applies a radial blur **to** the input.

参数名称: Center

参数描述: The center point **of** the ripples.

参数类型: vtPoint

当前值: fxpoint 150:150

最小值: fxpoint 0:0

默认值: fxpoint 150:150

最大值: fxpoint 65535:65535

参数名称: BlurAmount

参数描述: The scale **of** the blur (**as** a fraction **of** the input size).

参数类型: vtFloat

当前值: 1

最小值: .01

默认值: 1

最大值: 10

//-----

滤镜类名: TAffineFilter

滤镜名称: AffineTransform

滤镜描述: Applies an affine transform **to** an image.

参数名称: Center

参数描述: The center point **of** the rotation.

参数类型: vtPoint

当前值: fxpoint 150:150

最小值: fxpoint 0:0

默认值: fxpoint 150:150

最大值: fxpoint 65535:65535

参数名称: Rotation

参数描述: Rotation angle **in** degrees.

参数类型: vtFloat

当前值: 0

最小值: -180

默认值: 0

最大值: 180

参数名称: Scale

参数描述: Scale value **as** floating.

参数类型: vtFloat

当前值: 1

最小值: .05

默认值: 1

最大值: 4

//-----

滤镜类名: TPerspectiveFilter

滤镜名称: PerspectiveTransform

滤镜描述: Applies an perspective transform **to** an image.

参数名称: TopLeft

参数描述: Top left point **of** result transformation.

参数类型: vtPoint

当前值: fxpoint 0:0

最小值: fxpoint 0:0

默认值: fxpoint 0:0

最大值: fxpoint 65535:65535

参数名称: TopRight

参数描述: Top right point **of** result transformation.

参数类型: vtPoint

当前值: fxpoint 300:0

最小值: fxpoint 0:0

默认值: fxpoint 300:0

最大值: fxpoint 65535:65535

参数名称: BottomRight

参数描述: Bottom right point **of** result transformation.

参数类型: vtPoint

当前值: fxpoint 350:300

最小值: fxpoint 0:0

默认值: fxpoint 350:300

最大值: fxpoint 65535:65535

参数名称: BottomLeft

参数描述: Bottom left point **of** result transformation.

参数类型: vtPoint

当前值: fxpoint 0:300

最小值: fxpoint 0:0

默认值: fxpoint 0:300

最大值: fxpoint 65535:65535

//-----

滤镜类名: TCropFilter

滤镜名称: Crop

滤镜描述: The size **and** shape **of** the cropped image depend **on** the rectangle you specify.

参数名称: LeftTop

参数描述: Left-top corner **of** cropping rect

参数类型: vtPoint

当前值: fxpoint 0:0

最小值: fxpoint 0:0

默认值: fxpoint 0:0

最大值: fxpoint 65535:65535

参数名称: RightBottom

参数描述: Left-top corner **of** cropping rect

参数类型: vtPoint

当前值: fxpoint 150:150

最小值: fxpoint 0:0

默认值: fxpoint 150:150

最大值: fxpoint 65535:65535

//-----

滤镜类名: TBandedSwirlTransition

滤镜名称: BandedSwirlTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: Strength

参数描述: The amount **of** twist **to** the spiral.

参数类型: vtFloat

当前值: 1

最小值: 0

默认值: 1

最大值: 10

参数名称: Frequency

参数描述: The frequency **of** the spiral.

参数类型: vtFloat

当前值: 20

最小值: 0

默认值: 20

最大值: 100

参数名称: Center

参数描述: The center point **of** the ripples.

参数类型: vtPoint

当前值: fxpoint 150:150

最小值: fxpoint 0:0

默认值: fxpoint 150:150

最大值: fxpoint 65535:65535

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TBlindTransition

滤镜名称: BlindTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: NumberOfBlinds

参数描述: The number **of** Blinds strips

参数类型: vtFloat

当前值: 5

最小值: 2

默认值: 5

最大值: 15

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TBloodTransition

滤镜名称: BloodTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) of the transition from first texture to the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: RandomSeed

参数描述: The seed value that determines dripiness.

参数类型: vtFloat

当前值: .3

最小值: 0

默认值: .3

最大值: 1

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TCircleTransition

滤镜名称: CircleTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: FuzzyAmount

参数描述: The fuzziness factor.

参数类型: vtFloat

当前值: .1

最小值: 0

默认值: .1

最大值: 1

参数名称: Size

参数描述: The size **of** the circle.

参数类型: vtFloat

当前值: 1

最小值: 0

默认值: 1

最大值: 2

参数名称: Center

参数描述: The center point **of** effect.

参数类型: vtPoint

当前值: fxpoint 150:150

最小值: fxpoint 0:0

默认值: fxpoint 150:150

最大值: fxpoint 65535:65535

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TMagnifyTransition

滤镜名称: MagnifyTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: Center

参数描述: The center point **of** effect.

参数类型: vtPoint

当前值: fxpoint 150:150

最小值: fxpoint 0:0

默认值: fxpoint 150:150

最大值: fxpoint 65535:65535

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TCrumpleTransition

滤镜名称: CrumpleTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: RandomSeed

参数描述: The seed value that determines dripiness.

参数类型: vtFloat

当前值: 0

最小值: -1

默认值: 0

最大值: 1

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TDissolveTransition

滤镜名称: DissolveTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: RandomSeed

参数描述: The seed value that determines dripiness.

参数类型: vtFloat

当前值: 0

最小值: -1

默认值: 0

最大值: 1

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TDropTransition

滤镜名称: DropTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: RandomSeed

参数描述: The seed value that determines dripiness.

参数类型: vtFloat

当前值: 0

最小值: -1

默认值: 0

最大值: 1

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TFadeTransition

滤镜名称: FadeTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TBrightTransition

滤镜名称: BrightTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TPixelateTransition

滤镜名称: PixelateTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) of the transition from first texture to the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TBlurTransition

滤镜名称: BlurTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) of the transition from first texture to the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TWiggleTransition

滤镜名称: WiggleTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) of the transition from first texture to the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TShapeTransition

滤镜名称: ShapeTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TRippleTransition

滤镜名称: RippleTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TRotateCrumpleTransition

滤镜名称: RotateCrumpleTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: RandomSeed

参数描述: The seed value that determines dripiness.

参数类型: vtFloat

当前值: 0

最小值: -1

默认值: 0

最大值: 1

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TSaturateTransition

滤镜名称: SaturateTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TSlideInTransition

滤镜名称: SlideTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: SlideAmount

参数描述: The center point **of** the ripples.

参数类型: vtPoint

当前值: fxpoint 150:150

最小值: fxpoint -65535:-65535

默认值: fxpoint 150:150

最大值: fxpoint 65535:65535

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TSwirlTransition

滤镜名称: SwirlTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: Strength

参数描述: The amount **of** twist **to** the spiral.

参数类型: vtFloat

当前值: 30

最小值: -70

默认值: 30

最大值: 70

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TWaterTransition

滤镜名称: WaterTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) **of** the transition from first texture **to** the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: RandomSeed

参数描述: The seed value that determines dripiness.

参数类型: vtFloat

当前值: .3

最小值: 0

默认值: .3

最大值: 1

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TWaveTransition

滤镜名称: WaveTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) of the transition from first texture to the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TLineTransition

滤镜名称: LineTransition

滤镜描述: A transition effect.

参数名称: Progress

参数描述: The amount (%) of the transition from first texture to the second texture.

参数类型: vtFloat

当前值: 30

最小值: 0

默认值: 30

最大值: 100

参数名称: Origin

参数描述: The line origin.

参数类型: vtPoint

当前值: fxpoint 0:0

最小值: fxpoint 0:0

默认值: fxpoint 0:0

最大值: fxpoint 65535:65535

参数名称: Normal

参数描述: The line normal.

参数类型: vtPoint

当前值: fxpoint 150:150

最小值: fxpoint 0:0

默认值: fxpoint 150:150

最大值: fxpoint 65535:65535

参数名称: Offset

参数描述: The line offset.

参数类型: vtPoint

当前值: fxpoint 400:400

最小值: fxpoint 0:0

默认值: fxpoint 400:400

最大值: fxpoint 65535:65535

参数名称: FuzzyAmount

参数描述: The fuzziness factor.

参数类型: vtFloat

当前值: .1

最小值: 0

默认值: .1

最大值: 1

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TInvertFilter

滤镜名称: Invert

滤镜描述: An effect that inverts all colors.

//-----

滤镜类名: TMonochromeFilter

滤镜名称: Monochrome

滤镜描述: Remaps colors so they fall within shades **of** a single color.

//-----

滤镜类名: TColorKeyAlphaFilter

滤镜名称: ColorKeyAlpha

滤镜描述: An effect that makes pixels **of** a particular color transparent.

参数名称: ColorKey

参数描述: The color that becomes transparent.

参数类型: vtFloat

当前值: 0

最小值: -1

默认值: 0

最大值: 1

参数名称: Tolerance

参数描述: The tolerance **in** color differences.

参数类型: vtFloat

当前值: .3

最小值: 0

默认值: .3

最大值: 1

//-----

滤镜类名: TMaskToAlphaFilter

滤镜名称: MaskToAlpha

滤镜描述: Converts a grayscale image **to** a white image that **is** masked by alpha.

//-----

滤镜类名: THueAdjustFilter

滤镜名称: HueAdjust

滤镜描述: Changes the overall hue, **or** tint, **of** the source pixels.

参数名称: Hue

参数描述: The hue offset.

参数类型: vtFloat

当前值: 0

最小值: -1

默认值: 0

最大值: 1

//-----

滤镜类名: TContrastFilter

滤镜名称: Contrast

滤镜描述: An effect that controls brightness **and** contrast.

参数名称: Brightness

参数描述: The brightness offset.

参数类型: vtFloat

当前值: 0

最小值: -1

默认值: 0

最大值: 1

参数名称: Contrast

参数描述: The contrast multiplier.

参数类型: vtFloat

当前值: 1.5

最小值: 0

默认值: 1.5

最大值: 2

//-----

滤镜类名: TBloomFilter

滤镜名称: Bloom

滤镜描述: An effect that intensifies bright regions.

参数名称: BloomIntensity

参数描述: Intensity **of** the bloom image.

参数类型: vtFloat

当前值: 1

最小值: 0

默认值: 1

最大值: 1

参数名称: BaseIntensity

参数描述: Intensity **of** the base image.

参数类型: vtFloat

当前值: .5

最小值: 0

默认值: .5

最大值: 1

参数名称: BloomSaturation

参数描述: Saturation **of** the bloom image.

参数类型: vtFloat

当前值: 1

最小值: 0

默认值: 1

最大值: 1

参数名称: BaseSaturation

参数描述: Saturation **of** the base image.

参数类型: vtFloat

当前值: .5

最小值: 0

默认值: .5

最大值: 1

//-----

滤镜类名: TGloomFilter

滤镜名称: Gloom

滤镜描述: An effect that intensifies dark regions.

参数名称: GloomIntensity

参数描述: Intensity **of** the gloom image.

参数类型: vtFloat

当前值: 1

最小值: 0

默认值: 1

最大值: 1

参数名称: BaseIntensity

参数描述: Intensity **of** the base image.

参数类型: vtFloat

当前值: .5

最小值: 0

默认值: .5

最大值: 1

参数名称: GloomSaturation

参数描述: Saturation **of** the gloom image.

参数类型: vtFloat

当前值: 1

最小值: 0

默认值: 1

最大值: 1

参数名称: BaseSaturation

参数描述: Saturation **of** the base image.

参数类型: vtFloat

当前值: .5

最小值: 0

默认值: .5

最大值: 1

//-----

滤镜类名: TNormalBlendFilter

滤镜名称: NormalBlend

滤镜描述: Normal blending of two images.

参数名称: Target

参数描述: The target bitmap.

参数类型: vtBitmap

//-----

滤镜类名: TFillFilter

滤镜名称: Fill

滤镜描述: Generates a solid color.

参数名称: Color

参数描述: The fill color.

参数类型: vtColor

当前值: 4280299584

最小值: 0

默认值: 4280299584

最大值: 0

//-----

滤镜类名: TFillOpaqueFilter

滤镜名称: FillRGB

滤镜描述: Fill all pixels **with not** empty alpha.

参数名称: Color

参数描述: The fill color.

参数类型: vtColor

当前值: 4280299584

最小值: 0

默认值: 4280299584

最大值: 0

//-----

滤镜类名: TPixelateFilter

滤镜名称: Pixelate

滤镜描述:

参数名称: BlockCount

参数描述: The number **of** pixel blocks.

参数类型: vtFloat

当前值: 25

最小值: 1

默认值: 25

最大值: 1000

//-----

滤镜类名: TEmbossFilter

滤镜名称: Emboss

滤镜描述: An effect that embosses the input.

参数名称: Amount

参数描述: The amplitude **of** the embossing.

参数类型: vtFloat

当前值: .5

最小值: 0

默认值: .5

最大值: 1

参数名称: Width

参数描述: The separation between samples (**as** a fraction **of** input size).

参数类型: vtFloat

当前值: 3

最小值: 0

默认值: 3

最大值: 10

//-----

滤镜类名: TSharpenFilter

滤镜名称: Sharpen

滤镜描述: An effect that sharpens the input.

参数名称: Amount

参数描述: The amount **of** sharpening.

参数类型: vtFloat

当前值: 1

最小值: 0

默认值: 1

最大值: 2

//-----

滤镜类名: TToonFilter

滤镜名称: Toon

滤镜描述: An effect that applies cartoon-like shading (posterization).

参数名称: Levels

参数描述: The number **of** color levels **to** use.

参数类型: vtFloat

当前值: 5

最小值: 3

默认值: 5

最大值: 15

//-----

滤镜类名: TSepiaFilter

滤镜名称: Sepia

滤镜描述: Sepia effect.

参数名称: Amount

参数描述: The amount **of** sharpening.

参数类型: vtFloat

当前值: .5

最小值: 0

默认值: .5

最大值: 1

//-----

滤镜类名: TPaperSketchFilter

滤镜名称: PaperSketch

滤镜描述: An paper sketch effect.

参数名称: BrushSize

参数描述: The brush size **of** the sketch effect.

参数类型: vtFloat

当前值: 3

最小值: .6

默认值: 3

最大值: 10

//-----

滤镜类名: TPencilStrokeFilter

滤镜名称: PencilStroke

滤镜描述: An pencil stroke effect.

参数名称: BrushSize

参数描述: The brush size **of** the sketch effect.

参数类型: vtFloat

当前值: 5

最小值: 1

默认值: 5

最大值: 19

//-----

滤镜类名: TTilerFilter

滤镜名称: Tiler

滤镜描述: Pixel shader tiles the image across multiple rows **and** columns

参数名称: VerticalTileCount

参数描述: The number **of** verical tiles **to** add **to** the output. The higher the value the more tiles.

参数类型: vtFloat

当前值: 4

最小值: 0

默认值: 4

最大值: 20

参数名称: HorizontalTileCount

参数描述: The number **of** horizontal tiles **to** add **to** the output. The higher the value the more tiles.

参数类型: vtFloat

当前值: 3

最小值: 0

默认值: 3

最大值: 20

参数名称: HorizontalOffset

参数描述: Change the horizontal offset **of** each tile.

参数类型: vtFloat

当前值: 0

最小值: 0

默认值: 0

最大值: 1

参数名称: VerticalOffset

参数描述: Change the vertical offset **of** each tile.

参数类型: vtFloat

当前值: 0

最小值: 0

默认值: 0

最大值: 1

//-----

滤镜类名: TRippleFilter

滤镜名称: Ripple

滤镜描述: An effect that superimposes rippling waves upon the input.

参数名称: Center

参数描述: The center point **of** the ripples.

参数类型: vtPoint

当前值: fxpoint 150:150

最小值: fxpoint 0:0

默认值: fxpoint 150:150

最大值: fxpoint 65535:65535

参数名称: Amplitude

参数描述: The amplitude **of** the ripples.

参数类型: vtFloat

当前值: .1

最小值: 0

默认值: .1

最大值: 1

参数名称: Frequency

参数描述: The frequency **of** the ripples.

参数类型: vtFloat

当前值: 70

最小值: 0

默认值: 70

最大值: 100

参数名称: Phase

参数描述: The phase **of** the ripples.

参数类型: vtFloat

当前值: 0

最小值: -20

默认值: 0

最大值: 20

参数名称: AspectRatio

参数描述: The aspect ratio (width / height) **of** the input.

参数类型: vtFloat

当前值: 1.5

最小值: .5

默认值: 1.5

最大值: 2

//-----

滤镜类名: TSwirlFilter

滤镜名称: Swirl

滤镜描述: An effect that swirls the input **in** a spiral.

参数名称: Center

参数描述: The center point **of** the ripples.

参数类型: vtPoint

当前值: fxpoint 150:150

最小值: fxpoint 0:0

默认值: fxpoint 150:150

最大值: fxpoint 65535:65535

参数名称: Strength

参数描述: The amount **of** twist **to** the spiral.

