

DivWin
1.0v

Generated by Doxygen 1.7.3

Sun Sep 11 2011 13:58:39

Contents

Chapter 1

1. Using DivWin

[WTDivWin.h](#)

Using DivWin by William Thorley

DivWin is based on a Process / Renderable Object mentality. Process' are classes with behavioural code which tell them how to behave. They take inputs from the OS, signal other process' and tell Renderable objects how to display. Once a Process is written, the user should create a new instance. The system will record the information add it to the system and make the process enact it's behaviour, with no further input form the user (Fire and Forget). This way many similar objects can be created easily once the behavioural code is written. Render Objects should not have any behavioural code and are entirely directed by Process'.

The Process' are controlled by [cKernel](#) which will track and update all the process'.

Render Objects actually display on the screen. They are moved around in 3D space and [cCamera](#) renders them to the screen.

Collision Objects must be handed a Render Object at startup, which they will follow and determine collisions with other Collision Objects.

Files are loaded by the FileHandler and can be asked to provide media data for Render Objects and Collision Objects.

Lighting effects are controlled by the LightHandler.

The Main Loop:

```
*#include <WTDivWin.h>
*int WINAPI WinMain (HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR lpCmdLine,
                     int iCmdShow)
*{
//_START_PROGRAM(First_Process_Type,User_Settings_Type,Instance);
return _START_PROGRAM(cCore,cUserSettings,hInstance);

};
```

```
//Using DivWin in Linux
*#include <WTDivWin.h>

*int main ()
*{
    //_START_PROGRAM(First_Process_Type,User_Settings_Type);
    return _START_PROGRAM(cCore,cUserSettings);

*}
*
```

Main Loop Explanation: This will call the main function which will initialise the various components of the system. First_Process_Type must be a class type which inherits **cProcess** and should be the process that initialises and creates all the other processes required for the game. User_Settings_Type must be a class type which inherits **cSettings**. The virtual function UserSettings() in the derived class should set all the variables the user wishes to define. If the User does not wish to define ANY settings, use **cSettings** here and the defaults will be used. hInstance is a Windows only variable which is passed from the operating system to the WinMain function. Use the first HINSTANCE Passed to the function.

First _START_PROGRAM will set the settings for the game to use.

Then it will initialise the various components of the engine.

It will create an instance of the class type First_Process_Type which should initilise the game and create other processes to start the game.

While there are processes alive (and the system has not received an exit signal) the system will continue to cycle.

As the system exit, it will clear data, close links to devices and hardware and return a suitable exit signal for the Operating system. This should be returned from the funciton main().

Chapter 2

2. Using the cKernel

[WTKernel.h](#)

Using the kernel system by William Thorley

The kernel will create itself as soon as any Process is created. It will automatically link the process, and sort their run order. A Pointer to the [cKernel](#) can always be found by calling the macro _KERNEL ([cKernel::Instance\(\)](#)). If this function is called and there is no [cKernel](#), the function will create one. The Constructor is private, so there can be only one [cKernel](#). The Kernel is entirely automatic and should require no inputs from the user. However it can supply useful information and functionality to the user. KillAll(): Calling KillAll() will kill every running process. This will cause Update() to exit, and traditionally end the program. However it is possible to KillAll(), then create a new core process, and call Update() again, thereby ‘restarting’ the program.

_FIND_PROCESS(TYPE) Calling _FIND_PROCESS() will automatically search the [cKernel](#) for any processes of class type TYPE. It will return a pointer to the next process of class type TYPE everytime it is called. When there are no more processes of class type TYPE it will return 0.

Chapter 3

3. The First Process

[WTcProcess.h](#)

Using William Thorley's Process system

The First Process is exactly the same as every other process. i.e. it must inherit [cProcess](#). What is special about the Process passed to the `_START_PROGRAM()` call is that it is automatically called when the Kernel is initialized. This means that the loading code for `cCore` is the initialization code for the entire program. Usually the process' which will form the program are started here as well as loading files for the system. For these examples I wil call the class `cCore`.

This shows the declaration for `cCore`. Using `_PROCESS(Type)` is the same as calling class `Type : public cProcess` to inherit `cProcess`. Declaration:

```
_PROCESS(cCore)
{
    public:
        cCore();

    void Run();

    ~cCore();

    void AdditionalKillFunctionality();
    void AdditionalSleepFunctionality();
    void AdditionalWakeFunctionality();
};

};
```

`cCore()`: This shows the constructor for `cCore`. As it is the first object it will load media for the rest of the program and kick off the other processes.

```
cCore::cCore()
{
//Load IMF Files into memory.
    _LOAD_FILE("../src/User/Models/AShipModel.imf");
    _LOAD_FILE("../src/User/Models/StartShip.imf");
    _LOAD_FILE("../src/User/Textures/ATexture.imf");
    _LOAD_FILE("../src/User/CollisionObjects/ACollision.imf");
    _LOAD_FILE("../src/User/Objects/AIMFFileWithSeveralComponents.imf");
```

```

// Setup the camera.
_CAMERA->Far(1000.0f);
_CAMERA->Frustum();

//Create a new process and a renderable object.
mpProcessPointer=_CREATE(MyFirstProcess());
mpRenderObjectPointer=_CREATE(MyFirstRenderObject());

}

```

Run(): Run() is run once every frame as long as cCore is alive and controls the entire program. The Run() function should be explicitly designed to be re-written by the user to give each **cProcess** class their functionality. This code can be seen as the behaviour that the process should follow. It should rotate and position the Process' Render objects, send signals, receive inputs, anything the Process may want to do as part of its behaviour. The function **cCore::Run()** can be used like any other process, but is usually used to oversee the running of the program.

```

void cCore::Run()
{
    if (_KEY(KEY_SPACE)) _CAMERA->Far(10.0f);
    else _CAMERA->Far(1000.0f);
    _CAMERA->Frustum();

    mpProcessPointer=_CREATE(MyFirstProcess());
    mpRenderObjectPointer=_CREATE(MYFirstRenderObject());
}

}

```

~cCore(): This is called when the object is deleted, not when it is killed. It is difficult to predict reliably where this will be called, so it is best not to put code in this function.

AdditionalKillFunctionality(): This is called whenever the cCore Process is Killed. This will only activate if the process was alive and is now dead. This should be used to kill or transfer control of Render Objects that are owned by this process, or unload files that are no longer used.

```

*void cCore::AdditionalKillFunctionality()
*{
    _KILL(mpRenderObjectPointer);
    mpProcessPointer_SIGNAL(_S_SLEEP);
    *
}

```

AdditionalSleepFunctionality(): This is called whenever the cCore Process is made to Sleep. This will only activate if the process was awake and is now asleep. This is generally used to sleep Render Objects that are owned by this process.

```

*void cCore::AdditionalSleepFunctionality()
*{
    mpRenderObjectPointer->Signal(_S_SLEEP);
    *
}

```

AdditionalWakeFunctionality(): This is called whenever the cCore Process is made to Wake. This will only activate if the process was asleep and is now awake. This is generally used to wake Render Objects that were slept when cCore was sent to sleep.

```
void cCore::AdditionalSleepFunctionality() { mpRenderObjectPointer->Signal(_S_-  
WAKE); }
```


Chapter 4

4. How to use Process Objects

[WTcProcess.h](#)

Using William Thorley's Process Handler System

When properly implemented the process handler automatically links and runs all processes. A process must inherit [cProcess](#). It must also define the virtual function Run(). A process without a Run() function is useless, and will be deleted during the frame it is created.

Declaring a new process class called player:

```
//This is the same as _PROCESS(player)
*class player : public cProcess
*{
    *public :
    void Run();
    *};
    *
```

void [cProcess::Run\(\)](#); The function Run() is a virtual function already defined. This function is called every time the process must run its code (usually once a frame). The code that defines how the process acts goes in Run().

Creating a Process: Call the macro [_CREATE\(Type\)](#); This will return a pointer to the new process of type Type.

```
player *mpNewProcessPointer;
NewProcessPointer=_CREATE(player());
_CREATE(AnotherProcess(Argument1,Argument2));
```

Killing a Process: Process' must not be destroyed by deleting the process. Processes can be deleted either by calling the macro [_KILL\(\)](#); or by calling the Function Signal(SIGNAL lsFlags) with the flag [_S_KILL](#). If [_KILL\(\)](#) is called without a pointer it will automatically use the pointer this. Process' will not be destroyed when the signal is sent, they will be deactivated, but the memory will remain allocated until [cKernel](#) reaches the correct stage to delete the object. Objects can remain allocated into the next frame, but not the frame after that. Both pieces of following code have the same effect. The Process pointed to by mpPointerToAnotherProcess is Killed and then this process is killed.

```
void player::Run()
```

```

{
if (_KEY(KEY_k))
{
mpPointerToAnotherProcess->Signal(_S_KILL);
_KILL(this);
}

void player::Run()
{

if (_KEY(KEY_k))
{
_KILL(mpPointerToAnotherProcess);
_KILL();
}

}

```

Sleeping a Process: A Process can be sent to sleep by calling the [cSignal::Signal\(SIGNAL lsFlags\)](#) function with the value `_S_SLEEP`. Sleeping a process leaves the process in the process list, but stops the `Run()` function being called every frame. This allows the `Signal(SIGNAL lsFlags)` function to be used to return it to ‘active duty’. The memory will remain allocated. Sending repeated Sleep calls to a process will not affect the process or the stability of the system, the Process will remain asleep.

Waking a Process: A Process can be awakend by calling the [cSignal::Signal\(SIGNAL lsFlags\)](#) function with the value `_S_WAKE`. Sending repeated Waking calls to a process will not affect the process or the stability of the system, the Process will remain awake.

```

void player::Run()
{

if (_KEY(KEY_s)) mpPointerToAnotherProcess->Signal(_S_SLEEP);
if (_KEY(KEY_w)) mpPointerToAnotherProcess->Signal(_S_WAKE);

}

```

Removing a Process: A Process can be killed by the [cKernel](#) object. Calling the [cKernel::Remove\(\)](#) function will kill the process and free the memory. This must not be done to the currently running process or the system may crash. Use on the current process at your peril. The pointer is to the [cLinkedNode<vProcess>](#) which owns this process. Each Process has a pointer (`mpNode`) to its [cLinkedNode<vProcess>](#). *

```

void player::Run()
{

if (_KEY(KEY_k)) _KERNEL->Remove(mpPointerToAProcessesNode);
}

```

Chapter 5

5. How to use Render Objects

[WTcRenderObject.h](#)

Using William Thorley's Renderable Object System

All Render Objects are inherited from class [vRenderObject](#), through class [cRenderObject](#). The object also Inherits the class [cMatrix4](#). This is a 3D Translation class, and can handle 2D rotations (about X axis), 3D rotations, 3D Translations and 3D scaling. Calling the [cMatrix4](#) functions will move the [cRenderObject](#).

Creating RenderObjects: [cRenderObjects](#) can be created using the [_CREATE](#) macro as per a Process. This will return a pointer to the [RenderObject](#).

```
void player::Run()
{
    mpHull=_CREATE(cTexturedModel(mpShipNode));
    mpHull->Mesh(_GET_FILE(vMesh*,"Mesh"));
    mpHull->Texture(_GET_FILE(vTexture*,"Texture"));
    mpHull->Shader(_GET_SHADER("TexturingProgram"));
}
```

Currently there are lots of Render Objects:

class [cTexturedModel](#); class [cLandscape](#); class [cRenderNode](#); class [cBeamMesh](#); class [cImage](#); class [cTextureText](#); class [cLine](#); class [cParticle](#); class [cParticleGroup](#); class [cParticleHandler](#); class [cModelList](#); class [cPoint](#);

See the relevant documentation for each for how to use them.

The Renderable Object allows the user to develop their own Renderable Objects and links them to the renderer. A Renderable object must inherit [cRenderObject](#) and must also define the virtual function [Render\(\)](#), This will be called every time the renderable object needs to be rendered. It should also define all empty virtual functions in [vRenderObject](#) and [cRenderObject](#),

Declaration of a Renderable Object: class player : public [cRenderObject](#), public [cMatrix4](#) { public : [Render\(\)](#); };

[Render\(\)](#):

Chapter 6

6. Use of the cMatrix4 and cCameraMatrix Classes

cMatrix4.h

cCameraMatrix.h

Use of the Translation matrix classes by William Thorley.

The [cMatrix4](#) class and cCameraMatrix classes are very similar. The CameraMatrix maintains an inverted [cMatrix4](#) matrix as the translations applied to the camera must be inverted to give the same effect as to normal objects. Otherwise the effects are the same for [cMatrix4](#) and cCameraMatrix.

Most Render Objects have inherited the [cMatrix4](#) class meaning that these functions can be called to move and rotate Render Objects. Calling these functions, will apply the defined translations to the current object. Some are relative to the current translation and some are relative to the position of the Render Node controlling this object. Either way, the translations stack.

The X Axis points 90 to the Right perpendicular to the objects facing.

The Y Axis points 90 Upwards, perpendicular to the objects facing.

The Z Axis is the objects facing.

These Classes can be used for 2D or 3D Translations. There are different functions for 2D and 3D translations. The standard DivWin Objects automatically identify if they should be 2D or 3D.

3D Object Translation Functions:

void Set3D()

float *Position()

float X()

float Y()

float Z()

```
float *XVect()
float *YVect()
float *ZVect()
void Position(c2DVf *lpPosition)
void Position(float lfX,float lfY)
void Position(c3DVf *lpPosition)
void Position(float lfX,float lfY,float lfZ)
void PositionX(float lfX)
void PositionY(float lfY)
void PositionZ(float lfZ)
void AdvanceX(float lfDistance)
void AdvanceY(float lfDistance)
void AdvanceZ(float lfDistance)
void Advance(float lfX,float lfY, float lfZ)
void GAdvanceX(float lfX)
void GAdvanceY(float lfX)
void GAdvanceZ(float lfX)
void GAdvance(float lfX,float lfY,float lfZ)
void LRotate(float lfAngle)
void LRotateX(float lfAngle)
void LRotateY(float lfAngle)
void LRotateZ(float lfAngle)
void GRotateX(float lfAngle)
void GRotateY(float lfAngle)
void GRotateZ(float lfAngle)
void GRotateOriginX(float lfAngle)
void GRotateOriginY(float lfAngle)
void GRotateOriginZ(float lfAngle)
void GRotateX(float lfAngle,float lfX,float lfY,float lfZ)
void GRotateY(float lfAngle,float lfX,float lfY,float lfZ)
void GRotateZ(float lfAngle,float lfX,float lfY,float lfZ)
void Resize(float lfScale)
void LResizeX(float lfScale)
void LResizeY(float lfScale)
```

```
void LResizeZ(float lfScale)
void GResizeX(float lfScale)
void GResizeY(float lfScale)
void GResizeZ(float lfScale)
2D Object Translation Functions:
void Set2D()
void Advance(float lfX,float lfY)
void GAdvance(float lfX,float lfY)
void Angle(float lfAngle)
void Rotate(float lfAngle)
void GRotateOrigin(float lfAngle)
void GRotate(float lfAngle,float lfX,float lfY)
```


Chapter 7

7. Explanatory page for indicating the use of files

[WTcFile.h](#)

Using the file handler by William Thorley

The Filehandler is an automatic system for controlling files loaded in DivWin. All files should be of type IMF, though others can be loaded. When a file is loaded it is automatically linked into the system. This allows any process to access any loaded file. All Objects (blocks within IMF Files) are given a Reference (a character string) to allow them to be easily identified. And in the event of an OS clearing data (for instance Windows clearing Textures on Minimize) the file handler will automatically reload all the files. Non IMF Files will use the Filename as the Reference for the purposes of Identification.

To create a new File Class and make it use the file handler, have it inherit [cFile](#), This will require #include “WTcFile.h”. To load a new file use the macro _LOAD_-FILE(FileName).

IMF Files contain many types of data:

Models Textures Fonts Model Trees Reference Lists Shaders Shader Programs Collision Objects Landscape Models

See the specific files for each type for more information.

Chapter 8

8. Explanation of the Input System.

There are two main types on input. Mouse inputs and Key inputs. Both are updated every frame and buffered until the next frame. This means that a key press or mouse position will be consistent throughout the entire frame. THe inputs re received as interupts so will be recievied in line with the OS. Key states are boolean and can be accessed using the macro _KEY(Key Identifier). A list of Key Identifiers can be found on page [III. List of Key Identifiers for accessing the Key States](#) KeyIdentifiersList. True is key pressed, false is key not pressed.

The Mouse can be accessed using the macro _MOUSE() which is a reference to the [cMouse](#) Object. The mouse has three buttons (for now) left, right and middle, because that is enough for most people. Like keys these are boolean values, with true for pressed and false for not pressed. The mouse also has x,y and z which are the cursor position in pixels from the window position 0,0. Finally the mouse has xs and ys which is the amount the cursor has moved (in pixels) since the last frame.

Chapter 9

9. Using the Audio System

How to use the Audio System by William Thorley.

The class [cAudioObject](#) are used to play sounds from the sound card. Currently sound files are loaded by creating a new [cWav\(\)](#) with the file path and name as the first argument. They will be added to the file list in [cFileHandler](#) and can be found by searching the list for the filepath and name, once sound files are added to IMF files they will be loaded with the [_LOAD\(\)](#) macro.. A Sound file is passed to the buffer and played with the [Play\(\)](#) command. When the sound is played it will take up a channel on the sound card.

```
* // Load the Sound file
mWav = new cWav("./User/Audio/wave1.wav");
//Create the cAudioObject to buffer the sound file in.
mAo = new cAudioObject;
//Load the sound file into the buffer
mAo->SetBuffer(mWav);
//Play the sound
mAo->Play();
```


Chapter 10

10. class cSignal Explanation and Flags List

These correspond to the Flag values for [cSignal::Signal\(SIGNAL lsFlags\)](#).

```
Signal(_S_SLEEP);  
lpOtherProcess->Signal(_S_WAKE_TREE);
```

_S_KILL

- Kill the process. It will be deactivated and no longer run. Once the system is stable, the Kernel will delete the object. _S_SLEEP
- Sleep the process. It will be deactivated, but will not get deleted. The Process will continue to exist, but will not run. Objects this Process controls will remain and can be controlled from other processes. _S_WAKE
- Wake the Process. A Sleeping process will be reawakened and will start to run again. _S_KILL_TREE
- If the Variable WT_USE_PARENT_TREE is true, this will kill the signaled object and any child objects it created. This will act recursively until all children, grandchildren, great grandchildren etc. have been killed. _S_SLEEP_TREE
- If the Variable WT_USE_PARENT_TREE is true, this will Sleep the signaled object and any child objects it created. This will act recursively until all children, grandchildren, great grandchildren etc. have been Slept. _S_WAKE_TREE
- If the Variable WT_USE_PARENT_TREE is true, this will Wake the signaled object and any child objects it created. This will act recursively until all children, grandchildren, great grandchildren etc. have been Wakened.

Chapter 11

11. Examples of how to use Collision Objects

Collision Objects will only collide with other collision objects, NOT Render Objects. When a Collision Object is created it must be passed a pointer to a Render Object which will define it's translation. This should be the first argument passed to a Collision Object.

Collision Objects are created using the _CREATE() Macro and killed using the _KILL() Macro, as per a process. They can be slept and woken. Collision Objects need a media type to define how and when they collide. This is defined using the Type Function which gives the Collision Object the data it needs and sets the type of collision Object it will be. A Collision object should also be linked to a Process so that processes can transfer information or signals in the event of a collision.

Types of Collision Object: cCollisionSphere [cMeshCollision](#) cCollisionRay cCollisionBeam cCollisionBox cCollisionBoxFile

```
MyProcess::MyProcess()
{
    mpRender=_CREATE(TexturedModel());
    mpCollision=_CREATE(cCollisionObject(mpRender));

    mpCollision->SetLink(this);
    mpCollision->SetType(10.0f);
}

void MyProcess::Run()
{
    vProcess *lpProc;
    cCollisionList* lpList=_COLLISION_HANDLER->GenerateCollisionList(mpHullColl);//
    ,WT_ENEMY_UNIT
    _COLLISION_PROCESS_LIST(lpList,lpProc)
    {
        _KILL(lpProc);
    }
    delete lpList;
}
```


Chapter 12

12. How to use cRenderNode Objects

[WTcRenderNode.h](#)

How to use Render Nodes by William Thorley.

RenderNodes are a special type of Render Object. Render Nodes have no visual rendering to screen. Instead they manipulate other Render Objects. Specifically, a Translation applied to a Render Node will affect any Render Objects Controlled by the RenderNode. This means the objects controlled by the Render Node will move as if the Render Nodes position was 0,0,0.

```
mpNodePoint=_CREATE(cRenderNode());
mpNodePoint->Position(0.0f,0.0f,0.0f);
mpModel=_CREATE(cTexturedModel(mpNodePoint));
mpModel->Position(10.0f,0.0f,0.0f);

// mpModel is now at 10.0f,0.0f,0.0f.

mpNodePoint->Position(10.0f,0.0f,10.0f);

// mpModel is now at 20.0f,0.0f,10.0f.

mpNodePoint->RotateY(3.1416); //(rotate 90 degrees)

//Model is now at 0.0f,0.0f,-10.0f.
```

To control a Render Object by a Render Node object pass a pointer to the Render Node as the first argument in the Render Objects Constructor. Render Node objects can control any Render Object, including Render Nodes. This means that there can be many levels of Render Nodes before reaching a Renderable Object, making positioning of complex positional relationships easy.

See Also: [cModelList](#)

Chapter 13

IMF File Generation and Usage

IMF Files are generated by the IMF Compiler. The IMF Compiler is a separate program to the DivWin engine and has a text based interface. The Compiler should be run from the terminal, so the user can view and use the interface.

Everytime the user runs the program will start with an empty IMF File. IMF Files contain media blocks. Each block begins with a type identifier to identify the type of media stored in the block. This is followed by a size specifier defining the amount of data in the remainder of the block. Finally the Block has a character string storing the reference.

The main task the user will perform is to add Media to the IMF File. Each Media file added will require a reference (a character string which allows the media to be identified in the DivWin Engine) and often other data to fully define the object.

Media can often be converted into several different types of IMF Blocks. e.g. A image can be converted to a 2D Texture, a Landscape height map or if it is 64 times taller than it is wide into a font. Each of these require different information to generate the object.

An IMF File can contain many blocks all with different media types in. This allows the user to group media into sensible sets which are interdependant, eg a tank body model, a tank turret model, a tank shell model, a texture for the tank, and a model list representing the skeletal structure for the tank. This ensures that all inter dependant media can be loaded with a single call.

Each level of the menu defines the options available to the user, selecting 0 will always move the user back up to the previous level. Otherwise, the user should select the desired option, insert the number representing it and press enter.

To add each item, select 1 from the main menu. Each file Name (including file type) which is entered will be added to the IMF File as a new block. The system will request all the information required to generate the object, then add it to the IMF File as a new block. Take care when selecting the references as they are the only way to access the media in the DivWin Engine.

Once all the required media files are added to the IMF file, it can be written to the harddrive, by selecting option 7. The IMF file type should be included by the user.

IMF Files can be loaded and will add all their blocks to the end of all the blocks in the

current IMF File. This allows the user to add new media to previous groupings.

