Blotch3D

Generated by Doxygen 1.8.14

Contents

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Namespace Documentation

4.1 Blotch Namespace Reference

Classes

- class **BlDebug**

  This static class holds the debug flags. Initial flag values are often enabled for Debug builds and disabled for Release builds. Some flags enable exceptions for probable errors, and many flags cause warning messages to be sent to the console window, if it exist. For this reason you should test your app as a debug build console app.

- class BlGraphicsDeviceManager

  This holds everything having to do with an output device. BlWindow3D creates one of these for itself.

- class BlGuiControl

  A 2D GUI control. To create a GUI control: instantiate one of these, set its initial Texture (remember to create it in the 3D thread context), window position, and delegate, and then add it to BlWindow3D.GuiControls. (Any member can be dynamically changed.) The texture will be displayed, and then each frame the mouse is over it the delegate will be called. The delegate typically would examine the current mouse state (Mouse.GetState()) and the PrevMouseState member to detect button changes, etc. and perform an action. The delegate is called in the context of the window's 3D thread after the FrameProc method. You can use Graphics.TextToTexture to create a textual textures, or just load a texture from a content file. Remember to Dispose textures when you are done with them.

- class BlMipmap

  A mipmap of textures for a given BlSprite. You could load this from an image file and then assign it to the Mipmap member of a BlSprite. Note that this is a software mipmap (i.e. it isn't implemented in the 3D hardware). That is, only one resolution texture is used at time.

- class BlSprite

  A BlSprite is a single 3D object. Each sprite can also hold any number of subsprites, so you can make a sprite tree (a scene graph). In that case the child sprites 'follow' the orientation and position of the parent sprite. That is, they exist in the coordinate system of the parent sprite. The location and orientation of a sprite in its parent's coordinate system is defined by the sprite's Matrix member. Subsprites, LODs, and Mipmaps are NOT disposed when the sprite is disposed, so you can assign the same one to multiple sprites. Also see Matrix for more information.

- class BlWindow3D

  To create the 3D window, derive a class from BlWindow3D. Instantiate it and call its Run method from the same thread. When you instantiate it, it will create the 3D window and a separate thread we'll call the "3D thread". All model meshes, textures, fonts, etc. used by the 3D hardware must be created and accessed by the 3D thread, because supported hardware platforms require it. Its safest to assume all Blotch3D and MonoGame objects must be created and accessed in the 3D thread. Although it may apparently work in certain circumstances, do not have the window class constructor create or access any of these things, or have its instance initializers do it, because neither are executed by the 3D thread. To specify code to be executed in the context of the 3D thread, you can override the Setup, FrameProc, and/or FrameDraw methods, and other threads can pass a delegate to the EnqueueCommand and EnqueueCommandBlocking methods. When you override the Setup method it will be called once when the object is first created. You might put time-consuming overall initialization code in there like graphics setting initializations if

different from the defaults, loading of persistent content (models, fonts, etc.), creation of persistent BlSprites, etc. Do not draw things in the 3D window from the setup method. When you override the FrameProc method it will be called once per frame (see BlGraphicsDeviceManager.FramePeriod). You can put code there that should be called periodically. This is typically code that must run at a constant rate, like code that implements smooth sprite and camera movement, etc. Do not draw things in the 3D window from the FrameProc method. When you override the FrameDraw method, the 3D thread calls PrepareDraw just before calling FrameDraw once per frame, but more rarely if CPU is being exhausted. This is where you put drawing code (BlSprite.Draw, BlGraphicsDeviceManager.DrawText, etc.). Finally, if you are developing a multithreaded app, when other threads need to create, change, or destroy 3D resources or otherwise do something in a thread-safe way with the 3D thread, they can queue a delegate to EnqueueCommand or EnqueueCommandBlocking, which makes sure the code is done by the 3D thread sequentially at the end of the current FrameProc. If user input to the 3D window needs to be conveyed back to app threads, you can create thread-safe queues for that as well. This inherits from MonoGame's "Game" class.

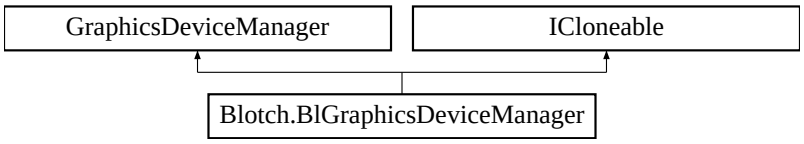
Chapter 5

Class Documentation

5.1 Blotch.BlGraphicsDeviceManager Class Reference

This holds everything having to do with an output device. BlWindow3D creates one of these for itself.

Inheritance diagram for Blotch.BlGraphicsDeviceManager:



Classes

- class Light

  Defines a light. See the BlSprite.Lights field. The default BasicShader supports up to three lights.

Public Member Functions

- BlGraphicsDeviceManager (BlWindow3D window)

  A single BlGraphicsDeviceManager object is automatically created when you create a BlGame object.
- void Initialize ()

  For internal use only. Apps should not normally call this. This initializes some values AFTER the BlWindow has been created.
- void ExtendClippingTo (BlSprite s)

  Informs the auto-clipping code of an object that should be included in the clipping region. This is mainly for internal use. Application code should control clipping with NearClip and FarClip.
- void SetSpriteToCamera (BlSprite sprite)

  Sets a sprite's Matrix to the current camera position and orientation. You could use this to implement a HUD, for example. Note: This only works correctly if the sprite has no parent (and is thus drawn directly) or it's parents are untransformed. If all you want is to set the sprite's position (but NOT orientation) to the camera, then set the sprite's Matrix.Translation = graphics.Eye
- void SetCameraToSprite (BlSprite sprite)

Sets the camera position and orientation to the current position and orientation of a sprite. You could use for cockpit view, for example. Note that the camera will lag sprite movement unless the following is done: For every frame you must first calculate the sprite's position and orientation, call this function, and then draw everything.

- void AdjustCameraZoom (double dif)

  Magnifies the current view. If dif is zero, then there is no change in zoom. Normally one would set zoom with the Zoom field. This is mainly for internal use.
- void AdjustCameraDolly (double dif)

  Migrates the current camera dolly (distance from LookAt) according to dif. If dif is zero, then there is no change in dolly.
- void AdjustCameraTruck (double difX, double difY=0)

  Adjusts camera truck (movement relative to camera direction) according to difX and difY. if difX and difY are zero, then truck position isn't changed.
- void AdjustCameraRotation (double difX, double difY=0)

  Adjusts camera rotation about the LookAt point according to difX and difY. if difX and difY are zero, then rotation isn't changed.
- void AdjustCameraPan (double difX, double difY=0)

  Adjusts camera pan (changing direction of camera) according to difX and difY. if difX and difY are zero, then pan direction isn't changed.
- Ray DoDefaultGui ()

  Updates Eye, LookAt, etc. according to mouse and certain key input. Specifically: Wheel=Dolly, CTRL-wheel=Zoom, Left-drag=Truck, Right-drag=Rotate, CTRL-left-drag=Pan, Esc=Reset. Also, SHIFT causes all the previous controls to be fine rather than coarse. If CTRL is pressed and mouse left or right button is clicked, then returns a ray into window at mouse position.
- void ResetCamera ()

  Sets Eye. LookAt, etc. back to default starting position.
- void SetCameraRollToZero ()

  Sets the camera 'roll' to be level with the XY plane
- Ray CalculateRay (Vector2 windowPosition)

  Returns a ray that that goes from the near clipping plane to the far clipping plane, at the specified window position.
- Vector3 GetWindowCoordinates (BlSprite sprite)

  Returns the window coordinates of the specified sprite
- Texture2D TextToTexture (string text, SpriteFont font, Microsoft.Xna.Framework.Color? color=null, Microsoft.Xna.Framework.Color? backColor=null)

  Returns a BlTexture2D containing the specified text. It's up to the caller to Dispose the returned texture.
- void DrawTexture (Texture2D texture, Rectangle windowRect, Microsoft.Xna.Framework.Color? color=null)

  Draws a texture in the window
- void DrawText (string text, SpriteFont font, Vector2 windowPos, Microsoft.Xna.Framework.Color? color=null)

  Draws text on the window
- Texture2D LoadFromImageFile (string fileName)

  Loads a texture directly from an image file
- void PrepareDraw (bool firstCallInDraw=true)

  This is automatically called once at the beginning of your Draw method. It calculates the latest View and Projection settings according to the current camera specifications (Zoom, Aspect, Eye, LookAt, etc.), and if firstCallInDraw is true it also may sleep in order to obey FramePeriod. It must also be called explicitly after any changes to the camera settings made later in the Draw method. Only in the first call should firstCallInDraw be true, and in any subsequent calls it should be false.
- Texture2D CloneTexture2D (Texture2D tex)

  Returns a deepcopy of the texture
- object **Clone** ()
- new void Dispose ()

  When finished with the object, you should call Dispose() from the same thread that created the object. You can call this multiple times, but once is enough. If it isn't called before the object becomes inaccessible, then the destructor will call it and, if BlDebug.EnableDisposeErrors is true (it is true by default for Debug builds), then it will get an exception saying that it wasn't called by the same thread that created it. This is because the platform's underlying 3D library (OpenGL, etc.) often requires 3D resources to be managed only by one thread. This inherits from MonoGame's GraphicsDeviceManager class.