参数类型: vtFloat

当前值: 10

最小值: -70

默认值: 10

最大值: 70

参数名称: AspectRatio

参数描述: The aspect ratio (width / height) **of** the input.

参数类型: vtFloat

当前值: 1.5

最小值: .5

默认值: 1.5

最大值: 2

// -----

滤镜类名: TMagnifyFilter

滤镜名称: Magnify

滤镜描述: An effect that magnifies a circular region.

参数名称: Center

参数描述: The center point **of** the ripples.

参数类型: vtPoint

当前值: fxpoint 150:150

最小值: fxpoint 0:0

默认值: fxpoint 150:150

最大值: fxpoint 65535:65535

参数名称: Radius

参数描述: The radius **of** the magnified region.

参数类型: vtFloat

当前值: .25

最小值: 0

默认值: .25

最大值: 1

参数名称: Magnification

参数描述: The magnification factor.

参数类型: vtFloat

当前值: 2

最小值: 1

默认值: 2

最大值: 5

参数名称: AspectRatio

参数描述: The aspect ratio (width / height) **of** the input.

参数类型: vtFloat

当前值: 1.5

最小值: .5

默认值: 1.5

最大值: 2

//-----

滤镜类名: TSmoothMagnifyFilter

滤镜名称: SmoothMagnify

滤镜描述: An effect that magnifies a circular region.

参数名称: Center

参数描述: The center point **of** the ripples.

参数类型: vtPoint

当前值: fxpoint 150:150

最小值: fxpoint 0:0

默认值: fxpoint 150:150

最大值: fxpoint 65535:65535

参数名称: InnerRadius

参数描述: The inner radius **of** the magnified region.

参数类型: vtFloat

当前值: .2

最小值: 0

默认值: .2

最大值: 1

参数名称: OuterRadius

参数描述: The outer radius **of** the magnified region.

参数类型: vtFloat

当前值: .4

最小值: 0

默认值: .4

最大值: 1

参数名称: Magnification

参数描述: The magnification factor.

参数类型: vtFloat

当前值: 2

最小值: 1

默认值: 2

最大值: 5

参数名称: AspectRatio

参数描述: The aspect ratio (width / height) **of** the input.

参数类型: vtFloat

当前值: 1.5

最小值: .5

默认值: 1.5

最大值: 2

//-----

滤镜类名: TBandsFilter

滤镜名称: Bands

滤镜描述: An effect that creates bands **of** bright regions.

参数名称: BandDensity

参数描述: The number **of** verical bands **to** add **to** the output. The higher the value the more bands.

参数类型: vtFloat

当前值: 65

最小值: 0

默认值: 65

最大值: 150

参数名称: BandIntensity

参数描述: Intensity **of** each band.

参数类型: vtFloat

当前值: .2

最小值: 0

默认值: .2

最大值: 1

//-----

滤镜类名: TWaveFilter

滤镜名称: Wave

滤镜描述: An effect that applies a wave pattern **to** the input.

参数名称: Time

参数描述: The moment **in** time. Animate this value over a long period **of** time. The speed depends **on** the size. The larger the size, the larger the increase **in** time **on** every frame, thus from 0 **to** 2048 **in** a smaller amount **of** time.

参数类型: vtFloat

当前值: 0

最小值: 0

默认值: 0

最大值: 2048

参数名称: WaveSize

参数描述: The distance between waves. (the higher the value the closer the waves are **to** their neighbor).

参数类型: vtFloat

当前值: 64

最小值: 32

默认值: 64

最大值: 256

//-----

滤镜类名: TWrapFilter

滤镜名称: Wrap

滤镜描述: Wrap image by two Bezier curves.

参数名称: LeftStart

参数描述: Left wrap curve start point

参数类型: vtFloat

当前值: 0

最小值: 0

默认值: 0

最大值: 1

参数名称: LeftControl1

参数描述: Left wrap curve control point 1

参数类型: vtFloat

当前值: .25

最小值: 0

默认值: .25

最大值: 1

参数名称: LeftControl2

参数描述: Left wrap curve control point 2

参数类型: vtFloat

当前值: .25

最小值: 0

默认值: .25

最大值: 1

参数名称: LeftEnd

参数描述: Left wrap curve **end** point

参数类型: vtFloat

当前值: 0

最小值: 0

默认值: 0

最大值: 1

参数名称: RightStart

参数描述: Right wrap curve start point

参数类型: vtFloat

当前值: 1

最小值: 0

默认值: 1

最大值: 1

参数名称: RightControl1

参数描述: Right wrap curve control point 1

参数类型: vtFloat

当前值: .75

最小值: 0

默认值: .75

最大值: 1

参数名称: RightControl2

参数描述: Right wrap curve control point 2

参数类型: vtFloat

当前值: .75

最小值: 0

默认值: .75

最大值: 1

参数名称: RightEnd

参数描述: Right wrap curve **end** point

参数类型: vtFloat

当前值: 1

最小值: 0

默认值: 1

最大值: 1

//-----

滤镜类名: TBandedSwirlFilter

滤镜名称: BandedSwirl

滤镜描述: An effect that swirls the input **in** alternating clockwise **and** counterclockwise bands.

参数名称: Center

参数描述: The center point **of** the ripples.

参数类型: vtPoint

当前值: fxpoint 150:150

最小值: fxpoint 0:0

默认值: fxpoint 150:150

最大值: fxpoint 65535:65535

参数名称: Bands

参数描述: The number **of** bands **in** the swirl.

参数类型: vtFloat

当前值: 10

最小值: 0

默认值: 10

最大值: 20

参数名称: Strength

参数描述: The amount **of** twist **to** the spiral.

参数类型: vtFloat

当前值: 30

最小值: -70

默认值: 30

最大值: 70

参数名称: AspectRatio

参数描述: The aspect ratio (width / height) **of** the input.

参数类型: vtFloat

当前值: 1.5

最小值: .5

默认值: 1.5

最大值: 2

//-----

滤镜类名: TPinchFilter

滤镜名称: Pinch

滤镜描述: An effect that pinches a circular region.

参数名称: Center

参数描述: The center point **of** the pinched region.

参数类型: vtPoint

当前值: fxpoint 150:150

最小值: fxpoint 0:0

默认值: fxpoint 150:150

最大值: fxpoint 65535:65535

参数名称: Radius

参数描述: The radius **of** the pinched region.

参数类型: vtFloat

当前值: .25

最小值: 0

默认值: .25

最大值: 1

参数名称: Strength

参数描述: The amount **of** twist **to** the spiral.

参数类型: vtFloat

当前值: 10

最小值: 0

默认值: 10

最大值: 20

参数名称: AspectRatio

参数描述: The aspect ratio (width / height) **of** the input.

参数类型: vtFloat

当前值: 1.5

最小值: .5

默认值: 1.5

最大值：2

//-----

Delphi XE2 之 FireMonkey 入门(15) - 滤镜：获取滤镜信息

滤镜类的继承关系：

TObject -> TPersistent -> TFilter -> TShaderFilter -> { 具体的滤镜类 }

//下面例子首先会用到 FMX.Filter 单元四个公用方法：

```
procedure FillCategory(AList: TStrings);  
procedure FillFiltersInCategory(const Category: string; AList: TStrings);  
function FilterByName(const AName: string): TFilter;  
function FilterClassByName(const AName: string): TFilterClass;
```

通过 FillCategory()方法可以获取滤镜的分类列表：

```
FillCategory(ListBox1.Items);
```

通过 FillFiltersInCategory()方法可以获取某个类别的滤镜列表：

```
FillFiltersInCategory('分类名称', ListBox2.Items);
```

通过 FilterByName()方法可以获取滤镜对象：

var

```
filter: TFilter;  
begin  
    filter := FilterByName('滤镜名称'); //这很方便, 省得手动建立了  
end;
```

通过 TFilter 的 FilterAttr()方法可以获取滤镜的信息:

```
var  
    filter: TFilter;  
    filterRec: TFilterRec; //滤镜信息被包装在 TFilterRec 类型的结构体中  
begin  
    filter := FilterByName('滤镜名称');  
    filterRec := filter.FilterAttr;  
    {因为 FilterAttr 是 class 方法, 所以也可以如下获取}  
    filterRec := FilterClassByName('滤镜名称').FilterAttr;  
end;
```

TFilterRec 结构:

```
TFilterRec = record  
    Name: string; //滤镜名称  
    Desc: string; //滤镜描述  
    Values: TShaderValueRecArray; //滤镜的参数数组, TShaderValueRec 类型  
end;
```

TShaderValueRec 结构:

```
TShaderValueRec = record

    Name: string;                // 参数名称

    Desc: string;                // 参数描述

    ValueType: TShaderValueType; // 参数类型: vtFloat, vtPoint, vtColor,
vtBitmap(数值、点、颜色值、图像)

    Value: Variant;              // 参数值

    Min, Max, Default: Variant; // 参数的最小、最大及默认值

end;
```

测试: 需要两个 TListBox、一个 TMemo, 还有 OnCreate 事件和 TListBox 的 OnClick 事件.

```
uses System.TypeInfo; // 用于获取枚举名称

procedure TForm1.FormCreate(Sender: TObject);
begin

    FillCategory(ListBox1.Items); // 获取分类名称列表

end;

procedure TForm1.ListBox1Click(Sender: TObject);
var
    strItem1: string;
begin
    strItem1 := ListBox1.Items[ListBox1.ItemIndex];

    FillFiltersInCategory(strItem1, ListBox2.Items); // 获取指定类别的滤
    镜列表

    ListBox2.ItemIndex := 0;
    ListBox2.OnClick(nil);
```

```
end;
```

```
procedure TForm1.ListBox2Click(Sender: TObject);
```

```
var
```

```
    strItem2: string;
```

```
    filter: TFilter;
```

```
    filterRec: TFilterRec;
```

```
    shaderValueRec: TShaderValueRec;
```

```
begin
```

```
    strItem2 := ListBox2.Items[ListBox2.ItemIndex];
```

```
    filter := FilterByName(strItem2); //通过滤镜名称获取滤镜对象
```

```
    filterRec := filter.FilterAttr; //获取滤镜对象的信息
```

```
// filterRec := FilterClassByName(strItem).FilterAttr; //同上一行
```

```
    Memo1.Text := '滤镜类名: ' + filter.ToString; //ClassName
```

```
    Memo1.Lines.Add('滤镜名称: ' + filterRec.Name);
```

```
    Memo1.Lines.Add('滤镜描述: ' + filterRec.Desc);
```

```
    Memo1.Lines.Add('-----  
' );
```

```
    for shaderValueRec in filterRec.Values do
```

```
    begin
```

```
        Memo1.Lines.Add('参数名称: ' + shaderValueRec.Name);
```

```
        Memo1.Lines.Add('参数描述: ' + shaderValueRec.Desc);
```

```
        Memo1.Lines.Add('参数类型: ' + GetEnumName(TypeInfo(TShaderValueType), Integer(shaderValueRec.ValueType)));
```

```
        if shaderValueRec.ValueType <> TShaderValueType.vtBitmap then
```

```
        begin
```

```
            Memo1.Lines.Add('当前值: ' + string(shaderValueRec.Value));
```

```
    Memo1.Lines.Add('最小值: ' + string(shaderValueRec.Min));  
  
    Memo1.Lines.Add('默认值: ' + string(shaderValueRec.Default));  
  
    Memo1.Lines.Add('最大值: ' + string(shaderValueRec.Max));  
  
    end;  
  
    Memo1.Lines.Add(EmptyStr);  
  
    end;  
end;
```

Delphi XE2 之 FireMonkey 入门(16) - 滤镜: 实例测试

窗体上需要 TImage、TOpenDialog 和六个按钮。

```
unit Unit1;
```

```
interface
```

```
uses
```

```
    System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,
```

```
    FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs, FMX.Objects;
```

```
type
```

```
    TForm1 = class (TForm)
```

```
        Image1: TImage;
```

```
        OpenDialog1: TOpenDialog;
```

```
        Button1: TButton;
```

```
        Button2: TButton;
```

```
        Button3: TButton;
```

```
        Button4: TButton;
```

```
        Button5: TButton;
```

```
Button6: TButton;

procedure FormCreate(Sender: TObject);
procedure FormDestroy(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure Button4Click(Sender: TObject);
procedure Button5Click(Sender: TObject);
procedure Button6Click(Sender: TObject);
end;

var
    Form1: TForm1;

implementation

{$R *.fmx}

uses FMX.Filter;

var
    bit: TBitmap;

procedure TForm1.FormCreate(Sender: TObject);
begin
    bit := TBitmap.Create(0, 0);
    if not OpenFileDialog1.Execute then Exit;
    bit.LoadFromFile(OpenDialog1.FileName);
    Image1.Bitmap.Assign(bit);
    Image1.SetBounds(10, 10, bit.Width, bit.Height);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
```



```
    bit.Free;
end;

{ 高斯模糊：一个参数 }

procedure TForm1.Button1Click(Sender: TObject);
var
    filter: TFilter;
begin
    filter := FilterByName('GaussianBlur');           //GaussianBlur
    是高斯模糊滤镜的名称

    filter.Values['BlurAmount'] := 0.25;              //BlurAmount 是高
    斯模糊参数的名称，默认值是 1 (0.1..10)

    filter.ValuesAsBitmap['input'] := bit;            //input 是给输入
    图片约定的索引名称，不区分大小写

    Image1.Bitmap := filter.ValuesAsBitmap['output']; //output 是给输出
    图片约定的索引名称
end;

{ 颜色翻转：无参数 }

procedure TForm1.Button2Click(Sender: TObject);
var
    filter: TFilter;
begin
    filter := FilterByName('Invert');                 //颜色翻转

    filter.ValuesAsBitmap['input'] := bit;

    Image1.Bitmap := filter.ValuesAsBitmap['output'];
end;

{ 灰度：无参数 }

procedure TForm1.Button3Click(Sender: TObject);
```

var

filter: TFilter;

begin

filter := FilterByName('Monochrome');

filter.ValuesAsBitmap['input'] := bit;

Image1.Bitmap := filter.ValuesAsBitmap['output'];

end;

{ 马赛克: 一个参数 }

procedure TForm1.Button4Click(Sender: TObject);**var**

filter: TFilter;

begin

filter := FilterByName('Pixelate');

filter.Values['BlockCount'] := 30; {25 : 1..1000}

filter.ValuesAsBitmap['input'] := bit;

Image1.Bitmap := filter.ValuesAsBitmap['output'];

end;

{ 对比度: 两个参数 }

procedure TForm1.Button5Click(Sender: TObject);**var**

filter: TFilter;

begin

filter := FilterByName('Contrast');

filter.Values['Brightness'] := -0.5; {-1..1}

filter.Values['Contrast'] := 1.5; {0..2}

filter.ValuesAsBitmap['input'] := bit;

Image1.Bitmap := filter.ValuesAsBitmap['output'];

end;

{ 透视: 参数是四个点 }

procedure TForm1.Button6Click(Sender: TObject);

var

filter: TFilter;

begin

filter := FilterByName('PerspectiveTransform');

filter.Values['TopLeft'] := VarFromPointXY(0, 0);

filter.Values['TopRight'] := VarFromPointXY(bit.Width, 0);

filter.Values['BottomRight'] := VarFromPointXY(bit.Width * 2, bit.Height * 2);

filter.Values['BottomLeft'] := VarFromPointXY(0, bit.Height);

filter.ValuesAsBitmap['input'] := bit;

Image1.Bitmap := filter.ValuesAsBitmap['output'];

Image1.SetBounds(10, 10, bit.Width * 2, bit.Height * 2);

end;**end.**

Transition 分类的滤镜需要两张图片:

uses FMX.Filter;**var**

bit1, bit2: TBitmap;

{ 百叶窗效果 }

procedure TForm1.FormCreate(Sender: TObject);**var**

filter: TFilter;

begin

bit1 := TBitmap.Create(0, 0);

bit2 := TBitmap.Create(0, 0);

bit1.LoadFromFile('C:\Temp\test1.jpg');

bit2.LoadFromFile('C:\Temp\test2.jpg');

```
filter := FilterByName('BlindTransition');

filter.Values['Progress'] := 50;           //变换进度: 0..100; 当以此进度
做动画时, PropertyName := 'Value'

filter.Values['NumberOfBlinds'] := 10; //分叶数: 2..15

filter.ValuesAsBitmap['input'] := bit1;

filter.ValuesAsBitmap['target'] := bit2; //target 也是约定的索引名称

Image1.Bitmap := filter.ValuesAsBitmap['output'];
Image1.SetBounds(10, 10, bit1.Width, bit1.Height);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    bit1.Free;
    bit2.Free;
end;
```

Delphi XE2 之 FireMonkey 入门(17) - 特效

刚打开 XE2 时, 就从 Tool Palette 窗口的 Effects 组中发现洋洋洒洒的六十多个特效...