Media Types Supported:

Shader Code:

.shd (text files containing the shader code)

Model Files:

.x

.obj

.q3d

Model Files can be converted into:

Meshes (3D Models, including Normals and UV if available)

Collision Objects (Currently only supporting convex faces)

Box Collision Objects

Images:

.bmp

Image Files can be converted into:

2D Textures

Fonts (are composed of 64 vertical characters, with equal width and height)

Landscape Height Maps (Produces a map with a polygon per pixel in the image, with RGB(0,0,0) being no height and RGB(1,1,1) being maximum terrain height)

User Generated Media Types:

All data in these types is entered by the user into the IMF Compiler and the files are produced from that data. To activate them, insert the desired media type instead of a filename, in the add file menu.

* Shader Programs:

Insert 'Shader Program' as file name.

Model Lists:

Insert 'Model List' as file name.

Render Trees:

Insert 'Render Tree' as file name.

Chapter 14

I. A List of Useful Macros for accessing functions and pointers

RANDOM_NUMBER

- returns a Randomly generated number between 0.0f and 1.0f.
- SIGNAL()
- A Class Type for passing Signals between Process', Render Objects and Collision Objects.
- _KEY(KEY_ID)
- Function for accessing the array of key states. Takes Key ident as an input. Key Idents are defined in the Key List.

_MOUSE

- Reference giving access the systems mouse states.
- _LOAD(FileName)
- Function for loading a file. Takes the filename as an input. Same as _LOAD_-FILE(FileName);
- _LOAD_FILE(FileName)
- Function for loading a file. Takes the filename as an input. Same as _LOAD(FileName);
- _GET_FILE(FileType,FileName)
- Function returning a pointer of type FileType to the IMF Object with the reference FileName.

_CREATE(Type)

- Function operating like the new operator. Will create a new Process, Render Object or Collision Object and return a pointer of type Type.

_KILL()

- Will Kill the current process.
`_KILL(Pointer)`
- Will Kill the process pointed to by Pointer.
`_SLEEP(Process)`
- Will Sleep the process pointer to by Process.
`_WAKE(Process)`
- Will Wake the process pointed to by Process.
`_SIGNAL(Flag)`
- Function which will send a signal to the process this function is called from, to signal the action, Flag. (Signal Flags are defined in the Signal List).
`_USE_SHADER(FileNamed)`
- Function telling the system to buffer for use the shader with reference FileNamed.
`_GET_SHADER(FileNamed)`
- Function returning a pointer to the `cShader` Object with the reference FileNamed.
`_USE_FIXED_FUNCTION()`
- Function telling the system to used the fixed function pipeline and not use shaders.
`_CAMERA`
- A Pointer to the `cCamera` Object.
`_LIGHT`
- A Pointer to the `cLightHandler` Object.
`_KERNEL`
- A Pointer to the `cKernel` Object.
`_FILE`
- A Pointer to the `cFileHandler` Object.
`_PAINTER`
- A Pointer to the `cPainter` Object.
`_COLLISION_HANDLER`
- A Pointer to the `cCollisionHandlerObject`.
`_RESET_COLLISION`
- This will reset a stepping Collision Search, ready for a fresh search. (Not issue for Generating Collision Lists)
`WT_TIME_IND`
- This is the current frame length in seconds. Multiply anything time dependant by this to make it time independant. (It is a float multiplier, think before use.)
`_COLLISION_PROCESS_LOOP(lpList,lpVar)`

- This will start a Loop to step through a collision list. lpList should point to the Collision List and lpVar will point at the current process in the collision list.

`_COLLISION_RENDER_LOOP(lpList,lpVar)`

- This will start a Loop to step through a collision list. lpList should point to the Collision List and lpVar will point at the current Render object in the collision list.

Chapter 15

II. A List of Settings which can be used to modify the way the engine works

WT_USE_PAINTER_ALGORITHM

- This determines whether the system will render objects in the order they sit in the Render Tree, or to use the Painter Algorithm and sort them for speed.

WT_OPENGL_LIGHTS

- This is the number of Lights that you wish OpenGL to use simultaneously to light an object.

USE_LIGHT_HANDLER

- The light handler will determine the closest WT_OPENGL_LIGHTS to the object and render the object only using their influence. This allows the user to improve performance and / or use more lights than OpenGL can support.

WT_COLLISION_HANDLER_TYPE WT_COLLISION_HANDLER_TYPE_TYPE

- This determines the Type of Collision handler used to sort Collisions. It can be set to either WT_COLLISION_HANDLER_TYPE_TYPE or WT_COLLISION_HANDLER_TYPE_BSP.

WT_RAY_ANGLE_RANGE

- Ummmm, something to do with Ray Angles.

WT_ALLOW_DYNAMIC_TEXTURES

- This determines whether the user wants to be able to update textures once loaded into memory. Being able to do so will result in a reduction in performance.

WT_COLLISION_HANDLER_SIZE

- This will set the number of slots that the Collision Handler will use for filtering collisions. See relevant documentation.

WT_COLLISION_HANDLER_DIMENSIONS

- This sets the number of dimensions the BSP Collision Handler should use (1,2 or 3) WT_COLLISION_HANDLER_DIMENSIONS_1D,WT_COLLISION_HANDLER_DIMENSIONS_2D, WT_COLLISION_HANDLER_DIMENSIONS_3D.

WT_COLLISION_SPACE_SIZE

- This is the spatial size of each slot for the BSP. It should be at least (preferably more) than half the largest object size, that will be involved in collisions.

WT_DEFINE_OS

- This is the OS that the engine is running under. So far OS_LINUX or OS_WINDOWS.

WT_COLLISION_DEPTH

- This shouldn't be used anymore. Why did I leave it in here. Was from the good ol' days of Collision Trees.

_GRAVITY_X

- This is the X Axis value of gravity, to be used by particle objects.

_GRAVITY_Y

- This is the Y Axis value of gravity, to be used by particle objects.

_GRAVITY_Z

- This is the Z Axis value of gravity, to be used by particle objects.

_WIND_X

- This is the X Axis value of wind, to be used by particle objects.

_WIND_Y

- This is the Y Axis value of wind, to be used by particle objects.

_WIND_Z

- This is the Z Axis value of wind, to be used by particle objects.

WT_MAX_PARTICLES

- This is the maximum particles that [cParticleHandler](#) should need to maintain at any one time.

WT_PARTICLE_HANDLER_UPDATE_PARTICLE_POSITIONS

- When true, the Particle handler will automatically update the particles speed and position based on Wind and Gravity.

WT_VERTEX_RANGE_CHECK_SIMILAR

- This is the maximum allowed distance between vertices in a collision object for them to be counted as a single vertex.

WT_USE_PARENT_STACK

- This will determine whether the system should automatically track each process' parent and children.

Chapter 16

III. List of Key Identifiers for accessing the Key States

These correspond to the Key codes for the Macro _KEY()

```
if(_KEY(KEY_SPACE)) _CREATE(Bullets());
if(_KEY(KEY_TAB)) _KERNEL->KillAll();
```

KEY_BACKSPACE

KEY_TAB

KEY_ENTER

KEY_RETURN

KEY_SHIFT

KEY_CONTROL

KEY_CTRL

KEY_ALTERNATE

KEY_ALT

KEY_PAUSE

KEY_CAPSLOCK

KEY_CAPITALLOCK

KEY_ESCAPE

KEY_ESC

KEY_SPACE

KEY_PGUP

KEY_PAGEUP

KEY_PGDN

KEY_PAGEDOWN
KEY_END
KEY_HOME
KEY_LEFT
KEY_LEFTARROW
KEY_RIGHT
KEY_RIGHTARROW
KEY_UP
KEY_UPARROW
KEY_DOWN
KEY_DOWNARROW
KEY_SELECT
KEY_EARLYPRINT
KEY_EARLYPRINTSCREEN
KEY_EXECUTE
KEY_PRINT
KEY_PRINTSCREEN
KEY_INS
KEY_INSERT
KEY_DEL
KEY_DELETE
KEY_HELP
KEY_0
KEY_1
KEY_2
KEY_3
KEY_4
KEY_5
KEY_6
KEY_7
KEY_8
KEY_9
KEY_A
KEY_B

KEY_C
KEY_D
KEY_E
KEY_F
KEY_G
KEY_H
KEY_I
KEY_J
KEY_K
KEY_L
KEY_M
KEY_N
KEY_O
KEY_P
KEY_Q
KEY_R
KEY_S
KEY_T
KEY_U
KEY_V
KEY_W
KEY_X
KEY_Y
KEY_Z
KEY_a
KEY_b
KEY_c
KEY_d
KEY_e
KEY_f
KEY_g
KEY_h
KEY_i
KEY_j

KEY_k
KEY_l
KEY_m
KEY_n
KEY_o
KEY_p
KEY_q
KEY_r
KEY_s
KEY_t
KEY_u
KEY_v
KEY_w
KEY_x
KEY_y
KEY_z

Chapter 17

Namespace Index

17.1 Namespace List

Here is a list of all namespaces with brief descriptions:

[cIMF](#) (This basically holds functions for loading IMF files) ??

Chapter 18

Class Index

18.1 Class Hierarchy

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

c2DVf	??
c2DVi	??
c3DVf	??
c3DVi	??
cAudioBuffer	??
cWav	??
cAudioDevice	??
cAudioObject	??
cCameraMatrix4	??
cCamera	??
cCameraPainter	??
cCluster	??
cCollisionHandler	??
cCollisionHandlerType	??
cCollisionHandlerBSP	??
cCollisionList	??
cCollisionListObject	??
cEventHandler	??
CEException	??
cFace	??
cFullFaceData	??
cFileHandler	??
cFog	??
cFrameRate	??
cIMFLoader	??
cMeshArray	??
cMeshFileCollision	??
cKernel	??

cKeyStore	??
cLightHandler	??
cLightSpot	??
cLimitedList< cX >	??
cLimitedList< cCluster >	??
cClusterList	??
cLimitedList< cFullFaceData >	??
cFullFaceList	??
cLimitedList< cPlane >	??
cPlaneList	??
cLimitedList< cPolygon >	??
cPolygonList	??
cLimitedList< cVertex >	??
cVertexList	??
cLimitedPointerList< cX >	??
cLinkedList< T >	??
cLinkedNode< T >	??
cMainThread< cX, cS >	??
cMaterial	??
cLandscape	??
cModelList	??
cTexturedModel	??
cMatrix	??
cMatrix4	??
vRenderObject	??
cRenderNode	??
cRenderObject	??
cBeamMesh	??
cImage	??
cLandscape	??
cLine	??
cModelList	??
cParticleGroup	??
cParticleHandler	??
cPoint	??
cText	??
cTexturedModel	??
cMeshTreeArray	??
cMeshTreeNode	??
cMinLL< T >	??
cMinLN< T >	??
cmLandscapeArray	??
cModelListNode	??
cMouse	??
cPainter	??
cParentStack	??
cParticleForGroup	??

cGravityParticle	??
cParticle	??
cWindAndGravityParticle	??
cParticleSettings	??
cPlane	??
cPolygon	??
cPushPopStack< cX >	??
cReferenceList	??
cRenderPointer	??
cSettings	??
cShaderArray	??
cSignal	??
cCollisionObject	??
cParticle	??
vProcess	??
cProcess	??
vRenderObject	??
cSync	??
cTextureArray	??
cUserSignal	??
vProcess	??
cVertex	??
cWindow	??
dwLog	??
v2DPolygon	??
vCollisionData	??
cSphereCollision	??
cBeamCollision	??
cRayCollision	??
cMeshCollision	??
cMeshFileCollision	??
vFile	??
cFile	??
cLandscapeMeshFile	??
cMeshFileCollision	??
cShaderProgram	??
cWav	??
vMesh	??
cMesh	??
vMeshTree	??
cMeshTree	??
vShader	??
cShader	??
vTexture	??
cTexture	??
vFont	??
cFont	??

vLight	??
cLight	??
cSpotLight	??
vmLandscape	??
cmLandscape	??
cLandscapeMeshFile	??
vShaderProgram	??
cShaderProgram	??

Chapter 19

Class Index

19.1 Class List

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

c2DVf (This is a 2 Dimensional float vector object)	??
c2DVi (This is a 2 Dimensional integer vector)	??
c3DVf (This is a 3 dimensional float vector object)	??
c3DVi (This is a 3 Dimensional integer vector)	??
cAudioBuffer (This class will create a buffer space to store sound data. This class will create and initialise an OpenAL buffer. The Buffer stores the sound data and is linked to a cAudioObject to allow it to be played)	??
cAudioDevice (This class will initialise the sound card. This class will initialise an audio device so sounds can be played on the system) . . .	??
cAudioObject (This class will allow a sound to be played. This class will link an audio source and a buffer, This allows the sound data stored in the buffer to be played through the source. Each source is an audio channel and can only play one sound at a time)	??
cBeamCollision	??
cBeamMesh (A Procedurally generated cylindrical Renderable Object. This class will generate a cylinder with specified dimensions and segments. The origin for the cylinder is in the radial center of one end of the cylinder)	??
cCamera (This class will control the rendering of the render tree. It will handle all the render objects and how to render to the screen. cCamera is the renderer for the entire program. It will contain a scene graph, and inherits cCameraMatrix4 as its view matrix. It can be considered as a camera in the program space, and will render every thing in its scene graph to the screen)	??

cCameraMatrix4	(This is a translation matrix for a camera object. This is a translation matrix for a camera. All the translations are inverted. Distances are 'reversed', Local rotations are 'globalised' and Global rotations are 'localised'. Effectively all the translations are inverted before they are applied to this matrix)	??
cCameraPainter	??
cCluster	(This class is an array of cFace . This is often used)	??
cClusterList	??
cCollisionHandler	(This is the Collision Handler. It will control any collision search the user performs. This Collision handler is created by using the Instance() and can ONLY be created using Instance() . _COLLISION_HANDLER is a quick pointer to the cCollisionHandler::Instance() pointer. This will store data for the search and will store the current position to resume searches. Calling the Function GenerateCollisionList() wil create a comprehensive list of pointers to cRenderobjects() that meet the collision parameters of GenerateCollisionList() . This list can be accessed by using NextCollisionItem())	??
cCollisionHandlerBSP	??
cCollisionHandlerType	??
cCollisionList	??
cCollisionListObject	??
cCollisionObject	??
cEventHandler	(This will handle events from the OS. Actually this will just store the Input data for the keyboard and mouse. It is easiest to access the input data using _KEY and _MOUSE . There can only be one cEventHandler , created using Instance())	??
CException	??
cFace	(This class will store data about faces for a 3D Mesh. This can be used for Models, Collision Meshes or any other object using 3D Faces. Includes code for loading and saving the object types to and from IMF Files. Uses cVertex and cPlane to store the data)	??
cFile	(This is the base code for files to be loaded from a hdd. Any file object loaded from a hddvshould inherit this class. It is best used for media files. This code will automatically add newly loaded files to cFileHandler . The files can be loaded using the filename or if loaded from an IMF file using the reference for each file)	??
cFileHandler	(This is the handler for the file system. This handles all files loaded from a hdd. It will give allow processes to use files loaded else where, using either the filenames or if loaded from an IMF a file reference. The files are stored in the list mpFiles)	??
cFog	(This class will create and control an OpenGL scenes fog effect)	??
cFont	(This class will store the data for a Font ready to be used for rendering cText . This should come from an IMF file and be composed of an image of 64 character images stacked vertically. This is a file class and should be handled entirely by the engine)	??
Processes Per Frame:		
Frames Per Second:		
Frame Time and Process Time:		
These are the amount of time that passes every Frame and Process Cycle measured		

in seconds. Multiplying any distances moved by this will convert them into distance per second. This allows the user to change the running frame rate without affecting the speed the user experiences. If the Frame rate and Process Rate are not going to be changed this can be ignored)??

cFullFaceData (This class is a cFace with expanded Functionality. Allows the user to check information about the face. See list of functions for information)	??
cFullFaceList (A dynamic linked list of cFullFaceData . As described. Can load object to and from IMFs)	??
cGravityParticle (CParticles which are affected by Gravity. These Particles have the code to be affected by the variables _GRAVITY_X,_GRAVITY_Y and _GRAVITY_Z. UpdatePos() will account use the current Gravity settings to calculate the speed and position)	??
cImage (A 2D renderable object)	??
cIMFLoader	??
cKernel (Kernel Object. Handles Processes. Tracks, runs and deletes current processes. Has complete control over every cProcess object. Will run all awake alive processes every process cycle, will delete dead processes. Also controls the activation of rendering frames and handling interactions with the operating system)	??
cKeyStore (This class stores all the input data for a single keyboard)	??
cLandscape (A height map based, matrix structured Landscape. Landscape is composed of a matrix of square polygons. The heights of each vertex is produced using the packaging software, and is generated from a bitmap)	??
cLandscapeMeshFile (This is the class which the Landscape File is stored in. This can be passed to any of cmLandscape , cLandscapeMeshIndividual or cLandscapeMeshRandom)	??
cLight (Creates an OpenGL light effect)	??
cLightHandler (CLightHandler will control the OpenGL Lights. It will turn off lights not required or possible for different renderings to increase speed and circumvent the OpenGL limit of active lights. OpenGL has a limited number of lights that can be used at any one time. This handler identifies the lights which will have the greatest effect on the current object and prepares the optimal selection of lights for rendering the scene)	??
cLightSpot	??
cLimitedList< cX >	??
cLimitedPointerList< cX >	??
cLine (A standard renderable Line object)	??
cLinkedList< T > (This is the control for a linked list. It controls a linked list of cLinkedNodes which each point to their item in the list. Each cLinkedNode poitns to the cLinkedNode 's either side of themselves and the data they own. cLinkedList is templated and so can be a linked list of any type)	??
cLinkedNode< T > (This is a node class to allow templating of the cLinkedList class. This node will store pointers to the nodes either side of this node in the linked list and a pointer to the object his node owns)	??
cMainThread< cX, cS >	??

cMaterial	(A class to store material data for an object. Defines the 'reflectiveness' of the surface)	??
cMatrix		??
cMatrix4		??
cMesh	(This class stores the data for a 3D model)	??
cMeshArray	(This is a temporary storage class to ease transformation of data from the hdd to the cMesh class)	??
cMeshCollision		??
cMeshFileCollision	(Mesh Collision Object with functionality to load a Collision Mesh from a file. This inherits cMeshCollision , so uses that code for defining the Mesh Collision Object)	??
cMeshTree	(This will store the data for a cModelList())	??
cMeshTreeArray	(This will load the information from an IMF file to be handed to a cMeshTree() This object will store the data for a cMeshTree() object from an IMF file)	??
cMeshTreeNode	(This object will store the data for a single item in a cMeshTree() . This stores the Position, Rotation, Mesh, Texture, Tree Level, for this object)	??
cMinLL< T >	(Linked List Lite. I'm not sure I use this. But quick small and simple templated Linked List)	??
cMinLN< T >	(Linked Nodes Lite. I'm not sure I use this. But quick small and simple templated Linked List nodes)	??
cmLandscape	(This will store the data for a landscape mesh. The data can be rendered through cLandscape)	??
cmLandscapeArray	(This is a class to transfer data from a hdd file to memory in a format that is quick and easy to render. This is a temporary storage class to ease transformation of data from the hdd to the cmLandscape class)	??
cModelList	(Model list is a 'static' render tree holding data for a series of renderable objects. It counts as a single renderable object)	??
cModelListNode	(Class for storing data about a single node on in the Model List)	??
cMouse	(This will store all the input data for a single mouse)	??
cPainter		??
cParentStack	(This class will automatically track the parents and children of each process. When a new process is created, the process that created is stored as the new processes parent. The new process is added as a child of the creating process. This allows Processes to get their parent (a rather inefficient method). It also allows use of the TREE signal commands to send a signal to a process, all it's children recursively. See cSignal for more information)	??
cParticle		??
cParticleForGroup		??
cParticleGroup		??
cParticleHandler		??
cParticleSettings		??
cPlane	(A class for handling plane data for 3D mesh objects. Is composed of 4 floats, X,Y,Z,N. X,Y,Z components of the Normalised normal vector and distance above the origin along the line of the Normal)	??

cPlaneList (A resizable array of cPlane Objects. Stores and Handles a list of cPlane objects using cLimitedList)	??
cPoint (A Renderable object for rendering single points)	??
cPolygon (This class stores Polygon Data for 3D Meshes. A Polygon is a single convex bounded Face in a single Plane with any number of vertices above 3. cFaces which are sharing an edge and in the same plane can be combined into a single cPolygon . This allows models to be represented with less Planes for the purposes of calculating Collisions. Uses cPlane and cVertex Objects)	??
cPolygonList (An array of cPolygon Objects. Creates an array of cPolygon objects using the class cLimitedList)	??
cProcess (This is the base code for a process. This will automatically create a new process. It will hand itself to cKernel to be processed every frame. Any Processes created by the user should inherit this type to be handled by cKernel automatically. Initialisation code should go in the constructor of the user type. Linking to cKernel is performed automatically by cProcess . Update code should go in the function Run() . Destruction code should go in the destructor of the user type. Code to handle interaction of two processes should go in either processes UserSignal() function)	??
cPushPopStack< cX >	??
cRayCollision	??
cReferenceList (This will Load from an IMF and store a list of string type references. This will load a list of string type references from an IMF filestream. The References can be accessed as if they were an array)	??
cRenderNode (This is a dynamic render tree branch. This class stores a dynamic list of cRenderObjects called mpObjects. cRenderNode inherits cRenderObject and so can be stored in mpObjects of other cRenderNodes . Any translations applied to this cRenderNode will modify the base coordinates of any objects stored in mpObjects. This allows objects to be grouped for the purposes of translations. A cRenderNode volume encompasses all objects beneath it in the render tree, this increases the speed of collision searches as a cRenderNode can remove all its sub objects from the search)	??
cRenderObject (This class contains the base code for all renderable objects. Any renderable object should inherit this class)	??
cRenderPointer (This class is a temporary store for a cRenderObjects data. This is the complete render matrix and all other data required to render the object)	??
cSettings	??
cShader	??
cShaderArray (This is the transfer class to transfer shader data from raw data to a cShader class. This will collect raw data from an IMF class and process it ready to be handed to a cShader() object)	??

cShaderProgram (A cShaderProgram() is a series of cShader() objects compiled into a program. cShaderProgram is a cFile() object. The cShaderProgram() finds a pointer to all the Shaders that compose it and compile them into a program. A cShader() object can be used in many different programs. The cShaderProgram() can be produced manually using the AttachShader() and Link() functions. The cShaderProgram() can be turned on with the use function) . . .	??
cSignal (Class for handling Signals sent between objects (cProcess , cRenderObject , cCollisionObject). Allows the user to wake, sleep and kill objects. For cProcess (while cParentStack is enabled) also allows signals to be sent to a process that will recursively affect all the children of that process. Possible signals to be passed in are _S_SLEEP,_S_WAKE,_S_KILL,_S_SLEEP_TREE, _S_WAKE_TREE,_S_KILL_TREE User Specified Signals are controlled by the class cUserSignal)	??
cSphereCollision	??
cSpotLight	??
cSync (This is the timer class. It allows the user to time events and pause the system)	??
cText (This class is a text renderable object)	??
cTexture	??
cTextureArray	??
cTexturedModel (A standard Textured Model renderable object)	??
cUserSignal (Class for handling user specified signals sent between classes inheriting cProcess . The function UserSignal should be defined for each object. The signals each class has defined can be independant. The code for processing the signal should be defined in UserSignal as there is no signal buffer. This should be used for dealing with Process interactions, to make a single point of detection of interaction and allowing both processes to handle the interaction. System signals (sleep, wake and kill) should be handled through cSignal) . . .	??
cVertex	??
cVertexList	??
cWav	??
cWindAndGravityParticle (CGavityParticles which are also affected by Wind. These Particles have the code to be affected by the variables __WIND_X,__WIND_Y and __WIND_Z. UpdatePos() will account use the current Wind and Gravity settings to calculate the speed and position of each particle)	??
cWindow (This class will create control and destroy desktop windows. This is ugly and disgusting. It will create new windows, link the window with OpenGL and do basic event handling. This is the users access to interact with the OS. Note it is all very OS specific)	??
dwLog	??
v2DPolygon (This stores mesh data for a single square quadrilateral. This is used for cTextureText)	??
vCollisionData (Virtual Class so cCollisionObject can access the Collision data object it needs)	??
vFile (This is the virtual file for cFile . It is a virtual representation of the base code for files loaded from a hdd)	??

vFont	??
vLight	??
vMesh (This is the virtual base class for a cMesh object)	??
vMeshTree (This is a virtual class for inherited by cMeshTree())	??
vmLandscape (This is the virtual base class for cmLandscape)	??
vProcess (This is the virtual base class for a system process, see cProcess)	??
vRenderObject (Virtual function for producing renderable objects. see cRenderObject)	??
vShader (This is the virutal base class for the Shader Files - both Vertex and Fragment)	??
vShaderProgram (This is a virtual class for a Shader Program. A shader program is a series of cShader() objects to be compiled into a single Shader Program)	??
vTexture	??