Public Attributes

- Microsoft.Xna.Framework.Matrix View

  This is the view matrix. Normally you would use the higher-level functions Eye, LookAt, Up, CameraToSprite, and DoDefaultGui intead of changing this directly.
- Microsoft.Xna.Framework.Matrix Projection

  The Projection matrix. Normally you would use the higher-level functions Zoom, Aspect, NearClip, or FarClip intead of changing this directly.
- Vector3 CameraUp

  Camera Up vector. Initially set to +Z. ResetCamera and SetCameraToSprite updates this.
- double DefGuiMinLookZ = -1

  Caues DoDefaultGui to limit the Z component of CameraForwardNormalized above this value. For example, set this to zero so that DoDefaultGui won't allow the camera to look downward
- double DefGuiMaxLookZ = 1

  Caues DoDefaultGui to limit the Z component of CameraForwardNormalized below this value. For example, set this to zero so that DoDefaultGui won't allow the camera to look upward
- DepthStencilState DepthStencilStateEnabled

  Assign DepthStencilState to this to enable depth buffering
- DepthStencilState DepthStencilStateDisabled

  Assign DepthStencilState to this to disable depth buffering
- Vector3 TargetEye

  The point that Eye migrates to, according to CameraSpeed. See Eye for more information.
- Vector3 TargetLookAt

  The point that LookAt migrates to, according to CameraSpeed. See LookAt for more information.
- double CameraSpeed = .4

  The responsiveness of the camera position to changes in TargetEye and TargetLookAt. Zero means it doesn't respond to changes, 1 means it immediately responds. See Eye and LookAt for more information.
- double Zoom =45

  The field of view, in degrees
- double Aspect =2

  The aspect ratio
- double NearClip = 0

  The near clipping plane, or 0 = autoclip
- double FarClip = 0

  The far clipping plane, or 0 = autoclip
- Microsoft.Xna.Framework.Color ClearColor =new Microsoft.Xna.Framework.Color(0,0,.1f)

  The background color
- double AutoRotate = 0

  How fast DoDefaultGui should auto-rotate the scene
- double FramePeriod = 1/60.0

  How much time between each frame
- List< Light > Lights = new List<Light>()

  Information for directional lights. Note: The BasicEffect shader only supports the first three. To handle more lights, you'll need to write your own shader.
- Vector3 AmbientLightColor = new Vector3(.1f, .1f, .1f)

  The ambient light color. If null, no ambient light is enabled. Note: There is no ambient color. Both diffuse and ambient light illuminates the model's Color. See the EsSprite.Color member.
- Vector3 FogColor = null

  If not null, color of fog
- float fogStart = 1

  How far away fog starts. See FogColor
- float fogEnd = 10

How far away fog ends. See FogColor

- SpriteBatch **MySpriteBatch** =null
- bool IsDisposed = false

  Set when the object is Disposed.

Properties

- Vector3 CameraForward `[get]`

  The vector between Eye and LookAt. Writes to Eye and LookAt and calls to SetCameraToSprite cause this to be updated. Also see CameraForwardNormalized and CameraForwardMag.
- Vector3 CameraForwardNormalized `[get]`

  Normalized form of CameraForward. Writes to Eye and LookAt, and calls to SetCameraToSprite cause this to be updated. Also see CameraForward and CameraForwardMag.
- float CameraForwardMag `[get]`

  The magnitude of CameraForward. Writes to Eye and LookAt, and calls to SetCameraToSprite cause this to be updated. Also see CameraForward and CameraForwardNormalized.
- Vector3 CameraRight `[get]`

  Camera Right vector. Writes to Eye and LookAt, and calls to SetCameraToSprite cause this to be updated.
- Vector3 Eye `[get]`

  The current camera position. Note: To change the camera position, set TargetEye. Also see CameraSpeed.
- Vector3 LookAt `[get]`

  The current camera LookAt position. Note: To change the camera LookAt, set TargetLookAt. Also see CameraSpeed.
- double CurrentAspect `[get]`

  Current aspect ratio. Same as Aspect unless Aspect==0.
- double CurrentNearClip `[get]`

  Current value of near clipping plane. See NearClip.
- double CurrentFarClip `[get]`

  Current value of far clipping plane. See FarClip.
- double MinCamDistance `[get]`

  Distance to nearest sprite less its radius. Note this is set to a very large number by PrepareDraw, and then as Draw is called it is set more reasonably.
- double MaxCamDistance `[get]`

  Distance to farthest sprite plus its radius. Note this is set to a very small number by PrepareDraw, and then as Draw is called it is set more reasonably.

5.1.1 Detailed Description

This holds everything having to do with an output device. BlWindow3D creates one of these for itself.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 BlGraphicsDeviceManager()

```
Blotch.BlGraphicsDeviceManager.BlGraphicsDeviceManager (
            BlWindow3D window )
```

A single BlGraphicsDeviceManager object is automatically created when you create a BlGame object.

Parameters

window	The BlWindow3D object for which this is to be the GraphicsDeviceManager

5.1.3 Member Function Documentation

5.1.3.1 AdjustCameraDolly()

```
void Blotch.BlGraphicsDeviceManager.AdjustCameraDolly (
            double dif )
```

Migrates the current camera dolly (distance from LookAt) according to dif. If dif is zero, then there is no change in dolly.

Parameters

dif	How much to dolly camera (plus = toward LookAt, minus = away)

5.1.3.2 AdjustCameraPan()

```
void Blotch.BlGraphicsDeviceManager.AdjustCameraPan (
            double difX,
            double difY = 0 )
```

Adjusts camera pan (changing direction of camera) according to difX and difY. if difX and difY are zero, then pan direction isn't changed.

Parameters

difX	How much to pan horizontally
difY	How much to pan vertically

5.1.3.3 AdjustCameraRotation()

```
void Blotch.BlGraphicsDeviceManager.AdjustCameraRotation (
            double difX,
            double difY = 0 )
```

Adjusts camera rotation about the LookAt point according to difX and difY. if difX and difY are zero, then rotation isn't changed.

Parameters

difX	How much to rotate the camera horizontally
difY	How much to rotate the camera vertically

5.1.3.4 AdjustCameraTruck()

```
void Blotch.BlGraphicsDeviceManager.AdjustCameraTruck (
            double difX,
            double difY = 0 )
```

Adjusts camera truck (movement relative to camera direction) according to difX and difY. if difX and difY are zero, then truck position isn't changed.

Parameters

difX	How much to truck the camera horizontally
difY	How much to truck the camera vertically

5.1.3.5 AdjustCameraZoom()

```
void Blotch.BlGraphicsDeviceManager.AdjustCameraZoom (
            double dif )
```

Magnifies the current view. If dif is zero, then there is no change in zoom. Normally one would set zoom with the Zoom field. This is mainly for internal use.

Parameters

dif	How much to zoom camera (plus = magnify, minus = reduce)

5.1.3.6 CalculateRay()

```
Ray Blotch.BlGraphicsDeviceManager.CalculateRay (
            Vector2 windowPosition )
```

Returns a ray that that goes from the near clipping plane to the far clipping plane, at the specified window position.

Parameters

windowPosition	The window's pixel coordinates

Returns

The Ray into the window at the specified pixel coordinates

5.1.3.7 CloneTexture2D()

```
Texture2D Blotch.BlGraphicsDeviceManager.CloneTexture2D (
            Texture2D tex )
```

Returns a deepcopy of the texture

Parameters

tex	

Returns

5.1.3.8 Dispose()

```
new void Blotch.BlGraphicsDeviceManager.Dispose ( )
```

When finished with the object, you should call Dispose() from the same thread that created the object. You can call this multiple times, but once is enough. If it isn't called before the object becomes inaccessible, then the destructor will call it and, if BlDebug.EnableDisposeErrors is true (it is true by default for Debug builds), then it will get an exception saying that it wasn't called by the same thread that created it. This is because the platform's underlying 3D library (OpenGL, etc.) often requires 3D resources to be managed only by one thread. This inherits from MonoGame's GraphicsDeviceManager class.

5.1.3.9 DoDefaultGui()

```
Ray Blotch.BlGraphicsDeviceManager.DoDefaultGui ( )
```

Updates Eye, LookAt, etc. according to mouse and certain key input. Specifically: Wheel=Dolly, CTR↩
L-wheel=Zoom, Left-drag=Truck, Right-drag=Rotate, CTRL-left-drag=Pan, Esc=Reset. Also, SHIFT causes all the previous controls to be fine rather than coarse. If CTRL is pressed and mouse left or right button is clicked, then returns a ray into window at mouse position.

Returns

If a mouse left or right click occurred, returns the Ray into the screen at that position. Otherwsie returns null

5.1.3.10 DrawText()

```
void Blotch.BlGraphicsDeviceManager.DrawText (
            string text,
            SpriteFont font,
            Vector2 windowPos,
            Microsoft.Xna.Framework.Color?  color = null )
```

Draws text on the window

Parameters

text	The text to draw
font	The font to use (typically created from SpriteFont content with Content.Load<SpriteFont>(...))
windowPos	The X and Y window location, in pixels
color	Foreground color of the font

5.1.3.11 DrawTexture()

```
void Blotch.BlGraphicsDeviceManager.DrawTexture (
            Texture2D texture,
            Rectangle windowRect,
            Microsoft.Xna.Framework.Color?  color = null )
```

Draws a texture in the window

Parameters

texture	The texture to draw
windowRect	The X and Y window location, in pixels
color	Foreground color of the font

5.1.3.12 ExtendClippingTo()

```
void Blotch.BlGraphicsDeviceManager.ExtendClippingTo (
            BlSprite s )
```

Informs the auto-clipping code of an object that should be included in the clipping region. This is mainly for internal use. Application code should control clipping with NearClip and FarClip.

Parameters

s	The sprite that should be included in the auto-clipping code

5.1.3.13 GetWindowCoordinates()

```
Vector3 Blotch.BlGraphicsDeviceManager.GetWindowCoordinates (
            BlSprite sprite )
```

Returns the window coordinates of the specified sprite

Parameters

sprite	The sprite to get the window coordinates of

Returns

The window coordinates of the sprite, in pixels

5.1.3.14 Initialize()

```
void Blotch.BlGraphicsDeviceManager.Initialize ( )
```

For internal use only. Apps should not normally call this. This initializes some values AFTER the BlWindow has been created.

5.1.3.15 LoadFromImageFile()

```
Texture2D Blotch.BlGraphicsDeviceManager.LoadFromImageFile (
            string fileName )
```

Loads a texture directly from an image file

Parameters

fileName	An image file of any standard type supported by MonoGame (jpg, png, etc.)