每个特效分别对应一个类, 分别来自 FMX.Effects 和 FMX.Filter.Effects 单元.

FMX.Effects 中的特效属于附加特效, FMX.Filter.Effects 中的特效只是某种滤镜效果; 它们都是滤镜的一种快捷应用.

每个特效有不同的参数; 因都是直接或间接地继承与 FMX.Types 中的 TEffect, 也都拥有 Trigger、Enabled、GetDisablePaint 等功能.

测试: 在窗体上放一个 TPanel

{ 为 Panel1 设置阴影特效 }

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  with TShadowEffect.Create(Self) do
  begin
    Parent := Panel1;           //特效也是作用于其父对象

    Distance := 3.0;           //距离

    Direction := 45.0;         //角度

    Softness := 0.3;           //柔和度

    Opacity := 0.6;            //透明度

    ShadowColor := claBlack;    //阴影色

    Trigger := 'IsMouseOver=true'; //指定启用效果的事件，从其父类 TEffect
    继承，现在不太好用(也可能是我不会用)

  end;
end;
```

Delphi XE2 之 FireMonkey 入门(18) - TLang(多语言切换的实现)

一个小小的 TLang 类，实现多语言切换，挺好的。它的工作思路是：

- 1、首先通过 AddLang('语言代码') 添加语言类别，如：AddLang('en')、AddLang('cn')。
- 2、每个语言代码对应一个 TStringList 列表，获取方式如：LangStr['en']、LangStr['cn']。
- 3、可以手动填充这些数据、可以通过 LoadFromFile() 方法载入之前 SaveToFile() 的数据(*.lng)、还可以在设计时提供的界面中操作这些数据。
- 4、切换时修改 Lang 属性即可，如 Lang := 'cn'。

5、它的作用域是当前工程的所有窗体及控件，但不包括绘图控件(如 `TText`)、控件中的文本和窗体标题等。

测试：

- 1、先在窗体上添加 `TLang`;
 - 2、添加三个 `TRadioButton`, 用于切换语言;
 - 3、添加 `TButton`、`TCheckBox`、`TLabel` 用于显示测试;
 - 4、激活 `RadioButton1`、`Button1` 和窗体的默认事件。
-

```
unit Unit1;
```

```
interface
```

```
uses
```

```
    System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,  
    FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs, FMX.Objects;
```

```
type
```

```
    TForm1 = class(TForm)  
        Lang1: TLang;  
        RadioButton1: TRadioButton;  
        RadioButton2: TRadioButton;  
        RadioButton3: TRadioButton;  
        Button1: TButton;  
        CheckBox1: TCheckBox;  
        Label1: TLabel;  
        procedure FormCreate(Sender: TObject);  
        procedure RadioButton1Change(Sender: TObject);  
        procedure Button1Click(Sender: TObject);  
    end;
```

var

Form1: TForm1;

implementation

{ \$R *.fmx }

procedure TForm1.FormCreate(Sender: TObject);

begin

RadioButton1.Text := 'English';

RadioButton2.Text := '简体中文';

RadioButton3.Text := '繁体中文';

RadioButton1.Tag := 0;

RadioButton2.Tag := 1;

RadioButton3.Tag := 2;

RadioButton2.OnChange := RadioButton1.OnChange;

RadioButton3.OnChange := RadioButton1.OnChange;

{ 这些标题应对应着 TLang 的相关设置 }

Button1.Text := 'Button';

CheckBox1.Text := 'CheckBox';

Label1.Text := 'Test';

Caption := 'Test';

{ 添加语言类别 }

Lang1.AddLang('en');

Lang1.AddLang('cn');

Lang1.AddLang('big');

{ Original: 这个原始的 TStrings 可有可无 }

with Lang1.Original **do**

begin

```
Add('Button');  
Add('CheckBox');  
Add('Test');  
end;
```

{en 作为默认也可以不设置}

```
with Lang1.LangStr['en'] do  
begin  
    Add('Button');  
    Add('CheckBox');  
    Add('Test');  
end;
```

{简体中文}

```
with Lang1.LangStr['cn'] do  
begin  
    Values['Button'] := '按钮';  
  
    Values['CheckBox'] := '复选框';  
  
    Values['Test'] := '测试';  
  
    // Values[Lang1.Original[0]] := '按钮';  
    // Values[Lang1.Original[1]] := '复选框';  
    // Values[Lang1.Original[2]] := '测试';  
  
end;
```

{繁体中文}

```
with Lang1.LangStr['big'] do  
begin  
    Values['Button'] := '按鈕';  
  
    Values['CheckBox'] := '復選框';
```



```
    Values['Test'] := '測試';

    end;
end;

{ 切换 }

procedure TForm1.RadioButton1Change(Sender: TObject);
begin
    case TRadioButton(Sender).Tag of
        0: Lang1.Lang := 'en';
        1: Lang1.Lang := 'cn';
        2: Lang1.Lang := 'big';
    end;
end;

{ 语言数据保存在 Resources 属性中, 它是嵌套的 TStrings 类型 }

procedure TForm1.Button1Click(Sender: TObject);
var
    i: Integer;
begin
    for i := 0 to Lang1.Resources.Count - 1 do
        ShowMessage(TStrings(Lang1.Resources.Objects[i]).Text);
    end;

end.
```

Delphi XE2 之 FireMonkey 入门(19) - TFmxObject 的子类们(表)

参考: [和 FMX 相关的类\(表\)](#)

TFmxObject	IFreeNotification						
	TAnimation	TBitmapAnimation					
		TBitmapListAnimation					
		TColorAnimation					
		TColorKeyAnimation					
		TFloatAnimation					
		TFloatKeyAnimation					
		TGradientAnimation					
		TPathAnimation					
		TRectAnimation					
	TBitmapObject						
	TBrushObject						
	TCommonDialog	TOpenDialog	TSaveDialog				
		TPageSetupDialog					
		TPrintDialog					
		TPrinterSetupDialog					
	TCustomPopupMenu	TPopupMenu	IItemsContainer				

	opupMenu						
	TEffect	TBevelEffect					
		TBlurEffect					
		TGlowEffect	TInnerGlowEffect				
		TImageFXEffect	TImageFXEffect -> 57 个				
		TReflectionEffect					
		TShadowEffect					
	TFilterBaseFilter	TFilterBaseFilter -> 57 个					
	TLang						
	TPathObject						
	TStyleBook						
	TTimer						
	TControl3D	IControl					
		TCamera					
		TDummy	TModel3D				
		TGrid3D					
		TLight					
		TProxyObject					
		TSelectionPoint3D					
		TShape3D	TCustomMesh	TCone			
				TCube			

				TCylinder			
				TMesh			
				TPlane			
				TRoundCube			
				TSphere			
			TExtrudedShape3D	TEllipse3D			
				TPath3D			
				TRectangle3D			
				TText3D			
			TStrokeCube				
		TAbstractLayer3D	IAlignableObject, IAlignRoot				
			TCustomBufferLayer3D	TBufferLayer3D			
				TCustomLayer3D	IScene, IContainerObject		
					TLayer3D		
					TTextLayer3D		
			TImage3D				
			TLayout3D				
	TControl	IControl, IContainerObject, IAlignRoot,					

		IAlignableObject					
		TContent	TScrollContent				
		TGridLayout					
		TImage					
		TLayout	TCustomBindNavigator	TBindNavigator			
		TPaintBox					
		TPlotGrid					
		TScaledLayout					
		TSelection					
		TSelectionPoint					
		TShape	TCustomPath	TPath			
			TEllipse	TArc			
				TCircle			
				TPie			
			TLine				
			TRectangle	TCalloutRectangle			
			TRoundRect				
			TText				
		TStyledControl	TAniIndicator				
			TArcDial				
			TCalendar				
			TColorBox				
			TColorPanel				
			TColorPicker				

			TColorQuad				
			TColumn	TCheckColumn			
				TImageColumn			
				TPopupColumn			
				TProgressColumn			
				TStringColumn			
			TComboColorBox				
			TCustomTrack	TTrack			
				TTrackBar	TBitmapTrackBar	TAlphaTrackBar	
						TBWTrackBar	
						THueTrackBar	
			TGradientEdit				
			TImageControl	TImageCell			
			TPanel	TCalloutPanel			
			TPathLabel				
			TPopup				
			TProgressBar	TProgressCell			

			TScrollBar	TSmallScrollBar			
			TScrollBox	TFramedScrollBar			
				TImageViewer			
				TVerticalScrollBar	TFramedVerticalScrollBar		
				TMemo	IIMNotification		
				TCustomGrid	IItemsContainer		
					TGrid		
					TStringGrid		
				TCustomListBox	IItemsContainer		
					TComboBoxEditListBox		
					TComboBoxListBox		
					TListBox	TColorListBox	
				TCustomTreeView	IItemsContainer		
					TTreeView		
			TSplitter				
			TStatusBar				

				TCalendarBox			
				TCheckBox	TCheckCell		
				TCustomButton	TButton		
					TColorButton		
					TCustomCornerButton	TCornerButton	TBindNavButton
						THeaderItem	
					TExpanderButton		
					TPopupBox	TPopupCell	
					TSpeedButton		
			TTextControl	TDropTarget			
				TExpander			
				TGroupBox			
				TLabel			
				TListBoxItem			
				TRadioButton			
				TTabItem			
				TMenuItem	IItemsContainer		
				TTreeViewI	IItemsContainer		

				tem	er		
			TThumb				
			TToolBar				
			TCustomEdit	IIMENotification			
				TCalendarEdit			
				TClearingEdit			
				TComboEdit			
				TComboTrackerBar			
				TEdit	TTextCell		
				TNumberBox			
				TSpinBox			
			TCustomComboBox	IItemsContainer			
				TComboBox	TColorComboBox		
			THeader	IItemsContainer			
			TTabControl	IItemsContainer			
			TMenuBar	IMenuView, IItemsCon			

				tainer			
			TSizeGrip	ISizeGrip			
		TViewport3D	IViewport3D				
	TMainMenu	IItemsContainer					
	TCommonCustomForm	IRoot, IContainerObject, IAlignRoot					
		TCustomForm	IScene				
			TForm				
		TCustomForm3D	IViewport3D				
			TForm3D				

TImageFXEffect 的 57 个子类:

TAffineTransformEffect
 TBandedSwirlEffect
 TBandedSwirlTransitionEffect
 TBandsEffect
 TBlindTransitionEffect
 TBloodTransitionEffect
 TBloomEffect
 TBlurTransitionEffect
 TBoxBlurEffect
 TBrightTransitionEffect
 TCircleTransitionEffect
 TColorKeyAlphaEffect
 TContrastEffect
 TCropEffect
 TCrumpleTransitionEffect

TDirectionalBlurEffect
TDissolveTransitionEffect
TDropTransitionEffect
TEmbossEffect
TFadeTransitionEffect
TFillEffect
TFillRGBEffect
TGaussianBlurEffect
TGloomEffect
THueAdjustEffect
TInvertEffect
TLineTransitionEffect
TMagnifyEffect
TMagnifyTransitionEffect
TMaskToAlphaEffect
TMonochromeEffect
TNormalBlendEffect
TPaperSketchEffect
TPencilStrokeEffect
TPerspectiveTransformEffect
TPinchEffect
TPixelateEffect
TPixelateTransitionEffect
TRadialBlurEffect
TRippleEffect
TRippleTransitionEffect
TRotateCrumpleTransitionEffect
TSaturateTransitionEffect
TSepiaEffect
TShapeTransitionEffect
TSharpenEffect
TSlideTransitionEffect
TSmoothMagnifyEffect
TSwirlEffect

TSwirlTransitionEffect
TTilerEffect
TToonEffect
TWaterTransitionEffect
TWaveEffect
TWaveTransitionEffect
TWiggleTransitionEffect
TWrapEffect

TFilterBaseFilter 的 57 个子类:

TFilterAffineTransform
TFilterBandedSwirl
TFilterBandedSwirlTransition
TFilterBands
TFilterBlindTransition
TFilterBloodTransition
TFilterBloom
TFilterBlurTransition
TFilterBoxBlur
TFilterBrightTransition
TFilterCircleTransition
TFilterColorKeyAlpha
TFilterContrast
TFilterCrop
TFilterCrumpleTransition
TFilterDirectionalBlur
TFilterDissolveTransition
TFilterDropTransition
TFilterEmboss
TFilterFadeTransition
TFilterFill
TFilterFillRGB

TFilterGaussianBlur
TFilterGloom
TFilterHueAdjust
TFilterInvert
TFilterLineTransition
TFilterMagnify
TFilterMagnifyTransition
TFilterMaskToAlpha
TFilterMonochrome
TFilterNormalBlend
TFilterPaperSketch
TFilterPencilStroke
TFilterPerspectiveTransform
TFilterPinch
TFilterPixelate
TFilterPixelateTransition
TFilterRadialBlur
TFilterRipple
TFilterRippleTransition
TFilterRotateCrumpleTransition
TFilterSaturateTransition
TFilterSepia
TFilterShapeTransition
TFilterSharpen
TFilterSlideTransition
TFilterSmoothMagnify
TFilterSwirl
TFilterSwirlTransition
TFilterTiler
TFilterToon
TFilterWaterTransition
TFilterWave
TFilterWaveTransition
TFilterWiggleTransition

TFilterWrap

Delphi XE2 之 FireMonkey 入门(20) - TStyleBook(皮肤、样式相关)

我觉得叫 "皮肤" 不如叫 "样式" 或 "风格", 因为它可以包含和动作关联的动画。

在 FMX 下, 控件可以任意绘制, 各部件个性化的属性可以统一保存成一个 *.style 文件。

XE2 在 "...\\Program Files\\Embarcadero\\RAD Studio\\9.0\\Redist\\styles\\Fmx\\" 下提供了如下样式文件:

Air.Style
Amakrits.Style
AquaGraphite.style
Blend.Style
dark.style
FMX.Platform.iOS.style
FMX.Platform.Mac.style
FMX.Platform.Win.style
GoldenGraphite.Style
iOS.Style
MacBlue.Style
MacGraphite.Style
RubyGraphite.style
Windows7.Style

文件是文本格式, 类似窗体文件, 可用 TStyleBook 读入并管理。

不过要给程序套用样式, 确简单得很:

```
procedure TForm1.Button1Click(Sender: TObject);  
begin
```

```
Application.StyleFileName := '样式文件名'; //如果样式文件和 exe 同目录,
还可以省略路径
end;
```

在窗体上添加一个 TStyleBook(StyleBook1), 可以载入、编辑、另存这些样式。
编辑 StyleBook1 后, 可以把它直接赋给窗体的 StyleBook 属性:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Self.StyleBook := StyleBook1; //只作用于当前窗体
end;
```

窗体的 StyleBook 属性既然也是 TStyleBook 对象, 可以直接使用它:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    StyleBook := TStyleBook.Create(Self); //默认情况下, 窗体的 StyleBook
    还没有建立
    StyleBook.FileName := '样式文件';      //或者用下一行
    //StyleBook.Resource.LoadFromFile('样式文件'); //TStyleBook 是用 Re
    source(TStrings) 储存数据的
end;
```

单独修改控件样式可以使用 StyleLookup 属性, 如:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Button1.StyleLookup := 'checkbox';
```

end;

//在设计时可从控件的右键菜单

*.style 文件中 StyleName 命名有约定(这是我猜的):

1、类名 (去掉前面的 T) + 'style' //这是独立控件的样式名称

2、类名 (去掉前面的 T) //这是控件子部件的样式名称

3、既然有了命名约定, 控件可以根据自己的类名去套用, 所以会发现大多数控件的 StyleLookup 属性并无赋值

4、修改控件或子部件的样式都应该通过 StyleLookup 属性, 而不是 StyleName(我都觉得 StyleName 这个属性有点多余)。

我倒是发现控件子部件的 StyleName 都可以读的出来:

procedure TForm1.Button1Click(Sender: TObject);

begin

 ShowMessage (Memo1.HScrollBar.StyleName);

 ShowMessage (Memo1.VScrollBar.StyleName);

end;

其它诸如 ApplyStyleLookup()、UpdateStyle()、FindStyleResource() 等相关方法, 一般都是被自动调用的。

程序会把默认样式嵌入到资源中(名称: defaultstyle, 格式: RT_RCDATA), 恢复程序的默认样式时只需:

procedure TForm1.Button1Click(Sender: TObject);

begin

```
Application.StyleFileName := ''; //程序会自动套用默认样式
```

```
Form1.StyleBook := nil; //如果窗体单独设置了样式，这样恢复
```

```
end;
```

Delphi XE2 之 FireMonkey 入门(21) - 和 FMX 相关的类(表)