Chapter 20

Namespace Documentation

20.1 cIMF Namespace Reference

This basically holds functions for loading IMF files.

Functions

- void [LoadIMF](#) (const char *lpPath)

This is the function which will load an IMF file from memory.

20.1.1 Detailed Description

This basically holds functions for loading IMF files.

20.1.2 Function Documentation

20.1.2.1 void [cIMF::LoadIMF](#) (const char * *lpPath*)

This is the function which will load an IMF file from memory.

Definition at line 5 of file WTcIMFLoader.cpp.

Chapter 21

Class Documentation

21.1 c2DVf Class Reference

This is a 2 Dimensional float vector object.

Public Member Functions

- double [Angle \(\)](#)

This will return the angle of the vector in radians.

- float [Magnitude \(\)](#)

This will return the absolute size of the vector.

- void [operator= \(c2DVf *lpValue\)](#)

This will equate this vector to the [c2DVf](#) pointed to by lpValue.

- float & [operator\[\] \(uint32 liPos\)](#)

Public Attributes

- float [v \[2\]](#)

21.1.1 Detailed Description

This is a 2 Dimensional float vector object.

Definition at line 4 of file WTMATH.h.

21.1.2 Member Function Documentation

21.1.2.1 double c2DVf::Angle()

This will return the angle of the vector in radians.

Definition at line 3 of file WTMATH.cpp.

21.1.2.2 float c2DVf::Magnitude()

This will return the absolute size of the vector.

Definition at line 8 of file WTMATH.cpp.

21.1.2.3 void c2DVf::operator=(c2DVf * lpValue)

This will equate this vector to the `c2DVf` pointed to by lpValue.

Definition at line 36 of file WTMATH.cpp.

21.1.2.4 float& c2DVf::operator[](uint32 liPos) [inline]

Definition at line 22 of file WTMATH.h.

21.1.3 Member Data Documentation

21.1.3.1 float c2DVf::v[2]

Definition at line 7 of file WTMATH.h.

21.2 c2DVi Class Reference

This is a 2 Dimensional integer vector.

Public Member Functions

- long `Angle()`

This will return the angle of the vector in radians.

- int `Magnitude()`

This will return the absolute size of the vector.

- `c2DVi * operator=(c2DVi *lpValue)`
- int & `operator[] (uint32 liPos)`

Public Attributes

- int `v` [2]

21.2.1 Detailed Description

This is a 2 Dimensional integer vector.

Definition at line 55 of file WTMath.h.

21.2.2 Member Function Documentation

21.2.2.1 long c2DVi::Angle ()

This will return the angle of the vector in radians.

Definition at line 75 of file WTMath.cpp.

21.2.2.2 int c2DVi::Magnitude ()

This will return the absolute size of the vector.

Definition at line 80 of file WTMath.cpp.

21.2.2.3 c2DVi * c2DVi::operator= (c2DVi * *lpValue*)

Definition at line 90 of file WTMath.cpp.

21.2.2.4 int& c2DVi::operator[](uint32 *lPos*) [inline]

Definition at line 70 of file WTMath.h.

21.2.3 Member Data Documentation

21.2.3.1 int c2DVi::v[2]

Definition at line 58 of file WTMath.h.

21.3 c3DVf Class Reference

This is a 3 dimensional float vector object.

Public Member Functions

- float `Magnitude ()`

This will return the absolute size of this vector.

- void `Normalise ()`

This will make the magnitude of this vector 1 while maintaining its direction.

- `c3DVf * operator= (c3DVf *lpValue)`
- `c3DVf operator= (c3DVf lpValue)`
- float * `operator= (float *lpValue)`
- `c3DVf operator+= (c3DVf lpValue)`
- `c3DVf * operator+= (c3DVf *lpValue)`
- float `Dot (c3DVf lpValue)`
- `c3DVf operator* (c3DVf lvOther)`
- float & `operator[] (uint32 liPos)`

Public Attributes

- float `v [3]`

21.3.1 Detailed Description

This is a 3 dimensional float vector object.

Definition at line 26 of file WTMath.h.

21.3.2 Member Function Documentation

21.3.2.1 float c3DVf::Dot (`c3DVf lpValue`)

Definition at line 143 of file WTMath.cpp.

21.3.2.2 float c3DVf::Magnitude ()

This will return the absolute size of this vector.

Definition at line 13 of file WTMath.cpp.

21.3.2.3 void c3DVf::Normalise ()

This will make the magnitude of this vector 1 while maintaining its direction.

Definition at line 18 of file WTMath.cpp.

21.3.2.4 c3DVf c3DVf::operator* (c3DVf *lvOther*)

Definition at line 27 of file WTMath.cpp.

21.3.2.5 c3DVf * c3DVf::operator+= (c3DVf * *lpValue*)

Definition at line 67 of file WTMath.cpp.

21.3.2.6 c3DVf c3DVf::operator+= (c3DVf *lpValue*)

Definition at line 59 of file WTMath.cpp.

21.3.2.7 float * c3DVf::operator= (float * *lpValue*)

Definition at line 53 of file WTMath.cpp.

21.3.2.8 c3DVf c3DVf::operator= (c3DVf *lpValue*)

Definition at line 47 of file WTMath.cpp.

21.3.2.9 c3DVf * c3DVf::operator= (c3DVf * *lpValue*)

Definition at line 41 of file WTMath.cpp.

21.3.2.10 float& c3DVf::operator[] (uint32 *liPos*) [inline]

Definition at line 51 of file WTMath.h.

21.3.3 Member Data Documentation

21.3.3.1 float c3DVf::v[3]

Definition at line 29 of file WTMath.h.

21.4 c3DVi Class Reference

This is a 3 Dimensional integer vector.

Public Member Functions

- int **Magnitude ()**

This will return the absolute size of the vector.

- `c3DVi operator* (c3DVi lvOther)`
- `c3DVi * operator= (c3DVi *lpValue)`
- `c3DVi operator= (c3DVi lpValue)`
- float `Dot (c3DVi lpValue)`
- int & `operator[] (uint32 liPos)`

Public Attributes

- int `v [3]`

21.4.1 Detailed Description

This is a 3 Dimensional integer vector.

Definition at line 74 of file WTMath.h.

21.4.2 Member Function Documentation

21.4.2.1 float c3DVi::Dot (`c3DVi lpValue`)

Definition at line 135 of file WTMath.cpp.

21.4.2.2 int c3DVi::Magnitude ()

This will return the absolute size of the vector.

Definition at line 85 of file WTMath.cpp.

21.4.2.3 c3DVi c3DVi::operator* (`c3DVi lvOther`)

Definition at line 115 of file WTMath.cpp.

21.4.2.4 c3DVi c3DVi::operator= (`c3DVi lpValue`)

Definition at line 106 of file WTMath.cpp.

21.4.2.5 c3DVi * c3DVi::operator= (`c3DVi *lpValue`)

Definition at line 96 of file WTMath.cpp.

21.4.2.6 int& c3DVi::operator[] (`uint32 liPos`) [inline]

Definition at line 92 of file WTMath.h.

21.4.3 Member Data Documentation

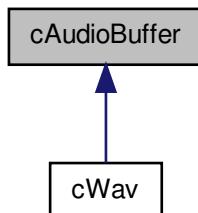
21.4.3.1 int c3DVi::v[3]

Definition at line 77 of file WTMath.h.

21.5 cAudioBuffer Class Reference

This class will create a buffer space to store sound data. This class will create and initialise an OpenAL buffer. The Buffer stores the sound data and is linked to a [cAudioObject](#) to allow it to be played.

Inheritance diagram for cAudioBuffer:



Public Member Functions

- [cAudioBuffer \(\)](#)

Constructor to Initialise a Buffer.

- [~cAudioBuffer \(\)](#)
- [ALuint Buffer \(\)](#)

This will return the ID of the buffer owned by this class.

Protected Attributes

- [ALuint miBuffer](#)

21.5.1 Detailed Description

This class will create a buffer space to store sound data. This class will create and initialise an OpenAL buffer. The Buffer stores the sound data and is linked to a [cAudioObject](#) to allow it to be played.

Definition at line 8 of file WTcAudioBuffer.h.

21.5.2 Constructor & Destructor Documentation

21.5.2.1 `cAudioBuffer::cAudioBuffer()`

Constructor to Initialise a Buffer.

Definition at line 3 of file WTcAudioBuffer.cpp.

21.5.2.2 `cAudioBuffer::~cAudioBuffer()`

Definition at line 13 of file WTcAudioBuffer.cpp.

21.5.3 Member Function Documentation

21.5.3.1 `ALuint cAudioBuffer::Buffer()`

This will return the ID of the buffer owned by this class.

Definition at line 19 of file WTcAudioBuffer.cpp.

21.5.4 Member Data Documentation

21.5.4.1 `ALuint cAudioBuffer::miBuffer [protected]`

Definition at line 12 of file WTcAudioBuffer.h.

21.6 cAudioDevice Class Reference

This class will initialise the sound card. This class will initialise an audio device so sounds can be played on the system.

Collaboration diagram for cAudioDevice:



Public Member Functions

- [cAudioDevice \(\)](#)
Constructor will start and initialise an OpenAL system.
- [~cAudioDevice \(\)](#)

Static Public Member Functions

- static [cAudioDevice * Instance \(\)](#)
This Function will return a pointer to the current OpenAL Audio Device.

21.6.1 Detailed Description

This class will initialise the sound card. This class will initialise an audio device so sounds can be played on the system.

Definition at line 7 of file WTcAudioDevice.h.

21.6.2 Constructor & Destructor Documentation

21.6.2.1 [cAudioDevice::cAudioDevice \(\)](#)

Constructor will start and initialise an OpenAL system.

Definition at line 5 of file WTcAudioDevice.cpp.

21.6.2.2 [cAudioDevice::~cAudioDevice \(\)](#)

Definition at line 25 of file WTcAudioDevice.cpp.

21.6.3 Member Function Documentation

21.6.3.1 `cAudioDevice *cAudioDevice::Instance() [static]`

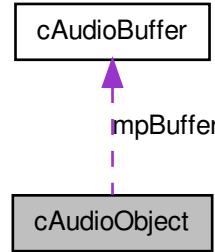
This Function will return a pointer to the current OpenAL Audio Device.

Definition at line 32 of file WTcAudioDevice.cpp.

21.7 cAudioObject Class Reference

This class will allow a sound to be played. This class will link an audio source and a buffer, This allows the sound data stored in the buffer to be played through the source. Each source is an audio channel and can only play one sound at a time.

Collaboration diagram for cAudioObject:



Public Member Functions

- `cAudioObject ()`

This will create an OpenAL source for this object, without linking any cAudioBuffers to it.

- `cAudioObject (cAudioBuffer *lpBuffer)`

This will create an OpenAL source and link this object to a buffer.

- `~cAudioObject ()`

- `void SetBuffer (cAudioBuffer *lpBuffer)`

This will link the OpenAL buffer pointed to by lpBuffer to this Audio Object ready to be played.

- `void Play ()`

This will play a sound through the OpenAL source from the buffer.

- ALuint [Source \(\)](#)

This will return the ID of the OpenAL ID.

21.7.1 Detailed Description

This class will allow a sound to be played. This class will link an audio source and a buffer. This allows the sound data stored in the buffer to be played through the source. Each source is an audio channel and can only play one sound at a time.

Definition at line 7 of file WTcAudioObject.h.

21.7.2 Constructor & Destructor Documentation

21.7.2.1 [cAudioObject::cAudioObject \(\)](#)

This will create an OpenAL source for this object, without linking any cAudioBuffers to it.

Definition at line 3 of file WTcAudioObject.cpp.

21.7.2.2 [cAudioObject::cAudioObject \(cAudioBuffer * lpBuffer \)](#)

This will create an OpenAL source and link this object to a buffer.

Definition at line 9 of file WTcAudioObject.cpp.

21.7.2.3 [cAudioObject::~cAudioObject \(\)](#)

Definition at line 17 of file WTcAudioObject.cpp.

21.7.3 Member Function Documentation

21.7.3.1 [void cAudioObject::Play \(\)](#)

This will play a sound through the OpenAL source from the buffer.

Definition at line 22 of file WTcAudioObject.cpp.

21.7.3.2 [void cAudioObject::SetBuffer \(cAudioBuffer * lpBuffer \)](#)

This will link the OpenAL buffer pointed to by lpBuffer to this Audio Object ready to be played.

Definition at line 27 of file WTcAudioObject.cpp.

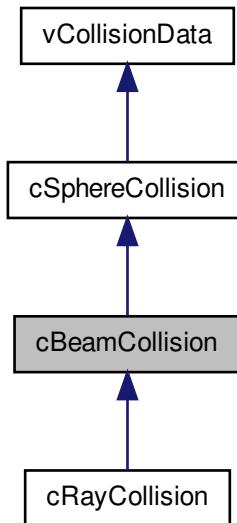
21.7.3.3 ALuint cAudioObject::Source()

This will return the ID of the OpenAL ID.

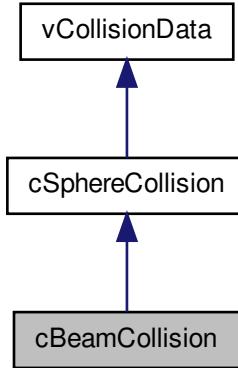
Definition at line 35 of file WTcAudioObject.cpp.

21.8 cBeamCollision Class Reference

Inheritance diagram for cBeamCollision:



Collaboration diagram for cBeamCollision:



Public Member Functions

- [cBeamCollision \(\)](#)
- [virtual ~cBeamCollision \(\)](#)
- [void BuildObject \(float lfLength, float lfRadius\)](#)

*This will generate a [cBeamCollision](#) data set of length *lfLength* and Radius *lfRadius*.*

- [virtual cBeamCollision * Beam \(\)](#)

Will return a pointer if this object contains a Beam collision data object. Otherwise returns 0;.

- [virtual cRayCollision * Ray \(\)](#)

Will return a pointer if this object contains a Ray collision data object. Otherwise returns 0;.

- [virtual void Update \(cMatrix4 &New\)](#)
- [float RayVector \(uint8 liPos\)](#)
- [float * RayVector \(\)](#)

Will return a float array with the beams global vector representing the direction it is pointing in. This should be normalised.

- [float Length \(\)](#)

Will return the length of the beam.

Protected Attributes

- float `mfRaySize` [3]
- float `mfLength`

21.8.1 Detailed Description

This class is for representing beams. Think energy beams. The system will assume that the object is a sphere of radius equal to the value set in `SetSize(float *lfSize)`. The system will project the beam from the centrepoint of the beams current position in the direction of the Z-axis for a distance of `mfLength`. This is a fast and perfect (if the beam is a cylinder capped with hemi-spheres) way of colliding straight energy beams. See `vCollisionData` for more information.

Definition at line 65 of file `WTvCollisionData.h`.

21.8.2 Constructor & Destructor Documentation

21.8.2.1 `cBeamCollision::cBeamCollision()`

Definition at line 154 of file `WTvCollisionData.cpp`.

21.8.2.2 `cBeamCollision::~cBeamCollision()` [virtual]

Definition at line 155 of file `WTvCollisionData.cpp`.

21.8.3 Member Function Documentation

21.8.3.1 `cBeamCollision * cBeamCollision::Beam()` [virtual]

Will return a pointer if this object contains a Beam collision data object. Otherwise returns 0;

Reimplemented from `cSphereCollision`.

Definition at line 157 of file `WTvCollisionData.cpp`.

21.8.3.2 `void cBeamCollision::BuildObject(float lfLength, float lfRadius)`

This will generate a `cBeamCollision` data set of length `lflength` and Radius `lfRadius`.

Definition at line 156 of file `WTvCollisionData.cpp`.

21.8.3.3 `float cBeamCollision::Length()`

Will return the length of the beam.

Definition at line 161 of file `WTvCollisionData.cpp`.

21.8.3.4 cRayCollision * cBeamCollision::Ray() [virtual]

Will return a pointer if this object contains a Ray collision data object. Otherwise returns 0;.

Reimplemented from [cSphereCollision](#).

Reimplemented in [cRayCollision](#).

Definition at line 158 of file WTvCollisionData.cpp.

21.8.3.5 float cBeamCollision::RayVector(uint8 liPos)

Definition at line 159 of file WTvCollisionData.cpp.

21.8.3.6 float * cBeamCollision::RayVector()

Will return a float array with the beams global vector representing the direction it is pointing in. This should be normalised.

Definition at line 160 of file WTvCollisionData.cpp.

21.8.3.7 void cBeamCollision::Update(cMatrix4 & New) [virtual]

Reimplemented from [cSphereCollision](#).

Reimplemented in [cRayCollision](#).

Definition at line 8 of file WTvCollisionData.cpp.

21.8.4 Member Data Documentation**21.8.4.1 float cBeamCollision::mfLength [protected]**

Definition at line 70 of file WTvCollisionData.h.

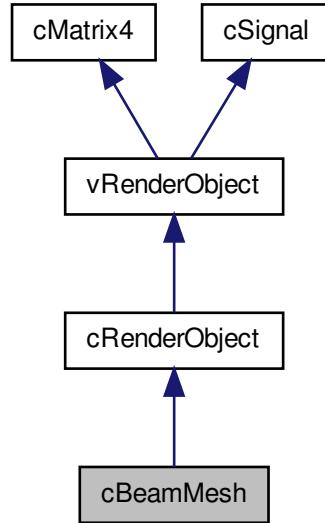
21.8.4.2 float cBeamCollision::mfRaySize[3] [protected]

Definition at line 69 of file WTvCollisionData.h.

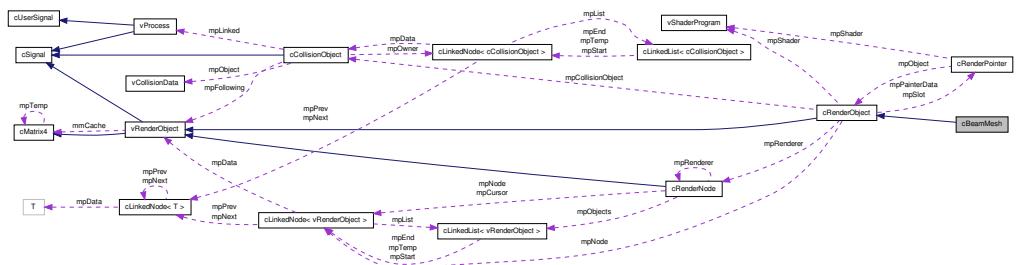
21.9 cBeamMesh Class Reference

A Procedurally generated cylindrical Renderable Object. This class will generate a cylinder with specified dimensions and segments. The origin for the cylinder is in the radial center of one end of the cylinder.

Inheritance diagram for cBeamMesh:



Collaboration diagram for cBeamMesh:



Public Member Functions

- `cBeamMesh (float Radius=0.1f, float Length=1.0f, uint16 Segments=6, cRenderNode *lpNode=cCamera::Instance()->RenderList())`

Constructor to Create a Beam with the specified Dimensions. Segments must be an even integer.

- `~cBeamMesh ()`
- `void Length (float Length)`
Sets the Length of the Cylinder.
- `void Radius (float Radius)`
Sets the Radius of the Cylinder.
- `void GenerateData (float Radius, float Length, uint16 Segments)`
Will regenerate the cylinder with the specified Specifications.
- `float Length ()`
Returns the current Length of the Cylinder.
- `float Radius ()`
Returns the current Radius of the Cylinder.
- `void BufferBeam ()`
- `void RenderBeam ()`
- `void RenderPainter (uint8 liLevel)`
virtual function to allow polymorphism. see `cCamera::RenderPainter()`;
- `void RenderToPainter ()`
virtual function to allow polymorphism. see `cCamera::RenderToPainter()`;
- `void Render ()`
virtual function to allow polymorphiss. see `cCamera::Render()`;

21.9.1 Detailed Description

A Procedurally generated cylindrical Renderable Object. This class will generate a cylinder with specified dimensions and segments. The origin for the cylinder is in the radial center of one end of the cylinder.

Definition at line 8 of file WTcBeamMesh.h.

21.9.2 Constructor & Destructor Documentation

21.9.2.1 `cBeamMesh::cBeamMesh (float Radius = 0.1f, float Length = 1.0f, uint16 Segments = 6, cRenderNode * lpNode = cCamera::Instance() ->RenderList ())`

Constructor to Create a Beam with the specified Dimensions. Segments must be an even integer.

Definition at line 12 of file WTcBeamMesh.cpp.

21.9.2.2 cBeamMesh::~cBeamMesh ()

Definition at line 3 of file WTcBeamMesh.cpp.

21.9.3 Member Function Documentation**21.9.3.1 void cBeamMesh::BufferBeam ()**

Definition at line 35 of file WTcBeamMesh.cpp.

21.9.3.2 void cBeamMesh::GenerateData (float Radius, float Length, uint16 Segments)

Will regenerate the cylinder with the specified Specifications.

Definition at line 151 of file WTcBeamMesh.cpp.

21.9.3.3 float cBeamMesh::Length () [inline]

Returns the current Length of the Cylinder.

Definition at line 32 of file WTcBeamMesh.h.

21.9.3.4 void cBeamMesh::Length (float Length)

Sets the Length of the Cylinder.

Definition at line 126 of file WTcBeamMesh.cpp.

21.9.3.5 void cBeamMesh::Radius (float Radius)

Sets the Radius of the Cylinder.

Definition at line 101 of file WTcBeamMesh.cpp.

21.9.3.6 float cBeamMesh::Radius () [inline]

Returns the current Radius of the Cylinder.

Definition at line 34 of file WTcBeamMesh.h.

21.9.3.7 void cBeamMesh::Render () [virtual]

virtual function to allow polymorphiss. see [cCamera::Render\(\)](#);

Implements [cRenderObject](#).

Definition at line 77 of file WTcBeamMesh.cpp.

21.9.3.8 void cBeamMesh::RenderBeam()

Definition at line 19 of file WTcBeamMesh.cpp.

21.9.3.9 void cBeamMesh::RenderPainter(uint8 *liLevel*) [virtual]

virtual function to allow polymorphism. see cCamera::RenderPainter();

Implements [cRenderObject](#).

Definition at line 50 of file WTcBeamMesh.cpp.