Returns

The texture that was loaded

5.1.3.16 PrepareDraw()

```
void Blotch.BlGraphicsDeviceManager.PrepareDraw (
            bool firstCallInDraw = true )
```

This is automatically called once at the beginning of your Draw method. It calculates the latest View and Projection settings according to the current camera specifications (Zoom, Aspect, Eye, LookAt, etc.), and if firstCallInDraw is true it also may sleep in order to obey FramePeriod. It must also be called explicitly after any changes to the camera settings made later in the Draw method. Only in the first call should firstCallInDraw be true, and in any subsequent calls it should be false.

Parameters

firstCallInDraw	True indicates this method should also sleep in order to obey FramePeriod.

5.1.3.17 ResetCamera()

```
void Blotch.BlGraphicsDeviceManager.ResetCamera ( )
```

Sets Eye. LookAt, etc. back to default starting position.

5.1.3.18 SetCameraRollToZero()

```
void Blotch.BlGraphicsDeviceManager.SetCameraRollToZero ( )
```

Sets the camera 'roll' to be level with the XY plane

5.1.3.19 SetCameraToSprite()

```
void Blotch.BlGraphicsDeviceManager.SetCameraToSprite (
            BlSprite sprite )
```

Sets the camera position and orientation to the current position and orientation of a sprite. You could use for cockpit view, for example. Note that the camera will lag sprite movement unless the following is done: For every frame you must first calculate the sprite's position and orientation, call this function, and then draw everything.

Parameters

sprite	The sprite that the camera should be connected to

5.1.3.20 SetSpriteToCamera()

```
void Blotch.BlGraphicsDeviceManager.SetSpriteToCamera (
            BlSprite sprite )
```

Sets a sprite's Matrix to the current camera position and orientation. You could use this to implement a HUD, for example. Note: This only works correctly if the sprite has no parent (and is thus drawn directly) or it's parents are untransformed. If all you want is to set the sprite's position (but NOT orientation) to the camera, then set the sprite's Matrix.Translation = graphics.Eye

Parameters

sprite	The sprite that should be connected to the camera

5.1.3.21 TextToTexture()

```
Texture2D Blotch.BlGraphicsDeviceManager.TextToTexture (
            string text,
            SpriteFont font,
            Microsoft.Xna.Framework.Color?  color = null,
            Microsoft.Xna.Framework.Color?  backColor = null )
```

Returns a BlTexture2D containing the specified text. It's up to the caller to Dispose the returned texture.

Parameters

text	The text to write to the texture
font	Font to use
color	If specified, color of the text. (Default is white)
backColor	If specified, background color, like Color.Transparent. If null, then do not clear the background)

Returns

The texture (as a RenderTarget2D). Caller is responsible for Disposing this!

5.1.4 Member Data Documentation

5.1.4.1 AmbientLightColor

```
Vector3 Blotch.BlGraphicsDeviceManager.AmbientLightColor = new Vector3(.1f, .1f, .1f)
```

The ambient light color. If null, no ambient light is enabled. Note: There is no ambient color. Both diffuse and ambient light illuminates the model's Color. See the EsSprite.Color member.

5.1.4.2 Aspect

```
double Blotch.BlGraphicsDeviceManager.Aspect =2
```

The aspect ratio

5.1.4.3 AutoRotate

```
double Blotch.BlGraphicsDeviceManager.AutoRotate = 0
```

How fast DoDefaultGui should auto-rotate the scene

5.1.4.4 CameraSpeed

```
double Blotch.BlGraphicsDeviceManager.CameraSpeed = .4
```

The responsiveness of the camera position to changes in TargetEye and TargetLookAt. Zero means it doesn't respond to changes, 1 means it immediately responds. See Eye and LookAt for more information.

5.1.4.5 CameraUp

```
Vector3 Blotch.BlGraphicsDeviceManager.CameraUp
```

Camera Up vector. Initially set to +Z. ResetCamera and SetCameraToSprite updates this.

5.1.4.6 ClearColor

```
Microsoft.Xna.Framework.Color Blotch.BlGraphicsDeviceManager.ClearColor =new Microsoft.Xna.↩
Framework.Color(0,0,.1f)
```

The background color

5.1.4.7 DefGuiMaxLookZ

```
double Blotch.BlGraphicsDeviceManager.DefGuiMaxLookZ = 1
```

Caues DoDefaultGui to limit the Z component of CameraForwardNormalized below this value. For example, set this to zero so that DoDefaultGui won't allow the camera to look upward

5.1.4.8 DefGuiMinLookZ

```
double Blotch.BlGraphicsDeviceManager.DefGuiMinLookZ = -1
```

Caues DoDefaultGui to limit the Z component of CameraForwardNormalized above this value. For example, set this to zero so that DoDefaultGui won't allow the camera to look downward

5.1.4.9 DepthStencilStateDisabled

```
DepthStencilState Blotch.BlGraphicsDeviceManager.DepthStencilStateDisabled
```

Initial value:

```
= new DepthStencilState()
        {
            DepthBufferEnable = false,
            DepthBufferWriteEnable = false,
            DepthBufferFunction = CompareFunction.Always
        }
```

Assign DepthStencilState to this to disable depth buffering

5.1.4.10 DepthStencilStateEnabled

```
DepthStencilState Blotch.BlGraphicsDeviceManager.DepthStencilStateEnabled
```

Initial value:

```
= new DepthStencilState()
        {
            DepthBufferEnable = true,
            DepthBufferWriteEnable = true,
            DepthBufferFunction = CompareFunction.LessEqual
        }
```

Assign DepthStencilState to this to enable depth buffering

5.1.4.11 FarClip

```
double Blotch.BlGraphicsDeviceManager.FarClip = 0
```

The far clipping plane, or 0 = autoclip

5.1.4.12 FogColor

```
Vector3 Blotch.BlGraphicsDeviceManager.FogColor = null
```

If not null, color of fog

5.1.4.13 fogEnd

```
float Blotch.BlGraphicsDeviceManager.fogEnd = 10
```

How far away fog ends. See FogColor

5.1.4.14 fogStart

```
float Blotch.BlGraphicsDeviceManager.fogStart = 1
```

How far away fog starts. See FogColor

5.1.4.15 FramePeriod

```
double Blotch.BlGraphicsDeviceManager.FramePeriod = 1/60.0
```

How much time between each frame

5.1.4.16 IsDisposed

```
bool Blotch.BlGraphicsDeviceManager.IsDisposed = false
```

Set when the object is Disposed.

5.1.4.17 Lights

```
List<Light> Blotch.BlGraphicsDeviceManager.Lights = new List<Light>()
```

Information for directional lights. Note: The BasicEffect shader only supports the first three. To handle more lights, you'll need to write your own shader.

5.1.4.18 NearClip

```
double Blotch.BlGraphicsDeviceManager.NearClip = 0
```

The near clipping plane, or 0 = autoclip

5.1.4.19 Projection

```
Microsoft.Xna.Framework.Matrix Blotch.BlGraphicsDeviceManager.Projection
```

The Projection matrix. Normally you would use the higher-level functions Zoom, Aspect, NearClip, or FarClip intead of changing this directly.

5.1.4.20 TargetEye

```
Vector3 Blotch.BlGraphicsDeviceManager.TargetEye
```

The point that Eye migrates to, according to CameraSpeed. See Eye for more information.

5.1.4.21 TargetLookAt

```
Vector3 Blotch.BlGraphicsDeviceManager.TargetLookAt
```

The point that LookAt migrates to, according to CameraSpeed. See LookAt for more information.

5.1.4.22 View

```
Microsoft.Xna.Framework.Matrix Blotch.BlGraphicsDeviceManager.View
```

This is the view matrix. Normally you would use the higher-level functions Eye, LookAt, Up, CameraToSprite, and DoDefaultGui intead of changing this directly.

5.1.4.23 Zoom

```
double Blotch.BlGraphicsDeviceManager.Zoom =45
```

The field of view, in degrees

5.1.5 Property Documentation

5.1.5.1 CameraForward

```
Vector3 Blotch.BlGraphicsDeviceManager.CameraForward [get]
```

The vector between Eye and LookAt. Writes to Eye and LookAt and calls to SetCameraToSprite cause this to be updated. Also see CameraForwardNormalized and CameraForwardMag.

5.1.5.2 CameraForwardMag

```
float Blotch.BlGraphicsDeviceManager.CameraForwardMag [get]
```

The magnitude of CameraForward. Writes to Eye and LookAt, and calls to SetCameraToSprite cause this to be updated. Also see CameraForward and CameraForwardNormalized.

5.1.5.3 CameraForwardNormalized

```
Vector3 Blotch.BlGraphicsDeviceManager.CameraForwardNormalized [get]
```

Normalized form of CameraForward. Writes to Eye and LookAt, and calls to SetCameraToSprite cause this to be updated. Also see CameraForward and CameraForwardMag.

5.1.5.4 CameraRight

```
Vector3 Blotch.BlGraphicsDeviceManager.CameraRight [get]
```

Camera Right vector. Writes to Eye and LookAt, and calls to SetCameraToSprite cause this to be updated.

5.1.5.5 CurrentAspect

```
double Blotch.BlGraphicsDeviceManager.CurrentAspect [get]
```

Current aspect ratio. Same as Aspect unless Aspect==0.

5.1.5.6 CurrentFarClip

```
double Blotch.BlGraphicsDeviceManager.CurrentFarClip [get]
```

Current value of far clipping plane. See FarClip.

5.1.5.7 CurrentNearClip

```
double Blotch.BlGraphicsDeviceManager.CurrentNearClip  [get]
```

Current value of near clipping plane. See NearClip.

5.1.5.8 Eye

```
Vector3 Blotch.BlGraphicsDeviceManager.Eye  [get]
```

The current camera position. Note: To change the camera position, set TargetEye. Also see CameraSpeed.

5.1.5.9 LookAt

```
Vector3 Blotch.BlGraphicsDeviceManager.LookAt  [get]
```

The current camera LookAt position. Note: To change the camera LookAt, set TargetLookAt. Also see Camera↩ Speed.