TO bje ct	TPersistent	TComponent	IInterface, IInterfaceComp onentReference				
			TBasicAction	TControlActio nLink			
			TApplication				
			TBindNavigator Controller				
			TPlatform	TPlatformCoo a			
			TScreen				
			TFmxObject	参见 TFmxObj ect 的子类们 (表)			
			TBasicBindCom ponent -> TContainedB indComponent -> -> TBaseBin	TBaseBindDBF ieldLink	TBaseBindDBF ieldControlLink	TCustomBind DBCheckLink TCustomBind DBEditLink	TBindDB CheckLin k TBindDB EditLink

			dDBControlLink			TCustomBindDBImageLink	TBindDBImageLink
						TCustomBindDBListLink	TBindDBListLink
				TCustomBindDBMemoLink	TBindDBMemoLink		
				TCustomBindDBTextLink	TBindDBTextLink		
		TBaseBindDBGridLink		ICustomDBGrid			
				TBaseBindDBGridControlLink	TCustomBindDBGridLink	TBindDBGridLink	
		TBitmapCodec	TBitmapCodecCocoa				
			TBitmapCodecQuartz				
		TBounds					
		TCanvasSaveState	TCocoaCanvasSaveState				
			TQuartzCanvasSaveState				
		TFilter	TShaderFilter	TShaderFilter -> 57 ↑			
		TFont					
		TGradient					

		TIndexBuffer					
		TMaterial					
		TMeshData					
		TModelImportSe rvices					
		TPosition					
		TPosition3D					
		TTransform					
		TVertexBuffer					
		TAbstractPrinter	TPrinter	TPrinterWin			
		TStrings	TListBoxStrings				
		TWideStrings	TListBoxStrings				
		TInterfacedPersi stent	IInterface				
			TBrushBitmap				
			TBrushGrab	IFreeNotificati on			
			TBrushResource	IFreeNotificati on			
			TCanvas	IFreeNotificati on			
				TCanvasCocoa			
				TCanvasQuart z			
				TPrinterCanva s			
			TContext3D	IFreeNotificati on			

				TContextOpen GL			
				TCustomDirectXContext			
			TPathData	IFreeNotification			
			TBitmap	IStreamPersist, IFreeNotification			
				TGEBitmap			
		TCollection	TGradientPoints				
			TKeys				
			TBaseDBGridLinkColumns	TDBGridLinkColumns			
		TCollectionItem	TGradientPoint				
			TKey	TColorKey			
				TFloatKey			
			TBaseDBGridLinkColumn	TDBGridLinkColumn			
	TList	TOrderedList	TStack	TEditActionStack			
	TSpline						
	TInterfacedObject	IInterface					
		TBindDBStringGridColumnCreator	IBindDBGridLinkControlManager				

		TModelImporter	IModelImporter				
			TASEModelImporter				
			TDAEModelImporter				
			TOBJModelImporter				
		TBindListEditor	IBindListEditor				
			TBindGridEditor	IBindGridEditor			
				TBindListGridEditor			
				TBindListStringGridEditor			
				TBindListComboBoxEditor			
				TBindListListBoxEditor			
		TBindCheckBoxEditor	IBindCheckBoxEditor				
			TBindStateCheckBoxEditor				
	TCustomModel	TASEModel					
		TDAEModel					
		TOBJModel					
	TGEMaterial	TASEMaterial					

	TBindDBColumnFactory	TBindDBStringGridColumnFactory					
	Exception	EASELexerError					
		EASEParserError					
		EBindNavException					
		EDAError					
		EInvalidFmxHandle					
		EPrinter					

TShaderFilter 的 57 个子类:

TAffineFilter
 TBandedSwirlFilter
 TBandedSwirlTransition
 TBandsFilter
 TBlindTransition
 TBloodTransition
 TBloomFilter
 TBlurFilter
 TBlurTransition
 TBrightTransition
 TCircleTransition
 TColorKeyAlphaFilter
 TContrastFilter
 TCropFilter
 TCrumpleTransition
 TDirectionalBlurFilter
 TDissolveTransition
 TDropTransition
 TEmbossFilter

TFadeTransition
TFillFilter
TFillOpaqueFilter
TGaussianBlurFilter
TGloomFilter
THueAdjustFilter
TInvertFilter
TLineTransition
TMagnifyFilter
TMagnifyTransition
TMaskToAlphaFilter
TMonochromeFilter
TNormalBlendFilter
TPaperSketchFilter
TPencilStrokeFilter
TPerspectiveFilter
TPinchFilter
TPixelateFilter
TPixelateTransition
TRippleFilter
TRippleTransition
TRotateCrumpleTransition
TSaturateTransition
TSepiaFilter
TShapeTransition
TSharpenFilter
TSlideInTransition
TSmoothMagnifyFilter
TSwirlFilter
TSwirlTransition
TTilerFilter
TToonFilter
TWaterTransition
TWaveFilter

TWaveTransition
TWiggleTransition
TWrapFilter
TZoomBlurFilter

Delphi XE2 之 FireMonkey 入门(22) - 数据绑定: BindingSource、BindingName、FindBinding()、Binding[]

在窗体上添加 TrackBar1、Edit1、Label1, 然后设置属性(可在设计时):

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    Edit1.BindingSource := TrackBar1; //将 TrackBar1 的值绑定在 Edit1  
    Label1.BindingSource := TrackBar1; //将 TrackBar1 的值绑定在 Label1  
end;  
  
{之后, Edit1.Text 和 Label1.Text 会同步 TrackBar1.Value 值}
```

不同类型值的沟通, 我想应该是依赖于 FM 组件新增的 Data 属性, 它是 Variant 类型的:

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    ShowMessage(TrackBar1.Data); //TrackBar1.Data <- TrackBar1.Value  
    ShowMessage(Label1.Data);    //Label1.Data <- Label1.Text  
    ShowMessage(Edit1.Text);     //Edit1.Data <- Edit1.Text  
end;
```

有些控件没有 BindingSource 属性(如 TButton、TPanel), 是因为它没必要有(但在使用专用绑定控件

时可以).

两个控件互为数据源后, 就可以互通数据了:

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    Edit1.BindingSource := Edit2;  
    Edit2.BindingSource := Edit1;  
end;
```

如果给拥有 BindingSource 属性的控件指定了 BindingName ...:

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    Edit1.BindingName := 'bn1'; //  
  
    {可以通过其上层控件的 FindBinding() 方法找到它}  
    ShowMessage(Self.FindBinding('bn1').ClassName);  
    ShowMessage(TEdit(FindBinding('bn1')).Text);  
  
    {可以通过其上层控件的 Binding[] 属性读取或修改其 Data 值}  
    Self.Binding['bn1'] := 'NewText';  
    ShowMessage(Self.Binding['bn1']); //NewText  
end;
```

XE2 的绑定功能(不仅仅在 FM)比我之前的想象要强大得多, 还有 TBindingsList、TBindingsScope、TBindScopeDB、TBindNavigator ...

Delphi XE2 之 FireMonkey 入门(23) - 数据绑定: TBindingsList: TBindExpression

准备用 TBindingsList 重做上一个例子。

可以先把 TBindingsList 理解为是一组绑定表达式(TBindExpression)的集合；

官方应该是提倡在设计时完成 TBindExpression 的建立与参数设置，但我觉得看运行时的代码会更容易理解。

- 1、先在窗体上添加 TrackBar1、Edit1、Label1、BindingsList1；
 - 2、激活窗体和 TrackBar1 的默认事件；
 - 3、然后将通过代码把 TrackBar1.Value 分别绑定到 Edit1.Text、Label1.Text, (这里把 TrackBar1 做源控件，把 Edit1、Label1 做目标控件)。
-

```
unit Unit1;
```

```
interface
```

```
uses
```

```
    System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,
```

```
    FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs, Data.Bind.EngExt,
```

```
    Fmx.Bind.DBEngExt, Data.Bind.Components, FMX.Edit;
```

```
type
```

```
    TForm1 = class (TForm)
```

```
        TrackBar1: TTrackBar;
```

```
        Edit1: TEdit;
```

```
        Label1: TLabel;
```

```
        BindingsList1: TBindingsList;
```

```
        procedure FormCreate(Sender: TObject);
```

```
        procedure TrackBar1Change(Sender: TObject);
```

```
    end;
```

```
var
```

```
Form1: TForm1;
```

implementation

```
{ $R *.fmx }
```

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
```

```
{给 BindingsList1 添加表达式, 并设置参数}
```

```
  with TBindExpression.Create(BindingsList1) do
```

```
  begin
```

```
    ControlComponent := Edit1;      // 目标控件
```

```
    ControlExpression := 'Text'; //目标控件属性
```

```
    SourceComponent := TrackBar1; //源控件
```

```
    SourceExpression := 'Value'; //源控件属性
```

```
    Active := True;                  //激活; 可以通过 Active := False 断开
```

```
    绑定
```

```
  end;
```

```
  with TBindExpression.Create(BindingsList1) do
```

```
  begin
```

```
    ControlComponent := Label1;
```

```
    ControlExpression := 'Text';
```

```
    SourceComponent := TrackBar1;
```

```
    SourceExpression := 'Value';
```

```
    Active := True;
```

```
  end;
```

```
end;
```

```
{在 TrackBar1 的 OnChange 中发送通知}
```

```
procedure TForm1.TrackBar1Change(Sender: TObject);
```

```
begin
```

```
BindingsList1.Notify(Sender, ''); //这里也可以写作: BindingsList1.N  
otify(TrackBar1, 'Value');  
end;  
  
end.
```

下面尝试在设计时完成以上工作:

- 1、先在窗体上添加 TrackBar1、Edit1、Label1;
 - 2、从 Edit1 的右键菜单 New LiveBinding...(从 Strucure 窗口 Edit1 的右键菜单、或从 Object Inspector 窗口的 LiveBindings 添加均可);
 - 3、确认添加 TBindingExpression, 其默认名称会是: BindExpressionEdit11, (此时也会有 BindingsList1 被自动添加);
 - 4、选定刚刚添加的 BindExpressionEdit11, 设置属性:
ControlComponent : 'Edit1'
ControlExpression: 'Text'
SourceComponent : 'TrackBar1'
SourceExpression : 'Value'
 - 5、重复步骤 2..4 同样设置 Label1
 - 6、在 TrackBar1 的 OnChange 事件中写上一句: BindingsList1.Notify(Sender, "");
 - 7、运行.
-

使用 System.Bindings.Helper 单元中 TBindings 类的静态方法实现如上绑定(先在窗体上添加 TrackBar1、Edit1、Label1):

```
unit Unit1;
```

```
interface
```

```
uses
```

```
System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,  
FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs, FMX.Edit;
```

type

```
TForm1 = class(TForm)  
    TrackBar1: TTrackBar;  
    Edit1: TEdit;  
    Label1: TLabel;  
    procedure FormCreate(Sender: TObject);  
    procedure TrackBar1Change(Sender: TObject);  
end;
```

var

```
Form1: TForm1;
```

implementation

```
{ $R *.fmx }
```

```
uses System.Bindings.Expression, System.Bindings.Helper;
```

```
procedure TForm1.FormCreate(Sender: TObject);
```

begin

```
TBindings.CreateManagedBinding(  
    [TBindings.CreateAssociationScope([Associate(TrackBar1, 'Input1'  
'))]],  
    'Input1.Value',  
    [TBindings.CreateAssociationScope([Associate(Edit1, 'Output1'  
'))]],  
    'Output1.Text',  
    nil  
); //其中的 Input1、Output1 都是随意的，前后一致即可
```

```
TBindings.CreateManagedBinding(  
  [TBindings.CreateAssociationScope([Associate(TrackBar1, 'Input1'  
'')]),  
    'Input1.Value',  
    [TBindings.CreateAssociationScope([Associate(Label1, 'Output1'  
'')]),  
      'Output1.Text',  
      nil  
    ]  
  ]  
);  
end;  
  
procedure TForm1.TrackBar1Change(Sender: TObject);  
begin  
  TBindings.Notify(Sender, '');  
end;  
  
end.
```

Delphi XE2 之 FireMonkey 入门 (24) - 数据绑定: TBindingsList: TBindExpression.Direction

在学习 BindingSource 属性时, 可以让两个控件互为绑定源; TBindExpression 对应的功能是 Direction 属性.

先在窗体上添加 Edit1、Edit2、BindingsList1; 然后激活 Edit1、Edit2 和窗体的默认事件.

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
  with TBindExpression.Create(BindingsList1) do  
    begin  
      ControlComponent := Edit2;
```

```
ControlExpression := 'Text';
SourceComponent := Edit1;
SourceExpression := 'Text';

// Direction := TExpressionDirection.dirSourceToControl; // 默认
// Direction := TExpressionDirection.dirControlToSource; // 反向

Direction := TExpressionDirection.dirBidirectional; // 双向

Active := True;
end;
end;

procedure TForm1.Edit1Change(Sender: TObject);
begin
    BindingsList1.Notify(Sender, '');
end;

procedure TForm1.Edit2Change(Sender: TObject);
begin
    BindingsList1.Notify(Sender, '');
end;
```

如果没有 `Direction` 属性, 代码或许会是:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    with TBindExpression.Create(BindingsList1) do
    begin
        ControlComponent := Edit2;
        ControlExpression := 'Text';
        SourceComponent := Edit1;
        SourceExpression := 'Text';
        Active := True;
```

```
end;

with TBindExpression.Create(BindingsList1) do
begin
    ControlComponent := Edit1;
    ControlExpression := 'Text';
    SourceComponent := Edit2;
    SourceExpression := 'Text';
    Active := True;
end;
end;

procedure TForm1.Edit1Change(Sender: TObject);
begin
    BindingsList1.Notify(Sender, '');
end;

procedure TForm1.Edit2Change(Sender: TObject);
begin
    BindingsList1.Notify(Sender, '');
end;
```

Delphi XE2 之 FireMonkey 入门(25) - 数据绑定: TBindingsList: 表达式的灵活性及表达式函数

绑定表达式中可以有简单的运算和字符串连接, 但字符串需放在双引号中.

还可以使用 TBindingsList.Methods 提供的一组表达式函数(分别来自 System.Bindings.Methods 和 Data.Bind.EngExt 单元):

```
ToStr()
ToVariant()
Round()
```



```
Format()  
UpperCase()  
LowerCase()  
FormatDateTime()  
StrToDateTime()  
Max()  
Min()  
CheckedState()  
SelectedItem()  
SelectedText()
```

示例：用三个 TLabel 分别呈现窗体的宽度、高度、面积。

现在窗体上添加 Label1、Label2、Label3、BindingsList1，并激活窗体的 OnCreate 和 OnResize 事件：

```
unit Unit1;
```

```
interface
```

```
uses
```

```
    System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,  
    FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs, Data.Bind.EngExt,  
    Fmx.Bind.DBEngExt, Data.Bind.Components;
```

```
type
```

```
    TForm1 = class(TForm)  
        Label1: TLabel;  
        Label2: TLabel;  
        Label3: TLabel;
```

```
BindingsList1: TBindingsList;  
procedure FormCreate(Sender: TObject);  
procedure FormResize(Sender: TObject);  
end;
```

var

```
Form1: TForm1;
```

implementation

```
{ $R *.fmx }
```

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
```

```
  with TBindExpression.Create(BindingsList1) do
```

```
  begin
```

```
    ControlComponent := Label1;
```

```
    ControlExpression := 'Text';
```

```
    SourceComponent := Form1;
```

```
    SourceExpression := '"宽度: " + ToString(Width)';
```

```
    Active := True;
```

```
  end;
```

```
  with TBindExpression.Create(BindingsList1) do
```

```
  begin
```

```
    ControlComponent := Label2;
```

```
    ControlExpression := 'Text';
```

```
    SourceComponent := Form1;
```

```
//    SourceExpression := '"高度: " + ToString(Height)';
```

```
    SourceExpression := 'Format("高度: %s", ToString(Height))'; //同上一
```

行; 在表达式中使用 `Format` 函数时, 后面的参数不能放在 `[]` 中

```
    Active := True;
```

```
end;

with TBindExpression.Create(BindingsList1) do
begin
  ControlComponent := Label3;
  ControlExpression := 'Text';
  SourceComponent := Form1;

  SourceExpression := '"面积: " +ToStr(Width * Height)';
  Active := True;
end;
end;

procedure TForm1.FormResize(Sender: TObject);
begin
  BindingsList1.Notify(Sender, 'Width');
  BindingsList1.Notify(Sender, 'Height');
end;

end.
```

在表达式中还可以使用关键字 Self、Owner.

Delphi XE2 之 FireMonkey 入门(26) - 数据绑定: TBindingsList: TBindExprItems

如果要给一对 "源控件" 和 "目标控件" 写多个表达式, 使用 TBindExpression 就不如 TBindExprItems 了.

TBindExprItems 中的表达式又分两组: FormatExpressions、ClearExpressions, 后者是在断开绑定时的表达式.

示例设想:

- 1、TrackBar1 为源, 把其 Value 值绑定给 Edit1.Text 和 Edit1.Width 和 Label1.Text;
- 2、断开绑定时, 在 Edit1 中显示 "已断开绑定".

