TSpriteEngine Component

Properties Methods Events

Unit

SpriteEngine

Description

TSpriteEngine components manage instances of TSprite-derived objects. Typically, sprites are intended to be rendered on a TDIBDrawingSurface, and the DIBDrawingSurface property is provided to hook the sprite engine into a single drawing surface component on the form. During each execution of the drawing surface's OnCustomPaint event,.you should execute the sprite engine's ProcessSprites method, then its RenderSprites method. If you wish to perform collision detection, you can also call the CollisionDetection method at this point.

To add a TSprite-derived object to the engine, call the AddSprite method. To remove a sprite, call the RemoveSprite method. To change the priority (Z-Order) of a sprite once it has been added to the engine, call the ChangeSpritePriority method.

You can determine the number of sprites in the engine through the SpriteCount property, and access any particular sprite through the Sprite indexed property.

The sprite engine has a DirtyRectangles property that determines whether a dirty rectangle system will be used when rendering sprites. You should use the dirty rectangle system if you have few sprites, or if only a few sprites will ever be moving simultaneously. When you modify the DirtyRectangles property, or assign a TDIBDrawingSurface to the DIBDrawingSurface property, the corresponding DirtyRectangle is synchronized with that of the sprite engine.

Properties

DIBDrawingSurface	DirtyRectangles	ExpectedSprites
Sprite	SpriteCount	

Methods

AddSprite	ChangeSpritePriority	CollisionDetection
ProcessSprites	RenderSprites	RemoveSprite

Events

OnCollision

Sprite Property

Applies To
TSpriteEngine

Declaration
```
property Sprite[n: integer]: TSprite;
```

Description
The Sprite property returns the specified TSprite-derived object from the engine.

SpriteCount Property

Applies To
TSpriteEngine

Declaration
```
property SpriteCount: integer;
```

Description
The SpriteCount property returns the number of sprites currently registered with the engine.

DIBDrawingSurface Property

Applies To

TSpriteEngine

Declaration

```
property DIBDrawingSurface: TDIBDrawingSurface;
```

Description

The DIBDrawingSurface property points to the TDIBDrawingSurface component that this sprite engine is associated with. All sprites are moved using the specific setting of this TDIBDrawingSurface. The TDIBDrawingSurface properties that can affect sprite movement are OffsetX, OffsetY, WrapHorizontal, WrapVertical, PhysicalWidth and PhysicalHeight. The sprites are also rendered onto this component when the RenderSprites method is invoked.

More than one sprite engine can be associated with a single TDIBDrawingSurface.

DirtyRectangles Property

Applies To

TSpriteEngine

Declaration

```
property DirtyRectangles: boolean;
```

Description

If DirtyRectangles is set to True, a dirty rectangle system will be employed that will render only sprites that have actually changed position. If False, all sprites are rendered during each call to RenderSprites, regardless of whether they moved or not. The dirty rectangle system incurs some overhead, so it should be employed only if there are a relatively few number of sprites, or if only a few sprites will ever be moving simultaneously.

ExpectedSprites Property

Applies To

TSpriteEngine

Declaration

```
property ExpectedSprites: integer;
```

Description

Set this property to the number of sprites you expect to register with the engine. This causes the resources for the sprite list to be allocated up-front, instead of gradually as sprites are added to the list.

AddSprite Method

Applies To

TSpriteEngine

Declaration

```
procedure AddSprite( spr: TSprite );
```

Description

Call the AddSprite method to register a TSprite-derived object with the sprite engine. The following example registers 25 TDIBSprites:

```
procedure Add;
var
   i: integer
begin
   for i := 1 to 25 do
      begin
         spr := TDIBSprite.CreateDIBSprite( DIB1 );
         spr.Position := Point( Random( 100 ), Random( 100 ) );
         spr.Destination := Point( Random( 100 ), Random( 100 ) );
         spr.MotionType := mtContinuous;
         SpriteEngine1.AddSprite( spr );
      end;
end;
```

ChangeSpritePriority Method

Applies To

TSpriteEngine

Declaration

```
procedure ChangeSpritePriority( spr: TSprite; n: integer );
```

Description

The ChangeSpritePriority changes a sprite's priority (or Z-Order) after the sprite has been registered with the engine. After a spriite is registered with AddSprite, you should not explicitly change its Priority property directly, because this could cause synchronization problems with the sprite engine.