5.1.5.10 MaxCamDistance

```
double Blotch.BlGraphicsDeviceManager.MaxCamDistance  [get]
```

Distance to farthest sprite plus its radius. Note this is set to a very small number by PrepareDraw, and then as Draw is called it is set more reasonably.

5.1.5.11 MinCamDistance

```
double Blotch.BlGraphicsDeviceManager.MinCamDistance  [get]
```

Distance to nearest sprite less its radius. Note this is set to a very large number by PrepareDraw, and then as Draw is called it is set more reasonably.

The documentation for this class was generated from the following file:

- C:/Users/kloum/Desktop/Source/Blotch3D/src/BlGraphicsDeviceManager.cs

5.2 Blotch.BlGuiControl Class Reference

A 2D GUI control. To create a GUI control: instantiate one of these, set its initial Texture (remember to create it in the 3D thread context), window position, and delegate, and then add it to BlWindow3D.GuiControls. (Any member can be dynamically changed.) The texture will be displayed, and then each frame the mouse is over it the delegate will be called. The delegate typically would examine the current mouse state (Mouse.GetState()) and the Prev↩ MouseState member to detect button changes, etc. and perform an action. The delegate is called in the context of the window's 3D thread after the FrameProc method. You can use Graphics.TextToTexture to create a textual textures, or just load a texture from a content file. Remember to Dispose textures when you are done with them.

Public Member Functions

- delegate void OnMouseChangeDelegate (BlGuiControl guiCtrl)

  Delegates for a BlGuiControl are of this type
- **BlGuiControl** (BlWindow3D window)
- bool HandleInput ()

  Periodically called by BlWindow3D. You shouldn't need to call this.

Public Attributes

- Texture2D Texture = null

  The texture to display for this control. Don't forget to dispose it when done.
- Vector2 Position = Vector2.Zero

  The pixel position in the BlWindow3D of this control
- OnMouseChangeDelegate OnMouseOver = null

  The delegate to call each frame (from the 3D thread) when the mouse is over the control. A typical delegate would make a decision according to guiCtrl.PrevMouseState and the current mouse state (Mouse.GetState).
- MouseState PrevMouseState = new MouseState()

  The previous mouse state. A delegte typiclly uses this along with the current mouse state to make a decision.
- BlWindow3D Window = null

  The window this BlGuiControl is in.

5.2.1 Detailed Description

A 2D GUI control. To create a GUI control: instantiate one of these, set its initial Texture (remember to create it in the 3D thread context), window position, and delegate, and then add it to BlWindow3D.GuiControls. (Any member can be dynamically changed.) The texture will be displayed, and then each frame the mouse is over it the delegate will be called. The delegate typically would examine the current mouse state (Mouse.GetState()) and the Prev↩ MouseState member to detect button changes, etc. and perform an action. The delegate is called in the context of the window's 3D thread after the FrameProc method. You can use Graphics.TextToTexture to create a textual textures, or just load a texture from a content file. Remember to Dispose textures when you are done with them.

5.2.2 Member Function Documentation

5.2.2.1 HandleInput()

```
bool Blotch.BlGuiControl.HandleInput ( )
```

Periodically called by BlWindow3D. You shouldn't need to call this.

Returns

True if mouse is over any control, false otherwise.

5.2.2.2 OnMouseChangeDelegate()

```
delegate void Blotch.BlGuiControl.OnMouseChangeDelegate (
            BlGuiControl guiCtrl )
```

Delegates for a BlGuiControl are of this type

Parameters

guiCtrl	

5.2.3 Member Data Documentation

5.2.3.1 OnMouseOver

```
OnMouseChangeDelegate Blotch.BlGuiControl.OnMouseOver = null
```

The delegate to call each frame (from the 3D thread) when the mouse is over the control. A typical delegate would make a decision according to guiCtrl.PrevMouseState and the current mouse state (Mouse.GetState).

5.2.3.2 Position

```
Vector2 Blotch.BlGuiControl.Position = Vector2.Zero
```

The pixel position in the BlWindow3D of this control

5.2.3.3 PrevMouseState

```
MouseState Blotch.BlGuiControl.PrevMouseState = new MouseState()
```

The previous mouse state. A delegte typiclly uses this along with the current mouse state to make a decision.

5.2.3.4 Texture

```
Texture2D Blotch.BlGuiControl.Texture = null
```

The texture to display for this control. Don't forget to dispose it when done.

5.2.3.5 Window

```
BlWindow3D Blotch.BlGuiControl.Window = null
```

The window this BlGuiControl is in.

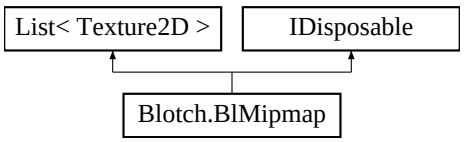
The documentation for this class was generated from the following file:

- C:/Users/kloum/Desktop/Source/Blotch3D/src/BlGuiControl.cs

5.3 Blotch.BlMipmap Class Reference

A mipmap of textures for a given BlSprite. You could load this from an image file and then assign it to the Mipmap member of a BlSprite. Note that this is a software mipmap (i.e. it isn't implemented in the 3D hardware). That is, only one resolution texture is used at time.

Inheritance diagram for Blotch.BlMipmap:



Public Member Functions

- BlMipmap (BlGraphicsDeviceManager graphics, Texture2D tex, int numMaps=999, bool reverseX=false, bool reverseY=false)

  Creates the mipmaps.
- void Dispose ()

  When finished with the object, you should call Dispose() from the same thread that created the object. You can call this multiple times, but once is enough. If it isn't called before the object becomes inaccessible, then the destructor will call it and, if BlDebug.EnableDisposeErrors is true (it is true by default for Debug builds), then it will get an exception saying that it wasn't called by the same thread that created it. This is because the platform's underlying 3D library (OpenGL, etc.) often requires 3D resources to be managed only by one thread.

Public Attributes

- bool IsDisposed = false

  Set when the object is Disposed.

5.3.1 Detailed Description

A mipmap of textures for a given BlSprite. You could load this from an image file and then assign it to the Mipmap member of a BlSprite. Note that this is a software mipmap (i.e. it isn't implemented in the 3D hardware). That is, only one resolution texture is used at time.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 BlMipmap()

```
Blotch.BlMipmap.BlMipmap (
            BlGraphicsDeviceManager graphics,
            Texture2D tex,
            int numMaps = 999,
            bool reverseX = false,
            bool reverseY = false )
```

Creates the mipmaps.

Parameters

graphics	Graphics device (typically the one owned by your BlWindow3D)
tex	Texture from which to create mipmaps, typically gotten from BlGraphics.LoadFromImageFile.
numMaps	Maximum number of mipmaps to create (none are created with lower resolution than 16x16)
reverseX	Whether to reverse pixels horizontally
reverseY	Whether to reverse pixels vertically

5.3.3 Member Function Documentation

5.3.3.1 Dispose()

```
void Blotch.BlMipmap.Dispose ( )
```

When finished with the object, you should call Dispose() from the same thread that created the object. You can call this multiple times, but once is enough. If it isn't called before the object becomes inaccessible, then the destructor will call it and, if BlDebug.EnableDisposeErrors is true (it is true by default for Debug builds), then it will get an exception saying that it wasn't called by the same thread that created it. This is because the platform's underlying 3D library (OpenGL, etc.) often requires 3D resources to be managed only by one thread.

5.3.4 Member Data Documentation

5.3.4.1 IsDisposed

```
bool Blotch.BlMipmap.IsDisposed = false
```

Set when the object is Disposed.

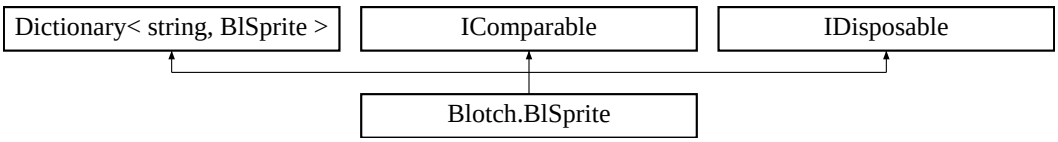
The documentation for this class was generated from the following file:

- C:/Users/kloum/Desktop/Source/Blotch3D/src/BlMipmap.cs

5.4 Blotch.BlSprite Class Reference

A BlSprite is a single 3D object. Each sprite can also hold any number of subsprites, so you can make a sprite tree (a scene graph). In that case the child sprites 'follow' the orientation and position of the parent sprite. That is, they exist in the coordinate system of the parent sprite. The location and orientation of a sprite in its parent's coordinate system is defined by the sprite's Matrix member. Subsprites, LODs, and Mipmaps are NOT disposed when the sprite is disposed, so you can assign the same one to multiple sprites. Also see Matrix for more information.

Inheritance diagram for Blotch.BlSprite:



Public Types

- enum PreDrawCmd { PreDrawCmd.Continue, PreDrawCmd.Abort, PreDrawCmd.UseCurrentAbsoluteMatrix }

  Return code from PreDraw callback. This tells Draw what to do next.
- enum PreSubspritesCmd { PreSubspritesCmd.Continue, PreSubspritesCmd.Abort, PreSubspritesCmd.DontDrawSubsprites }

  Return code from PreSubsprites callback. This tells Draw what to do next.
- enum PreMeshDrawCmd { PreMeshDrawCmd.Continue, PreMeshDrawCmd.Abort, PreMeshDrawCmd.Skip }

  Return code from PreSubsprites callback. This tells Draw what to do next.
- enum PreLocalCmd { PreLocalCmd.Continue, PreLocalCmd.Abort }

  Return code from PreSubsprites callback. This tells Draw what to do next.