先在窗体上添加: TrackBar1、Edit1、Label1、BindingsList1、CheckBox1, 然后激活 TrackBar1、ChengkBox1 和窗体的默认事件:

```
unit Unit1;
```

```
interface
```

```
uses
```

```
    System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,  
    FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs, Data.Bind.EngExt,  
    Fmx.Bind.DBEngExt, Data.Bind.Components, FMX.Edit;
```

```
type
```

```
    TForm1 = class(TForm)  
        TrackBar1: TTrackBar;  
        Edit1: TEdit;  
        Label1: TLabel;  
        BindingsList1: TBindingsList;  
        CheckBox1: TCheckBox;  
        procedure FormCreate(Sender: TObject);  
        procedure TrackBar1Change(Sender: TObject);  
        procedure CheckBox1Change(Sender: TObject);  
    end;
```

```
var
```

```
    Form1: TForm1;
```

implementation

```
{ $R *.fmx }
```

var

```
bindExprItems: TBindExprItems;
```

```
procedure TForm1.FormCreate(Sender: TObject);
```

begin

```
bindExprItems := TBindExprItems.Create(BindingsList1);
```

```
with bindExprItems do
```

begin

```
ControlComponent := Edit1;    //目标控件
```

```
SourceComponent := TrackBar1; //源控件
```

```
{把 TrackBar1.Value 绑定到 Edit1.Text}
```

```
with FormatExpressions.AddExpression do
```

begin

```
ControlExpression := 'Text';
```

```
SourceExpression := 'Value';
```

```
end;
```

```
{把 TrackBar1.Value 绑定到 Edit1.Width}
```

```
with FormatExpressions.AddExpression do
```

begin

```
ControlExpression := 'Width';
```

```
SourceExpression := 'Value';
```

```
end;
```

```
{把 TrackBar1.Value 绑定到 Label1.Text}
```

```
with FormatExpressions.AddExpression do
```

begin

```
ControlExpression := 'Owner.Label1.Text'; //Owner
```

```
SourceExpression := 'Value';
```

```
end;

{ 当要断开绑定时... }

with ClearExpressions.AddExpression do
begin
    ControlExpression := 'Self.Text'; //这里的 Self 是指目标控件本身,
    可省略
    SourceExpression := '"已断开绑定"';
end;
Active := True;
end;
CheckBox1.IsChecked := True;
TrackBar1.Max := ClientWidth;
end;

procedure TForm1.TrackBar1Change(Sender: TObject);
begin
    BindingsList1.Notify(Sender, 'Value');
end;

procedure TForm1.CheckBox1Change(Sender: TObject);
begin
    bindExprItems.Active := TCheckBox(Sender).IsChecked;
end;

end.
```

Delphi XE2 之 FireMonkey 入门(27) - 数据绑定: TBindingsList: TBindScope

如果在编写表达式时, 如果能够随意指认需要的控件就好了(通过 Owner 也可以勉强做到), TBindScope 就是解决这个问题的.

示例设想：把三个 TEdit 的 Text 绑定到一个 TLabel.

在窗体上添加 Label1、Edit1、Edit2、Edit3、BindingsList1、BindScope1; 激活 Edit1 和窗体的默认事件.

```
unit Unit1;
```

```
interface
```

```
uses
```

```
    System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,  
    FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs, Data.Bind.EngExt,  
    Fmx.Bind.DBEngExt, Data.Bind.Components, FMX.Layouts, FMX.Edit;
```

```
type
```

```
    TForm1 = class(TForm)  
        Edit1: TEdit;  
        Edit2: TEdit;  
        Edit3: TEdit;  
        Label1: TLabel;  
        BindingsList1: TBindingsList;  
        BindScope1: TBindScope;  
        procedure FormCreate(Sender: TObject);  
        procedure Edit1Change(Sender: TObject);  
    end;
```

```
var
```

```
    Form1: TForm1;
```

```
implementation
```

```
{ $R *.fmx }
```

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    with TBindExpression.Create(BindingsList1) do  
        begin  
            ControlComponent := Label1;  
            ControlExpression := 'Text';  
  
            SourceComponent := BindScope1; //把 BindScope1 指定为源组件, 之后可  
            以在表达式中直接使用控件名  
            SourceExpression := 'Format("%s,%s,%s", Edit1.Text, Edit2.Text,  
            Edit3.Text)';  
            Active := True;  
        end;  
        BindScope1.Active := True; //  
        Edit2.OnChange := Edit1.OnChange;  
        Edit3.OnChange := Edit1.OnChange;  
end;  
  
procedure TForm1.Edit1Change(Sender: TObject);  
begin  
    BindingsList1.Notify(Sender, '');  
end;  
  
end.
```

Delphi XE2 之 FireMonkey 入门(28) - 数据绑定: TBindingsList: 表达式函数测试: SelectedText()、CheckedState()

示例构想：用 Label1 显示 ListBox1 的选项，用 Label2 显示 CheckBox1 的状态。

- 1、放控件：Label1、Label2、ListBox1、CheckBox1、BindingsList1、BindScope1；
 - 2、激活 ListBox1 的 OnClick 事件和窗体的默认事件。
-

```
unit Unit1;
```

```
interface
```

```
uses
```

```
    System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,  
    FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs, Data.Bind.EngExt,  
    Fmx.Bind.DBEngExt, Data.Bind.Components, FMX.Layouts, FMX.ListBox;  
;
```

```
type
```

```
    TForm1 = class(TForm)  
    Label1: TLabel;  
    Label2: TLabel;  
    ListBox1: TListBox;  
    CheckBox1: TCheckBox;  
    BindingsList1: TBindingsList;  
    BindScope1: TBindScope;  
    procedure FormCreate(Sender: TObject);  
    procedure ListBox1Click(Sender: TObject);  
end;
```

```
var
```

```
    Form1: TForm1;
```

```
implementation
```

```
{ $R *.fmx }
```

```
uses Fmx.Bind.Editors; //使用表达式函数 SelectedText、SelectedItem、CheckedState 时，需要支持单元
```

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
var
```

```
    i: Integer;
```

```
begin
```

```
    for i := 1 to 9 do
```

```
        ListBox1.Items.Add('Item_' + IntToStr(i));
```

```
    with TBindExpression.Create(BindingsList1) do
```

```
    begin
```

```
        ControlComponent := Label1;
```

```
        ControlExpression := 'Text';
```

```
        SourceComponent := BindScope1;
```

```
        SourceExpression := 'SelectedText(ListBox1)'; //同下一行
```

```
//    SourceExpression := 'ListBox1.Selected.Text';
```

```
        Active := True;
```

```
    end;
```

```
    with TBindExpression.Create(BindingsList1) do
```

```
    begin
```

```
        ControlComponent := Label2;
```

```
        ControlExpression := 'Text';
```

```
        SourceComponent := BindScope1;
```

```
        SourceExpression := 'CheckedState(CheckBox1)'; //Checked: 'True'; Unchecked: 'False'; Grayed: ''
```

```
//    SourceExpression := 'CheckBox1.IsChecked'; //在本例中，这等同于上一行
```

```
    Active := True;
end;

CheckBox1.OnClick := ListBox1.OnClick;
end;

procedure TForm1.ListBox1Click(Sender: TObject);
begin
    BindingsList1.Notify(Sender, '');
end;

end.
```

Delphi XE2 之 FireMonkey 入门(29) - 数据绑定: TBindingsList: 表达式的 Evaluate() 方法

TBindingsList 中可能不止一个表达式, 通过表达式的 Evaluate 方法可单独提交绑定, 并可在 Active = False 时提交.

在 TBindExprItems 中对应的方法是 EvaluateFormat.

测试设想: Label1、Label2 的绑定源同是 Edit1, 分别提交绑定.

- 1、在窗体上加控件: Label1、Label2、Edit1、BindingsList1;
 - 2、激活 Edit1 的 OnKeyUp、OnChange 事件, 还有窗体的默认事件.
-

```
unit Unit1;
```

```
interface
```

```
uses
```

```
System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,  
FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs, FMX.Edit, Data.Bind.EngExt,  
FMX.Bind.DBEngExt, Data.Bind.Components;
```

type

```
TForm1 = class(TForm)  
    Label1: TLabel;  
    Label2: TLabel;  
    Edit1: TEdit;  
    BindingsList1: TBindingsList;  
    procedure FormCreate(Sender: TObject);  
    procedure Edit1KeyUp(Sender: TObject; var Key: Word; var KeyChar: Char; Shift: TShiftState);  
    procedure Edit1Change(Sender: TObject);  
end;
```

var

```
Form1: TForm1;
```

implementation

```
{ $R *.fmx }
```

var

```
bindExpression1, bindExpression2: TBindExpression;
```

```
procedure TForm1.FormCreate(Sender: TObject);
```

begin

```
    bindExpression1 := TBindExpression.Create(BindingsList1);
```

```
    bindExpression2 := TBindExpression.Create(BindingsList1);
```

```
    with bindExpression1 do
```

```
begin
    ControlComponent := Label1;
    ControlExpression := 'Text';
    SourceComponent := Edit1;
    SourceExpression := 'Text';
    // Active := True;
end;

with bindExpression2 do
begin
    ControlComponent := Label2;
    ControlExpression := 'Text';
    SourceComponent := Edit1;
    SourceExpression := 'UpperCase(Text)';
    // Active := True;
end;
end;

procedure TForm1.Edit1KeyUp(Sender: TObject; var Key: Word; var KeyC
har: Char; Shift: TShiftState);
begin
    bindExpression1.Evaluate;
end;

procedure TForm1.Edit1Change(Sender: TObject);
begin
    bindExpression2.Evaluate;
end;

end.
```

Delphi XE2 之 FireMonkey 入门(30) - 数据绑定: TBindingsList: TBindExpression 的 OnAssigningValue 事件

表达式中的函数有限, 譬如我想通过绑定输出文本的长度(譬如在 Label1 中绑定输出 Edit1.Text 的长度)就没有相应的函数;

这可在 TBindExpression 的 OnAssigningValue 事件中处理.

TBindExpression 和它的兄弟们 (TBindExprItems、TBindLink、TBindListLink、TBindGridLink、TBindPosition、TBindList、TBindGridList、TBindDBEditLink、TBindDBTextLink、TBindDBListLink、TBindDBImageLink、TBindDBMemoLink、TBindDBCheckLink、TBindDBGridLink) 拥有相同的事件.

先在窗体上添加 Label1、Edit1、BindingsList1, 然后激活 Edit1 的 OnKeyUp 事件和窗体的默认事件;

代码中手动完成了 OnAssigningValue 事件, 在设计时添加事件会更方便.

```
unit Unit1;

interface

uses
  System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants, FMX.Types,
  FMX.Controls, FMX.Forms, FMX.Dialogs, Data.Bind.EngExt, Fmx.Bind.DBEngExt, System.Rtti,
  System.Bindings.Outputs, FMX.Layouts, Data.Bind.Components, FMX.Edit;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
```

```
    BindingsList1: TBindingsList;
    procedure FormCreate(Sender: TObject);
    procedure Edit1KeyUp(Sender: TObject; var Key: Word; var KeyChar:
    Char; Shift: TShiftState);
    procedure MyOnAssigningValue(Sender: TObject; AssignValueRec: TB
    indingAssignValueRec; var Value: TValue; var Handled: Boolean);
    end;

var
    Form1: TForm1;

implementation

{$R *.fmx}

procedure TForm1.FormCreate(Sender: TObject);
begin
    with TBindExpression.Create(BindingsList1) do
    begin
        ControlComponent := Label1;
        ControlExpression := 'Text';
        SourceComponent := Edit1;
        SourceExpression := 'Text';
        OnAssigningValue := MyOnAssigningValue; //
        Active := True;
    end;
end;

procedure TForm1.Edit1KeyUp(Sender: TObject; var Key: Word; var KeyC
    har: Char; Shift: TShiftState);
begin
    BindingsList1.Notify(Sender, '');
end;
```

```
procedure TForm1.MyOnAssigningValue(Sender: TObject; AssignValueRec:
  TBindingAssignValueRec; var Value: TValue; var Handled: Boolean);
begin
  Value := Length(Value.ToString); //
end;

end.
```

Delphi XE2 之 FireMonkey 入门(31) - 数据绑定: 绑定数据库

一、全设计时操作:

先在窗体上放置控件:

```
DataSource1      : TDataSource;
ClientDataSet1   : TClientDataSet;
Label1           : TLabel;
Edit1            : TEdit;
Memo1            : TMemo;
ImageControl1    : TImageControl;
BindNavigator1   : TBindNavigator;
```

{ 在连接过程中, 会自动添加下面部件 }

```
BindingsList1      : TBindingsList;
BindScopeDB1       : TBindScopeDB;

DBLinkLabel1SpeciesNo1 : TBindDBTextLink;
DBLinkEdit1Category1  : TBindDBEditLink;
DBLinkMemo1Notes1     : TBindDBMemoLink;
DBLinkImageControl1Graphic1 : TBindDBImageLink;
```

测试使用官方提供的测试数据 biolife.xml(或 biolife.cds), 其路径通常是: C:\Program Files\Common Files\CodeGear Shared\Data\biolife.xml

连接步骤:

1、置 DataSource1 的 DataSet 属性为 ClientDataSet1;

2、置 ClientDataSet 的 FileName 属性为 'C:\Program Files\Common Files\CodeGear Shared\Data\biolife.xml';

3、将四个控件分别关联到 biolife.xml 中的字段:

Label1 -> 右键 -> Link To DB Field... -> 选择 Species No (数字)

Edit1 -> 右键 -> Link To DB Field... -> 选择 Category (**string**)

Memo1 -> 右键 -> Link To DB Field... -> 选择 Notes (Text)

ImageControl -> 右键 -> Link To DB Field... -> 选择 Graphic (Graphics)

4、置 BindNavigator1 的 BindScope 属性为 BindScopeDB1;

5、置 ClientDataSet1 的 Active 属性为 True.

二、运行时连接:

先在窗体上放置控件(其它在运行时完成):

```
DataSource1      : TDataSource;  
ClientDataSet1   : TClientDataSet;  
Label1           : TLabel;  
Edit1            : TEdit;  
Memo1            : TMemo;  
ImageControl1    : TImageControl;  
BindNavigator1   : TBindNavigator;  
BindingsList1    : TBindingsList;  
BindScopeDB1     : TBindScopeDB;
```

与设计时功能相同的代码:

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    ClientDataSet1.FileName := 'C:\Program Files\Common Files\CodeGear  
Shared\Data\biolife.xml';  
    DataSource1.DataSet := ClientDataSet1;  
    BindScopeDB1.DataSource := DataSource1;  
    BindNavigator1.BindScope := BindScopeDB1;  
  
    with TBindDBTextLink.Create(BindingsList1) do  
        begin  
            ControlComponent := Label1;  
            DataSource := BindScopeDB1;  
            FieldName := 'Species No';  
        end;  
  
    with TBindDBEditLink.Create(BindingsList1) do  
        begin
```

```
ControlComponent := Edit1;
DataSource := BindScopeDB1;
FieldName := 'Category';
end;

with TBindDBMemoLink.Create(BindingsList1) do
begin
ControlComponent := Memo1;
DataSource := BindScopeDB1;
FieldName := 'Notes';
end;

with TBindDBImageLink.Create(BindingsList1) do
begin
ControlComponent := ImageControll1;
DataSource := BindScopeDB1;
FieldName := 'Graphic';
end;

ClientDataSet1.Active := True;
end;
```

要绑定到 TStringGrid, 在设计时(在上面的基础上)只需: StringGrid1 -> Link to DB DataSource... -> BindScopeDB1

下面是运行时绑定到 TStringGrid 的代码:

```
uses Fmx.Bind.Editors, Data.Bind.DBLinks, Fmx.Bind.DBLinks;

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
ClientDataSet1.FileName := 'C:\Program Files\Common Files\CodeGear Shared\Data\biolife.xml';
DataSource1.DataSet := ClientDataSet1;
BindScopeDB1.DataSource := DataSource1;
BindNavigator1.BindScope := BindScopeDB1;
ClientDataSet1.Active := True;

with TBindDBGridLink.Create(BindingsList1) do //还可使用 TBindGridLink
begin
    GridControl := StringGrid1;
    DataSource := BindScopeDB1;
    Active := True;
end;
end;
```

Delphi XE2 之 FireMonkey 入门(32) - 数据绑定: TBindingsList: TBindList、TBindPosition [未完成...]

Delphi XE2 之 FireMonkey 入门(33) - 控件基础: TFmxObject: SaveToStream、LoadFromStream、SaveToBinStream、LoadFromBinStream

SaveToStream()、LoadFromStream() 对应文本流;

SaveToBinStream()、LoadFromBinStream() 对应二进制流; 相对文本流, 二进制流会小一些.

有这几个方法, 控件数据与状态的序列化就太容易了.

示例:

1、控件: ListBox1、Button1、Button2

2、事件: Button1.OnClick、Button2.OnClick、Form1.OnCreate