21.9.3.10 void cBeamMesh::RenderToPainter() [virtual]

virtual function to allow polymorphism. see cCamera::RenderToPainter();

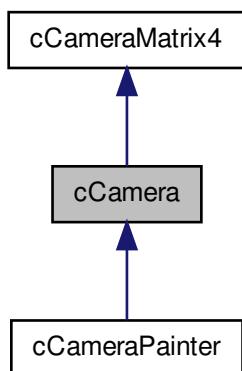
Implements [cRenderObject](#).

Definition at line 59 of file WTcBeamMesh.cpp.

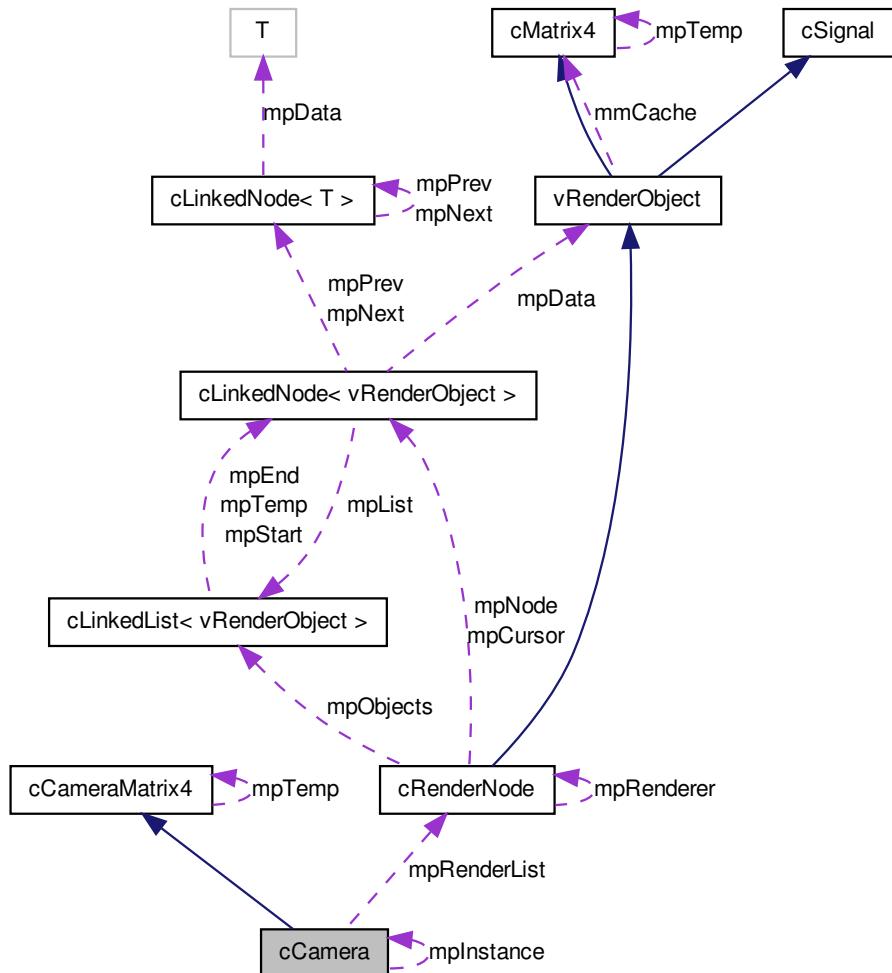
21.10 cCamera Class Reference

This class will control the rendering of the render tree. It will handle all the render objects and how to render to the screen. [cCamera](#) is the renderer for the entire program. It will contain a scene graph, and inherits [cCameraMatrix4](#) as its view matrix. It can be considered as a camera in the program space, and will render every thing in its scene graph to the screen.

Inheritance diagram for cCamera:



Collaboration diagram for cCamera:



Public Member Functions

- `~cCamera ()`

This is cCameras destructor and will delete the entire scene graph stored in mpRenderList.

- `void RemoveAll ()`

This function will automatically remove all cRenderObjects from the scene graph pointed to by mpRenderList.

- virtual void [Render \(\)](#)

This function will render the scene graph to the screen. This is used if WT_USE_PAINTER is set to false.

- void [SetClearColor \(float lfRed, float lfGreen, float lfBlue, float lfAlpha\)](#)

This function will set the color the camera will clear the screen to every frame.

- void [Orthographic \(\)](#)

This creates the camera model view matrix using an Orthographic algorithm.

- void [Frustum \(\)](#)

This creates the camera model view matrix using a Frustum algorithm.

- [cRenderNode * RenderList \(\)](#)

This will return a pointer to the scene graph.

- [vRenderObject * vRenderList \(\)](#)

This will return a virtual pointer to the the scene graph.

- float [Near \(\)](#)

This will return the closest distance [cCamera](#) will render polygons.

- float [Far \(\)](#)

This will return the Furthest distance [cCamera](#) will render polygons.

- void [Near \(float lfN\)](#)

This will set the closest distance [cCamera](#) will render polygons.

- void [Far \(float lfF\)](#)

This will set the Furthest distance [cCamera](#) will render polygons.

- void [ResetGLMatrix \(\)](#)

Static Public Member Functions

- static [cCamera * Instance \(\)](#)

Protected Member Functions

- [cCamera \(\)](#)

Protected Attributes

- float `mfZoom`
- float `mfNear`
- float `mfFar`
- `cRenderNode * mpRenderList`

Static Protected Attributes

- static `cCamera * mpInstance = 0`

21.10.1 Detailed Description

This class will control the rendering of the render tree. It will handle all the render objects and how to render to the screen. `cCamera` is the renderer for the entire program. It will contain a scene graph, and inherits `cCameraMatrix4` as its view matrix. It can be considered as a camera in the program space, and will render every thing in its scene graph to the screen.

Definition at line 11 of file WTcCamera.h.

21.10.2 Constructor & Destructor Documentation

21.10.2.1 `cCamera::cCamera()` [protected]

Definition at line 3 of file WTcCamera.cpp.

21.10.2.2 `cCamera::~cCamera()`

This is `cCameras` destructor and will delete the entire scene graph stored in `mpRenderList`.

Definition at line 15 of file WTcCamera.cpp.

21.10.3 Member Function Documentation

21.10.3.1 `float cCamera::Far()`

This will return the Furthest distance `cCamera` will render polygons.

Definition at line 101 of file WTcCamera.cpp.

21.10.3.2 `void cCamera::Far(float fF)`

This will set the Furthest distance `cCamera` will render polygons.

Definition at line 147 of file WTcCamera.cpp.

21.10.3.3 void cCamera::Frustum()

This creates the camera model view matrix using a Frustum algorithm.

Definition at line 87 of file WTcCamera.cpp.

21.10.3.4 cCamera * cCamera::Instance() [static]

This function will return a pointer to the current [cCamera](#) object.

Definition at line 115 of file WTcCamera.cpp.

21.10.3.5 float cCamera::Near()

This will return the closest distance [cCamera](#) will render polygons.

Definition at line 96 of file WTcCamera.cpp.

21.10.3.6 void cCamera::Near(float fN)

This will set the closest distance [cCamera](#) will render polygons.

Definition at line 141 of file WTcCamera.cpp.

21.10.3.7 void cCamera::Orthographic()

This creates the camera model view matrix using an Orthographic algorithm.

Definition at line 78 of file WTcCamera.cpp.

21.10.3.8 void cCamera::RemoveAll()

This function will automatically remove all [cRenderObjects](#) from the scene graph pointed to by mpRenderList.

Definition at line 72 of file WTcCamera.cpp.

21.10.3.9 void cCamera::Render() [virtual]

This function will render the scene graph to the screen. This is used if [WT_USE_PAINTER](#) is set to false.

Reimplemented in [cCameraPainter](#).

Definition at line 29 of file WTcCamera.cpp.

21.10.3.10 cRenderNode * cCamera::RenderList()

This will return a pointer to the scene graph.

Definition at line 128 of file WTcCamera.cpp.

21.10.3.11 void cCamera::ResetGLMatrix()

Definition at line 106 of file WTcCamera.cpp.

21.10.3.12 void cCamera::SetClearColor(float lfRed, float lfGreen, float lfBlue, float lfAlpha)

This function will set the color the camera will clear the screen to every frame.

Parameters

<i>lfRed</i>	This is the red component of the color that cCamera will clear the screen to.
<i>lfGreen</i>	This is the green component of the color that cCamera will clear the screen to.
<i>lfBlue</i>	This is the blue component of the color that cCamera will clear the screen to.
<i>lfAlpha</i>	This is the alpha component of the color that cCamera will clear the screen to. This function sets the color that the camera will clear the screen to every frame. RGB defines the color while lfAlpha sets the translucency of the screen clear - this can be used to leave screen echos.

Definition at line 67 of file WTcCamera.cpp.

21.10.3.13 vRenderObject * cCamera::vRenderList()

This will return a virtual pointer to the the scene graph.

Definition at line 135 of file WTcCamera.cpp.

21.10.4 Member Data Documentation

21.10.4.1 float cCamera::mfFar [protected]

Definition at line 22 of file WTcCamera.h.

21.10.4.2 float cCamera::mfNear [protected]

Definition at line 20 of file WTcCamera.h.

21.10.4.3 float cCamera::mfZoom [protected]

Definition at line 18 of file WTcCamera.h.

21.10.4.4 cCamera * cCamera::mpInstance = 0 [static, protected]

Definition at line 15 of file WTcCamera.h.

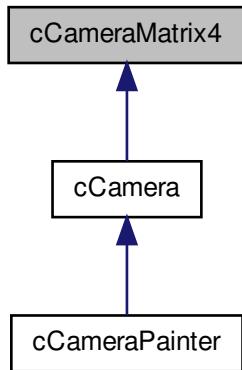
21.10.4.5 cRenderNode* cCamera::mpRenderList [protected]

Definition at line 28 of file WTcCamera.h.

21.11 cCameraMatrix4 Class Reference

This is a translation matrix for a camera object. This is a translation matrix for a camera. All the translations are inverted. Distances are 'reversed', Local rotations are 'globalised' and Global rotations are 'localised'. Effectively all the translations are inverted before they are applied to this matrix.

Inheritance diagram for cCameraMatrix4:



Collaboration diagram for cCameraMatrix4:



Public Member Functions

- [cCameraMatrix4 \(\)](#)

This will create this matrix, and initialise all the static data for operations.

- float [Determinant \(\)](#)

This will return the determinant of this matrix.

- [cCameraMatrix4 Transpose \(\)](#)

This will return the transpose of this objects matrix ready for multiplications etc.

- void [InvSign \(\)](#)

This will make all terms of the matrix, the opposite sign to what they area.

- float * [Matrix \(\)](#)

This will return a pointer to this objects matrix.

- float * [Position \(\)](#)

This will return the position vector of this objects matrix.

- float [X \(\)](#)

This will return the X position of this objects matrix.

- float [Y \(\)](#)

This will return the Y position of this objects matrix.

- float [Z \(\)](#)

This will return the Z position of this objects matrix.

- void [Identity \(\)](#)

This will restore this objects matrix to an Identity matrix.

- void [Zero \(\)](#)

This will make the entire objects matrix equal to zero.

- **cCameraMatrix4 operator*** (float &lVal)
This will multiply the matrix by a float.
- **cCameraMatrix4 operator*** (const float lVal)
This will multiply the matrix by a float.
- **cCameraMatrix4 operator*** (cCameraMatrix4 &lVal)
This will multiply this matrix by the matrix lVal.
- **cCameraMatrix4 * operator*** (cCameraMatrix4 *lVal)
This will multiply this matrix by the matrix pointed to by lVal.
- **cCameraMatrix4 operator/** (float &lVal)
This will devide this objects matrix by the float lVal.
- **cCameraMatrix4 operator/** (const float lVal)
This will devide this objects matrix by the float lVal.
- **cCameraMatrix4 operator+** (float &lVal)
This will add lVal to this objects matrix. lVal is added to every position in the matrix.
- **cCameraMatrix4 operator+** (const float lVal)
This will add lVal to this objects matrix. lVal is added to every position in the matrix.
- **cCameraMatrix4 operator+** (cCameraMatrix4 &lVal)
This will add this objects matrix and the matrix lVal.
- **cCameraMatrix4 operator-** (float &lVal)
This will deduct lVal from this objects matrix. lVal is deducted from every position in the matrix.
- **cCameraMatrix4 operator-** (const float lVal)
This will deduct lVal from this objects matrix. lVal is deducted from every position in the matrix.
- **cCameraMatrix4 operator-** (cCameraMatrix4 &lVal)
This will deduct the matrix lVal from this objects matrix.
- float & **operator[]** (uint16 liPos)
This will return the float in the position liPos in this objects matrix.
- float & **operator()** (uint16 liColumn, uint16 liRow)
This will return the float in the position [liColumn,liRow] in this objects matrix.
- float **operator=** (float &lVal)

This will set every float in this objects matrix to lVal.

- float **operator=** (const float lVal)

This will set every float in this objects matrix to lVal.

- **cCameraMatrix4 operator= (cCameraMatrix4 &lVal)**

This will set this objects matrix to be the same as the matrix lVal.

- **cCameraMatrix4 * operator= (cCameraMatrix4 *lVal)**

This will set this objects matrix to be the same as the matrix lVal.

- void **Position (c2DVf *lpPosition)**

This will set this objects matrix position (2D - X,Y) vector to be the same as lpPosition.

- void **Position (float lfX, float lfY)**

This will set this objects matrix position vector to be lfX,lfY.

- void **Position (c3DVf *lpPosition)**

This will set this objects matrix position vector to be the same as lpPosition.

- void **Position (float lfX, float lfY, float lfZ)**

This will set this objects matrix position vector to be lfX,lfY,lfZ.

- void **PositionX (float lfX)**

This will set this objects X position to he lfX.

- void **PositionY (float lfY)**

This will set this objects Y position to he lfY.

- void **PositionZ (float lfZ)**

This will set this objects Z position to he lfZ.

- void **AdvanceX (float lfDistance)**

This will advance the position of this objects X position by lfDistance along local axis.

- void **AdvanceY (float lfDistance)**

This will advance the position of this objects Y position by lfDistance along local axis.

- void **AdvanceZ (float lfDistance)**

This will advance the position of this objects Z position by lfDistance along local axis.

- void **Advance (float lfX, float lfY, float lfZ)**

This will advance the position of this objects X,Y and Z positions by lfX,lfY and lfZ along local axis.

- void **GAdvanceX (float lfDistance)**

This will advance the position of this objects X position by lfDistance along global axis.

- void **GAdvanceY** (float lfDistance)

This will advance the position of this objects Y position by lfDistance along global axis.

- void **GAdvanceZ** (float lfDistance)

This will advance the position of this objects Z position by lfDistance along global axis.

- void **GAdvance** (float lfX, float lfY, float lfZ)

This will advance the position of this objects X,Y and Z positions by lfX,lfY and lfZ along global axis.

- void **LRotate** (float lfAngle)

This will locally rotate the object by lfAngle radians about the Z axis. This is suitable to be used by 2D objects.

- void **LRotateX** (float lfAngle)

This will rotate the object by lfAngle radians about its local X axis. This is suitable for use by 3D objects.

- void **LRotateY** (float lfAngle)

This will rotate the object by lfAngle radians about its local Y axis. This is suitable for use by 3D objects.

- void **LRotateZ** (float lfAngle)

This will rotate the object by lfAngle radians about its local Z axis. This is suitable for use by 3D objects.

- void **GRotateX** (float lfAngle)

This will rotate the object by lfAngle radians about its global X axis. This is suitable for use by 3D objects.

- void **GRotateY** (float lfAngle)

This will rotate the object by lfAngle radians about its global Y axis. This is suitable for use by 3D objects.

- void **GRotateZ** (float lfAngle)

This will rotate the object by lfAngle radians about its global Z axis. This is suitable for use by 3D objects.

- void **GRotateX** (float lfAngle, float lfX, float lfY, float lfZ)

This will rotate the object by lfAngle radians about the global X axis of the point lfX,lfY,lfZ. This is suitable for use by 3D objects.

- void **GRotateY** (float lfAngle, float lfX, float lfY, float lfZ)

This will rotate the object by lfAngle radians about the global Y axis of the point lfX,lfY,lfZ. This is suitable for use by 3D objects.

- void **GRotateZ** (float lfAngle, float lfX, float lfY, float lfZ)

This will rotate the object by lfAngle radians about the global Z axis of the point lfX,lfY,lfZ. This is suitable for use by 3D objects.

- void **Angle** (float lfAngle)

This will set the current objects matrix to be rotated to the absolute angle lfAngle. This is suitable for 2D objects.

- void **Resize** (float lfScale)

This will scale this objects matrix by a factor of lfScale.

- void **LResizeX** (float lfScale)

This will scale this objects local X axis by a factor of lfScale.

- void **LResizeY** (float lfScale)

This will scale this objects local Y axis by a factor of lfScale.

- void **LResizeZ** (float lfScale)

This will scale this objects local Z axis by a factor of lfScale.

- void **GResizeX** (float lfScale)

This will scale this objects globally along the X axis by a factor of lfScale.

- void **GResizeY** (float lfScale)

This will scale this objects globally along the Y axis by a factor of lfScale.

- void **GResizeZ** (float lfScale)

This will scale this objects globally along the Z axis by a factor of lfScale.

- uint32 **Distance3D** (float *lpOther)

This will return the 3D distance between this matrix and the matrix pointed to by lpOther.

- uint32 **Distance2D** (float *lpOther)

This will return the 2D distance between this matrix and the matrix pointed to by lpOther.

- void **Follow** (cMatrix4 *lpOther, float lfDist)

- void **PointAt** (float *mpPos)

Public Attributes

- float `mpPosition [3]`

The position of this matrix has been separated from the rest of the matrix as the camera should rotate around 0,0,0, not itself?

21.11.1 Detailed Description

This is a translation matrix for a camera object. This is a translation matrix for a camera. All the translations are inverted. Distances are 'reversed', Local rotations are 'globalised' and Global rotations are 'localised'. Effectively all the translations are inverted before they are applied to this matrix.

Definition at line 10 of file WTcCameraMatrix4.h.

21.11.2 Constructor & Destructor Documentation

21.11.2.1 `cCameraMatrix4::cCameraMatrix4()`

This will create this matrix, and initialise all the static data for operations.

Definition at line 9 of file WTcCameraMatrix4.cpp.

21.11.3 Member Function Documentation

21.11.3.1 `void cCameraMatrix4::Advance(float IfX, float IfY, float IfZ)`

This will advance the position of this objects X,Y and Z positions by IfX,IfY and IfZ along local axis.

Definition at line 742 of file WTcCameraMatrix4.cpp.

21.11.3.2 `void cCameraMatrix4::AdvanceX(float IfDistance)`

This will advance the position of this objects X position by IfDistance along local axis.

Definition at line 720 of file WTcCameraMatrix4.cpp.

21.11.3.3 `void cCameraMatrix4::AdvanceY(float IfDistance)`

This will advance the position of this objects Y position by IfDistance along local axis.

Definition at line 727 of file WTcCameraMatrix4.cpp.

21.11.3.4 `void cCameraMatrix4::AdvanceZ(float IfDistance)`

This will advance the position of this objects Z position by IfDistance along local axis.

Definition at line 734 of file WTcCameraMatrix4.cpp.

21.11.3.5 void cCameraMatrix4::Angle (float *lfAngle*)

This will set the current objects matrix to be rotated to the absolute angle *lfAngle*. This is suitable for 2D objects.

Definition at line 513 of file WTcCameraMatrix4.cpp.

21.11.3.6 float cCameraMatrix4::Determinant ()

This will return the determinant of this matrix.

Definition at line 350 of file WTcCameraMatrix4.cpp.

21.11.3.7 uint32 cCameraMatrix4::Distance2D (float * *lpOther*)

This will return the 2D distance between this matrix and the matrix pointed to by *lpOther*.

Definition at line 861 of file WTcCameraMatrix4.cpp.

21.11.3.8 uint32 cCameraMatrix4::Distance3D (float * *lpOther*)

This will return the 3D distance between this matrix and the matrix pointed to by *lpOther*.

Definition at line 852 of file WTcCameraMatrix4.cpp.

21.11.3.9 void cCameraMatrix4::Follow (cMatrix4 * *lpOther*, float *lfDist*)

Definition at line 869 of file WTcCameraMatrix4.cpp.

21.11.3.10 void cCameraMatrix4::GAdvance (float *lfX*, float *lfY*, float *lfZ*)

This will advance the position of this objects X,Y and Z positions by *lfX*,*lfY* and *lfZ* along global axis.

Definition at line 758 of file WTcCameraMatrix4.cpp.

21.11.3.11 void cCameraMatrix4::GAdvanceX (float *lfDistance*)

This will advance the position of this objects X position by *lfDistance* along global axis.

Definition at line 749 of file WTcCameraMatrix4.cpp.

21.11.3.12 void cCameraMatrix4::GAdvanceY (float lfDistance)

This will advance the position of this objects Y position by lfDistance along global axis.

Definition at line 752 of file WTcCameraMatrix4.cpp.

21.11.3.13 void cCameraMatrix4::GAdvanceZ (float lfDistance)

This will advance the position of this objects Z position by lfDistance along global axis.

Definition at line 755 of file WTcCameraMatrix4.cpp.

21.11.3.14 void cCameraMatrix4::GResizeX (float lfScale)

This will scale this objects globally along the X axis by a factor of lfScale.

Definition at line 637 of file WTcCameraMatrix4.cpp.

21.11.3.15 void cCameraMatrix4::GResizeY (float lfScale)

This will scale this objects globally along the Y axis by a factor of lfScale.

Definition at line 645 of file WTcCameraMatrix4.cpp.

21.11.3.16 void cCameraMatrix4::GResizeZ (float lfScale)

This will scale this objects globally along the Z axis by a factor of lfScale.

Definition at line 653 of file WTcCameraMatrix4.cpp.

21.11.3.17 void cCameraMatrix4::GRotateX (float lfAngle)

This will rotate the object by lfAngle radians about its global X axis. This is suitable for use by 3D objects.

Definition at line 661 of file WTcCameraMatrix4.cpp.

21.11.3.18 void cCameraMatrix4::GRotateX (float lfAngle, float lfX, float lfY, float lfZ)

This will rotate the object by lfAngle radians about the global X axis of the point lfX,lfY,lfZ. This is suitable for use by 3D objects.

Definition at line 765 of file WTcCameraMatrix4.cpp.

21.11.3.19 void cCameraMatrix4::GRotateY (float lfAngle)

This will rotate the object by lfAngle radians about its global Y axis. This is suitable for use by 3D objects.

Definition at line 681 of file WTcCameraMatrix4.cpp.

21.11.3.20 void cCameraMatrix4::GRotateY (float lfAngle, float lfX, float lfY, float lfZ)

This will rotate the object by lfAngle radians about the global Y axis of the point lfX,lfY,lfZ. This is suitable for use by 3D objects.

Definition at line 795 of file WTcCameraMatrix4.cpp.

21.11.3.21 void cCameraMatrix4::GRotateZ (float lfAngle)

This will rotate the object by lfAngle radians about its global Z axis. This is suitable for use by 3D objects.

Definition at line 702 of file WTcCameraMatrix4.cpp.

21.11.3.22 void cCameraMatrix4::GRotateZ (float lfAngle, float lfX, float lfY, float lfZ)

This will rotate the object by lfAngle radians about the global Z axis of the point lfX,lfY,lfZ. This is suitable for use by 3D objects.

Definition at line 825 of file WTcCameraMatrix4.cpp.

21.11.3.23 void cCameraMatrix4::Identity ()

This will restore this objects matrix to an Identity matrix.