Public Member Functions

- delegate PreDrawCmd PreDrawType (BISprite sprite)

  See PreDraw
- delegate PreSubspritesCmd PreSubspritesType (BISprite sprite)

  See PreSubsprites
- delegate PreMeshDrawCmd PreMeshDrawType (BISprite sprite, ModelMesh mesh)

  See PreMeshDraw
- delegate PreLocalCmd PreLocalType (BISprite sprite)

  See PreLocal
- delegate void DrawCleanupType (BISprite sprite)

  See DrawCleanup
- **BISprite** (BIGraphicsDeviceManager graphicsIn, string name)
- void **Add** (BISprite s)
- Vector2 GetViewCoords ()

  Returns the current view coordinates of the sprite (for passing to DrawText, for example), or null if it's behind the camera.
- void SetAllMaterialBlack ()

  Sets all material colors to black.
- double DoesRayIntersect (Ray ray)

  Returns the distance along the ray to the first point the ray enters the bounding sphere (BoundSphere), or null if it doesn't enter the sphere.
- List< BISprite > GetRayIntersections (Ray ray, ulong flags=0xFFFFFFFFFFFFFFFF, List< BISprite > sprites=null)

  Returns a list of subsprites that the ray hit (i.e. those that were within their radius of the ray)
- void Draw (Matrix? worldMatrixIn=null, ulong flagsIn=0xFFFFFFFFFFFFFFFF)

  Draws the sprite and the subsprites.
- override string **ToString** ()
- int CompareTo (object obj)

  This makes a Sort operation sort sprites far to near. That is, the nearer sprites are later in the list. For sorting near to far, use something like myList.Sort(new Comparison<EsSprite>((b, a) => a.CompareTo(b)));
- void Dispose ()

  When finished with the object, you should call Dispose() from the same thread that created the object. You can call this multiple times, but once is enough. If it isn't called before the object becomes inaccessible, then the destructor will call it and, if BIDebug.EnableDisposeErrors is true (it is true by default for Debug builds), then it will get an exception saying that it wasn't called by the same thread that created it. This is because the platform's underlying 3D library (OpenGL, etc.) often requires 3D resources to be managed only by one thread.

Static Public Member Functions

- static Vector3 NearestPointOnLine (Vector3 point1, Vector3 point2, Vector3 nearPoint)

  Returns the point on the line between point1 and point2 that is nearest to nearPoint

Public Attributes

- ulong Flags = 0xFFFFFFFFFFFFFFFF

  The Flags field can be used by callbacks of Draw (PreDraw, PreSubspriteDraw, PreLocalDraw, and PreMeshDraw) to indicate various user attributes of the sprite. Also, GetRayIntersections aborts if the bitwise AND of this value and the flags argument passed to it is zero.
- List< object > LODs = new List<object>()

The object drawn for this sprite. Specifically, this is a list of levels of detail (LOD), where only one is drawn depending on the ApparentSize. Each element can be a Model, a triangle list (VertexPositionNormalTexture[]), or null (indicating nothing should be drawn). Elements with lower indices are higher LODs. So index 0 is the highest, index 1 is second highest, etc. LOD decreases (the index increases) for every halving of the object's apparent size. You can adjust how close the LODs must be to the camera with LodScale (see LodScale). When the calculated LOD index (see LodCurrentIndex) is higher than the last element, then the last element is used. So the simplest way to use this is to add a single element of the object you want drawn. You can also add multiple references of the same object so multiple consecutive LODs draw the same object. You can also set an element to null so it doesn't draw anything, which is typically the last element. A model can be assigned to multiple sprites. These are NOT disposed when the sprite is disposed.

- double LodScale = 9

  Defines the LOD scaling. The higher this value, the closer you must be to see a given LOD. A value of 9 (default) indicates that the highest LOD (LODs[0]) occurs when an object with a diameter of 1 roughly fills the window.

- BlMipmap Mipmap = null

  Mipmap textures to apply to the model. These work the same as LODs (see LODs for more information). The texture used depends on the apparent size of the model. The next higher mipmap is used for every doubling of model size, where element zero is the highest resolution, used when the apparent size is largest. If a mipmap is not available for the apparent size, the next higher available on is used. So, for example, you can specify only one texture to be used as all mipmaps if you like. Note that for a texture to display, the model must include texture coordinates. Most graphics subsystems do support mipmaps, but these are supported at the app level. Therefore only one image is used over a model for a given model apparent size, rather than nearer portions of the model showing higher-level mipmaps. These are NOT disposed when the sprite is disposed. A given BlMipmap may be assigned to multiple sprites.

- double MipmapScale = 5

  Defines the mipmap (Textures) scaling. The higher this value, the closer you must be to see a given mipmap.

- BoundingSphere BoundSphere = null

  The bounding sphere for this sprite. This is automatically updated when a model is drawn, but not if vertices are drawn. In that case you should set/update it explicitly if any of the internal functions may need it to be roughly correct, like if auto-clipping is enabled or a mouse selection or ray may hit the sprite and the hit be properly detected.

- bool SphericalBillboard = false

  Spherically billboard the model. Specifically, keep the model's 'forward' direction pointing at the camera and keep its 'Up' direction pointing in the same direction as the camera's 'Up' direction. Also see CylindricalBillboardX, CylindricalBillboardY, CylindricalBillboardZ, and ConstSize.

- Vector3 CylindricalBillboardX = Vector3.Zero

  If non-zero, this is the rotation vector and magnitude of cylindrical billboarding where the angle calculation assumes this vector is the X axis, even though it may not be. The more this varies from that axis, the more eccentric the billboarding behavior. The amount of billboarding is equal to: 2∗mag^2 - 1/mag^2. So if this vector's magnitude is unity (1), then full cylindrical billboarding occurs. A vector magnitude of 0.605 produces double reverse cylindrical billboarding. Also see SphericalBillboard, CylindricalBillboardY, CylindricalBillboardZ, and ConstSize.

- Vector3 CylindricalBillboardY = Vector3.Zero

  If non-zero, this is the rotation vector and magnitude of cylindrical billboarding where the angle calculation assumes this vector is the Y axis, even though it may not be. The more this varies from that axis, the more eccentric the billboarding behavior. The amount of billboarding is equal to: 2∗mag^2 - 1/mag^2. So if this vector's magnitude is unity (1), then full cylindrical billboarding occurs. A vector magnitude of 0.605 produces double reverse cylindrical billboarding. Also see SphericalBillboard, CylindricalBillboardX, CylindricalBillboardZ, and ConstSize.

- Vector3 CylindricalBillboardZ = Vector3.Zero

  If non-zero, this is the rotation vector and magnitude of cylindrical billboarding where the angle calculation assumes this vector is the Z axis, even though it may not be. The more this varies from that axis, the more eccentric the billboarding behavior. The amount of billboarding is equal to: 2∗mag^2 - 1/mag^2. So if this vector's magnitude is unity (1), then full cylindrical billboarding occurs. A vector magnitude of 0.605 produces double reverse cylindrical billboarding. Also see SphericalBillboard, CylindricalBillboardX, CylindricalBillboardY, and ConstSize.

- bool ConstSize = false

  If true, maintain a constant apparent size for the sprite regardless of camera distance or zoom. This is typically used along with one of the Billboarding effects (see SphericalBillboard, CylindricalBillboardX, etc.). If both ConstSize and any Billboarding is enabled and you have asymmetric scaling (different scaling for each dimension), then you'll need to separate those operations into different levels of the sprite tree to obtain the desired behavior. You'll also probably want to disable the depth stencil buffer and control which sprite is drawn first so that certain sprites are 'always on top'. See the examples.

- Matrix AbsoluteMatrix = Matrix.Identity

The Draw method takes an incoming 'world' matrix parameter which is the coordinate system of its parent. Absolute↩ Matrix is that incoming world matrix parameter times the Matrix member and altered according to Billboarding and ConstSize. This is not read-only because a callback (see PreDraw, PreSubspritesDraw, PreLocalDraw, and Pre↩ MeshDraw) may need to change it from within the Draw method. This is the matrix that is also passed to subsprites as their 'world' matrix.

- Matrix Matrix = Matrix.Identity

  The matrix for this sprite. This defines the sprite's orientation and position relative to the parent coordinate system. For more detailed information, see AbsoluteMatrix.
- BlGraphicsDeviceManager Graphics = null

  Current incoming graphics parameter to the Draw method. Typically this would be of interest to a callback function (see PreDraw, PreSubspritesDraw, PreLocalDraw, and PreMeshDraw).
- Matrix LastWorldMatrix = null

  Current incoming world matrix parameter to the Draw method. Typically this would be of interest to a callback function (see PreDraw, PreSubspritesDraw, PreLocalDraw, and PreMeshDraw).
- bool IncludeInAutoClipping = true

  Whether to use depth testing, and whether to participate in autoclipping calculations when they are enabled.
- ulong FlagsParameter = 0

  Current incoming flags parameter to the Draw method. Typically this would be of interest to a callback function (see PreDraw, PreSubspritesDraw, PreLocalDraw, and PreMeshDraw).
- Vector3 Color = new Vector3(.5f, .5f, 1)

  The color of the material. This is lit by both diffuse and ambient light. If null, MonoGame's default color is kept.
- Vector3 EmissiveColor = new Vector3(.1f, .1f, .2f)

  The emissive color. If null, MonoGame's default is kept.
- Vector3 SpecularColor = null

  The specular color. If null, MonoGame's default is kept.
- float SpecularPower = 8

  If a specular color is specified, this is the specular power.
- PreDrawType PreDraw = null

  If not null, Draw method calls this at the beginning before doing anything else. From this function one might examine and/or alter any public writable EsSprite field, and/or control the further execution of the Draw method.
- PreSubspritesType PreSubsprites = null

  If not null, Draw method calls this after the matrix calculations for AbsoluteMatrix (including billboards, CamDistance, ConstSize, etc.) but before drawing the subsprites or local model. From this function one might examine and/or alter any public writable EsSprite field.
- PreMeshDrawType PreMeshDraw = null

  If not null, Draw method calls this before each model mesh is drawn for the local model. From this function one might examine and/or alter any public writable EsSprite field. If the return value is true, then the mesh will not be drawn.
- PreLocalType PreLocal = null

  If not null, Draw method calls this after drawing subsprites (if appropriate) but before drawing the local model. From this function one might examine and/or alter any public writable EsSprite field, and/or abort the Draw method.
- DrawCleanupType DrawCleanup = null

  If not null, Draw method calls this at the end of the Draw method.
- string Name

  The name of the EsSprite
- bool IsDisposed = false

  Set when the object is Disposed.