```
unit Unit1;
```

```
interface
```

```
uses
```

```
    System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,  
    FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs, FMX.Layouts, FMX.ListBox;
```

```
type
```

```
    TForm1 = class(TForm)  
        ListBox1: TListBox;  
        Button1: TButton;  
        Button2: TButton;  
        procedure FormCreate(Sender: TObject);  
        procedure Button1Click(Sender: TObject);  
        procedure Button2Click(Sender: TObject);  
    end;
```

```
var
```

```
    Form1: TForm1;
```

```
implementation
```

```
{ $R *.fmx }
```

```
{ 给 ListBox1 加点内容 }
```

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
var
```

```
    i: Integer;
```

```
begin
```

```
for i := 0 to 9 do
    ListBox1.Items.Add('Item' + IntToStr(i));
end;

{存取 ListBox1}

procedure TForm1.Button1Click(Sender: TObject);
const
    path = 'c:\temp\ListBox1Stream.txt';
var
    stream: TFileStream;
begin
    {1}
    stream := TFileStream.Create(path, fmCreate);
    ListBox1.SaveToStream(stream);
    stream.Free;

    {2}
    ListBox1.Clear;
    ShowMessage('a');

    {3}
    stream := TFileStream.Create(path, fmOpenRead);
    ListBox1.LoadFromStream(stream);
    stream.Free;
end;

{存取当前窗体}

procedure TForm1.Button2Click(Sender: TObject);
const
    path = 'c:\temp\Form1Stream.dat';
var
    stream: TFileStream;
begin
```

```
{1}
stream := TFileStream.Create('c:\temp\Form1Stream.dat', fmCreat
e);
Self.SaveToBinStream(stream);
stream.Free;

{2}
Button1.Free;
Button2.Free;
ListBox1.Free;
Realign;
ShowMessage('a');

{3}
stream := TFileStream.Create('c:\temp\Form1Stream.dat', fmOpenRea
d);
Self.LoadFromBinStream(stream);
stream.Free;
end;

end.
```

状态保存测试:

- 1、控件: Button1、Button2、Panel1; 在 Panel1 中再放三个 TCheckBox
 - 2、事件: Button1.OnClick、Button2.OnClick、Form1.OnCreate、Form1.OnDestroy
-

```
unit Unit1;
```

```
interface
```

```
uses
```

```
System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,  
FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs;
```

type

```
TForm1 = class(TForm)  
    Panel1: TPanel;  
    CheckBox1: TCheckBox;  
    CheckBox2: TCheckBox;  
    CheckBox3: TCheckBox;  
    Button1: TButton;  
    Button2: TButton;  
    procedure FormCreate(Sender: TObject);  
    procedure FormDestroy(Sender: TObject);  
    procedure Button2Click(Sender: TObject);  
    procedure Button1Click(Sender: TObject);  
end;
```

var

```
Form1: TForm1;
```

implementation

```
{ $R *.fmx }
```

var

```
stream: TMemoryStream;
```

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    stream := TMemoryStream.Create;  
    Button2.Enabled := False;  
end;
```



```
procedure TForm1.FormDestroy(Sender: TObject);  
begin  
    stream.Free;  
end;  
  
{ 调整三个 CheckBox 的状态后保存}  
  
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    Panel1.SaveToStream(stream);  
    ShowMessage(IntToStr(stream.Size));  
    Button2.Enabled := True;  
end;  
  
{ 恢复}  
  
procedure TForm1.Button2Click(Sender: TObject);  
begin  
    stream.Position := 0;  
    Panel1.LoadFromStream(stream);  
end;  
  
end.
```

Delphi XE2 之 FireMonkey 入门(34) - 控件基础: TFmxObject: 克隆对象

有两个和克隆相关的方法: Clone()、CloneChildFromStream().

Clone() 很好用, 但 CloneChildFromStream() 的源码很明显地写错了(是小问题, 谁外语好去报一下).

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
var
    i: Integer;
begin
    for i := 0 to 9 do
        ListBox1.Items.Add('Item' + IntToStr(i));
    end;

    {把 ListBox1 克隆到 Panel1}

    procedure TForm1.Button1Click(Sender: TObject);
    begin
        Panel1.AddObject(ListBox1.Clone(Self));
    end;
```

Delphi XE2 之 FireMonkey 入门(35) - 控件基础: TFmxObject: 其它

TFmxObject 增加了 TagObject、TagFloat、TagString, 算上从 TComponent 继承的 Tag, 可以暂存多种类型的数据了。

ChildrenCount、Children[] 代替了之前的 ControlCount、Controls[]。

有增删 Children 的几个方法: AddObject()、InsertObject()、RemoveObject()、Exchange()、DeleteChildren()

Index 属性是控件在 Children 队列中的序号, 给它赋值可调整控件的前后次序;

SendToBack()、BringToFront() 方法也会修改控件的 Index 值。

有个只读的 Root 属性, 但可通过 SetRoot() 方法赋值。

窗体是实现了 IRoot 接口的, 控件的 Root 属性就代表着它所在的窗体, 可测试下:

```
procedure TForm1.Button1Click(Sender: TObject);
```

begin

```
TForm(Button1.Root).Caption := 'abc';
```

end;

布尔属性 **Stored** 取代了原来的 **Stored** 关键字, 因为所有的对象都可以序列化了.

可通过 **IsIControl()** 判断是否是 **TControl**(它实现了 **IControl** 接口); 通过 **AsIControl()** 转换到 **IControl** 接口.

其它的(动画、**Binding** 等), 或者已探讨过, 或者感觉不重要了.

Delphi XE2 之 FireMonkey 入门(36) - 控件基础: TForm

当我第一次读取 **Form1.StyleLookup** 并期待出现 "formstyle" 时, 给的确是 "backgroundstyle"

...

现在明白了, 原来窗体上覆盖着一个 **TRectangle** 对象:

```
uses FMX.Objects;
```

```
procedure TForm1.Button1Click(Sender: TObject);
```

begin

```
(Self.Children[0] as TRectangle).Fill.Color := claRed; //默认情况下,
```

通过 **Children[0]** 可读出这个矩形, 但其次序(**Index**)是可变的

```
ShowMessage(Children[0].ToString); //TRectangle
```

```
{Panel1 上也是盖着一个矩形对象, 应该还有其它也是这样}
```

```
TRectangle(Panel1.Children[0]).Fill.Color := claYellow;
```

```
end;
```

现在的 FMX 窗体只有 8 个事件, 我回头数了数传统 VCL 窗体的事件有 40 个之多.

8 个是少了点, 但其上级有不少可以覆盖的虚方法, 如:

MouseDown、MouseUp、MouseMove、MouseLeave、MouseWheel、DragDrop、DragEnter、DragOver、DragLeave、KeyDown、KeyUp 等等.

它们在窗体中没有以事件的方式出现, 可能是设计者认为在窗体中很少用到这些事件.

测试: 通过覆盖 MouseMove 响应鼠标移动事件:

```
unit Unit1;
```

```
interface
```

```
uses
```

```
System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,
```

```
FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs;
```

```
type
```

```
TForm1 = class(TForm)
```

```
private
```

```
public
```

```
{ 在 TForm1 声明区 -> Ctrl+空格 -> 输入 MouseM... -> 回车 -> Shift+Ctrl+C }
```

```
procedure MouseMove(Shift: TShiftState; X: Single; Y: Single); o
```

```
override;
```

```
end;
```

var

Form1: TForm1;

implementation

{ \$R *.fmx }

procedure TForm1.MouseMove (Shift: TShiftState; X, Y: Single);

begin

inherited;

 Caption := Format('%f, %f', [X, Y]);

end;

end.

TForm 的继承链: TFmxObject -> TCommonCustomForm -> TCustomForm -> TForm

TCommonCustomForm 提出或实现了 TForm 和 TForm3D 的公有功能;

TCustomForm 通过实现 IScene 接口, 主要提供了 Canvas 和 Style 相关的功能;

TForm 没有功能扩展, 只是把部分成员改在 published 下以便在设计时可用.

{ TCommonCustomForm }

public

constructor Create(...); **override**; //

constructor CreateNew(...); **virtual**; //它比 Create() 多出一个默认参数

Dummy, 但没看出什么用途(可能是给 3D 窗口用的吧); 暂时它和 Create() 没什么区别

destructor Destroy; **override**; //

procedure InitializeNewForm; **virtual**; //窗体初始化, 可通过覆盖此方法

初始化窗体属性

procedure AfterConstruction; **override**; //建立后

```

procedure BeforeDestruction; override;    //销毁前

procedure AddObject(...); override;        //添加对象

{ children }

function ObjectAtPoint(...): IControl; virtual; //指定位置上的控件

{ called from Platform }

procedure SetMarkedText(...); virtual; //使用输入法时, 确认输入前的文

```

本; 覆盖此方法可截获

```

procedure PaintRects(...); virtual;        //更新指定的矩形(数组)范围; TC
CustomForm 里才有它的实现代码

```

```

procedureMouseDown(...); virtual;          //响应鼠标事件
procedureMouseMove(...); virtual;          //...
procedureMouseUp(...); virtual;             //...
procedureMouseWheel(...); virtual;          //...
procedureMouseLeave; virtual;                //...

procedureKeyDown(...); virtual;             //响应键盘事件
procedureKeyUp(...); virtual;               //...

function GetImeWindowRect: TRectF; virtual; //获取拥有输入焦点对象的

```

范围矩形

```

procedure Activate;                          //激活

procedure Deactivate;                        //取消激活

procedure DragEnter(...); virtual;          //响应拖放事件
procedure DragOver(...); virtual;           //...
procedure DragDrop(...); virtual;           //...
procedure DragLeave; virtual;                 //...

procedure EnterMenuLoop;                     //调用 TMenuBar 对象的 StartMe

```

nuLoop 方法

```

{ settings }

procedure SetBounds(...); virtual;           //设置位置与大小

function ClientToScreen(...): TPointF;      //点转换
function ScreenToClient(...): TPointF;      //...

function CloseQuery: Boolean; virtual;      //触发 OnCloseQuery 事件

function ClientRect: TRectF;                //客户区矩形

procedure Release;                          //释放

procedure Close;                            //关闭

procedure Show;                             //显示

procedure Hide;                            //隐藏

function ShowModal: TModalResult;           //显示为模态窗口

procedure CloseModal;                      //触发 OnCloseQuery、OnClose
事件

procedure Invalidate;                      //使无效而强制刷新

procedure BeginUpdate;                     //开始更新

procedure EndUpdate;                       //结束更新

property Handle: TFmxHandle ...;           //窗口句柄

property ContextHandle: THandle ...;       //环境句柄

property ModalResult: TModalResult ...;    //模态窗口的返回值

property FormState: TFmxFormStates ...;    //窗口模式状态

property Designer: IDesignerHook ...;     //设计时相关的接口

{ IRoot }

property Captured: IControl ...;           //被鼠标选中的控件

property Focused: IControl ...;           //拥有焦点的控件

```

property Hovered: IControl ...; // 鼠标下的控件

property IsActive: Boolean ...; // 是否是激活状态

property BiDiMode: TBiDiMode ...; // 不同语言的方向

property Caption: string ...; // 标题

property Cursor: TCursor ...; // 光标类型

property BorderStyle: TFmxFormBorderStyle ...; // 边框样式

property BorderIcons: TBorderIcons ...; // 工具条上的按钮

property ClientHeight: Integer ...; // 客户区高

property ClientWidth: Integer ...; // 客户区宽

property Margins: TBound ...; // 内边距

property Position: TFormPosition ...; // 相对位置

property Width: Integer ...; // 宽

property Height: Integer ...; // 高

property ShowActivated: Boolean ...; // 打开时是否激活

property StaysOpen: Boolean ...; // 是否延迟打开

property Transparency: Boolean ...; // 能否透明

property TopMost: Boolean ...; // 是否在最顶层

property Visible: Boolean ...; // 是否隐藏

property WindowState: TWindowState ...; // 窗口状态 (常态、最大化、最小化)

property OnCreate: TNotifyEvent ...; // 建立时

property OnDestroy: TNotifyEvent ...; // 销毁时

```

property OnClose: TCloseEvent ...;           //关闭时

property OnCloseQuery: TCloseQueryEvent ...;  //将要关闭时

property OnActivate: TNotifyEvent ...;       //被激活时

property OnDeactivate: TNotifyEvent ...;     //丢激活时

property OnResize: TNotifyEvent ...;         //调整大小时

published

property Left: Integer ...; //左

property Top: Integer ...; //上

end;

{ TCustomForm }

public

constructor Create(...); override; //

constructor CreateNew(...); override; //

destructor Destroy; override;         //释放

procedure InitializeNewForm; override;    //初始化 Form, 应该是内
部自动调用的

procedure AddObject(...); override;      //添加对象

procedure UpdateStyle;                  //更新样式

property Canvas: TCanvas ...;           //绘图层

property Fill: TBrush ...;              //画刷

property StyleBook: TStyleBook ...;      //样式表

property ActiveControl: TStyledControl ...; //激活控件或获取被激活的
控件

property StyleLookup: string ...;       //指定控件样式名

```

```

property OnPaint: TOnPaintEvent ...;           //重绘时
end;

```

Delphi XE2 之 FireMonkey 入门(37) - 控件基础: TControl 概览

```

{ TControl }
public

  constructor Create(...); override;           //
  destructor Destroy; override;                 //

  procedure AddObject(...); override;           //添加对象

  procedure RemoveObject(...); override;         //移除对象

  procedure SetNewScene(...); virtual;           //设置新场景

  procedure SetBounds(...); virtual;             //设置位置、大小

  function AbsoluteToLocal(...): TPointF; virtual; //点转换
  function LocalToAbsolute(...): TPointF; virtual; //...
  function AbsoluteToLocalVector(...): TVector; virtual; //...
  function LocalToAbsoluteVector(...): TVector; virtual; //...

  function PointInObject(...): Boolean; virtual; //判断指定点
  是否在对象范围内

  procedure RecalcUpdateRect; virtual; //重计算...
  procedure RecalcNeedAlign; virtual; //...
  procedure RecalcOpacity; virtual; //...
  procedure RecalcAbsolute; virtual; //...
  procedure RecalcEnabled; virtual; //...
  procedure RecalcHasEffect; virtual; //...

  function MakeScreenshot: TBitmap; //将控件捕获为 Bitmap

  procedure ShowCaretProc; //显示输入光标

```

```
procedure SetCaretPos(...);           // 设置输入光标的位置

procedure SetCaretSize(...);          // 设置输入光标的大小

procedure SetCaretColor(...);         // 设置输入光标的颜色

procedure HideCaret;                  // 隐藏输入光标

procedure BeginUpdate; virtual;       // 开始更新

procedure EndUpdate; virtual;         // 结束更新

procedure Realign; virtual;           // 重新排列

procedure ApplyEffect;                // 应用特效

procedure Painting; virtual;          // DoPaint 前

procedure DoPaint; virtual;           // 重绘过程

procedure AfterPaint; virtual;        // DoPaint 后

procedure UpdateEffects;              // 更新特效

procedure SetFocus;                  // 设置为焦点控件

procedure PaintTo(...);               // 将控件绘制到指定 Canvas

procedure Repaint;                   // 重绘

procedure InvalidateRect(...);        // 使指定矩形范围无效而强制更新

procedure Lock;                      // 锁定

property AbsoluteMatrix: TMatrix ...; // 实际的矩阵

property AbsoluteOpacity: Single ...;  // 实际的透明度

property AbsoluteWidth: Single ...;    // 实际的宽度

property AbsoluteHeight: Single ...;   // 实际的高度

property AbsoluteScale: TPointF ...;   // 实际的比例
```

property AbsoluteEnabled: Boolean ...; //实际的可用状态; 可能会随父对象而不可用

property HasEffect: Boolean ...; //是否有特效

property HasDisablePaintEffect: Boolean ...; //?

property HasAfterPaintEffect: Boolean ...; //?

property ChildrenRect: TRectF ...; //获取子对象占据的矩形

property InvertAbsoluteMatrix: TMatrix ...; //翻转后的矩阵

property InPaintTo: Boolean ...; //?