Definition at line 302 of file WTcCameraMatrix4.cpp.

21.11.3.24 void cCameraMatrix4::InvSign ()

This will make all terms of the matrix, the opposite sign to what they area.

Definition at line 903 of file WTcCameraMatrix4.cpp.

21.11.3.25 void cCameraMatrix4::LResizeX (float lfScale)

This will scale this objects local X axis by a factor of lfScale.

Definition at line 610 of file WTcCameraMatrix4.cpp.

21.11.3.26 void cCameraMatrix4::LResizeY (float lfScale)

This will scale this objects local Y axis by a factor of lfScale.

Definition at line 618 of file WTcCameraMatrix4.cpp.

21.11.3.27 void cCameraMatrix4::LResizeZ (float lfScale)

This will scale this objects local Z axis by a factor of lfScale.

Definition at line 627 of file WTcCameraMatrix4.cpp.

21.11.3.28 void cCameraMatrix4::LRotate (float lfAngle)

This will locally rotate the object by lfAngle radians about the Z axis. This is suitable to be used by 2D objects.

Definition at line 489 of file WTcCameraMatrix4.cpp.

21.11.3.29 void cCameraMatrix4::LRotateX (float lfAngle)

This will rotate the object by lfAngle radians about its local X axis. This is suitable for use by 3D objects.

Definition at line 520 of file WTcCameraMatrix4.cpp.

21.11.3.30 void cCameraMatrix4::LRotateY (float lfAngle)

This will rotate the object by lfAngle radians about its local Y axis. This is suitable for use by 3D objects.

Definition at line 544 of file WTcCameraMatrix4.cpp.

21.11.3.31 void cCameraMatrix4::LRotateZ (float lfAngle)

This will rotate the object by lfAngle radians about its local Z axis. This is suitable for use by 3D objects.

Definition at line 568 of file WTcCameraMatrix4.cpp.

21.11.3.32 float* cCameraMatrix4::Matrix () [inline]

This will return a pointer to this objects matrix.

Definition at line 38 of file WTcCameraMatrix4.h.

21.11.3.33 float & cCameraMatrix4::operator() (uint16 *iColumn*, uint16 *iRow*)

This will return the float in the position [*iColumn*,*iRow*] in this objects matrix.

Definition at line 443 of file WTcCameraMatrix4.cpp.

21.11.3.34 cCameraMatrix4 cCameraMatrix4::operator* (float & *IVal*)

This will multiplty the matrix by a float.

Definition at line 148 of file WTcCameraMatrix4.cpp.

21.11.3.35 cCameraMatrix4 cCameraMatrix4::operator* (const float *IVal*)

This will multiplty the matrix by a float.

Definition at line 171 of file WTcCameraMatrix4.cpp.

21.11.3.36 cCameraMatrix4 * cCameraMatrix4::operator* (cCameraMatrix4 * *IVal*)

This will multiply this matrix by the matrix pointed to by *IVal*.

Parameters

<i>IVal</i>	is a pointer to a cCameraMatrix4 object to be multiplied by this matrix. This will multiply the cCameraMatrix4 object pointed to by <i>IVal</i> by this objects matrix. This. <i>IVal</i>
-------------	---

Definition at line 412 of file WTcCameraMatrix4.cpp.

21.11.3.37 cCameraMatrix4 cCameraMatrix4::operator* (cCameraMatrix4 & *IVal*)

This will multiply this matrix by the matrix *IVal*.

Parameters

<i>IVal</i>	is a cCameraMatrix4 object to be multiplied by this matrix. This will multiply a cCameraMatrix4 object by this objects matrix. This. <i>IVal</i>
-------------	--

Definition at line 386 of file WTcCameraMatrix4.cpp.

21.11.3.38 cCameraMatrix4 cCameraMatrix4::operator+ (float & *IVal*)

This will add *IVal* to this objects matrix. *IVal* is added to every position in the matrix.

Definition at line 61 of file WTcCameraMatrix4.cpp.

21.11.3.39 cCameraMatrix4 cCameraMatrix4::operator+ (const float lVal)

This will add lVal to this objects matrix. lVal is added to every position in the matrix.

Definition at line 83 of file WTcCameraMatrix4.cpp.

21.11.3.40 cCameraMatrix4 cCameraMatrix4::operator+ (cCameraMatrix4 & lVal)

This will add this objects matrix and the matrix lVal.

Definition at line 307 of file WTcCameraMatrix4.cpp.

21.11.3.41 cCameraMatrix4 cCameraMatrix4::operator- (float & lVal)

This will deduct lVal from this objects matrix. lVal is deducted from every position in the matrix.

Definition at line 104 of file WTcCameraMatrix4.cpp.

21.11.3.42 cCameraMatrix4 cCameraMatrix4::operator- (const float lVal)

This will deduct lVal from this objects matrix. lVal is deducted from every position in the matrix.

Definition at line 127 of file WTcCameraMatrix4.cpp.

21.11.3.43 cCameraMatrix4 cCameraMatrix4::operator- (cCameraMatrix4 & lVal)

This will deduct the matrix lVal from this objects matrix.

Definition at line 328 of file WTcCameraMatrix4.cpp.

21.11.3.44 cCameraMatrix4 cCameraMatrix4::operator/ (float & lVal)

This will devide this objects matrix by the float lVal.

Definition at line 193 of file WTcCameraMatrix4.cpp.

21.11.3.45 cCameraMatrix4 cCameraMatrix4::operator/ (const float lVal)

This will devide this objects matrix by the float lVal.

Definition at line 215 of file WTcCameraMatrix4.cpp.

21.11.3.46 cCameraMatrix4 cCameraMatrix4::operator= (cCameraMatrix4 & lVal)

This will set this objects matrix to be the same as the matrix lVal.

Definition at line 285 of file WTcCameraMatrix4.cpp.

21.11.3.47 float cCameraMatrix4::operator= (const float lVal)

This will set every float in this objects matrix to lVal.

Definition at line 262 of file WTcCameraMatrix4.cpp.

21.11.3.48 float cCameraMatrix4::operator= (float & lVal)

This will set every float in this objects matrix to lVal.

Definition at line 238 of file WTcCameraMatrix4.cpp.

21.11.3.49 cCameraMatrix4 * cCameraMatrix4::operator= (cCameraMatrix4 * lVal)

This will set this objects matrix to be the same as the matrix lVal.

Definition at line 291 of file WTcCameraMatrix4.cpp.

21.11.3.50 float & cCameraMatrix4::operator[] (uint16 liPos)

This will return the float in the position liPos in this objects matrix.

Definition at line 438 of file WTcCameraMatrix4.cpp.

21.11.3.51 void cCameraMatrix4::PointAt (float * mpPos)

Definition at line 884 of file WTcCameraMatrix4.cpp.

21.11.3.52 void cCameraMatrix4::Position (c2DVF * lpPosition)

This will set this objects matrix position (2D - X,Y) vector to be the same as lpPosition.

Definition at line 462 of file WTcCameraMatrix4.cpp.

21.11.3.53 void cCameraMatrix4::Position (float lfX, float lfY, float lfZ)

This will set this objects matrix position vector to be lfX,lfY,lfZ.

Definition at line 454 of file WTcCameraMatrix4.cpp.

21.11.3.54 float* cCameraMatrix4::Position () [inline]

This will return the position vector of this objects matrix.

Definition at line 40 of file WTcCameraMatrix4.h.

21.11.3.55 void cCameraMatrix4::Position (float lfX, float lfY)

This will set this objects matrix position vector to be lfX,lfY.

Definition at line 448 of file WTcCameraMatrix4.cpp.

21.11.3.56 void cCameraMatrix4::Position (c3DVf * lpPosition)

This will set this objects matrix position vector to be the same as lpPosition.

Definition at line 469 of file WTcCameraMatrix4.cpp.

21.11.3.57 void cCameraMatrix4::PositionX (float lfX)

This will set this objects X position to he lfX.

Definition at line 476 of file WTcCameraMatrix4.cpp.

21.11.3.58 void cCameraMatrix4::PositionY (float lfY)

This will set this objects Y position to he lfY.

Definition at line 480 of file WTcCameraMatrix4.cpp.

21.11.3.59 void cCameraMatrix4::PositionZ (float lfZ)

This will set this objects Z position to he lfZ.

Definition at line 484 of file WTcCameraMatrix4.cpp.

21.11.3.60 void cCameraMatrix4::Resize (float lfScale)

This will scale this objects matrix by a factor of lfScale.

Definition at line 593 of file WTcCameraMatrix4.cpp.

21.11.3.61 cCameraMatrix4 cCameraMatrix4::Transpose ()

This will return the transpose of this objects matrix ready for multiplications etc.

Definition at line 359 of file WTcCameraMatrix4.cpp.

21.11.3.62 float cCameraMatrix4::X () [inline]

This will return the X position of this objects matrix.

Definition at line 46 of file WTcCameraMatrix4.h.

21.11.3.63 float cCameraMatrix4::Y() [inline]

This will return the Y position of this objects matrix.

Definition at line 48 of file WTcCameraMatrix4.h.

21.11.3.64 float cCameraMatrix4::Z() [inline]

This will return the Z position of this objects matrix.

Definition at line 50 of file WTcCameraMatrix4.h.

21.11.3.65 void cCameraMatrix4::Zero()

This will make the entire objects matrix equal to zero.

Definition at line 297 of file WTcCameraMatrix4.cpp.

21.11.4 Member Data Documentation

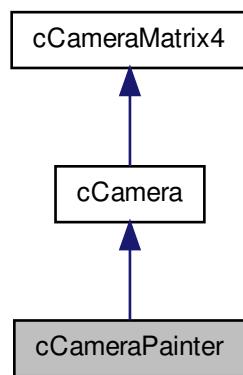
21.11.4.1 float cCameraMatrix4::mpPosition[3]

The position of this matrix has been seperated from the rest of the matrix as the camera should rotate around 0,0,0, not itself?

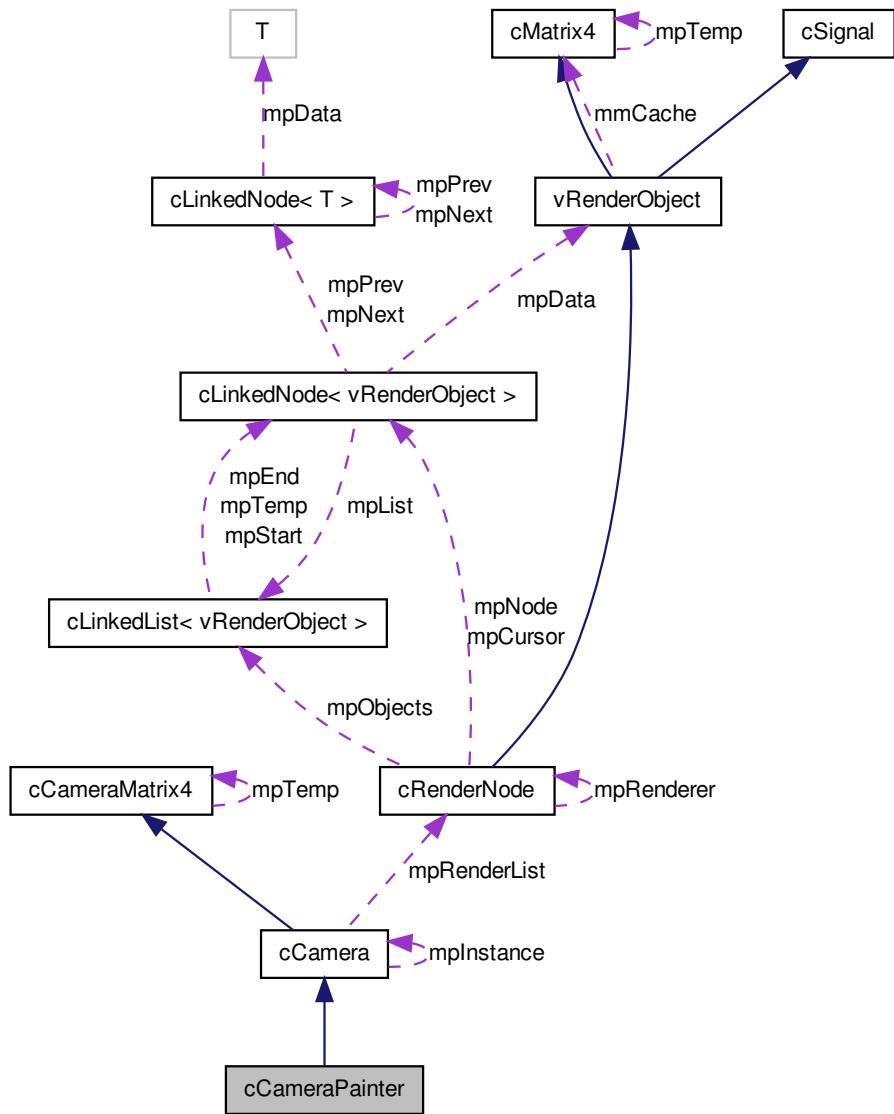
Definition at line 40 of file WTcCameraMatrix4.h.

21.12 cCameraPainter Class Reference

Inheritance diagram for cCameraPainter:



Collaboration diagram for cCameraPainter:



Public Member Functions

- `void Render()`

This function will render the scene graph to `cPainter` instead of the screen. This is

used if WT_USE_PAINTER is set to true.

Friends

- class [cCamera](#)

21.12.1 Detailed Description

Definition at line 82 of file WTcCamera.h.

21.12.2 Member Function Documentation

21.12.2.1 void [cCameraPainter::Render\(\)](#) [virtual]

This function will render the scene graph to [cPainter](#) instead of the screen. This is used if WT_USE_PAINTER is set to true.

Reimplemented from [cCamera](#).

Definition at line 47 of file WTcCamera.cpp.

21.12.3 Friends And Related Function Documentation

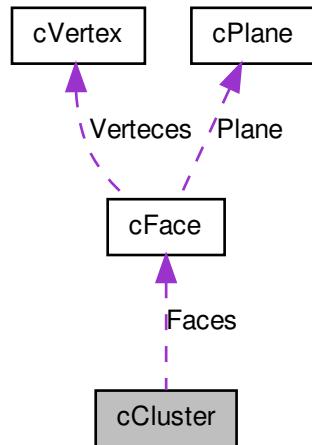
21.12.3.1 friend class [cCamera](#) [friend]

Definition at line 88 of file WTcCamera.h.

21.13 cCluster Class Reference

This class is an array of [cFace](#). This is often used.

Collaboration diagram for cCluster:



Public Member Functions

- `~cCluster ()`
- `cCluster ()`
- `void Init (uint32 liSize)`
- `void Optimise ()`
- `uint32 FileSize ()`
- `void Order ()`
- `void Strip ()`
- `void Switch (uint32 li1, uint32 li2)`
- `void Add (cFullFaceData &lpData)`
- `void Remove (uint32 li1)`
- `uint32 Items ()`
- `cFace Face (uint32 li1)`
- `void Resize ()`
- `void OutputIMFClusters (ofstream &FileStream)`
- `void LoadIMFClusters (ifstream &FileStream)`

21.13.1 Detailed Description

This class is an array of `cFace`. This is often used.

Definition at line 8 of file WTCcluster.h.

21.13.2 Constructor & Destructor Documentation

21.13.2.1 cCluster::~cCluster()

Definition at line 171 of file WTcCluster.cpp.

21.13.2.2 cCluster::cCluster()

Definition at line 172 of file WTcCluster.cpp.

21.13.3 Member Function Documentation

21.13.3.1 void cCluster::Add (cFullFaceData & lpData)

Definition at line 18 of file WTcCluster.cpp.

21.13.3.2 cFace cCluster::Face (uint32 li1)

Definition at line 189 of file WTcCluster.cpp.

21.13.3.3 uint32 cCluster::FileSize ()

Definition at line 176 of file WTcCluster.cpp.

21.13.3.4 void cCluster::Init (uint32 liSize)

Definition at line 173 of file WTcCluster.cpp.

21.13.3.5 uint32 cCluster::Items ()

Definition at line 188 of file WTcCluster.cpp.

21.13.3.6 void cCluster::LoadIMFClusters (ifstream & FileStream)

Definition at line 214 of file WTcCluster.cpp.

21.13.3.7 void cCluster::Optimise ()

Definition at line 174 of file WTcCluster.cpp.

21.13.3.8 void cCluster::Order ()

Definition at line 91 of file WTcCluster.cpp.

21.13.3.9 void cCluster::OutputIMFClusters (ostream & *FileStream*)

Definition at line 204 of file WTcCluster.cpp.

21.13.3.10 void cCluster::Remove (uint32 *li1*)

Definition at line 140 of file WTcCluster.cpp.

21.13.3.11 void cCluster::Resize ()

Definition at line 190 of file WTcCluster.cpp.

21.13.3.12 void cCluster::Strip ()

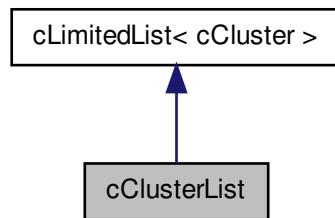
Definition at line 149 of file WTcCluster.cpp.

21.13.3.13 void cCluster::Switch (uint32 *li1*, uint32 *li2*)

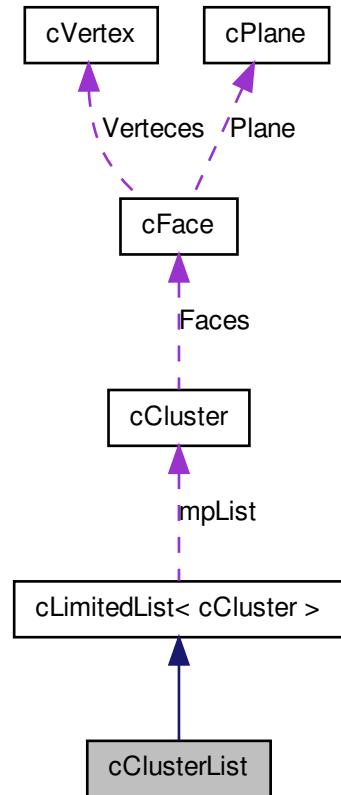
Definition at line 162 of file WTcCluster.cpp.

21.14 cClusterList Class Reference

Inheritance diagram for cClusterList:



Collaboration diagram for cClusterList:



Public Member Functions

- `cClusterList ()`
- `~cClusterList ()`
- `void Optimise ()`
- `void GenerateClusters (cFullFaceList &ClusterData)`
- `void GenerateConvex (cFullFaceList &ClusterData)`
- `void GenerateConcaves (cFullFaceList &ClusterData)`
- `void OutputIMFClusters (ofstream &FileStream)`
- `void LoadIMFClusters (ifstream &FileStream)`
- `uint32 FileSize ()`

21.14.1 Detailed Description

Definition at line 36 of file WTcCluster.h.

21.14.2 Constructor & Destructor Documentation

21.14.2.1 cClusterList::cClusterList() [inline]

Definition at line 40 of file WTcCluster.h.

21.14.2.2 cClusterList::~cClusterList() [inline]

Definition at line 41 of file WTcCluster.h.

21.14.3 Member Function Documentation

21.14.3.1 uint32 cClusterList::FileSize() [inline]

Definition at line 68 of file WTcCluster.h.

21.14.3.2 void cClusterList::GenerateClusters(cFullFaceList & ClusterData)

Definition at line 80 of file WTcCluster.cpp.

21.14.3.3 void cClusterList::GenerateConcaves(cFullFaceList & ClusterData)

Definition at line 48 of file WTcCluster.cpp.

21.14.3.4 void cClusterList::GenerateConvex(cFullFaceList & ClusterData)

Definition at line 24 of file WTcCluster.cpp.

21.14.3.5 void cClusterList::LoadIMFClusters(ifstream & FileStream) [inline]

Definition at line 57 of file WTcCluster.h.

21.14.3.6 void cClusterList::Optimise()

Definition at line 9 of file WTcCluster.cpp.

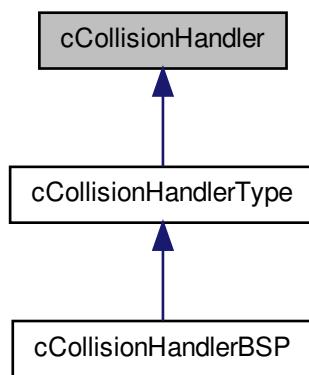
21.14.3.7 void cClusterList::OutputIMFClusters(ofstream & FileStream) [inline]

Definition at line 47 of file WTcCluster.h.

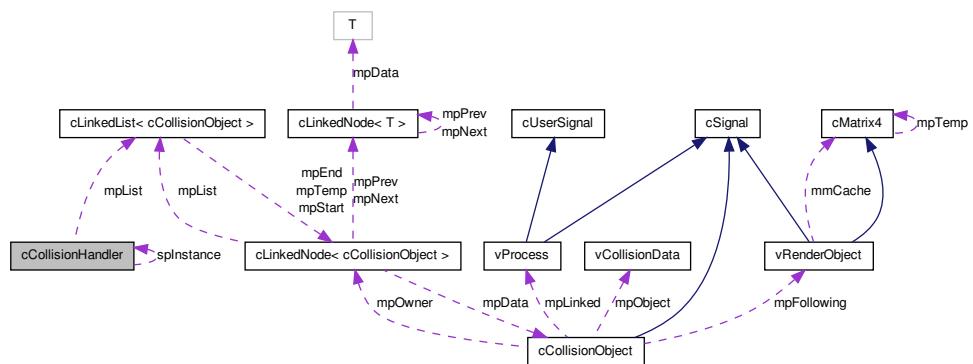
21.15 cCollisionHandler Class Reference

This is the Collision Handler. It will control any collision search the user performs. This Collision handler is created by using the [Instance\(\)](#) and can ONLY be created using [Instance\(\)](#). `_COLLISION_HANDLER` is a quick pointer to the `cCollisionHandler::Instance()` pointer. This will store data for the search and will store the current position to resume searches. Calling the Function [GenerateCollisionList\(\)](#) wil create a comprehensive list of pointers to `cRenderobjects()` that meet the collision parameters of [GenerateCollisionList\(\)](#). This list can be accessed by using [NextCollisionItem\(\)](#).

Inheritance diagram for `cCollisionHandler`:



Collaboration diagram for `cCollisionHandler`:



Public Member Functions

- virtual `cCollisionList * GenerateCollisionList (cCollisionObject *lpObj, uint32 lpType)=0`
`cCollisionObject * NextCollision (cCollisionObject *lpObj, uint32 lpType=0)`
This will step through the CollisionList to find the next collision.
- virtual `~cCollisionHandler ()`
This will deconstruct the class.
- virtual void `ResetCursors ()=0`
This will reset both the cursors used to track position through the collision object lists.

Static Public Member Functions

- static `cCollisionHandler * Instance ()`
This will return a pointer to the classes current instance and if there is none it will create one.

Protected Member Functions

- `cCollisionHandler ()`
- virtual `cLinkedNode< cCollisionObject > * Add (cCollisionObject *lpTemp)=0`
This will add the `cCollisionObject` pointed to by `lpObject` to the list `mpList`.
- virtual void `Remove (cLinkedNode< cCollisionObject > *lpOld)`
This will turn off Collisions for the `cLinkedNode` `lpOld`. This should in turn call `RemoveFromList()`.
- virtual void `RemoveFromList (cLinkedNode< cCollisionObject > *lpOld)`
This will acutally remove the clinkedNode from the relevant list.
- virtual bool `NextListItem ()`
This will return the Next item in the lists in order. (The item is pointed to by `mpColCur`). If an item is found will return true, else will return false.
- virtual bool `NextListItem (uint32 lpType)`
This will return the Next item in the list storing `lpType` in order. (the item is pointed to by `mpColCur`).If an item is found will return true, else will return false.
- virtual uint32 `FindSlot (cCollisionObject *lpObj)`
This will find the appropriate array slot for the `cCollisionObject` `lpObj`. It will return the array position of the slot.