Properties

- double ApparentSize `[get]`

  This is proportional to the apparent 2D size of the sprite. (Calculated from the last Draw operation that occurred, but before any effect of ConstSize)
- double LodTarget `[get]`

This read-only value is the log of the reciprocal of ApparentSize. It is used in the calculation of the LOD and the mipmap level. See LODs and Mipmap for more information.

- BasicEffect VerticesEffect `[get, set]`

  BasicEffect used to draw vertices. If not explicitly set, then use a default BasicEffect and dispose it when the BlSprite is disposed. If explicitly set, then don't dispose it when the BlSprite is disposed.

- double CamDistance `[get]`

  Distance to the camera.

5.4.1 Detailed Description

A BlSprite is a single 3D object. Each sprite can also hold any number of subsprites, so you can make a sprite tree (a scene graph). In that case the child sprites 'follow' the orientation and position of the parent sprite. That is, they exist in the coordinate system of the parent sprite. The location and orientation of a sprite in its parent's coordinate system is defined by the sprite's Matrix member. Subsprites, LODs, and Mipmaps are NOT disposed when the sprite is disposed, so you can assign the same one to multiple sprites. Also see Matrix for more information.

5.4.2 Member Enumeration Documentation

5.4.2.1 PreDrawCmd

`enum Blotch.BlSprite.PreDrawCmd [strong]`

Return code from PreDraw callback. This tells Draw what to do next.

Enumerator

Continue	Continue Draw method execution
Abort	Draw should immediately return
UseCurrentAbsoluteMatrix	Continue Draw method execution, but don't bother re-calculating AbsoluteMatrix. One would typically return this if, for example, its known that AbsoluteMatrix will not change from its current value because the Draw parameters will be the same as they were the last time Draw was called. This happens, for example, when multiple calls are being made in the same draw iteration for graphic operations that require multiple passes, like proper handling of translucency, etc.

5.4.2.2 PreLocalCmd

`enum Blotch.BlSprite.PreLocalCmd [strong]`

Return code from PreSubsprites callback. This tells Draw what to do next.

Enumerator

Continue	Continue Draw method execution
Abort	Draw should immediately return

5.4.2.3 PreMeshDrawCmd

```
enum Blotch.BlSprite.PreMeshDrawCmd  [strong]
```

Return code from PreSubsprites callback. This tells Draw what to do next.

Enumerator

Continue	Continue Draw method execution
Abort	Draw should immediately return
Skip	Draw should skip the current mesh

5.4.2.4 PreSubspritesCmd

```
enum Blotch.BlSprite.PreSubspritesCmd  [strong]
```

Return code from PreSubsprites callback. This tells Draw what to do next.

Enumerator

Continue	Continue Draw method execution
Abort	Draw should immediately return
DontDrawSubsprites	Skip drawing subsprites

5.4.3 Member Function Documentation

5.4.3.1 CompareTo()

```
int Blotch.BlSprite.CompareTo (
            object obj )
```

This makes a Sort operation sort sprites far to near. That is, the nearer sprites are later in the list. For sorting near to far, use something like myList.Sort(new Comparison<EsSprite>((b, a) => a.CompareTo(b)));

Parameters

obj	

Returns

5.4.3.2 Dispose()

```
void Blotch.BlSprite.Dispose ( )
```

When finished with the object, you should call Dispose() from the same thread that created the object. You can call this multiple times, but once is enough. If it isn't called before the object becomes inaccessible, then the destructor will call it and, if BlDebug.EnableDisposeErrors is true (it is true by default for Debug builds), then it will get an exception saying that it wasn't called by the same thread that created it. This is because the platform's underlying 3D library (OpenGL, etc.) often requires 3D resources to be managed only by one thread.

5.4.3.3 DoesRayIntersect()

```
double Blotch.BlSprite.DoesRayIntersect (
            Ray ray )
```

Returns the distance along the ray to the first point the ray enters the bounding sphere (BoundSphere), or null if it doesn't enter the sphere.

Parameters

ray	
boundingSphere	

Returns

5.4.3.4 Draw()

```
void Blotch.BlSprite.Draw (
            Matrix?  worldMatrixIn = null,
            ulong flagsIn = 0xFFFFFFFFFFFFFFFF )
```

Draws the sprite and the subsprites.

Parameters

worldMatrixIn	Defines the position and orientation of the sprite
flagsIn	Copied to LastFlags for use by any callback of Draw (PreDraw, PreSubspriteDraw, PreLocalDraw, and PreMeshDraw) that wants it

5.4.3.5 DrawCleanupType()

```
delegate void Blotch.BlSprite.DrawCleanupType (
            BlSprite sprite )
```

See DrawCleanup

Parameters

sprite	

5.4.3.6 GetRayIntersections()

```
List<BlSprite> Blotch.BlSprite.GetRayIntersections (
            Ray ray,
            ulong flags = 0xFFFFFFFFFFFFFFFF,
            List< BlSprite > sprites = null )
```

Returns a list of subsprites that the ray hit (i.e. those that were within their radius of the ray)

Parameters

ray	
flags	
sprites	

Returns

5.4.3.7 GetViewCoords()

```
Vector2 Blotch.BlSprite.GetViewCoords ( )
```

Returns the current view coordinates of the sprite (for passing to DrawText, for example), or null if it's behind the camera.

Returns

5.4.3.8 NearestPointOnLine()

```
static Vector3 Blotch.BlSprite.NearestPointOnLine (
            Vector3 point1,
            Vector3 point2,
            Vector3 nearPoint )  [static]
```

Returns the point on the line between point1 and point2 that is nearest to nearPoint

Parameters

point1	
point2	
nearPoint	

Returns

5.4.3.9 PreDrawType()

```
delegate PreDrawCmd Blotch.BlSprite.PreDrawType (
            BlSprite sprite )
```

See PreDraw

Parameters

sprite	

Returns

5.4.3.10 PreLocalType()

```
delegate PreLocalCmd Blotch.BlSprite.PreLocalType (
            BlSprite sprite )
```

See PreLocal

Parameters

sprite	

Returns

5.4.3.11 PreMeshDrawType()

```
delegate PreMeshDrawCmd Blotch.BlSprite.PreMeshDrawType (
            BlSprite sprite,
            ModelMesh mesh )
```

See PreMeshDraw

Parameters

sprite	
mesh	

Returns

5.4.3.12 PreSubspritesType()

```
delegate PreSubspritesCmd Blotch.BlSprite.PreSubspritesType (
            BlSprite sprite )
```

See PreSubsprites

Parameters

sprite	

Returns

5.4.3.13 SetAllMaterialBlack()

```
void Blotch.BlSprite.SetAllMaterialBlack ( )
```

Sets all material colors to black.

5.4.4 Member Data Documentation

5.4.4.1 AbsoluteMatrix

```
Matrix Blotch.BlSprite.AbsoluteMatrix = Matrix.Identity
```

The Draw method takes an incoming 'world' matrix parameter which is the coordinate system of its parent. AbsoluteMatrix is that incoming world matrix parameter times the Matrix member and altered according to Billboarding and ConstSize. This is not read-only because a callback (see PreDraw, PreSubspritesDraw, PreLocal↩ Draw, and PreMeshDraw) may need to change it from within the Draw method. This is the matrix that is also passed to subsprites as their 'world' matrix.

5.4.4.2 BoundSphere

```
BoundingSphere Blotch.BlSprite.BoundSphere = null
```

The bounding sphere for this sprite. This is automatically updated when a model is drawn, but not if vertices are drawn. In that case you should set/update it explicitly if any of the internal functions may need it to be roughly correct, like if auto-clipping is enabled or a mouse selection or ray may hit the sprite and the hit be properly detected.

5.4.4.3 Color

```
Vector3 Blotch.BlSprite.Color = new Vector3(.5f, .5f, 1)
```

The color of the material. This is lit by both diffuse and ambient light. If null, MonoGame's default color is kept.

5.4.4.4 ConstSize

```
bool Blotch.BlSprite.ConstSize = false
```

If true, maintain a constant apparent size for the sprite regardless of camera distance or zoom. This is typically used along with one of the Billboarding effects (see SphericalBillboard, CylindricalBillboardX, etc.). If both ConstSize and any Billboarding is enabled and you have asymmetric scaling (different scaling for each dimension), then you'll need to separate those operations into different levels of the sprite tree to obtain the desired behavior. You'll also probably want to disable the depth stencil buffer and control which sprite is drawn first so that certain sprites are 'always on top'. See the examples.

5.4.4.5 CylindricalBillboardX

```
Vector3 Blotch.BlSprite.CylindricalBillboardX = Vector3.Zero
```

If non-zero, this is the rotation vector and magnitude of cylindrical billboarding where the angle calculation assumes this vector is the X axis, even though it may not be. The more this varies from that axis, the more eccentric the billboarding behavior. The amount of billboarding is equal to: 2∗mag^2 - 1/mag^2. So if this vector's magnitude is unity (1), then full cylindrical billboarding occurs. A vector magnitude of 0.605 produces double reverse cylindrical billboarding. Also see SphericalBillboard, CylindricalBillboardY, CylindricalBillboardZ, and ConstSize.

5.4.4.6 CylindricalBillboardY

```
Vector3 Blotch.BlSprite.CylindricalBillboardY = Vector3.Zero
```

If non-zero, this is the rotation vector and magnitude of cylindrical billboarding where the angle calculation assumes this vector is the Y axis, even though it may not be. The more this varies from that axis, the more eccentric the billboarding behavior. The amount of billboarding is equal to: 2∗mag^2 - 1/mag^2. So if this vector's magnitude is unity (1), then full cylindrical billboarding occurs. A vector magnitude of 0.605 produces double reverse cylindrical billboarding. Also see SphericalBillboard, CylindricalBillboardX, CylindricalBillboardZ, and ConstSize.