property LocalRect: TRectF ...; //获取局部的矩形范围

property AbsoluteRect: TRectF ...; //获取实际的矩形范围

property UpdateRect: TRectF ...; //获取要更新的矩形范围

property BoundsRect: TRectF ...; //获取或设置矩形范围; 设置时可同时指定位置, 但再获取到的矩形都是 (0,0,Width,Height)

property ParentedRect: TRectF ...; //获取相对于父对象的矩形范围

property ParentedVisible: Boolean ...; //判断上级对象是否可见

property ClipRect: TRectF ...; //获取剪辑区域矩形

property Canvas: TCanvas ...; //绘图层

property Scene: IScene ...; //获取场景接口

property AutoCapture: Boolean ...; //能否在 MouseDown 时自动捕获控件

property CanFocus: Boolean ...; //能否设置焦点

property DisableFocusEffect: Boolean ...; //是否禁用焦点特效

property DisableDefaultAlign: Boolean ...; //是否禁用默认对齐

property TabOrder: TTabOrder ...; //Tab 序号

published

property IsMouseOver: Boolean ...; //鼠标是否在其上

property IsDragOver: Boolean ...; //是否有拖放经过

property IsFocused: Boolean ...; //是否拥有焦点; 只读, 可通过 SetFocus()

cus() 设置

property IsVisible: Boolean ...; //是否可见; 同 visible, 但只读

property Align: TAlignLayout ...; //对齐方式

property Cursor: TCursor ...; //光标

property DragMode: TDragMode ...; //拖放模式

property EnableDragHighlight: Boolean ...; //拖放时是否高亮显示

property Enabled: Boolean ...; //是否可用

property Position: TPosition ...; //位置

property RotationAngle: Single ...; //旋转角度

property RotationCenter: TPosition ...; //旋转中心点

property Locked: Boolean ...; //是否锁定控件; 使用 Lock() 方法锁定更好

property Width: Single ...; //宽

property Height: Single ...; //高

property Margins: TBounds ...; //内边界

property Padding: TBounds ...; //外边界

property Opacity: Single ...; //透明度: 0..1

property ClipChildren: Boolean ...; //是否隐藏超出边界的子控件

property ClipParent: Boolean ...; //?

property HitTest: Boolean ...; //当前版本未完成 Hint 功能

property CanClip: Boolean ...; // 是否同意被父对象剪辑

property PopupMenu: TCustomPopupMenu ...; // 指定右键菜单

property Scale: TPosition ...; // 缩放比例

property Visible: Boolean ...; // 是否显示

property DesignVisible: Boolean ...; // 是否在设计时显示

property OnDragEnter: TDragEnterEvent ...; // 拖放进入时

property OnDragLeave: TNotifyEvent ...; // 拖放离开时

property OnDragOver: TDragOverEvent ...; // 拖放经过时

property OnDragDrop: TDragDropEvent ...; // 拖放放下时

property OnDragEnd: TNotifyEvent ...; // 拖放结束时

property OnKeyDown: TKeyEvent ...; // 按键时

property OnKeyUp: TKeyEvent ...; // 离键时

property OnClick: TNotifyEvent ...; // 单击时

property OnDblClick: TNotifyEvent ...; // 双击时

property OnCanFocus: TCanFocusEvent ...; // SetFocus() 时

property OnEnter: TNotifyEvent ...; // 获取焦点时

property OnExit: TNotifyEvent ...; // 失去焦点时

property OnMouseDown: TMouseEvent ...; // 鼠标按下时

property OnMouseMove: TMouseMoveEvent ...; // 鼠标移动时

property OnMouseUp: TMouseEvent ...; // 鼠标按键释放时

property OnMouseWheel: TMouseWheelEvent ...; // 鼠标 (滚动轮) 滚动时

property OnMouseEnter: TNotifyEvent ...; // 鼠标进入时

```
property OnMouseLeave: TNotifyEvent ...;           // 鼠标离开时
property OnPainting: TOnPaintEvent ...;           // 重绘开始时
property OnPaint: TOnPaintEvent ...;             // 重绘时
property OnResize: TNotifyEvent ...;             // 调整大小时
property OnApplyStyleLookup: TNotifyEvent ...;    // 应用新样式时
end;
```

Delphi XE2 之 FireMonkey 入门(38) - 控件基础: TPopupMenu、TMenuItem、TMenuBar、TMainMenu

相关控件: TMenuBar、TPopupMenu、TMainMenu; 它们都是要包含 TMenuItem; 在设计时添加 TMenuItem 很容易.

其中的 TMainMenu 暂不能应用其他样式; TMenuBar 只有一个值得注意 UseOSMenu 属性.

控件 PopupMenu 属性用于指定右键菜单.

暂时无法直接为窗体指定右键菜单, 因为窗体现在没有 PopupMenu 属性; 我想到的办法是在窗体上覆盖一个 TPanel 或 TRectangle:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Panel1.Align := TAlignLayout.alClient;
    Panel1.StyleLookup := StyleLookup;
    Panel1.PopupMenu := PopupMenu1;
end;
```

也可通过 TPopupMenu 的 Popup() 方法:

```
procedure TForm1.MouseDown(Button: TMouseButton; Shift: TShiftState;
  X, Y: Single);
var
  pt: TPointF;
begin
  inherited;
  if Button = TMouseButton.mbRight then
  begin
    pt := PointF(x, y);
    pt := ClientToScreen(pt);
    PopupMenu1.Popup(pt.X, pt.Y);
  end;
end;
```

Popup() 方法用于控件的例子(如 TRectangle):

```
procedure TForm1.Rectangle1MouseDown(Sender: TObject; Button: TMous
eButton; Shift: TShiftState; X, Y: Single);
var
  pt: TPointF;
begin
  if Button = TMouseButton.mbRight then
  begin
    pt := PointF(x, y);
    pt := TControl(Sender).LocalToAbsolute(pt);
    pt := ClientToScreen(pt);
    PopupMenu1.Popup(pt.X, pt.Y);
  end;
end;
```

TPopupMenu 的功能很简单,更多需要在 TMenuItem 中.

以下测试都需要在空白窗体上先放置 Rectangle1、PopupMenu1.

动态添加菜单项:

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    Rectangle1.PopupMenu := PopupMenu1;  
  
    with TMenuItem.Create(PopupMenu1) do  
    begin  
        Parent := PopupMenu1;  
        Text := 'Item1';  
    end;  
  
    with TMenuItem.Create(PopupMenu1) do  
    begin  
        Parent := PopupMenu1;  
        Text := 'Item2';  
    end;  
  
    with TMenuItem.Create(PopupMenu1) do  
    begin  
        Parent := PopupMenu1;  
        Text := '-';  
    end;  
  
    with TMenuItem.Create(PopupMenu1) do  
    begin  
        Parent := PopupMenu1;  
        Text := 'Item3';
```

```
end;  
end;
```

嵌套菜单项:

```
procedure TForm1.FormCreate(Sender: TObject);  
var  
    item: TMenuItem;  
begin  
    Rectangle1.PopupMenu := PopupMenu1;  
  
    item := TMenuItem.Create(Self);  
    item.Parent := PopupMenu1;  
    item.Text := 'Item1';  
  
    with TMenuItem.Create(Self) do  
        begin  
            Parent := item;  
            Text := 'Item1_1';  
        end;  
    with TMenuItem.Create(Self) do  
        begin  
            Parent := item;  
            Text := 'Item1_2';  
        end;  
  
    with TMenuItem.Create(Self) do  
        begin  
            Parent := PopupMenu1;  
            Text := 'Item2';  
        end;  
    end;  
end;
```

指定快捷键:

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    Rectangle1.PopupMenu := PopupMenu1;  
  
    with TMenuItem.Create(PopupMenu1) do  
    begin  
        Parent := PopupMenu1;  
        Text := 'Item1';  
        ShortCut := scCtrl or Byte('A'); //Ctrl + A  
    end;  
  
    with TMenuItem.Create(PopupMenu1) do  
    begin  
        Parent := PopupMenu1;  
        Text := 'Item2';  
        ShortCut := scShift or scCtrl or scAlt or Ord('A'); //Shift + Ctrl + Alt + A  
    end;  
  
    with TMenuItem.Create(PopupMenu1) do  
    begin  
        Parent := PopupMenu1;  
        Text := '-';  
    end;  
  
    with TMenuItem.Create(PopupMenu1) do  
    begin  
        Parent := PopupMenu1;  
        Text := 'Item3';
```

```
ShortCut := 112; //F1
end;
end;
```

复选菜单项:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Rectangle1.PopupMenu := PopupMenu1;

    with TMenuItem.Create(PopupMenu1) do
    begin
        Parent := PopupMenu1;
        Text := 'Item1';
        AutoCheck := True;
    end;

    with TMenuItem.Create(PopupMenu1) do
    begin
        Parent := PopupMenu1;
        Text := 'Item2';
        AutoCheck := True;
    end;

    with TMenuItem.Create(PopupMenu1) do
    begin
        Parent := PopupMenu1;
        Text := '-';
    end;

    with TMenuItem.Create(PopupMenu1) do
    begin
```

```
    Parent := PopupMenu1;  
    Text := 'Item3';  
    AutoCheck := True;  
end;  
end;
```

单选(分组)菜单项:

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    Rectangle1.PopupMenu := PopupMenu1;  
  
    with TMenuItem.Create(PopupMenu1) do  
    begin  
        Parent := PopupMenu1;  
        Text := 'Item1';  
        AutoCheck := True;  
        RadioItem := True;  
        GroupIndex := 1;  
        IsChecked := True;  
    end;  
  
    with TMenuItem.Create(PopupMenu1) do  
    begin  
        Parent := PopupMenu1;  
        Text := 'Item2';  
        AutoCheck := True;  
        RadioItem := True;  
        GroupIndex := 1;  
    end;  
  
    with TMenuItem.Create(PopupMenu1) do
```

```
begin
    Parent := PopupMenu1;
    Text := '-';
end;

with TMenuItem.Create(PopupMenu1) do
begin
    Parent := PopupMenu1;
    Text := 'Item3';
    AutoCheck := True;
    RadioItem := True;
    GroupIndex := 2;
end;
with TMenuItem.Create(PopupMenu1) do
begin
    Parent := PopupMenu1;
    Text := 'Item4';
    AutoCheck := True;
    RadioItem := True;
    GroupIndex := 2;
end;
end;
```

菜单文本格式:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Rectangle1.PopupMenu := PopupMenu1;

    with TMenuItem.Create(PopupMenu1) do
    begin
        Parent := PopupMenu1;
```

```
Text := 'Item1';  
Font.Style := [TFontStyle.fsBold, TFontStyle.fsItalic];  
end;  
end;
```

图标:

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    Rectangle1.PopupMenu := PopupMenu1;  
  
    with TMenuItem.Create(Self) do  
    begin  
        Parent := PopupMenu1;  
        Text := 'Item1';  
        Bitmap.LoadFromFile('c:\temp\test.png');  
    end;  
end;
```

指定事件:

```
unit Unit1;  
  
interface  
  
uses  
    System.SysUtils, System.Types, System.UITypes, System.Classes, Sy  
stem.Variants,  
    FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs, FMX.Menus, FMX.O  
bjects;
```

type

```
TForm1 = class (TForm)
    Rectangle1: TRectangle;
    PopupMenu1: TPopupMenu;
    procedure FormCreate(Sender: TObject);
    procedure ItemOnClick(Sender: TObject);
end;
```

var

```
Form1: TForm1;
```

implementation

```
{ $R *.fmx }
```

```
procedure TForm1.FormCreate(Sender: TObject);
```

begin

```
    Rectangle1.PopupMenu := PopupMenu1;
```

```
    with TMenuItem.Create(PopupMenu1) do
```

begin

```
        Parent := PopupMenu1;
```

```
        Text := 'Item1';
```

```
        OnClick := ItemOnClick;
```

```
    end;
```

```
    with TMenuItem.Create(PopupMenu1) do
```

begin

```
        Parent := PopupMenu1;
```

```
        Text := 'Item2';
```

```
        OnClick := ItemOnClick;
```

```
    end;
```

```
end;
```



```
procedure TForm1.ItemOnClick(Sender: TObject);
begin
    ShowMessage(TTextControl(Sender).Text);
end;

end.
```

Delphi XE2 之 FireMonkey 入门(39) - 控件基础: TScrollBox、TVertScrollBox、TFramedScrollBox、TFramedVertScrollBox

TScrollBox 是不少控件(TMemo、TListBox、TStringGrid、TTreeView、TImageViewer 等)的基础, 所以先学.

TVertScrollBox 只比 TScrollBox 少了竖滚动条.

TFramedScrollBox、TFramedVertScrollBox 只是套用了不同的样式, 有可视的框架.

```
{ TScrollBox }
public
    constructor Create(...); override; //
    destructor Destroy; override; //
    procedure AddObject(...); override; //
    procedure Sort(...); override; //根据指定的排序函数给内部对象排序
    procedure MouseDown(...); override; //
    procedure MouseMove(...); override; //
    procedure MouseUp(...); override; //
    procedure MouseWheel(...); override; //
    procedure Realign; override; //
    procedure Centre; //滚动到中间位置
```

```
procedure ScrollTo(...);           //滚动指定的距离

procedure InViewRect(...);         //暂未实现

function ClientWidth: Single;      //客户区宽度

function ClientHeight: Single;     //客户区高度

property HScrollBar: TScrollBar ...; //横滚动条对象

property VScrollBar: TScrollBar ...; //竖滚动条对象

published

    property AutoHide: Boolean ...;   //是否根据需要自动隐藏或显示
滚动条; 默认 True

    property Animated: Boolean ...;   //是否使用滚动动画; 默认 True
e

    property DisableMouseWheel: Boolean ...; //是否禁用鼠标滚动轮; 默认
False, 未禁用

    property MouseTracking: Boolean ...; //能否用鼠标直接拖动(拖到控
件, 滚动条联动); 默认 False

    property ShowScrollBars: Boolean ...; //是否显示滚动条; 默认 True

    property ShowSizeGrip: Boolean ...; //是否显示 Grip(右下角的拖
拽标识); 默认 False

    property UseSmallScrollBars: Boolean ...; //是否使用小的滚动条

end;
```

示例:

```
uses FMX.Layouts, FMX.Objects;

var
    ScrollBox1: TScrollBox;
    Rectangle1: TRectangle;

procedure TForm1.FormCreate(Sender: TObject);
begin
    ScrollBox1 := TScrollBox.Create(Self);
    ScrollBox1.Parent := Self;
    ScrollBox1.Align := TAlignLayout.alClient;
    ScrollBox1.MouseTracking := True;

    Rectangle1 := TRectangle.Create(Self);
    Rectangle1.Parent := ScrollBox1;
    Rectangle1.SetBounds(100, 100, ClientWidth*2, ClientHeight*2);
end;
```

Delphi XE2 之 FireMonkey 入门(40) - 控件基础: TMemo

值得注意的变化:

1、其父类 TScrollBox 的许多特性也很有用处, 如:

```
Memo1.UseSmallScrollBars := True;
```

2、内部使用了一个栈列表管理动作, 现在可以执行多步撤销(UnDo).

3、使用了一个新的结构体 TCaretPosition 来描述当前位置, 并为该结构提供了一个公用的快速构造函数 ComposeCaretPos().

```
TCaretPosition = record
    Line, Pos: Integer;
end;
```

4、提供枚举属性 `CharCase` 控制大小写.

```
Memo1.CharCase := TEditCharCase.ecUpperCase;
```

5、提供布尔属性 `AutoSelect` 决定获取焦点时是否自动全选.

6、现在的 `Lines` 属性读取的是自动换行后的文本集合; 原始集合用 `UnwrapLines` 读取.

```
Memo1.CharCase := TEditCharCase.ecUpperCase;
```

7、还提供了 `InsertAfter()`、`DeleteFrom()` 等新方法.

```
{ TMemo }  
  
public  
    constructor Create(...); override; //  
    destructor Destroy; override;      //  
  
    procedure CopyToClipboard;           //复制  
  
    procedure PasteFromClipboard;        //粘贴  
  
    procedure CutToClipboard;            //剪切  
  
    procedure ClearSelection;            //取消选择  
  
    procedure SelectAll;                 //全选  
  
    procedure GoToTextEnd;               //到最后  
  
    procedure GoToTextBegin;            //到开始  
  
    procedure GotoLineEnd;              //到行尾  
  
    procedure GoToLineBegin;            //到行首  
  
    function GetPositionPoint(...): TPointF; //获取当前位置  
  
    procedure Undo;                      //撤销  
  
    procedure InsertAfter(...);          //插入
```

```

procedure DeleteFrom(...); //删除指定范围的内容

function TextPosToPos(...): TCaretPosition; //根据指定的文本长度计算
光标位置

function PosToTextPos(...): Integer; //TextPosToPos() 的反相
计算

property SelStart: Integer ...; //选区文本的起始位置

property SelLength: Integer ...; //选区文本的长度

property SelText: string ...; //选区文本

property CaretPosition: TCaretPosition ...; //输入光标的位置

property LineWidth[LineNum: Integer]: Single ...; //获取指定行的宽度

property UnwrapLines: TStrings ...; //没有换行的原始文本集合;
在 WordWrap = False 时同 Lines

property FontFill: TBrush ...; //文本笔刷

property SelectionFill: TBrush ...; //选区笔刷

published

property Cursor default crIBeam; //鼠标光标

property CanFocus default True; //能否拥有焦点

property DisableFocusEffect; //是否禁用焦点特效

property TabOrder; //Tab 序

property AutoSelect: Boolean ...; //是否在获取焦点时自动全选

property CharCase: TEditCharCase ...; //控制大小写的选项

property Enabled; //是否可用

property HideSelection: Boolean ...; //?

property Lines: TStrings ...; //自动换行后的文本集合

```

```
property MaxLength: Integer ...;           //最大长度

property ReadOnly: Boolean ...;            //是否只读

property OnChange: TNotifyEvent ...;       //有改变时

property OnChangeTracking: TNotifyEvent ...; //发生在 OnChange 之前
的事件

property WordWrap: Boolean ...;            //是否自动换行

property Font: TFont ...;                  //字体

property Text: string ...;                 //内容

property TextAlign: TTextAlign ...;        //文本对齐方式

property StyleLookup;                      //指定样式

end;
```

Delphi XE2 之 FireMonkey 入门(41) - 控件基础: TListBox

TScrollBar -> TCustomListBox -> TListBox; 其元素项是 TListBoxItem 类型.

TListBox 的功能在 TCustomListBox 里就完成了.

值得注意的变化是:

- 1、复选框(相关属性: ShowCheckboxes、TListBoxItem.IsChecked)
- 2、交替背景(通过继承还可以调整交替的背景色)
- 3、TListBoxItem 可调整大小、容纳其它对象.