- virtual uint32 [FindSlot](#) (float *lpPos)
- virtual [cLinkedList< cCollisionObject >](#) * [FindSlot](#) (uint32 *lpPos)

This will return the list for the spatial slot lpPos[0],lpPos[1],lpPos[2]. (Array slots not spatial co-ordinates).

- virtual void [Position](#) (float *lpTemp)

This will set the current Position of the Spatial Array.

- virtual float * [Position](#) ()

This will return the current Position of the Spatial Array.

Protected Attributes

- [cLinkedList< cCollisionObject >](#) * [mpList](#)

This is an array for storing all Collision Objects, that are currently on. The size of the array is set by WT_COLLISION_HANDLER_ARRAY_SIZE.

Static Protected Attributes

- static [cCollisionHandler](#) * [spInstance](#) = 0

This is a pointer to the classes current instance. There can only be one...

Friends

- class [cCollisionObject](#)

21.15.1 Detailed Description

This is the Collision Handler. It will control any collision search the user performs. This Collision handler is created by using the [Instance\(\)](#) and can ONLY be created using [Instance\(\)](#). [_COLLISION_HANDLER](#) is a quick pointer to the [cCollisionHandler::Instance\(\)](#) pointer. This will store data for the search and will store the current position to resume searches. Calling the Function [GenerateCollisionList\(\)](#) wil create a comprehensive list of pointers to [cRenderobjects\(\)](#) that meet the collision parameters of [GenerateCollisionList\(\)](#). This list can be accessed by using [NextCollisionItem\(\)](#). The sizes and positions of objects are calculated during a render cycle. [Objects](#) will not collide until after their first render cycle.

There are three types of Collision Searches. Tree, Type and Binary Spatial Position. (defined by setting [WT_COLLISION_HANDLER_TYPE](#) to [WT_COLLISION_HANDLER_TYPE_TYPE](#) or [WT_COLLISION_HANDLER_TYPE_BSP](#).

Tree searches are performed by traversing the render tree. Each objects size is based on the size of the objects beneath it so if a Node does not collide all objects beneath that node can be ignored.

Type searches are filtered by type. cCollisionObject's are given a [cCollisionObject::CollisionFilter\(uint32 liID\)](#) using which they are sorted into separate lists of Collision objects. If a search is performed with a defined ID term, only the list containing objects with the desired filter value are searched.

Binary Spatial Position Collision Handlers sort cCollisionObjects into the spatial boxes they contact with. This means it is only necessary to check other objects within the boxes the current object resides within.

Definition at line 30 of file WTcCollisionHandler.h.

21.15.2 Constructor & Destructor Documentation

21.15.2.1 cCollisionHandler::cCollisionHandler() [inline, protected]

Definition at line 35 of file WTcCollisionHandler.h.

21.15.2.2 cCollisionHandler::~cCollisionHandler() [virtual]

This will deconstruct the class.

Definition at line 62 of file WTcCollisionHandler.cpp.

21.15.3 Member Function Documentation

21.15.3.1 virtual cLinkedNode<cCollisionObject>* cCollisionHandler::Add (cCollisionObject * lpTemp) [protected, pure virtual]

This will add the [cCollisionObject](#) pointed to by lpObject to the list mpList.

Implemented in [cCollisionHandlerType](#).

21.15.3.2 virtual cLinkedList<cCollisionObject>* cCollisionHandler::FindSlot (uint32 * lpPos) [inline, protected, virtual]

This will return the list for the spatial slot lpPos[0],lpPos[1],lpPos[2]. (Array slots not spatial co-ordinates).

Reimplemented in [cCollisionHandlerType](#), and [cCollisionHandlerBSP](#).

Definition at line 59 of file WTcCollisionHandler.h.

21.15.3.3 virtual uint32 cCollisionHandler::FindSlot (cCollisionObject * *lpObj*)
[inline, protected, virtual]

This will find the appropriate array slot for the [cCollisionObject](#) lpObj. It will return the array position of the slot.

Reimplemented in [cCollisionHandlerType](#), and [cCollisionHandlerBSP](#).

Definition at line 55 of file WTcCollisionHandler.h.

21.15.3.4 virtual uint32 cCollisionHandler::FindSlot (float * *lpPos*) [inline,
protected, virtual]

Reimplemented in [cCollisionHandlerType](#), and [cCollisionHandlerBSP](#).

Definition at line 57 of file WTcCollisionHandler.h.

21.15.3.5 virtual cCollisionList* cCollisionHandler::GenerateCollisionList (
[cCollisionObject](#) * *lpObj*, uint32 *lpType*) [pure virtual]

Implemented in [cCollisionHandlerType](#), and [cCollisionHandlerBSP](#).

21.15.3.6 cCollisionHandler * cCollisionHandler::Instance () [static]

This will return a pointer to the classes current instance and if there is none it will create one.

Definition at line 5 of file WTcCollisionHandler.cpp.

21.15.3.7 cCollisionObject * cCollisionHandler::NextCollision (cCollisionObject * *lpObj*,
uint32 *lpType* = 0)

This will step through the CollisionList to find the next collision.

Definition at line 213 of file WTcCollisionHandler.cpp.

21.15.3.8 virtual bool cCollisionHandler::NextListItem () [inline, protected,
virtual]

This will return the Next item in the lists in order. (The item is pointed to by mpCol-Cur). If an item is found will return true, else will return false.

Reimplemented in [cCollisionHandlerType](#).

Definition at line 51 of file WTcCollisionHandler.h.

21.15.3.9 virtual bool cCollisionHandler::NextListItem (uint32 *lpType*) [inline, protected, virtual]

This will return the Next item in the list storing lpType in order. (the item is pointed to by mpColCur).If an item is found will return true, else will return false.

Reimplemented in [cCollisionHandlerType](#).

Definition at line 53 of file WTcCollisionHandler.h.

21.15.3.10 virtual float* cCollisionHandler::Position () [inline, protected, virtual]

This will return the current Position of the Spatial Array.

Reimplemented in [cCollisionHandlerType](#), and [cCollisionHandlerBSP](#).

Definition at line 64 of file WTcCollisionHandler.h.

21.15.3.11 virtual void cCollisionHandler::Position (float * *lpTemp*) [inline, protected, virtual]

This will set the current Position of the Spatial Array.

Reimplemented in [cCollisionHandlerType](#), and [cCollisionHandlerBSP](#).

Definition at line 62 of file WTcCollisionHandler.h.

21.15.3.12 virtual void cCollisionHandler::Remove (cLinkedNode< cCollisionObject > * *lpOld*) [inline, protected, virtual]

This will turn off Collisions for the [cLinkedNode](#) lpOld. This should in turn call [RemoveFromList\(\)](#).

Reimplemented in [cCollisionHandlerType](#).

Definition at line 46 of file WTcCollisionHandler.h.

21.15.3.13 virtual void cCollisionHandler::RemoveFromList (cLinkedNode< cCollisionObject > * *lpOld*) [inline, protected, virtual]

This will acutally remove the clinkedNode from the relevant list.

Reimplemented in [cCollisionHandlerType](#).

Definition at line 48 of file WTcCollisionHandler.h.

21.15.3.14 virtual void cCollisionHandler::ResetCursors () [pure virtual]

This will reset both the cursors used to track position through the collision object lists.

Implemented in [cCollisionHandlerType](#).

21.15.4 Friends And Related Function Documentation

21.15.4.1 **friend class cCollisionObject [friend]**

Definition at line 83 of file WTcCollisionHandler.h.

21.15.5 Member Data Documentation

21.15.5.1 **cLinkedList< cCollisionObject >* cCollisionHandler::mpList [protected]**

This is an array for storing all Collision Objects, that are currently on. The size of the array is set by WT_COLLISION_HANDLER_ARRAY_SIZE.

Definition at line 40 of file WTcCollisionHandler.h.

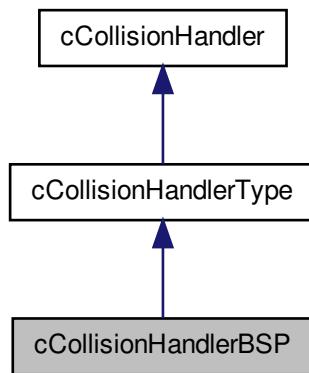
21.15.5.2 **cCollisionHandler * cCollisionHandler::spInstance = 0 [static, protected]**

This is a pointer to the classes current instance. There can only be one...

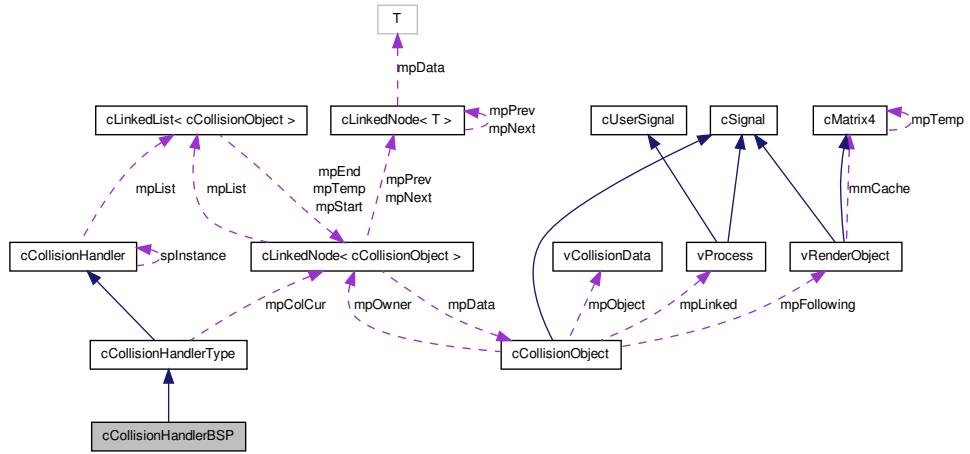
Definition at line 35 of file WTcCollisionHandler.h.

21.16 cCollisionHandlerBSP Class Reference

Inheritance diagram for cCollisionHandlerBSP:



Collaboration diagram for cCollisionHandlerBSP:



Public Member Functions

- `cCollisionList * GenerateCollisionList (cCollisionObject *lpObj, uint32 lpType=0)`

This will Actually search the Collision Lists and create a mpCollisionList with all the detected collisions with objects of type lpType.
- `void Position (float *lpTemp)`

This will set the current Position of the Spatial Array.
- `float * Position ()`

This will return the current Position of the Spatial Array.
- `uint32 FindSlot (cCollisionObject *lpObj)`

This will find the appropriate array slot for the `cCollisionObject` lpObj. It will return the array position of the slot.
- `uint32 FindSlot (float *lpPos)`

This will Find the Spatial Slot for the position lpPos.
- `cLinkedList<cCollisionObject> * FindSlot (uint32 *lpPos)`

This will return the list for the spatial slot lpPos[0],lpPos[1],lpPos[2]. (Array slots not spatial co-ordinates).
- `virtual ~cCollisionHandlerBSP ()`

Protected Member Functions

- [cCollisionHandlerBSP \(\)](#)

Protected Attributes

- float [mfCentre \[3\]](#)

This stores the Current central position of the Spatial Array.

Static Protected Attributes

- static uint32 [mpAxisOrder \[3\] = {0,2,1}](#)

This makes arrays dimensions order in X,Z,Y without hard coding it. (1D is X Axis, 2D is X,Z axis and 3D is X,Z,Y axis).

Friends

- class [cCollisionHandler](#)

21.16.1 Detailed Description

Definition at line 143 of file WTcCollisionHandler.h.

21.16.2 Constructor & Destructor Documentation

21.16.2.1 [cCollisionHandlerBSP::cCollisionHandlerBSP \(\) \[protected\]](#)

Definition at line 139 of file WTcCollisionHandler.cpp.

21.16.2.2 [cCollisionHandlerBSP::~cCollisionHandlerBSP \(\) \[virtual\]](#)

Definition at line 78 of file WTcCollisionHandler.cpp.

21.16.3 Member Function Documentation

21.16.3.1 [uint32 cCollisionHandlerBSP::FindSlot \(cCollisionObject * lpObj \) \[virtual\]](#)

This will find the appropriate array slot for the [cCollisionObject](#) lpObj. It will return the array position of the slot.

Reimplemented from [cCollisionHandlerType](#).

Definition at line 190 of file WTcCollisionHandler.cpp.

21.16.3.2 uint32 cCollisionHandlerBSP::FindSlot (float * *lpPos*) [virtual]

This will Find the Spatial Slot for the position lpPos.

Reimplemented from [cCollisionHandlerType](#).

Definition at line 161 of file WTcCollisionHandler.cpp.

21.16.3.3 cLinkedList< cCollisionObject > * cCollisionHandlerBSP::FindSlot (uint32 * *lpPos*) [virtual]

This will return the list for the spatial slot lpPos[0],lpPos[1],lpPos[2]. (Array slots not spatial co-ordinates).

Reimplemented from [cCollisionHandlerType](#).

Definition at line 197 of file WTcCollisionHandler.cpp.

21.16.3.4 cCollisionList * cCollisionHandlerBSP::GenerateCollisionList (cCollisionObject * *lpObj*, uint32 *lpType* = 0) [virtual]

This will Actually search the Collision Lists and create a mpCollisionList with all the detected collisions with objects of type lpType.

Reimplemented from [cCollisionHandlerType](#).

Definition at line 20 of file WTcCollisionHandler.cpp.

21.16.3.5 float* cCollisionHandlerBSP::Position () [inline, virtual]

This will return the current Position of the Spatial Array.

Reimplemented from [cCollisionHandlerType](#).

Definition at line 159 of file WTcCollisionHandler.h.

21.16.3.6 void cCollisionHandlerBSP::Position (float * *lpTemp*) [inline, virtual]

This will set the current Position of the Spatial Array.

Reimplemented from [cCollisionHandlerType](#).

Definition at line 157 of file WTcCollisionHandler.h.

21.16.4 Friends And Related Function Documentation**21.16.4.1 friend class cCollisionHandler [friend]**

Reimplemented from [cCollisionHandlerType](#).

Definition at line 169 of file WTcCollisionHandler.h.

21.16.5 Member Data Documentation

21.16.5.1 `float cCollisionHandlerBSP::mfCentre[3]` [protected]

This stores the Current central position of the Spatial Array.

Definition at line 148 of file WTcCollisionHandler.h.

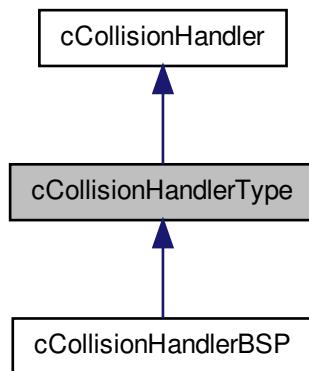
21.16.5.2 `uint32 cCollisionHandlerBSP::mpAxisOrder = {0,2,1}` [static, protected]

This makes arrays dimensions order in X,Z,Y without hard coding it. (1D is X Axis, 2D is X,Z axis and 3D is X,Z,Y axis).

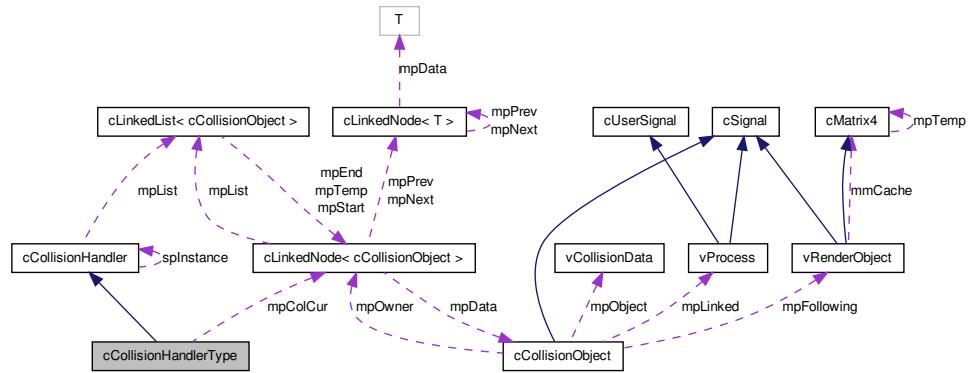
Definition at line 150 of file WTcCollisionHandler.h.

21.17 cCollisionHandlerType Class Reference

Inheritance diagram for cCollisionHandlerType:



Collaboration diagram for cCollisionHandlerType:



Public Member Functions

- virtual `cCollisionList * GenerateCollisionList (cCollisionObject *lpObj, uint32 lpType=0)`

This will Actually search the Collision Lists and create a mpCollisionList with all the detected collisions with objects of type lpType.

- void `ResetCursors ()`

This will reset both the cursors used to track position through the collision object lists.

- virtual `~cCollisionHandlerType ()`

This will deconstruct the class.

Protected Member Functions

- `cCollisionHandlerType ()`

Private Constructor.

- `cLinkedNode< cCollisionObject > * Add (cCollisionObject *lpTemp)`

This will add the `cCollisionObject` pointed to by `lpObject` to the list `mpList`.

- void `Remove (cLinkedNode< cCollisionObject > *lpOld)`

This will turn off Collisions for the `cLinkedNode` `lpOld`. This should in turn call `RemoveFromList()`.

- void `RemoveFromList (cLinkedNode< cCollisionObject > *lpOld)`

This will acutally remove the `clinkedNode` from the relevant list.

- bool [NextListItem \(\)](#)

This will return the Next item in the lists in order. (The item is pointed to by mpColCur). If an item is found will return true, else will return false.

- bool [NextListItem \(uint32 lpType\)](#)

This will return the Next item in the list storing lpType in order. (the item is pointed to by mpColCur). If an item is found will return true, else will return false.

- virtual uint32 [FindSlot \(cCollisionObject *lpObj\)](#)

This will find the appropriate array slot for the [cCollisionObject](#) lpObj. It will return the array position of the slot.

- virtual void [Position \(float *lpTemp\)](#)

This will set the current Position of the Spatial Array.

- virtual float * [Position \(\)](#)

This will return the current Position of the Spatial Array.

- virtual uint32 [FindSlot \(float *lpPos\)](#)

This will Find the Spatial Slot for the position lpPos.

- virtual [cLinkedList< cCollisionObject > * FindSlot \(uint32 *lpPos\)](#)

This will return the list for the spatial slot lpPos[0],lpPos[1],lpPos[2]. (Array slots not spatial co-ordinates).

Protected Attributes

- uint32 [miCurPos](#)

This is the current cursor position in the array mpList.

- [cLinkedList< cCollisionObject > * mpColCur](#)

This is the current cursor position (the current [cLinkedList](#)) in the List mpList[miTypeCur].

Friends

- class [cCollisionHandler](#)

21.17.1 Detailed Description

Definition at line 87 of file WTcCollisionHandler.h.

21.17.2 Constructor & Destructor Documentation

21.17.2.1 **cCollisionHandlerType::cCollisionHandlerType()** [protected]

Private Constructor.

Definition at line 130 of file WTcCollisionHandler.cpp.

21.17.2.2 **cCollisionHandlerType::~cCollisionHandlerType()** [virtual]

This will deconstruct the class.

Definition at line 68 of file WTcCollisionHandler.cpp.

21.17.3 Member Function Documentation

21.17.3.1 **cLinkedNode< cCollisionObject > * cCollisionHandlerType::Add(cCollisionObject * lpTemp)** [protected, virtual]

This will add the [cCollisionObject](#) pointed to by lpObject to the list mpList.

Implements [cCollisionHandler](#).

Definition at line 121 of file WTcCollisionHandler.cpp.

21.17.3.2 **virtual uint32 cCollisionHandlerType::FindSlot(float * lpPos)** [inline, protected, virtual]

This will Find the Spatial Slot for the position lpPos.

Reimplemented from [cCollisionHandler](#).

Reimplemented in [cCollisionHandlerBSP](#).

Definition at line 124 of file WTcCollisionHandler.h.

21.17.3.3 **virtual cLinkedList<cCollisionObject>* cCollisionHandlerType::FindSlot(uint32 * lpPos)** [inline, protected, virtual]

This will return the list for the spatial slot lpPos[0],lpPos[1],lpPos[2]. (Array slots not spatial co-ordinates).

Reimplemented from [cCollisionHandler](#).

Reimplemented in [cCollisionHandlerBSP](#).

Definition at line 126 of file WTcCollisionHandler.h.

21.17.3.4 uint32 cCollisionHandlerType::FindSlot (cCollisionObject * lpObj)
[protected, virtual]

This will find the appropriate array slot for the [cCollisionObject](#) lpObj. It will return the array position of the slot.

Reimplemented from [cCollisionHandler](#).

Reimplemented in [cCollisionHandlerBSP](#).

Definition at line 15 of file WTcCollisionHandler.cpp.

21.17.3.5 cCollisionList * cCollisionHandlerType::GenerateCollisionList (cCollisionObject * lpObj, uint32 lpType = 0) [virtual]

This will Actually search the Collision Lists and create a mpCollisionList with all the detected collisions with objects of type lpType.

Implements [cCollisionHandler](#).

Reimplemented in [cCollisionHandlerBSP](#).

Definition at line 34 of file WTcCollisionHandler.cpp.

21.17.3.6 bool cCollisionHandlerType::NextListItem () [protected, virtual]

This will return the Next item in the lists in order. (The item is pointed to by mpColCur). If an item is found will return true, else will return false.

Reimplemented from [cCollisionHandler](#).

Definition at line 90 of file WTcCollisionHandler.cpp.

21.17.3.7 bool cCollisionHandlerType::NextListItem (uint32 lpType) [protected, virtual]

This will return the Next item in the list storing lpType in order. (the item is pointed to by mpColCur).If an item is found will return true, else will return false.

Reimplemented from [cCollisionHandler](#).

Definition at line 104 of file WTcCollisionHandler.cpp.

21.17.3.8 virtual float* cCollisionHandlerType::Position () [inline, protected, virtual]

This will return the current Position of the Spatial Array.

Reimplemented from [cCollisionHandler](#).

Reimplemented in [cCollisionHandlerBSP](#).

Definition at line 121 of file WTcCollisionHandler.h.

21.17.3.9 virtual void cCollisionHandlerType::Position (float * *lpTemp*) [inline, protected, virtual]

This will set the current Position of the Spatial Array.

Reimplemented from [cCollisionHandler](#).

Reimplemented in [cCollisionHandlerBSP](#).

Definition at line 119 of file WTcCollisionHandler.h.

21.17.3.10 void cCollisionHandlerType::Remove (cLinkedNode< cCollisionObject > * *lpOld*) [protected, virtual]

This will turn off Collisions for the [cLinkedNode](#) *lpOld*. This should in turn call [RemoveFromList\(\)](#).

Reimplemented from [cCollisionHandler](#).

Definition at line 151 of file WTcCollisionHandler.cpp.

21.17.3.11 void cCollisionHandlerType::RemoveFromList (cLinkedNode< cCollisionObject > * *lpOld*) [protected, virtual]

This will acutally remove the clinkedNode from the relevant list.

Reimplemented from [cCollisionHandler](#).

Definition at line 114 of file WTcCollisionHandler.cpp.