CollisionDetection Method

Applies To

TSpriteEngine

Declaration

```
procedure CollisionDetection( se: TSpriteEngine );
```

Description

If you want to be notified of collisions between sprites, call the CollisionDetection method just after the ProcessSprites and RenderSprites methods. The parameter of this method is another TSpriteEngine, although you can pass the same engine if you want to check for collisions in a single sprite engine. The reason that the sprite engine parameter is provided is that you may have multiple sprite engines registered to a single TDIBDrawingSurface. One of the sprite engines may be reserved for missiles, that will destroy any other sprites they contact. Missiles will not, however, destroy other missiles. On the TDIBDrawingSurface's OnCustomPaint event, you would code the following:

```
procedure DIBDrawingSurface1CustomPaint( Sender: TObject );
begin
   SpriteEngineMissile.ProcessSprites;
   SpriteEngineOther.ProcessSprites;
   SpriteEngineMissile.RenderSprites;
   SpriteEngineOther.RenderSprites;
   SpriteEngineMissile.CollisionDetection( SpriteEngineOther );
end;
```

The above code will perform collision detection of missiles against other objects only, avoiding unnecessary processing.

The OnCollision event is triggered for each collision that is detected.

In order for collisions to be detected properly, valid values must be assigned to all sprites' Width and Height properties.

ProcessSprites Method

Applies To

TSpriteEngine

Declaration

```
procedure ProcessSprites;
```

Description

ProcessSprites executes the Move method of all sprites registered in the sprite engine. This call should be made within the OnCustomPaint event of the TDIBDrawingSurface that the sprite engine is associated with.

RenderSprites Method

Applies To
TSpriteEngine

Declaration
```
procedure RenderSprites;
```

Description
RenderSprites executes the Render method of all sprites registered in the sprite engine. This call should be made within the OnCustomPaint event of the TDIBDrawingSurface that the sprite engine is associated with.

OnCollsion Event

Applies To

TSpriteEngine

Declaration

```
TCollisionEvent = procedure( Sender: TObject; Sprite, TargetSprite: TSprite ) of object;
property OnCollision: TCollisionEvent;
```

Description

OnCollision events are triggered in response to the execution of the CollisionDetection method. The event returns the two sprites that were involved in a collsion.

In order for collisions to be detected properly, valid values must be assigned to all sprites' Width and Height properties.

RemoveSprite Method

Applies To
TSpriteEngine

Declaration
```
procedure RemoveSprite( spr: TSprite );
```

Description
The RemoveSprite method removes the specified sprite from the engine, and Frees the TSprite-derived object.

TSprite Class

Properties Methods

Unit

Sprite

Description

TSprite encapsulates a movable sprite object that can be registered with a TSpriteEngineTSPRITEENGINE and can be rendered on a TDIBDrawingSurface. TSprite is an abstract base class. In order to create sprites, you must derive a class from TSprite and override the Render method, which implements how the sprite is rendered on a TDIBDrawingSurface. Two derived classes are included as examples, TDIBSprite, which renders itself using a TDIB component, and TPolygonSprite, which renders itself by drawing lines.

All classes derived from TSprite should make sure to assign values to the Width and Height properties. These properties are used in calculating the sprite's DirtyRect and BoundingRect, as well as in the CollisionDetection method of TSpriteEngine.

The sprite's Move method controls its movement pattern. TSprite provides a default Move method that you can use in your derived classes. It moves the sprite toward its Destination property at a constant Speed. The sprite's MotionType controls whether or not the sprite stops at its Destination. You are free to augment or replace the Move method for more specialized control.

The sprite's position is available through the Position and PhysicalPosition properties. Position reports the logical position of the sprite within the DIBDrawingSurface, with the OffsetX and OffsetY properties taken into account. The PhysicalPosition returns the absolute physical position of the sprite on the surface.

Properties

BoundingRect	Destination		Dead
DIBDrawingSurface	Dirty	DirtyRect	
Height	MarginBottom		MarginLeft
MarginRight	MarginTop		MotionType
Moved	PhysicalPosition		Position
Priority	Speed		Tag
Visible	Width		

Methods

FudgedDistance	Move	RefreshBackground
Render		

BoundingRect Property

Applies To

TSprite

Declaration

```
property BoundingRect: TRect;
```

Description

Read Only. The BoundingRect property returns a TRect that describes the bounding area of the sprite, within the logical coordinates of the associated DIBDrawingSurface. The properties that influence a sprite's BoundingRect are Width, Height, Position, and the margin properties. This property is used by the TSpriteEngine and would rarely be accessed by the developer.

Destination Property

Applies To
TSprite

Declaration
```
property Destination: TPoint;
```

Description
The Destination property contains the sprite's destination in the logical coordinates of its associated DIBDrawingSurface. The default Move method moves the sprite towards this point at a constant Speed. Change this property to change the direction of the sprite.