```
{ TCustomListBox }

public

    constructor Create(...); override;    //
```

```

destructor Destroy; override; //
procedure Assign(...); override; //
procedure Clear; virtual; //清空

function DragChange(...): Boolean; dynamic; //调用 OnDragChange 事件

procedure SelectAll; //全选

procedure ClearSelection; //取消选择

procedure SelectRange(...); //选择指定范围

function ItemByPoint(...): TListBoxItem; //获取指定位置的项

function ItemByIndex(...): TListBoxItem; //获取指定序号的项

procedure Exchange(...); //交换项

procedure AddObject(...); override; //添加项

procedure RemoveObject(...); override; //删除项

procedure Sort(...); override; //排序

property Count: Integer ...; //项总数

property Selected: TListBoxItem ...; //当前选择的项

property Items: TStrings ...; //元素文本的集合

property ListItems[Index: Integer]: TListBoxItem ...; //根据索引获取项

property ItemIndex: Integer ...; //索引
end;

{ TListBox }
published
property StyleLookup; //
property AllowDrag; //是否允许拖放

```

```

property CanFocus; //
property DisableFocusEffect; //
property TabOrder; //

property AlternatingRowBackground; //是否使用交替背景

property Columns; //列数; 默认 1

property HideSelectionUnfocused; //在失去焦点时是否隐藏选择

property Items; //
property ItemIndex; //
property ItemWidth; //项宽
property ItemHeight; //项高

property ListStyle; //列表样式; TListStyle = (lsVertical, lsHorizontal);

property MultiSelect; //是否允许多选; 为 True 时, 配合 Ctrl 键多选

property Sorted; //
property ShowCheckboxes; //是否显示复选框; 默认 False

property BindingSource; //绑定源

property OnChange; //有变化时

property OnChangeCheck; //调整复选框时

property OnCompare; //排序比较时

property OnDragChange; //拖放项时

end;

{ TListBoxItem }
public
    constructor Create(...); override; //
    property Data: TObject ...; //

```

```
    property Index: Integer ...;           //
published

    property IsChecked: Boolean ...;      //复选值
    property IsSelected: Boolean ...;    //
    property AutoTranslate ...;          //
    property Font;                        //
    property StyleLookup;                 //
    property Text;                        //
    property TextAlign ...;               //
    property WordWrap;                   //
end;
```

测试:

```
procedure TForm1.FormCreate(Sender: TObject);
var
    i: Integer;
begin
    ListBox1.Align := TAlignLayout.alLeft;
    ListBox1.ShowCheckboxes := True;
    ListBox1.AlternatingRowBackground := True;
    for i := 0 to 9 do
        begin
            ListBox1.Items.Add('Itme' + IntToStr(i));
            ListBox1.ListItems[i].IsChecked := Odd(i);
        end;
    end;
```

TListBox 有两个兄弟 TComboBox、TComboEditListBox;

TComboBox、TComboEdit 虽不是不是从它们继承, 但分别包含了它们, 所以使用起来都有点像 TListBox.

TComboBox 更像 TListBox, 比 TComboEdit 多出了 Selected 等成员;

TComboEdit 是从 TCustomEdit 继承, 和 TEdit 是兄弟, 比 TComboBox 多出了 Text 等成员.

它们的公共常用属性:

DropDownCount // 下拉列表行的数

ItemHeight //

ItemIndex //

Items //

Count //

测试:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  { ComboBox1 }
  for i := 0 to 9 do
    ComboBox1.Items.Add(Format('Item_%d', [i]));
  with ComboBox1 do
  begin
    ItemIndex := 0;
    DropDownCount := 5;
    ListBox.UseSmallScrollBars := True;
```

```
TListBox(ListBox).AlternatingRowBackground := True; //这个兄弟转
换用得有点悬，只是为了让 AlternatingRowBackground 属性暴露出来

end;

{ ComboEdit1 }
ComboEdit1.Items.Assign(ComboBox1.Items);
with ComboEdit1 do
begin
  ItemIndex := 0;
  DropDownCount := 5;
  ListBox.UseSmallScrollBars := True;
  TListBox(ListBox).AlternatingRowBackground := True;
end;
// ComboEdit1.Text := 'Text';
end;
```

Delphi XE2 之 FireMonkey 入门(43) - 控件基础: TStringGrid、TGrid

TStringGrid、TGrid 都是从 TCustomGrid 继承; 区别有:

- 1、它们的列对象分别是: TStringColumn、TColumn;
- 2、TStringGrid 比 TGrid 多出了 Cells[] 属性.

因为 TGrid 没有 Cells[] 属性, 暂时不方便使用; 我尝试取其当前单元值时竟然用了这样的代码:
(Grid1.Columns[Grid1.ColumnIndex].CellControlByRow(Grid1.Selected) as TTextCell).Text

TStringGrid 测试:

```
{ 设计时放好 StringGrid1, 运行时填充数据 }
```

```
procedure TForm1.FormCreate(Sender: TObject);  
var  
    i,c,r: Integer;  
begin  
    StringGrid1.AlternatingRowBackground := True;  
    StringGrid1.UseSmallScrollBars := True;  
    for i := 0 to 5 do //从设计时添加列比这方便  
    begin  
        with TStringColumn.Create(Self) do  
        begin  
            Parent := StringGrid1;  
            Width := StringGrid1.ClientWidth / 6;  
        end;  
    end;  
    StringGrid1.RowCount := 20;  
    for c := 0 to StringGrid1.ColumnCount - 1 do  
        for r := 0 to StringGrid1.RowCount - 1 do  
            StringGrid1.Cells[c, r] := Format('%d,%d', [c, r]);  
end;  
  
    { 取当前单元值 }  
  
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    ShowMessage(StringGrid1.Cells[StringGrid1.ColumnIndex, StringGrid  
1.Selected]);  
end;
```

成员概览:

```
{ TCustomGrid }  
public
```

```

constructor Create(...); override;           //
destructor Destroy; override;                 //

function ColumnByIndex(...): TColumn;          //根据索引获取列对象
function ColumnByPoint(...): TColumn;          //根据位置获取列对象
function RowByPoint(...): Integer;              //根据位置获取行号
procedure AddObject(...); override;           //

property TopRow: Integer ...;                  //获取可见的首行的行号
property VisibleRows: Integer ...;             //获取可见的行总数
property ColumnCount: Integer ...;             //列数(也是只读)
property ColumnIndex: Integer ...;            //获取或设置列索引
property Columns[Index: Integer]: TColumn ...; //以数组索引的方式获取
列对象

property RowCount: Integer ...;                //行数(可读写)
property Selected: Integer ...;               //当前行号
property OnGetValue: TOnGetValue ...;         //取值时
property OnSetValue: TOnSetValue ...;        //赋值时
published
property StyleLookup;                          //
property AlternatingRowBackground: Boolean ...; //是否使用交替背景;
默认 False
property CanFocus default True;               //
property DisableFocusEffect default True;    //是否取消焦点特效
property RowHeight: Single ...;               //行高
property ShowSelectedCell: Boolean ...;       //是否呈现单元选择效果; 默认
True

```

```
property ShowVertLines: Boolean ...; //是否显示竖格线

property ShowHorzLines: Boolean ...; //是否显示横格线

property ShowHeader: Boolean ...; //是否显示表格头

property ReadOnly: Boolean ...; //是否只读; 默认 False

property TabOrder; //

property OnEdititingDone: TOnEdititingDone ...; //输入时

end;

{ TGrid }
TGrid = class(TCustomGrid)
published
    property RowCount; //
    property OnGetValue; //
    property OnSetValue; //
end;

{ TStringGrid }
public
    property Cells[ACol, ARow: Integer]: string ...; //
published
    property RowCount; //
end;
```

Delphi XE2 之 FireMonkey 入门(44) - 控件基础: TTreeView、TTreeViewItem

TScrollBar -> TCustomTreeView -> TTreeView 子项类型是 TTreeViewItem.

测试, 先在窗体上放一个 TTreeView, 并需要它的 OnChange 事件:

```
unit Unit1;
```

```
interface
```

```
uses
```

```
    System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,  
    FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs, FMX.Layouts, FMX.TreeView;
```

```
type
```

```
    TForm1 = class(TForm)  
        TreeView1: TTreeView;  
        procedure FormCreate(Sender: TObject);  
        procedure TreeView1Change(Sender: TObject);  
    end;
```

```
var
```

```
    Form1: TForm1;
```

```
implementation
```

```
{ $R *.fmx }
```

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
var
```

```
    item: TTreeViewItem;  
    i: Integer;
```

```
begin
```

```
    TreeView1.Align := TAlignLayout.alLeft;  
    TreeView1.UseSmallScrollBars := True;  
    TreeView1.AlternatingRowBackground := True;  
    TreeView1.ShowCheckboxes := True;
```

```
item := TTreeViewItem.Create(Self);
item.Parent := TreeView1;
item.Text := 'Item1';

item := TTreeViewItem.Create(Self);
item.Parent := TreeView1;
item.Text := 'Item2';
  for i := 0 to 19 do
    with TTreeViewItem.Create(Self) do
      begin
        Parent := item;
        Text := item.Text + '_' + IntToStr(i+1);
        IsChecked := Odd(i);
      end;
    item.IsExpanded := True;

item := TTreeViewItem.Create(Self);
item.Parent := TreeView1;
item.Text := 'Item3';
end;

procedure TForm1.TreeView1Change(Sender: TObject);
var
  item: TTreeViewItem;
begin
  item := TreeView1.Selected;
  if item <> nil then
    begin
      ShowMessageFmt(
        'TreeView1.GlobalCount: %d'#13#10 +
        'TreeView1.Count: %d'#13#10 +
        'TTreeViewItem.Count: %d'#13#10 +
        'TTreeViewItem.Level: %d'#13#10 +
```



```
'TTreeViewItem.GlobalIndex: %d'#13#10 +  
'TTreeViewItem.Index: %d'#13#10  
    , [TreeView1.GlobalCount, TreeView1.Count, item.Count, item.Le  
vel, item.GlobalIndex, item.Index]  
    );  
end;  
end;  
  
end.
```

成员概览:

```
{ TCustomTreeView }  
public  
    constructor Create(...); override; //  
    destructor Destroy; override;      //  
  
    procedure EndUpdate; override;      //结束更新  
  
    procedure Clear;                    //清空  
  
    procedure ExpandAll;                //全部展开  
  
    procedure CollapseAll;              //全部关闭  
  
    function ItemByText(...): TTreeViewItem; //根据标题获取项  
  
    function ItemByPoint(...): TTreeViewItem; //根据位置获取项  
  
    function ItemByIndex(...): TTreeViewItem; //根据索引获取项  
  
    function ItemByGlobalIndex(...): TTreeViewItem; //根据总索引获取项  
  
    procedure AddObject(...); override; //  
    procedure RemoveObject(...); override; //  
    procedure MouseDown(...); override; //  
    procedure MouseMove(...); override; //
```

```

procedure MouseUp (...); override;           //
procedure KeyDown (...); override;           //
procedure KeyUp (...); override;             //
procedure DragOver (...); override;          //
procedure DragDrop (...); override;          //

property Count: Integer ...;                  //第一层项总数

property GlobalCount: Integer ...;            //项总数

property CountExpanded: Integer ...;          //???

property Selected: TTreeViewItem ...;         //当前选择的项

property Items[Index: Integer]: TTreeViewItem ...; //索引到项

property AllowDrag: Boolean ...;               //是否允许拖拽

property AlternatingRowBackground: Boolean ...; //是否使用交替
背景

property ItemHeight: Single ...;               //项高

property HideSelectionUnfocused: Boolean ...; //是否在失去焦
点时隐藏选择

property MultiSelect: Boolean ...;             //能否多选

property ShowCheckboxes: Boolean ...;          //是否显示多选框

property Sorted: Boolean ...;                  //
property OnChange: TNotifyEvent ...;          //
property OnChangeCheck: TNotifyEvent ...;     //
property OnCompare: TOnCompareTreeViewItemEvent ...; //
property OnDragChange: TOnTreeViewDragChange ...; //

end;

{ TTreeView }
published
    property StyleLookup;                        //
    property CanFocus default True;            //

```

```
property DisableFocusEffect; //
property TabOrder; //
property AllowDrag default False; //
property AlternatingRowBackground default False; //
property ItemHeight; //
property HideSelectionUnfocused default False; //
property MultiSelect default False; //
property ShowCheckboxes default False; //
property Sorted default False; //
property OnChange; //
property OnChangeCheck; //
property OnCompare; //
property OnDragChange; //
end;

{ TTreeViewItem }
public
    constructor Create(...); override; //
    procedure Paint; override; //
    procedure AddObject(...); override; //
    procedure RemoveObject(...); override; //
    procedure Sort(...); override; //

    function ItemByPoint(...): TTreeViewItem; //根据位置获取项

    function ItemByIndex(...): TTreeViewItem; //根据索引获取项

    property Count: Integer ...; //子项数

    property GlobalIndex: Integer ...; //全局索引

    function TreeView: TCustomTreeView; //拥有它的 TreeView

    function Level: Integer; //在第几层

    function ParentItem: TTreeViewItem; //父项

    property Items[Index: Integer]: TTreeViewItem ...; //索引到子项
```

published

```
    property IsChecked: Boolean ...;      //是否选中复选框

    property IsExpanded: Boolean ...;     //是否展开

    property IsSelected: Boolean ...;     //是否选择

    property AutoTranslate default True; //是否自动接收语言翻译

    property Font;                        //
    property StyleLookup;                //
    property Text;                        //
    property TextAlign default TTextAlign.taLeading; //
end;
```

Delphi XE2 之 FireMonkey 入门(45) - 结题与问题

很喜欢 FMX 的一些新控件, 如: TExpander、TArcDial、TComboTrackBar、TPathLabel 等等, 没时间继续学习了.

对 FMX 的整体感觉: 还不成熟, 但肯定有前景; 它的构架师有远见、了不起, 很难估计他开启的是多大一扇门!

本将继续学习:

- 1、TCanvas、TBrush、TApplication;
- 2、FMX 中的 GDI+、D2D、DirectX;
- 3、FMX 3D;
- 4、XE2 中新增的其他内容(譬如新增的 TZipFile 类, 已测试过、下帖附上).

又要忙其它事情了, 但愿能尽早有时间和心情回来继续学习.

正在学习但没有学完的东西是关于拖放的; 现在的 FMX 窗体可以直接响应拖放了, 譬如: 拖拽文本文件到窗体让 TMemo 打开.

```
unit Unit1;
```

```
interface
```

```
uses
```

```
    System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,  
    FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs, FMX.Layouts, FMX.Memo;
```

```
type
```

```
    TForm1 = class(TForm)
```

```
        Memo1: TMemo; //先在空白窗体上放 Memo1
```

```
    private
```

```
    public
```

```
        { 覆盖这个两个方法 }
```

```
        procedure DragOver(const Data: TDragObject; const Point: TPointF;  
var Accept: Boolean); override;  
        procedure DragDrop(const Data: TDragObject; const Point: TPointF); override;  
    end;
```

```
var
```

```
    Form1: TForm1;
```

```
implementation
```

```
{ $R *.fmx }
```

```
{ TForm1 }
```

```
procedure TForm1.DragOver(const Data: TDragObject; const Point: TPoint; var Accept: Boolean);  
begin  
    inherited;  
  
    { 如果拖拽的是文件, 且是 *.txt 文件, 则让 Accept := True 以让后面的 DragDrop 事件响应 }  
  
    Accept := (Length(Data.Files) > 0) and (CompareText(ExtractFileExt(Data.Files[0]), '.txt') = 0);  
end;  
  
procedure TForm1.DragDrop(const Data: TDragObject; const Point: TPoint; var Accept: Boolean);  
begin  
    inherited;  
  
    Memo1.Lines.LoadFromFile(Data.Files[0]);  
end;  
  
end.
```

<结束>