21.17.3.12 void cCollisionHandlerType::ResetCursors () [virtual]

This will reset both the cursors used to track position through the collision object lists.

Implements [cCollisionHandler](#).

Definition at line 145 of file WTcCollisionHandler.cpp.

21.17.4 Friends And Related Function Documentation

21.17.4.1 friend class cCollisionHandler [friend]

Reimplemented in [cCollisionHandlerBSP](#).

Definition at line 140 of file WTcCollisionHandler.h.

21.17.5 Member Data Documentation

21.17.5.1 uint32 cCollisionHandlerType::miCurPos [protected]

This is the current cursor position in the array mpList.

Definition at line 94 of file WTcCollisionHandler.h.

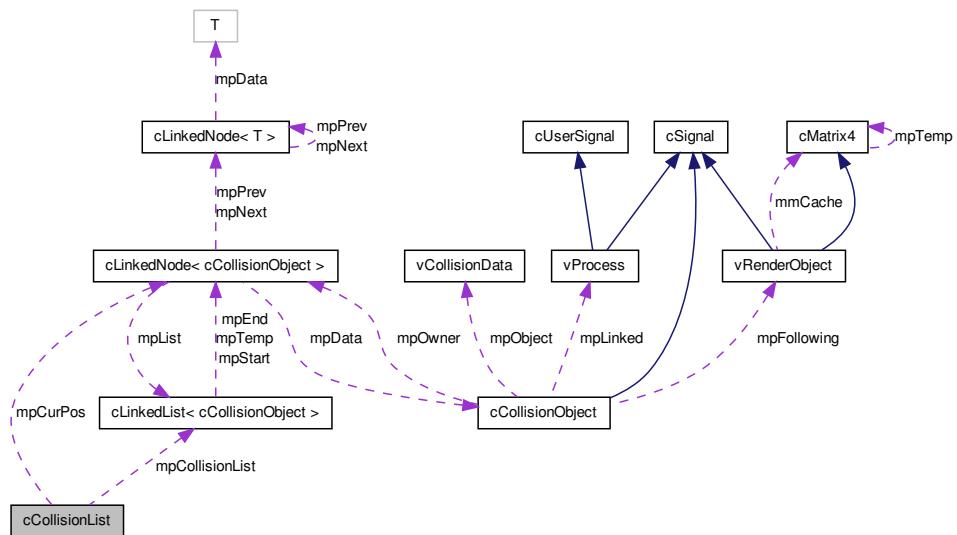
21.17.5.2 `cLinkedNode< cCollisionObject *>* cCollisionHandlerType::mpColCur` [protected]

This is the current cursor position (the current `cLinkedNode`) in the List `mpList[miTypeCur]`.

Definition at line 96 of file WTcCollisionHandler.h.

21.18 cCollisionList Class Reference

Collaboration diagram for cCollisionList:



Public Member Functions

- `cCollisionList ()`

The Constructor for `cCollisionList`.

- `void AddCollision (cCollisionObject *lpObject)`

This will Add the object `lpObject` to the list of objects colliding with the current searching object.

- `cCollisionObject * NextCollisionItem ()`

This will return the next item from the list mpCollisionList that has been stocked by GenerateCollisionList() as a CollisionObject pointer.

- **vProcess * NextCollisionP ()**

This will return the process owning renderable object creating the next detected collision.

- **vRenderObject * NextCollisionR ()**

This will return the renderable object involved in the next detected collision.

- **void ResetList ()**

This will reset the collision search to the start of mpList.

- **~cCollisionList ()**

This is the destructor for cCollisionList.

- **void SortByDistance ()**

*This will sort the list of collisionobjects into distance from the colliding object order.
This allows the user to resolve the collisions in 'Chronological' order.*

21.18.1 Detailed Description

Definition at line 13 of file WTcCollisionList.h.

21.18.2 Constructor & Destructor Documentation

21.18.2.1 cCollisionList::cCollisionList()

The Constructor for **cCollisionList**.

Definition at line 61 of file WTcCollisionList.cpp.

21.18.2.2 cCollisionList::~cCollisionList()

This is the destructor for **cCollisionList**.

Definition at line 55 of file WTcCollisionList.cpp.

21.18.3 Member Function Documentation

21.18.3.1 void cCollisionList::AddCollision (cCollisionObject * lpObject)

This will Add the object lpObject to the list of objects colliding with the current searching object.

Definition at line 5 of file WTcCollisionList.cpp.

21.18.3.2 cCollisionObject * cCollisionList::NextCollisionItem()

This will return the next item from the list mpCollisionList that has been stocked by GenerateCollisionList() as a CollisionObject pointer.

Definition at line 10 of file WTcCollisionList.cpp.

21.18.3.3 vProcess * cCollisionList::NextCollisionP()

This will return the process owning renderable object creating the next detected collision.

Definition at line 22 of file WTcCollisionList.cpp.

21.18.3.4 vRenderObject * cCollisionList::NextCollisionR()

This will return the renderable object involved in the next detected collision.

Definition at line 34 of file WTcCollisionList.cpp.

21.18.3.5 void cCollisionList::ResetList()

This will reset the collision search to the start of mpList.

Definition at line 49 of file WTcCollisionList.cpp.

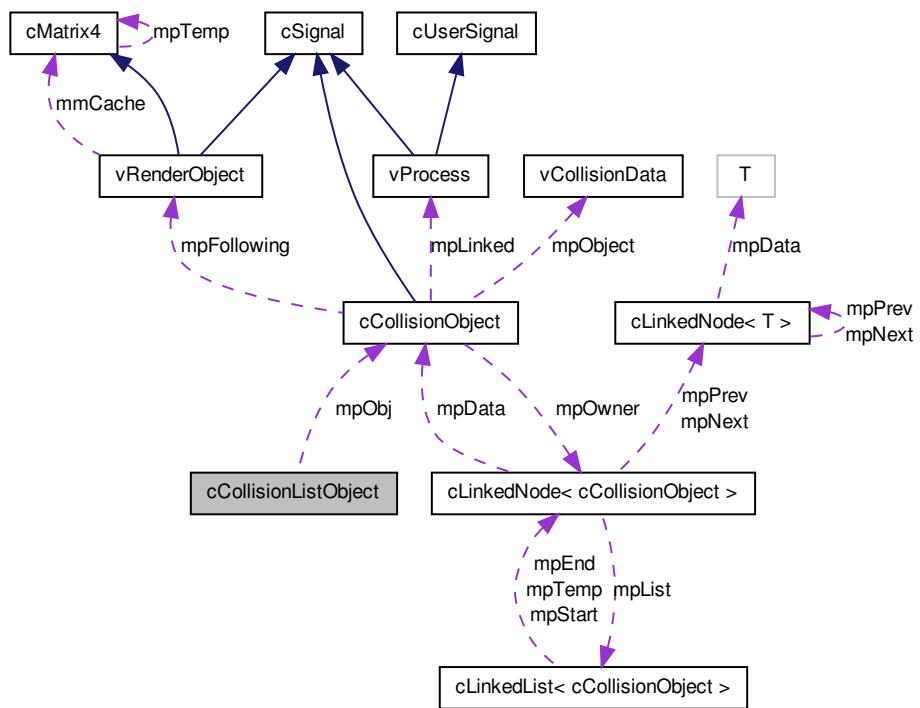
21.18.3.6 void cCollisionList::SortByDistance()

This will sort the list of collisionobjects into distance from the colliding object order. This allows the user to resolve the collisions in 'Chronological' order.

Definition at line 68 of file WTcCollisionList.cpp.

21.19 cCollisionListObject Class Reference

Collaboration diagram for cCollisionListObject:



Public Member Functions

- `cCollisionListObject (cCollisionObject *lpObj)`
- float `GetDistance ()`

21.19.1 Detailed Description

Definition at line 4 of file `WTcCollisionList.h`.

21.19.2 Constructor & Destructor Documentation

21.19.2.1 **cCollisionListObject::cCollisionListObject (*cCollisionObject * lpObj*)**
[inline]

Definition at line 9 of file WTcCollisionList.h.

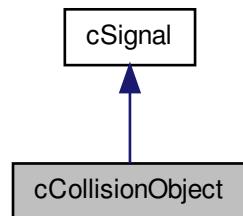
21.19.3 Member Function Documentation

21.19.3.1 **float cCollisionListObject::GetDistance ()** [inline]

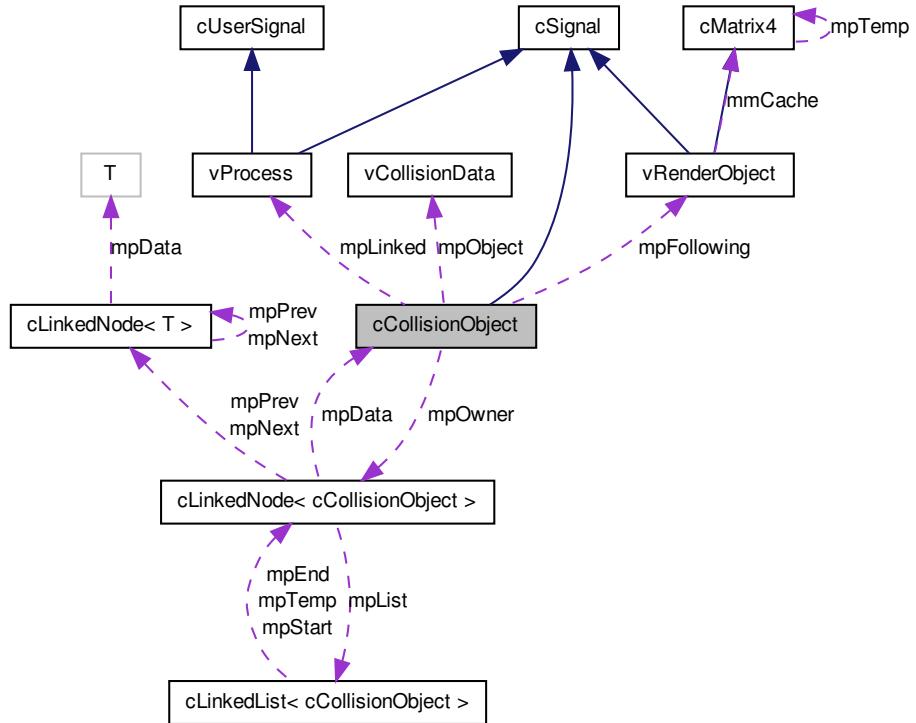
Definition at line 10 of file WTcCollisionList.h.

21.20 cCollisionObject Class Reference

Inheritance diagram for cCollisionObject:



Collaboration diagram for cCollisionObject:



Public Member Functions

- `cCollisionObject (vRenderObject *lpFollow, vProcess *lpLinked=0, uint32 liFilterType=0)`
- `cCollisionObject (cBeamMesh *lpFollow, vProcess *lpLinked=0, uint32 liFilterType=0)`
- `void Initialise (vRenderObject *lpFollow, vProcess *lpLinked, uint32 liFilterType)`
- `~cCollisionObject ()`
- `bool CreatedThisFrame ()`
- `void NotCreatedThisFrame ()`
- `vCollisionData * CollisionData ()`
- `void Signal (SIGNAL liFlags)`

This is the function that will handle a system signal. Possible signals to be passed in are S_SLEEP, S_WAKE, S_KILL, S_SLEEP_TREE, S_WAKE_TREE, S_KILL_TREE.

- `bool Procedural ()`
- `bool Loaded ()`
- `void ClearProcedural ()`
- `cSphereCollision * Sphere ()`
- `cMeshCollision * Mesh ()`
- `cRayCollision * Ray ()`
- `cBeamCollision * Beam ()`
- `void SetType (cSphereCollision *lpSphere)`

This will take a SphereCollision Object loaded from a file, and set this Collision Object to use it.

- `void SetType (cMeshCollision *lpMesh)`

This will take a MeshCollision Object loaded from a file, and set this Collision Object to use it.

- `void SetType (cBeamCollision *lpBeam)`

This will take a BeamCollision Object loaded from a file, and set this Collision Object to use it.

- `void OnFile (vCollisionData *lpPoint)`

- `void OnProcedural ()`

- `cSphereCollision * SetType (float lfSize)`

This will procedurally generate a Sphere or radius 1.0f;.

- `cBeamCollision * SetType (float lfLength, float lfRadius)`

This will procedurally generate a Beam of Radius lfRadius and Length lfLength.

- `cMeshCollision * SetType (float *lpBounds)`

*This will procedurally generate a Box collision object from the array of 6 floats lpBounds. see cGeneratedBoxCollision::BuildObject(float *lpBounds) for more information.*

- `cRayCollision * SetType (cRenderObject *lpObj)`

*This will make a ray object. See cRayCollision::BuildObject(float *lpBounds) for more information. Makes Ray radius to be same as object and ray to follow it's movement.*

- `cBeamCollision * SetType (cBeamMesh *lpBeam)`

This will make a Beam object to match a rendered Beam. (Nice and easy eh?)

- `cSphereCollision * GetCollisionData ()`

- `uint8 Type ()`

This is the accurate way of determining the type of these objects.

- `bool CheckCollision (cCollisionObject *lpOther)`

This will check for a collision between this object and the object lpOther.

- bool `TouchCollision (cCollisionObject *lpOther)`

- `cMatrix4 & CacheMatrix ()`

This will return the Objects Cached Matrix mmCache.

- `vRenderObject * RenderObject ()`

This will return a pointer the `cRenderObject` linked to this.

- uint32 `CollisionFilter ()`

This will return the current Collision Filter ID of this object.

- void `CollisionFilter (uint32 liID)`

This will set the Collision Filter ID of this object.

- bool `Collision (cCollisionObject *lpOther, uint32 lpCol)`

- bool `Collision (cCollisionObject *lpOther)`

- bool `CompareRanges (float lf1, float lf2, float lfR)`

*This will do a sphere collision between two points. lf1 and lf2 are $x*x+y*y+z*z$. lfR is $(Sum\ of\ Radii)*(Sum\ of\ Radii)$.*

- void `SetLink (vProcess *lpLink)`

This sets the process that this renderable object will return collision signals to.

- `vProcess * GetLink ()`

Returns the process currently linked to this renderable object.

- void `GenerateCollisionList (uint32 liType=0)`

This will generate a collision list of all objects colliding with this object filtered by type liType.

- float * `GetPos ()`

This will return the position of the selected object.

- void `PreUpdateCache ()`

This will remove the `cCollisionObject` from it's Spatial Slots before its position is updated.

- void `PostUpdateCache ()`

This will Add the `cCollisionObject` to the relevant Spatial Slots after it has found it's new position.

- bool `SphereSphere (cCollisionObject *lpOther)`

- bool `ModelModel (cCollisionObject *lpOther)`

- bool `RayRay (cCollisionObject *lpOther)`

- bool `SphereModel (cCollisionObject *lpOther)`

- bool `SphereRay (cCollisionObject *lpOther)`

- bool `RayModel (cCollisionObject *lpOther)`

- float `GetCollisionSize ()`

Static Public Attributes

- static float **mfStaticSize** [3] = {0.0f,0.0f,0.0f}

This is a temporary storage for transferring sizes between objects.

21.20.1 Detailed Description

Definition at line 14 of file WTcCollisionObject.h.

21.20.2 Constructor & Destructor Documentation

21.20.2.1 cCollisionObject::cCollisionObject (vRenderObject * *lpFollow*, vProcess * *lpLinked* = 0, uint32 *liFilterType* = 0)

Definition at line 839 of file WTcCollisionObject.cpp.

21.20.2.2 cCollisionObject::cCollisionObject (cBeamMesh * *lpFollow*, vProcess * *lpLinked* = 0, uint32 *liFilterType* = 0)

Definition at line 856 of file WTcCollisionObject.cpp.

21.20.2.3 cCollisionObject::~cCollisionObject ()

Definition at line 754 of file WTcCollisionObject.cpp.

21.20.3 Member Function Documentation

21.20.3.1 cBeamCollision* cCollisionObject::Beam () [inline]

This is a dynamic cast on the type of collision object this is. Note ALL cCollisionData objects return true to Sphere. This applies to all the functions [Sphere\(\)](#),[Box\(\)](#),[Mesh\(\)](#),[Ray\(\)](#) and [Beam\(\)](#).

Definition at line 74 of file WTcCollisionObject.h.

21.20.3.2 cMatrix4& cCollisionObject::CacheMatrix () [inline]

This will return the Objects Cached Matrix mmCache.

Definition at line 142 of file WTcCollisionObject.h.

21.20.3.3 bool cCollisionObject::CheckCollision (cCollisionObject * *lpOther*)

This will check for a collision between this object and the object lpOther.

*

Parameters

<i>lpOther</i>	is a pointer to the other collision object to check against. It will check both objects have collisions on. Then it will do an initial quick check to see if a collision is a possibility. Finally if required it will do a much more detailed collision check.
----------------	---

Definition at line 673 of file WTcCollisionObject.cpp.

21.20.3.4 void cCollisionObject::ClearProcedural ()

Definition at line 871 of file WTcCollisionObject.cpp.

21.20.3.5 bool cCollisionObject::Collision (cCollisionObject * *lpOther*)

Definition at line 665 of file WTcCollisionObject.cpp.

21.20.3.6 bool cCollisionObject::Collision (cCollisionObject * *lpOther*, uint32 *lpCol*)

This Function will control the collision detection between this object and *lpOther*. It will check the filter *lpCol* to cehck that the other object is of a suitable type. Then will call [CheckCollision\(cCollisionObject *lpOther\)](#)

Definition at line 654 of file WTcCollisionObject.cpp.

21.20.3.7 vCollisionData* cCollisionObject::CollisionData () [inline]

Definition at line 53 of file WTcCollisionObject.h.

21.20.3.8 uint32 cCollisionObject::CollisionFilter () [inline]

This will return the current Collision Filter ID of this object.

Definition at line 152 of file WTcCollisionObject.h.

21.20.3.9 void cCollisionObject::CollisionFilter (uint32 *IID*)

This will set the Collision Filter ID of this object.

Definition at line 616 of file WTcCollisionObject.cpp.

21.20.3.10 bool cCollisionObject::CompareRanges (float *lf1*, float *lf2*, float *lfR*)

This will do a sphere collision between two points. *lf1* and *lf2* are $x*x+y*y+z*z$. *lfR* is $(\text{Sum of Radii})*(\text{Sum of Radii})$.

21.20.3.11 bool cCollisionObject::CreatedThisFrame() [inline]

Definition at line 49 of file WTcCollisionObject.h.

21.20.3.12 void cCollisionObject::GenerateCollisionList(uint32 liType = 0)

This will generate a collision list of all objects colliding with this object filtered by type liType.

Definition at line 605 of file WTcCollisionObject.cpp.

21.20.3.13 cSphereCollision* cCollisionObject::GetCollisionData() [inline]

Definition at line 97 of file WTcCollisionObject.h.

21.20.3.14 float cCollisionObject::GetCollisionSize()

Definition at line 610 of file WTcCollisionObject.cpp.

21.20.3.15 vProcess* cCollisionObject::GetLink() [inline]

Returns the process currently linked to this renderable object.

Definition at line 174 of file WTcCollisionObject.h.

21.20.3.16 float * cCollisionObject::GetPos()

This will return the position of the selected object.

Definition at line 600 of file WTcCollisionObject.cpp.

21.20.3.17 void cCollisionObject::Initialise(vRenderObject * lpFollow, vProcess * lpLinked, uint32 liFilterType)

Definition at line 883 of file WTcCollisionObject.cpp.

21.20.3.18 bool cCollisionObject::Loaded() [inline]

Definition at line 58 of file WTcCollisionObject.h.

21.20.3.19 cMeshCollision* cCollisionObject::Mesh() [inline]

This is a dynamic cast on the type of collision object this is. Note ALL cCollisionData objects return true to Sphere. This applies to all the functions [Sphere\(\)](#), [Box\(\)](#), [Mesh\(\)](#), [Ray\(\)](#) and [Beam\(\)](#).

Definition at line 67 of file WTcCollisionObject.h.

21.20.3.20 bool cCollisionObject::ModelModel (cCollisionObject * *lpOther*)

Definition at line 11 of file WTcCollisionObject.cpp.

21.20.3.21 void cCollisionObject::NotCreatedThisFrame () [inline]

Definition at line 51 of file WTcCollisionObject.h.

21.20.3.22 void cCollisionObject::OnFile (vCollisionData * *lpPoint*)

Definition at line 777 of file WTcCollisionObject.cpp.

21.20.3.23 void cCollisionObject::OnProcedural ()

Definition at line 778 of file WTcCollisionObject.cpp.

21.20.3.24 void cCollisionObject::PostUpdateCache ()

This will Add the [cCollisionObject](#) to the relevant Spatial Slots after it has found it's new position.

Definition at line 637 of file WTcCollisionObject.cpp.

21.20.3.25 void cCollisionObject::PreUpdateCache ()

This will remove the [cCollisionObject](#) from it's Spatial Slots before its position is updated.

Definition at line 630 of file WTcCollisionObject.cpp.

21.20.3.26 bool cCollisionObject::Procedural () [inline]

Definition at line 57 of file WTcCollisionObject.h.

21.20.3.27 cRayCollision* cCollisionObject::Ray () [inline]

This is a dynamic cast on the type of collision object this is. Note ALL cCollisionData objects return true to Sphere. This applies to all the functions [Sphere\(\)](#), [Box\(\)](#), [Mesh\(\)](#), [Ray\(\)](#) and [Beam\(\)](#). [cRayCollision](#) Objects will also return true to [Beam\(\)](#).

Definition at line 71 of file WTcCollisionObject.h.

21.20.3.28 bool cCollisionObject::RayModel (cCollisionObject * *lpOther*)

Definition at line 504 of file WTcCollisionObject.cpp.

21.20.3.29 bool cCollisionObject::RayRay (cCollisionObject * *lpOther*)

Definition at line 68 of file WTcCollisionObject.cpp.

21.20.3.30 vRenderObject* cCollisionObject::RenderObject () [inline]

This will return a pointer the [cRenderObject](#) linked to this.

Definition at line 146 of file WTcCollisionObject.h.

21.20.3.31 void cCollisionObject::SetLink (vProcess * *lpLink*) [inline]

This sets the process that this renderable object will return collision signals to.

Parameters

<i>lpLink</i>	Points to the process that this renderable object will return collision signals to.
---------------	---

Definition at line 172 of file WTcCollisionObject.h.

21.20.3.32 void cCollisionObject::SetType (cMeshCollision * *lpMesh*)

This will take a MeshCollision Object loaded from a file, and set this Collision Object to use it.

Definition at line 773 of file WTcCollisionObject.cpp.

21.20.3.33 cRayCollision * cCollisionObject::SetType (cRenderObject * *lpObj*)

This will make a ray object. See [cRayCollision::BuildObject\(float *lpBounds\)](#) for more information. Makes Ray radius to be same as object and ray to follow it's movement.

Definition at line 735 of file WTcCollisionObject.cpp.

21.20.3.34 cSphereCollision * cCollisionObject::SetType (float *lfSize*)

This will procedurally generate a Sphere or radius 1.0f;

Definition at line 780 of file WTcCollisionObject.cpp.

21.20.3.35 cBeamCollision * cCollisionObject::SetType (cBeamMesh * *lpBeam*)

This will make a Beam object to match a rendered Beam. (Nice and easy eh?)

Definition at line 724 of file WTcCollisionObject.cpp.

21.20.3.36 cMeshCollision * cCollisionObject::SetType (float * *lpBounds*)

This will procedurally generate a Box collision object from the array of 6 floats lpBounds. see [cGeneratedBoxCollision::BuildObject\(float *lpBounds\)](#) for more information.