Dead Property

Applies To
TSprite

Declaration
```
property Dead: boolean;
```

Description
The Dead property is a flag that indicates if the sprite should be removed from a TSpriteEngine. This property is set by the RemoveSprite method of TSpriteEngine, but can also be set manually in the TSprite-derived class for sprites with short life spans. For an example, see the Move method of the sample TMissile and TExplosion classes provided in the TurboSprite package.

DIBDrawingSurface Property

Applies To

TSprite

Declaration

```
property DIBDrawingSurface: TDIBDrawingSurface;
```

Description

This property returns the TDIBDrawingSurface that the sprite is associated with. This property is set during the AddSprite method of TSpriteEngine.

Dirty Property

Applies To

TSprite

Declaration

```
property Dirty: boolean;
```

Description

The Dirty property is a flag that indicates if this sprite's BoundingRect intersects any dirty rectangles. When the DirtyRectangle property of the sprite's TSpriteEngine is True, the sprite engine uses this flag to determine which sprites to draw during the RenderSprites method. There should rarely be any reason to access this property directly.

DirtyRect Property

Applies To

TSprite

Declaration

```
property DirtyRect: TRect;
```

Description

Read Only. This property returns the sprite's dirty rectangle, which is the union of its current BoundingRect and its BoundingRect from the previous call to RenderSprites. The TSpriteEngine uses this property when its DirtyRectangles property is True, and there should be no need to access it explicitly.

Height Property

Applies To

TSprite

Declaration

```
property Height: integer;
```

Description

This property specifies the height of the sprite in pixels. Classes derived from TSprite must provide a valid value for this property, usually in the constructor. Otherwise, dirty rectangle processing will not be correct.

Width Property

Applies To

TSprite

Declaration

```
property Width: integer;
```

Description

This property specifies the width of the sprite in pixels. Classes derived from TSprite must provide a valid value for this property, usually in the constructor. Otherwise, dirty rectangle processing will not be correct.

Margin Properties

Applies To

TSprite

Declaration

```
property MarginBottom: integer;
property MarginLeft: integer;
property MarginRight: integer;
property MarginTop: integer;
```

Description

The Margin properties provide a way to reduce the perceived size of the sprite for collision detection purposes only. Sometimes, the actual size of the sprite is too large for collision detection. The Margin properties are a way to trim away pixels from each of the sides of the sprite, reducing its collision detection area.

MotionType Property

Applies To

TSprite

Declaration

```
TMotionType = ( mtStopAtDest, mtContinuous );
property MotionType: TMotionType;
```

Description

The MotionType property controls the behavior of TSprite's default Move method. If the value is mtStopAtDest, the sprite will cease moving as soon as it reaches its Destination. If it is mtContinuous, the sprite will continue in a straight line even after it reaches its Destination.

Moved Property

Applies To

TSprite

Declaration

```
property Moved: boolean;
```

Description

The Moved property is a flag that specifies whether or not the sprite moved during the last execution of its Move method. You can assign True to this property to force the sprite to be redrawn when the DirtyRectangle system of TSpriteEngine is in use.

PhysicalPosition Property

Applies To

TSprite

Declaration

```
property PhysicalPosition: TPoint;
```

Description

The PhysicalPosition property returns the phsical position of the sprite on its DIBDrawingSurface, with the OffsetX and OffsetY applied. PhysicalPosition is defined as follows:

```
PhysicalPosition.X := Position.X - DIBDrawingSurface.OffsetX;
PhysicalPosition.Y := Position.Y - DIBDrawingSurface.OffsetY;
```

When you override the Render method of TSprite, render the sprite using its PhysicalPosition, not its logical Position property.

Position Property

Applies To

TSprite

Declaration

```
property Position: TPoint;
```

Description

The Position property returns the sprite's logical position on the DIBDrawingSurface. You can modify this property to move the sprite to any desired position, but be sure to set the sprite's Destination to the same point, otherwise the sprite will toward it's Destination.

Priority Property

Applies To

TSprite

Declaration

```
property Priority: integer;
```

Description

The Priority property specifies the sprite's Z-Order. Sprites with a lower priority value will appear above other sprites. You can assign a value to this property when initializing a sprite, but once a sprite is registered with a TSpriteEngine you should use the sprite engine's ChangeSpritePriority method to change the value.

Speed Property

Applies To

TSprite

Declaration

```
property Speed: integer;
```

Description

The Speed property controls how fast the sprite moves toward its Destination. The lower the value, the faster the sprite moves.