5.4.4.7 CylindricalBillboardZ

```
Vector3 Blotch.BlSprite.CylindricalBillboardZ = Vector3.Zero
```

If non-zero, this is the rotation vector and magnitude of cylindrical billboarding where the angle calculation assumes this vector is the Z axis, even though it may not be. The more this varies from that axis, the more eccentric the billboarding behavior. The amount of billboarding is equal to: 2∗mag^2 - 1/mag^2. So if this vector's magnitude is unity (1), then full cylindrical billboarding occurs. A vector magnitude of 0.605 produces double reverse cylindrical billboarding. Also see SphericalBillboard, CylindricalBillboardX, CylindricalBillboardY, and ConstSize.

5.4.4.8 DrawCleanup

```
DrawCleanupType Blotch.BlSprite.DrawCleanup = null
```

If not null, Draw method calls this at the end of the Draw method.

5.4.4.9 EmissiveColor

```
Vector3 Blotch.BlSprite.EmissiveColor = new Vector3(.1f, .1f, .2f)
```

The emissive color. If null, MonoGame's default is kept.

5.4.4.10 Flags

```
ulong Blotch.BlSprite.Flags = 0xFFFFFFFFFFFFFFFF
```

The Flags field can be used by callbacks of Draw (PreDraw, PreSubspriteDraw, PreLocalDraw, and PreMeshDraw) to indicate various user attributes of the sprite. Also, GetRayIntersections aborts if the bitwise AND of this value and the flags argument passed to it is zero.

5.4.4.11 FlagsParameter

```
ulong Blotch.BlSprite.FlagsParameter = 0
```

Current incoming flags parameter to the Draw method. Typically this would be of interest to a callback function (see PreDraw, PreSubspritesDraw, PreLocalDraw, and PreMeshDraw).

5.4.4.12 Graphics

```
BlGraphicsDeviceManager Blotch.BlSprite.Graphics = null
```

Current incoming graphics parameter to the Draw method. Typically this would be of interest to a callback function (see PreDraw, PreSubspritesDraw, PreLocalDraw, and PreMeshDraw).

5.4.4.13 IncludeInAutoClipping

```
bool Blotch.BlSprite.IncludeInAutoClipping = true
```

Whether to use depth testing, and whether to participate in autoclipping calculations when they are enabled.

5.4.4.14 IsDisposed

```
bool Blotch.BlSprite.IsDisposed = false
```

Set when the object is Disposed.

5.4.4.15 LastWorldMatrix

```
Matrix Blotch.BlSprite.LastWorldMatrix = null
```

Current incoming world matrix parameter to the Draw method. Typically this would be of interest to a callback function (see PreDraw, PreSubspritesDraw, PreLocalDraw, and PreMeshDraw).

5.4.4.16 LODs

```
List<object> Blotch.BlSprite.LODs = new List<object>()
```

The object drawn for this sprite. Specifically, this is a list of levels of detail (LOD), where only one is drawn depending on the ApparentSize. Each element can be a Model, a triangle list (VertexPositionNormalTexture[]), or null (indicating nothing should be drawn). Elements with lower indices are higher LODs. So index 0 is the highest, index 1 is second highest, etc. LOD decreases (the index increases) for every halving of the object's apparent size. You can adjust how close the LODs must be to the camera with LodScale (see LodScale). When the calculated LOD index (see LodCurrentIndex) is higher than the last element, then the last element is used. So the simplest way to use this is to add a single element of the object you want drawn. You can also add multiple references of the same object so multiple consecutive LODs draw the same object. You can also set an element to null so it doesn't draw anything, which is typically the last element. A model can be assigned to multiple sprites. These are NOT disposed when the sprite is disposed.

5.4.4.17 LodScale

```
double Blotch.BlSprite.LodScale = 9
```

Defines the LOD scaling. The higher this value, the closer you must be to see a given LOD. A value of 9 (default) indicates that the highest LOD (LODs[0]) occurs when an object with a diameter of 1 roughly fills the window.

5.4.4.18 Matrix

```
Matrix Blotch.BlSprite.Matrix = Matrix.Identity
```

The matrix for this sprite. This defines the sprite's orientation and position relative to the parent coordinate system. For more detailed information, see AbsoluteMatrix.

5.4.4.19 Mipmap

```
BlMipmap Blotch.BlSprite.Mipmap = null
```

Mipmap textures to apply to the model. These work the same as LODs (see LODs for more information). The texture used depends on the apparent size of the model. The next higher mipmap is used for every doubling of model size, where element zero is the highest resolution, used when the apparent size is largest. If a mipmap is not available for the apparent size, the next higher available on is used. So, for example, you can specify only one texture to be used as all mipmaps if you like. Note that for a texture to display, the model must include texture coordinates. Most graphics subsystems do support mipmaps, but these are supported at the app level. Therefore only one image is used over a model for a given model apparent size, rather than nearer portions of the model showing higher-level mipmaps. These are NOT disposed when the sprite is disposed. A given BlMipmap may be assigned to multiple sprites.

5.4.4.20 MipmapScale

```
double Blotch.BlSprite.MipmapScale = 5
```

Defines the mipmap (Textures) scaling. The higher this value, the closer you must be to see a given mipmap.

5.4.4.21 Name

```
string Blotch.BlSprite.Name
```

The name of the EsSprite

5.4.4.22 PreDraw

```
PreDrawType Blotch.BlSprite.PreDraw = null
```

If not null, Draw method calls this at the beginning before doing anything else. From this function one might examine and/or alter any public writable EsSprite field, and/or control the further execution of the Draw method.

5.4.4.23 PreLocal

```
PreLocalType Blotch.BlSprite.PreLocal = null
```

If not null, Draw method calls this after drawing subsprites (if appropriate) but before drawing the local model. From this function one might examine and/or alter any public writable EsSprite field, and/or abort the Draw method.

5.4.4.24 PreMeshDraw

```
PreMeshDrawType Blotch.BlSprite.PreMeshDraw = null
```

If not null, Draw method calls this before each model mesh is drawn for the local model. From this function one might examine and/or alter any public writable EsSprite field. If the return value is true, then the mesh will not be drawn.

5.4.4.25 PreSubsprites

```
PreSubspritesType Blotch.BlSprite.PreSubsprites = null
```

If not null, Draw method calls this after the matrix calculations for AbsoluteMatrix (including billboards, CamDistance, ConstSize, etc.) but before drawing the subsprites or local model. From this function one might examine and/or alter any public writable EsSprite field.

5.4.4.26 SpecularColor

```
Vector3 Blotch.BlSprite.SpecularColor = null
```

The specular color. If null, MonoGame's default is kept.

5.4.4.27 SpecularPower

```
float Blotch.BlSprite.SpecularPower = 8
```

If a specular color is specified, this is the specular power.

5.4.4.28 SphericalBillboard

```
bool Blotch.BlSprite.SphericalBillboard = false
```

Spherically billboard the model. Specifically, keep the model's 'forward' direction pointing at the camera and keep its 'Up' direction pointing in the same direction as the camera's 'Up' direction. Also see CylindricalBillboardX, CylindricalBillboardY, CylindricalBillboardZ, and ConstSize.

5.4.5 Property Documentation

5.4.5.1 ApparentSize

```
double Blotch.BlSprite.ApparentSize  [get]
```

This is proportional to the apparent 2D size of the sprite. (Calculated from the last Draw operation that occurred, but before any effect of ConstSize)

5.4.5.2 CamDistance

```
double Blotch.BlSprite.CamDistance  [get]
```

Distance to the camera.

5.4.5.3 LodTarget

```
double Blotch.BlSprite.LodTarget  [get]
```

This read-only value is the log of the reciprocal of ApparentSize. It is used in the calculation of the LOD and the mipmap level. See LODs and Mipmap for more information.

5.4.5.4 VerticesEffect

```
BasicEffect Blotch.BlSprite.VerticesEffect  [get], [set]
```

BasicEffect used to draw vertices. If not explicitly set, then use a default BasicEffect and dispose it when the BlSprite is disposed. If explicitly set, then don't dispose it when the BlSprite is disposed.

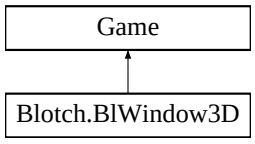
The documentation for this class was generated from the following file:

- C:/Users/kloum/Desktop/Source/Blotch3D/src/BlSprite.cs

5.5 Blotch.BlWindow3D Class Reference

To create the 3D window, derive a class from BlWindow3D. Instantiate it and call its Run method from the same thread. When you instantiate it, it will create the 3D window and a separate thread we'll call the "3D thread". All model meshes, textures, fonts, etc. used by the 3D hardware must be created and accessed by the 3D thread, because supported hardware platforms require it. Its safest to assume all Blotch3D and MonoGame objects must be created and accessed in the 3D thread. Although it may apparently work in certain circumstances, do not have the window class constructor create or access any of these things, or have its instance initializers do it, because neither are executed by the 3D thread. To specify code to be executed in the context of the 3D thread, you can override the Setup, FrameProc, and/or FrameDraw methods, and other threads can pass a delegate to the EnqueueCommand and EnqueueCommandBlocking methods. When you override the Setup method it will be called once when the object is first created. You might put time-consuming overall initialization code in there like graphics setting initializations if different from the defaults, loading of persistent content (models, fonts, etc.), creation of persistent BlSprites, etc. Do not draw things in the 3D window from the setup method. When you override the FrameProc method it will be called once per frame (see BlGraphicsDeviceManager.FramePeriod). You can put code there that should be called periodically. This is typically code that must run at a constant rate, like code that implements smooth sprite and camera movement, etc. Do not draw things in the 3D window from the FrameProc method. When you override the FrameDraw method, the 3D thread calls PrepareDraw just before calling FrameDraw once per frame, but more rarely if CPU is being exhausted. This is where you put drawing code (BlSprite.Draw, BlGraphicsDeviceManager.DrawText, etc.). Finally, if you are developing a multithreaded app, when other threads need to create, change, or destroy 3D resources or otherwise do something in a thread-safe way with the 3D thread, they can queue a delegate to EnqueueCommand or EnqueueCommandBlocking, which makes sure the code is done by the 3D thread sequentially at the end of the current FrameProc. If user input to the 3D window needs to be conveyed back to app threads, you can create thread-safe queues for that as well. This inherits from MonoGame's "Game" class.

Inheritance diagram for Blotch.BlWindow3D:



Public Member Functions

- delegate void Command (BlWindow3D win)

  See EnqueueCommand, EnqueueCommandBlocking, and BlWindow3D for more info
- BlWindow3D ()

  See BlWindow3D for details.
- void EnqueueCommand (Command cmd)

  Since all operations accessing 3D resources must be done by the 3D thread, this allows other threads to send commands to execute in the 3D thread. For example, you might need another thread to be able to create, move, and delete BlSprites. You can also use this for general thred safety of various operations. This method does not block. Also see BlWindow3D and the (blocking) EnqueueCommandBlocking for more details.
- void EnqueueCommandBlocking (Command cmd)

  Since all operations accessing 3D resources must be done by the 3D thread, this allows other threads to send commands to execute in the 3D thread. For example, you might need another thread to be able to create, move, and delete BlSprites. You can also use this for general thred safety of various operations. This method blocks until the command has executed. Also see BlWindow3D and the (non-blocking) EnqueueCommand for more details.
- new void Dispose ()

  When finished with the object, you should call Dispose() from the same thread that created the object. You can call this multiple times, but once is enough. If it isn't called before the object becomes inaccessible, then the destructor will call it and, if BlDebug.EnableDisposeErrors is true (it is true by default for Debug builds), then it will get an exception saying that it wasn't called by the same thread that created it. This is because the platform's underlying 3D library (OpenGL, etc.) often requires 3D resources to be managed only by one thread.

Public Attributes

- BlGraphicsDeviceManager Graphics

  The BlGraphicsDeviceManager associated with this window. This is automatically created when you create the BlWindow3D.
- ConcurrentDictionary< string, BlGuiControl > GuiControls = new ConcurrentDictionary<string, BlGuiControl>()

  The GUI controls for this window. See BlGuiControl for details.
- bool IsDisposed = false

  Set when the object is Disposed.

Protected Member Functions

- override void Initialize ()

  Used internally, Do NOT override. Use Setup instead.
- override void LoadContent ()

  Used internally, Do NOT override. Use Setup instead.
- virtual void Setup ()

  Override this and put all initialization and global content creation code in it. See BlWindow3D for details.
- override void Update (GameTime timeInfo)

  Used internally, Do NOT override. Use FrameProc instead.
- virtual void FrameProc (GameTime timeInfo)

  See BlWindow3D for details.
- override void Draw (GameTime timeInfo)

  Used internally, Do NOT override. Use FrameDraw instead.
- virtual void FrameDraw (GameTime timeInfo)

  See BlWindow3D for details.

5.5.1 Detailed Description

To create the 3D window, derive a class from BlWindow3D. Instantiate it and call its Run method from the same thread. When you instantiate it, it will create the 3D window and a separate thread we'll call the "3D thread". All model meshes, textures, fonts, etc. used by the 3D hardware must be created and accessed by the 3D thread, because supported hardware platforms require it. Its safest to assume all Blotch3D and MonoGame objects must be created and accessed in the 3D thread. Although it may apparently work in certain circumstances, do not have the window class constructor create or access any of these things, or have its instance initializers do it, because neither are executed by the 3D thread. To specify code to be executed in the context of the 3D thread, you can override the Setup, FrameProc, and/or FrameDraw methods, and other threads can pass a delegate to the EnqueueCommand and EnqueueCommandBlocking methods. When you override the Setup method it will be called once when the object is first created. You might put time-consuming overall initialization code in there like graphics setting initializations if different from the defaults, loading of persistent content (models, fonts, etc.), creation of persistent BlSprites, etc. Do not draw things in the 3D window from the setup method. When you override the FrameProc method it will be called once per frame (see BlGraphicsDeviceManager.FramePeriod). You can put code there that should be called periodically. This is typically code that must run at a constant rate, like code that implements smooth sprite and camera movement, etc. Do not draw things in the 3D window from the FrameProc method. When you override the FrameDraw method, the 3D thread calls PrepareDraw just before calling FrameDraw once per frame, but more rarely if CPU is being exhausted. This is where you put drawing code (BlSprite.Draw, BlGraphicsDeviceManager.DrawText, etc.). Finally, if you are developing a multithreaded app, when other threads need to create, change, or destroy 3D resources or otherwise do something in a thread-safe way with the 3D thread, they can queue a delegate to EnqueueCommand or EnqueueCommandBlocking, which makes sure the code is done by the 3D thread sequentially at the end of the current FrameProc. If user input to the 3D window needs to be conveyed back to app threads, you can create thread-safe queues for that as well. This inherits from MonoGame's "Game" class.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 BlWindow3D()

```
Blotch.BlWindow3D.BlWindow3D ( )
```

See BlWindow3D for details.

5.5.3 Member Function Documentation

5.5.3.1 Command()

```
delegate void Blotch.BlWindow3D.Command (
            BlWindow3D win )
```

See EnqueueCommand, EnqueueCommandBlocking, and BlWindow3D for more info

Parameters

win	The BlWindow3D object

5.5.3.2 Dispose()

```
new void Blotch.BlWindow3D.Dispose ( )
```

When finished with the object, you should call Dispose() from the same thread that created the object. You can call this multiple times, but once is enough. If it isn't called before the object becomes inaccessible, then the destructor will call it and, if BlDebug.EnableDisposeErrors is true (it is true by default for Debug builds), then it will get an exception saying that it wasn't called by the same thread that created it. This is because the platform's underlying 3D library (OpenGL, etc.) often requires 3D resources to be managed only by one thread.

5.5.3.3 Draw()

```
override void Blotch.BlWindow3D.Draw (
            GameTime timeInfo )  [protected]
```

Used internally, Do NOT override. Use FrameDraw instead.

Parameters

timeInfo	

5.5.3.4 EnqueueCommand()

```
void Blotch.BlWindow3D.EnqueueCommand (
            Command cmd )
```

Since all operations accessing 3D resources must be done by the 3D thread, this allows other threads to send commands to execute in the 3D thread. For example, you might need another thread to be able to create, move, and delete BlSprites. You can also use this for general thred safety of various operations. This method does not block. Also see BlWindow3D and the (blocking) EnqueueCommandBlocking for more details.

Parameters

cmd	

5.5.3.5 EnqueueCommandBlocking()

```
void Blotch.BlWindow3D.EnqueueCommandBlocking (
            Command cmd )
```

Since all operations accessing 3D resources must be done by the 3D thread, this allows other threads to send commands to execute in the 3D thread. For example, you might need another thread to be able to create, move, and delete BlSprites. You can also use this for general thred safety of various operations. This method blocks until the command has executed. Also see BlWindow3D and the (non-blocking) EnqueueCommand for more details.

Parameters

cmd	

5.5.3.6 FrameDraw()

```
virtual void Blotch.BlWindow3D.FrameDraw (
            GameTime timeInfo )  [protected], [virtual]
```

See BlWindow3D for details.

Parameters

timeInfo	

5.5.3.7 FrameProc()

```
virtual void Blotch.BlWindow3D.FrameProc (
            GameTime timeInfo )  [protected], [virtual]
```

See BlWindow3D for details.

Parameters

timeInfo	

5.5.3.8 Initialize()

```
override void Blotch.BlWindow3D.Initialize ( )  [protected]
```

Used internally, Do NOT override. Use Setup instead.

5.5.3.9 LoadContent()

```
override void Blotch.BlWindow3D.LoadContent ( )  [protected]
```

Used internally, Do NOT override. Use Setup instead.

5.5.3.10 Setup()

```
virtual void Blotch.BlWindow3D.Setup ( )  [protected], [virtual]
```

Override this and put all initialization and global content creation code in it. See BlWindow3D for details.

5.5.3.11 Update()

```
override void Blotch.BlWindow3D.Update (
            GameTime timeInfo )  [protected]
```

Used internally, Do NOT override. Use FrameProc instead.

Parameters

timeInfo	

5.5.4 Member Data Documentation

5.5.4.1 Graphics

```
BlGraphicsDeviceManager Blotch.BlWindow3D.Graphics
```

The BlGraphicsDeviceManager associated with this window. This is automatically created when you create the BlWindow3D.

5.5.4.2 GuiControls

```
ConcurrentDictionary<string, BlGuiControl> Blotch.BlWindow3D.GuiControls = new Concurrent↩
Dictionary<string, BlGuiControl>()
```

The GUI controls for this window. See BlGuiControl for details.

5.5.4.3 IsDisposed

```
bool Blotch.BlWindow3D.IsDisposed = false
```

Set when the object is Disposed.

The documentation for this class was generated from the following file:

- C:/Users/kloum/Desktop/Source/Blotch3D/src/BlWindow3D.cs

5.6 Blotch.BlGraphicsDeviceManager.Light Class Reference

Defines a light. See the BlSprite.Lights field. The default BasicShader supports up to three lights.

Public Attributes

- Vector3 **LightDirection** = new Vector3(1, 0, 0)
- Vector3 **LightDiffuseColor** = new Vector3(1, 0, 1)
- Vector3 **LightSpecularColor** = new Vector3(0, 1, 0)

5.6.1 Detailed Description

Defines a light. See the BlSprite.Lights field. The default BasicShader supports up to three lights.

The documentation for this class was generated from the following file:

- C:/Users/kloum/Desktop/Source/Blotch3D/src/BlGraphicsDeviceManager.cs