Tag Property

Applies To
TSprite

Declaration
```
property Tag: integer;
```

Description
The Tag property is simply an integer value you can use to stash miscellaneous information related to a sprite instance.

Visible Property

Applies To

TSprite

Declaration

```
property Visible: boolean;
```

Description

The Visible property is a means to control whether or not the sprite is displayed, without destroying it. Sprites with a Visible property of False will still collide with other sprites during CollisionDetection.

FudgedDistance Method

Applies To

TSprite

Declaration

```
function FudgedDistance( spr: TSprite ): word;
```

Description

The FudgedDistance method is a means to perform a quick distance check between two sprites. The method simply adds the absolute values of the X and Y differences in the two sprites Positions. This method is used in the CollisionDetection method of TSpriteEngine.

Move Method

Applies To

TSprite

Declaration

```
procedure Move; dynamic;
```

Description

The Move method of TSprite provides a default move behavior. The default Move method simply moves the sprite towards its Destination at a constant Speed. You are free to augment or rewrite the Move method in derived classes.

You will never directly call the Move method, but it is called by a TSpriteEngine during the execution of its ProcessSprites method.

RefreshBackground Method

Applies To

TSprite

Declaration

```
procedure RefreshBackground;
```

Description

The RefreshBackground method restores the background of the sprite when it is moved. This method is called by a TSpriteEngine when its DirtyRectangle property is True. You will normally never have to call this method directly.

Render Method

Applies To
TSprite

Declaration
```
procedure Render; dynamic;
```

Description
The Render method is responsible for drawing the sprite on its DIBDrawingSurface. You must override the Render method in classes derived from TSprite. When doing so, it is important to call inherited Render, as this updates some information required by the dirty rectangle system.

When writing your implementation of the Render method, remember to render the sprite using its PhysicalPosition and not Position.

TDIBSprite Class

Unit
DIBSprite

Description
TDIBSprite is a descendant of TSprite. It renders itself by using image data stored in a TDIB.

```
constructor TDIBSprite.CreateDIBSprite( dib: TDIB );
```
The TDIBSprite constructor takes a TDIB as the single parameter. This TDIB contains the image data that will be rendered on the sprite's DIBDrawingSurface.

```
property DIB: TDIB;
```
The DIB property specifies the TDIB that will be used when rendering the sprite. This value is initialized during the call to the TDIBSprite constructor. You are free to modify this value at run time. If the specified TDIB has FramesX and FramesY properties greater than 1, the TDIBSprite's FrameX and FrameY properties should be set to indicate which frame of the image to use when rendering.

```
Property FrameX: integer;
```
The FrameX property controls which frame of image data to display when rendering the sprite. The DIB associated with the sprite must have a FramesX property greater than 1.

```
property FrameY: integer;
```
The FrameY property controls which frame of image data to display when rendering the sprite. The DIB associated with the sprite must have a FramesY property greater than 1.

TPolygonSprite Class

Unit
PolygonSprite

Description
TPolygonSprite is a descendant of TSprite. It renders itself by drawing lines on the DIBDrawingSurface. TPolygonSprite is itself an abstract base class. Derived classes must override the AddVertices method which specifies the polygon vertices.

```
constructor TPolygonSprite.Create( nCol: byte );
```
The TPolygonSprite constructor takes a single parameter, the color index that the polygon lines will be drawn in.

```
procedure AddVertices; virtual; abstract;
```
The AddVertices method is an abstract method that must be overrrided in derived classes. It specifies the vertices of the polygon. The TPolygonSprite contains an instance of a Tpolygon object (which is declared in GRAFIX.PAS), named poly. The implementation of this method should call poly's AddVertex method for each TVertex object that makes up the polygon. Below is a sample implementation of the TSquareSprite derived class:

```
procedure TSquareSprite.AddVertices;
begin
  with poly do
     begin
        AddVertex( TVertex.CreateVertex( -5, -5 ) );
        AddVertex( TVertex.CreateVertex( 5, -5 ) );
        AddVertex( TVertex.CreateVertex( 5, 5 ) );
        AddVertex( TVertex.CreateVertex( -5, 5 ) );
     end;
end;
```

```
property Angle: integer;
```
The Angle property controls the angle of rotation of the polygon.

```
property ColorIndex: byte;
```
The ColorIndex property specifies the color index of the lines that will be drawn when rendering the sprite.

```
property Spin: TRotation;
```
The Spin property controls the direction of spin of the sprite. Possible values are rotNone, rotClockwise, and rotCounterClockwise.

```
property SpinSpeed: integer;
```
The SpinSpeed property determines how quickly the polygon spins. The value equals the number of degrees the polygon will rotate during each call to its Render method.