This will procedurally generate a Box collision object from the array of 6 floats lpBounds. see [cMeshCollision::BuildObject\(float *lpBounds\)](#) for more information.

Definition at line 794 of file WTcCollisionObject.cpp.

21.20.3.37 cBeamCollision * cCollisionObject::SetType (float *lfLength*, float *lfRadius*)

This will procedurally generate a Beam of Radius lfRadius and Length lfLength.

Definition at line 792 of file WTcCollisionObject.cpp.

21.20.3.38 void cCollisionObject::SetType (cSphereCollision * *lpSphere*)

This will take a SphereCollision Object loaded from a file, and set this Collision Object to use it.

Definition at line 771 of file WTcCollisionObject.cpp.

21.20.3.39 void cCollisionObject::SetType (cBeamCollision * *lpBeam*)

This will take a BeamCollision Object loaded from a file, and set this Collision Object to use it.

Definition at line 775 of file WTcCollisionObject.cpp.

21.20.3.40 void cCollisionObject::Signal (SIGNAL *IsSignal*) [virtual]

This is the function that will handle a system signal. Possible signals to be passed in are _S_SLEEP,_S_WAKE,_S_KILL,_S_SLEEP_TREE,_S_WAKE_TREE,_S_KILL_TREE.

Reimplemented from [cSignal](#).

Definition at line 808 of file WTcCollisionObject.cpp.

21.20.3.41 cSphereCollision* cCollisionObject::Sphere () [inline]

This is a dynamic cast on the type of collision object this is. Note ALL cCollisionData objects return true to Sphere. This applies to all the functions [Sphere\(\)](#),[Box\(\)](#),[Mesh\(\)](#),[Ray\(\)](#)

and [Beam\(\)](#).

Definition at line 64 of file WTcCollisionObject.h.

21.20.3.42 `bool cCollisionObject::SphereModel (cCollisionObject * lpOther)`

Definition at line 400 of file WTcCollisionObject.cpp.

21.20.3.43 `bool cCollisionObject::SphereRay (cCollisionObject * lpOther)`

Definition at line 446 of file WTcCollisionObject.cpp.

21.20.3.44 `bool cCollisionObject::SphereSphere (cCollisionObject * lpOther)`

Definition at line 747 of file WTcCollisionObject.cpp.

21.20.3.45 `bool cCollisionObject::TouchCollision (cCollisionObject * lpOther)`

It is used for quickly detecting if two objects MAY be colliding. This function is will assume all objects are either Spheres or Beams. This is a highly inaccurate way of colliding objects, but should filter the majority of collisions from the much slower more accurate collision detection performed elsewhere.

Definition at line 704 of file WTcCollisionObject.cpp.

21.20.3.46 `uint8 cCollisionObject::Type () [inline]`

This is the accurate way of determining the type of these objects.

Definition at line 107 of file WTcCollisionObject.h.

21.20.4 Member Data Documentation

21.20.4.1 `float cCollisionObject::mfStaticSize = {0.0f,0.0f,0.0f} [static]`

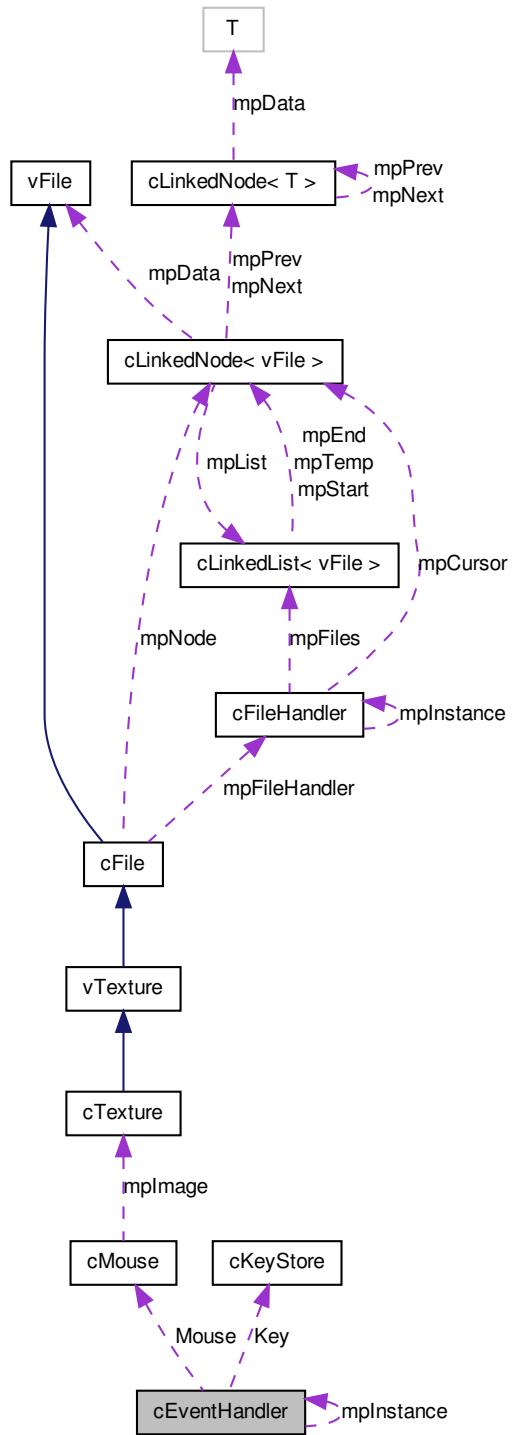
This is a temporary storage for transferring sizes between objects.

Definition at line 139 of file WTcCollisionObject.h.

21.21 cEventHandler Class Reference

This will handle events from the OS. Actually this will just store the Input data for the keyboard and mouse. It is easiest to access the input data using _KEY and _MOUSE. There can only be one [cEventHandler](#), created using [Instance\(\)](#).

Collaboration diagram for cEventHandler:



Static Public Member Functions

- static [cEventHandler * Instance \(\)](#)

This will return a pointer to the current cEventhandler instance. If there is no instance it will create one.

Public Attributes

- [cKeyStore Key](#)

This will store all the keyboard input data. see _KEY.

- [cMouse Mouse](#)

This will store the mouse input data. see _MOUSE.

21.21.1 Detailed Description

This will handle events from the OS. Actually this will just store the Input data for the keyboard and mouse. It is easiest to access the input data using _KEY and _MOUSE. There can only be one [cEventHandler](#), created using [Instance\(\)](#).

Definition at line 10 of file WTcEventHandler.h.

21.21.2 Member Function Documentation

21.21.2.1 [cEventHandler * cEventHandler::Instance \(\) \[static\]](#)

This will return a pointer to the current cEventhandler instance. If there is no instance it will create one.

Definition at line 5 of file WTcEventHandler.cpp.

21.21.3 Member Data Documentation

21.21.3.1 [cKeyStore cEventHandler::Key](#)

This will store all the keyboard input data. see _KEY.

Definition at line 20 of file WTcEventHandler.h.

21.21.3.2 [cMouse cEventHandler::Mouse](#)

This will store the mouse input data. see _MOUSE.

Definition at line 22 of file WTcEventHandler.h.

21.22 CException Class Reference

Public Member Functions

- [CException \(const char *m\)](#)
- [CException \(string &m\)](#)

Public Attributes

- string [message](#)

21.22.1 Detailed Description

Definition at line 8 of file CException.h.

21.22.2 Constructor & Destructor Documentation

21.22.2.1 [CException::CException \(const char * m \) \[inline\]](#)

Definition at line 11 of file CException.h.

21.22.2.2 [CException::CException \(string & m \) \[inline\]](#)

Definition at line 12 of file CException.h.

21.22.3 Member Data Documentation

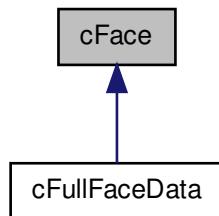
21.22.3.1 [string CException::message](#)

Definition at line 10 of file CException.h.

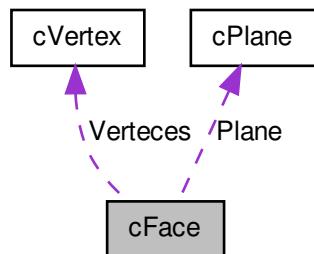
21.23 cFace Class Reference

This class will store data about faces for a 3D Mesh. This can be used for Models, Collision Meshes or any other object using 3D Faces. Includes code for loading and saving the object types to and from IMF Files. Uses [cVertex](#) and [cPlane](#) to store the data.

Inheritance diagram for cFace:



Collaboration diagram for cFace:



Public Member Functions

- [cFace & operator= \(cFace &lpOther\)](#)
- [uint32 FileSize \(\)](#)
- [void OutputIMFFace \(ofstream &FileStream\)](#)
- [void LoadIMFFace \(ifstream &FileStream\)](#)

Public Attributes

- [cVertex Verteces \[3\]](#)
- [cPlane Plane](#)

21.23.1 Detailed Description

This class will store data about faces for a 3D Mesh. This can be used for Models, Collision Meshes or any other object using 3D Faces. Includes code for loading and saving the object types to and from IMF Files. Uses [cVertex](#) and [cPlane](#) to store the data.

Definition at line 7 of file WTcFace.h.

21.23.2 Member Function Documentation

21.23.2.1 `uint32 cFace::FileSize()`

Reimplemented in [cFullFaceData](#).

Definition at line 12 of file WTcFace.cpp.

21.23.2.2 `void cFace::LoadIMFFace(ifstream & FileStream)`

Definition at line 33 of file WTcFace.cpp.

21.23.2.3 `cFace & cFace::operator=(cFace & lpOther)`

Definition at line 3 of file WTcFace.cpp.

21.23.2.4 `void cFace::OutputIMFFace(ofstream & FileStream)`

Definition at line 24 of file WTcFace.cpp.

21.23.3 Member Data Documentation

21.23.3.1 `cPlane cFace::Plane`

Definition at line 12 of file WTcFace.h.

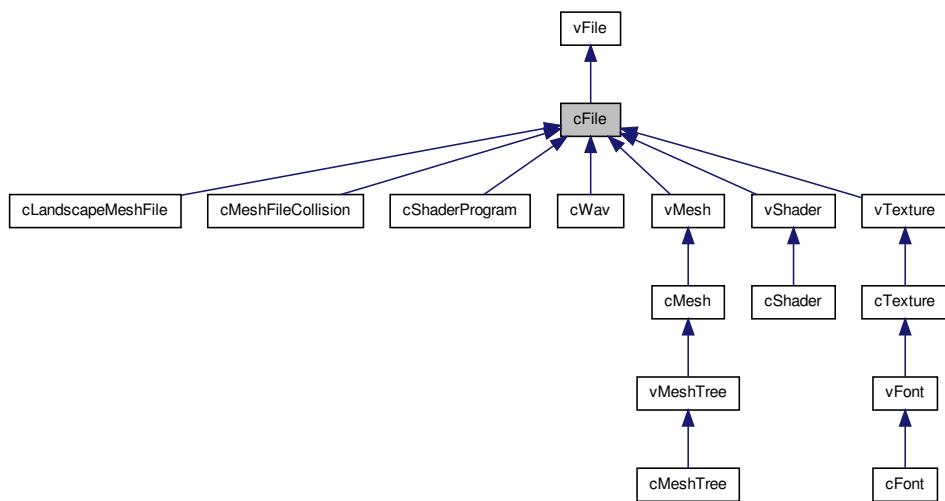
21.23.3.2 `cVertex cFace::Verteces[3]`

Definition at line 11 of file WTcFace.h.

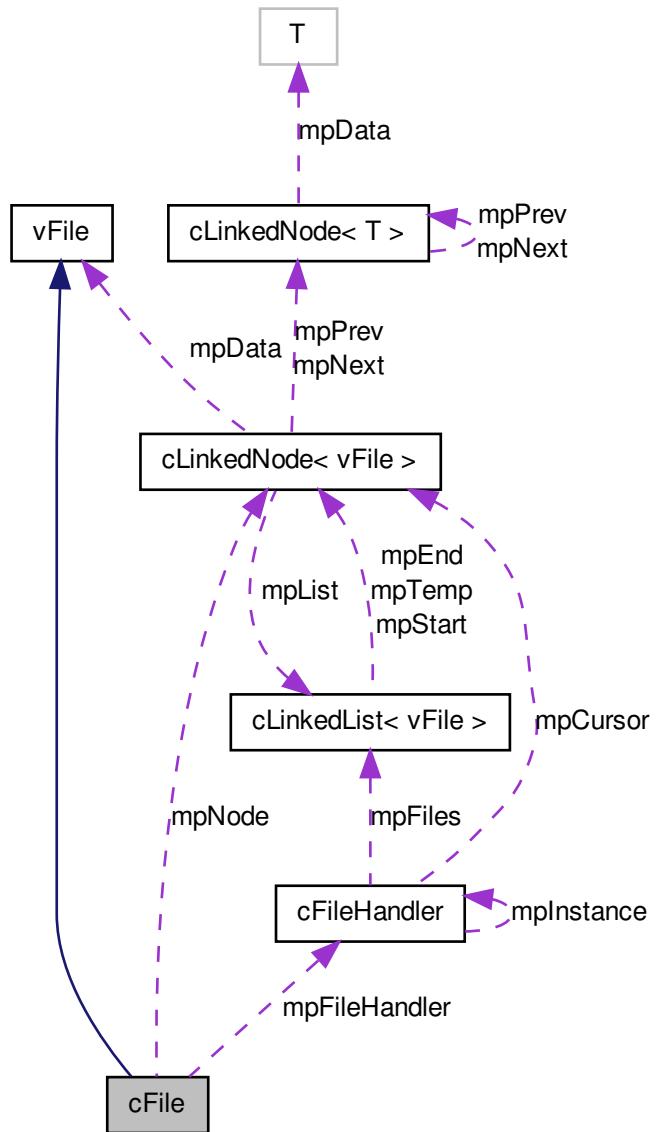
21.24 cFile Class Reference

This is the base code for files to be loaded from a hdd. Any file object loaded from a hddvshould inherit this class. It is best used for media files. This code will automatically add newly loaded files to [cFileHandler](#). The files can be loaded using the filename or if loaded from an IMF file using the reference for each file.

Inheritance diagram for cFile:



Collaboration diagram for cFile:



Public Member Functions

- [cFile \(\)](#)

This constructor will automatically load the file from memory and add it to the list in cFilehandler.

- `~cFile ()`
- `char * FileName ()`

This will return the files filename.

- `void Load ()`

This is the function that will actually load the file from a hdd.

- `void Delete ()`

This will delete the file from memory.

Public Attributes

- `cLinkedNode< vFile > * mpNode`

This is a pointer to the `cLinkedNode` which owns this file.

- `char mpFileName [64]`

This will store the files filename.

Protected Attributes

- `cFileHandler * mpFileHandler`

This is a pointer to the `cFileHandler` which owns this file.

Friends

- class `cFileHandler`

21.24.1 Detailed Description

This is the base code for files to be loaded from a hdd. Any file object loaded from a hdd should inherit this class. It is best used for media files. This code will automatically add newly loaded files to `cFileHandler`. The files can be loaded using the filename or if loaded from an IMF file using the reference for each file.

Definition at line 11 of file WTcFile.h.

21.24.2 Constructor & Destructor Documentation

21.24.2.1 `cFile::cFile()`

This constructor will automatically load the file from memory and add it to the list in `cFilehandler`.

Definition at line 8 of file `WTcFile.cpp`.

21.24.2.2 `cFile::~cFile()`

Definition at line 17 of file `WTcFile.cpp`.

21.24.3 Member Function Documentation

21.24.3.1 `void cFile::Delete()`

This will delete the file from memory.

Definition at line 22 of file `WTcFile.cpp`.

21.24.3.2 `char * cFile::fileName() [virtual]`

This will return the files filename.

Implements [vFile](#).

Definition at line 27 of file `WTcFile.cpp`.

21.24.3.3 `void cFile::Load() [virtual]`

This is the function that will actually load the file from a hdd.

Implements [vFile](#).

Definition at line 32 of file `WTcFile.cpp`.

21.24.4 Friends And Related Function Documentation

21.24.4.1 `friend class cFileHandler [friend]`

Definition at line 15 of file `WTcFile.h`.

21.24.5 Member Data Documentation

21.24.5.1 `cFileHandler* cFile::mpFileHandler [protected]`

This is a pointer to the [cFileHandler](#) which owns this file.

Definition at line 37 of file WTcFile.h.

21.24.5.2 char cFile::mpFileName[64]

This will store the files filename.

Definition at line 24 of file WTcFile.h.

21.24.5.3 cLinkedNode<vFile>* cFile::mpNode

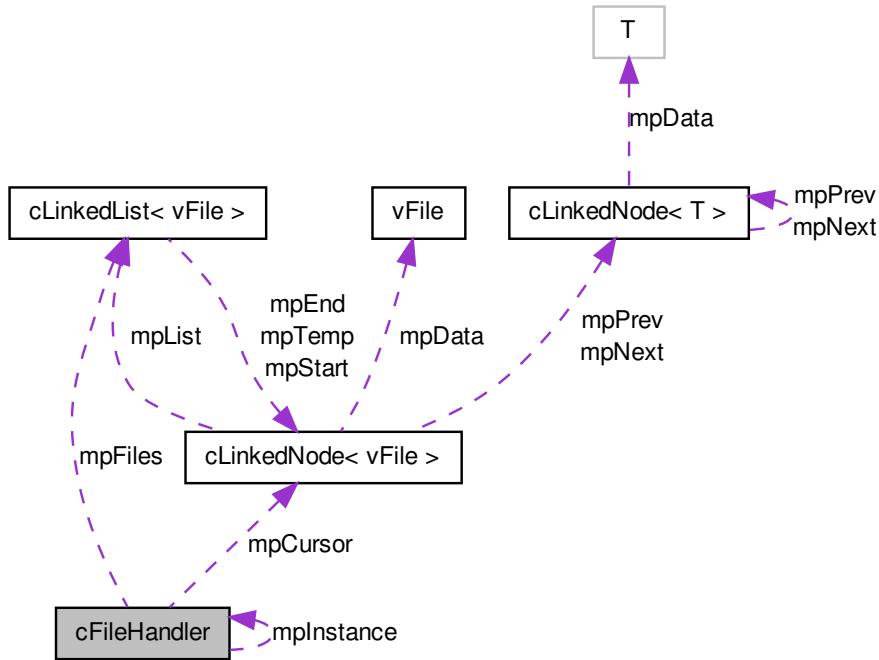
This is a pointer to the [cLinkedNode](#) which owns this file.

Definition at line 17 of file WTcFile.h.

21.25 cFileHandler Class Reference

This is the handler for the file system. This handles all files loaded from a hdd. It will give allow processes to use files loaded else where, using either the filenames or if loaded from an IMF a file reference. The files are stored in the list mpFiles.

Collaboration diagram for cFileHandler:



Public Member Functions

- `~cFileHandler ()`
- `void Reload ()`

This will reload all the files from a hdd into memory. (Note this is not currently complete).

- `void Unload ()`

This will remove all the files from memory. (Note this is not currently complete).

- `void * File (const char *lpFilename)`

This will return a pointer to a file with the filename or file reference lpFilename.

- `cLinkedNode< vFile > * Add (vFile *lpNew)`

This will add the file pointed to by lpNew to the current file list mpFile.

- `void Delete (cLinkedNode< vFile > *lpNode)`

This will delete the file owned by lpNode from the file list mpFile.

Static Public Member Functions

- static [cFileHandler * Instance \(\)](#)

This will return a pointer to the current [cFileHandler](#). If there is no current instance it will create one and return the pointer.

21.25.1 Detailed Description

This is the handler for the file system. This handles all files loaded from a hdd. It will give allow processes to use files loaded else where, using either the filenames or if loaded from an IMF a file reference. The files are stored in the list mpFiles.

Definition at line 10 of file WTcFileHandler.h.

21.25.2 Constructor & Destructor Documentation

21.25.2.1 [cFileHandler::~cFileHandler \(\)](#)

Definition at line 24 of file WTcFileHandler.cpp.

21.25.3 Member Function Documentation

21.25.3.1 [cLinkedNode< vFile > * cFileHandler::Add \(vFile * lpNew \)](#)

This will add the file pointed to by lpNew to the current file list mpFile.

Definition at line 43 of file WTcFileHandler.cpp.

21.25.3.2 [void cFileHandler::Delete \(cLinkedNode< vFile > * lpNode \)](#)

This will delete the file owned by lpNode from the file list mpFile.

Definition at line 52 of file WTcFileHandler.cpp.

21.25.3.3 [void * cFileHandler::File \(const char * lpFilename \)](#)

This will return a pointer to a file with the filename or file reference lpFilename.

Definition at line 9 of file WTcFileHandler.cpp.

21.25.3.4 cFileHandler * cFileHandler::Instance() [static]

This will return a pointer to the current [cFileHandler](#). If there is no current instance it will create one and return the pointer.

Definition at line 33 of file WTcFileHandler.cpp.

21.25.3.5 void cFileHandler::Reload()

This will reload all the files from a hdd into memory. (Note this is not currently complete).

Definition at line 39 of file WTcFileHandler.cpp.

21.25.3.6 void cFileHandler::Unload()

This will remove all the files from memory. (Note this is not currently complete).

Definition at line 41 of file WTcFileHandler.cpp.

21.26 cFog Class Reference

This class will create and control an OpenGL scenes fog effect.

Public Member Functions

- [cFog\(\)](#)
Creates a clear fog effect.
- [cFog\(float lfStart, float lfEnd\)](#)
Creates a new fog object between distance lfStart and lfEnd.
- [void SetColor\(float lfRed, float lfGreen, float lfBlue, float lfAlpha\)](#)
Will set the RGBA color of the fog.
- [void SetMode\(char ltMode\)](#)
Will set the fog equation the fog effect will use. (GL_EXP, GL_EXP2, GL_LINEAR)
- [void SetDensity\(float lfDensity\)](#)
Will set the effect the fog has on the screen colors. 1.0f is Opaque, 0.0f is completely transparent.
- [void SetRange\(float lfStart, float lfEnd\)](#)
Will set the near and far distances, from the camera, of the fog effect.
- [bool On\(\)](#)

Will return whether the fog effect is on or off.

- void **SetOn ()**
Will set the fog effect on.
- void **SetOff ()**
Will set the fog effect off.

21.26.1 Detailed Description

This class will create and control an OpenGL scenes fog effect.

Definition at line 7 of file WTcFog.h.

21.26.2 Constructor & Destructor Documentation

21.26.2.1 **cFog::cFog ()**

Creates a clear fog effect.

Definition at line 3 of file WTcFog.cpp.

21.26.2.2 **cFog::cFog (float lfStart, float lfEnd)**

Creates a new fog object between distance lfStart and lfEnd.

Definition at line 18 of file WTcFog.cpp.

21.26.3 Member Function Documentation

21.26.3.1 **bool cFog::On ()**

Will return whether the fog effect is on or off.

Definition at line 77 of file WTcFog.cpp.

21.26.3.2 **void cFog::SetColor (float lfRed, float lfGreen, float lfBlue, float lfAlpha)**

Will set the RGBA color of the fog.

Definition at line 50 of file WTcFog.cpp.

21.26.3.3 **void cFog::SetDensity (float lfDensity)**

Will set the effect the fog has on the screen colors. 1.0f is Opaque, 0.0f is completely transparent.

Definition at line 64 of file WTcFog.cpp.

21.26.3.4 void cFog::SetMode (char *lMode*)

Will set the fog equation the fog effect will use. (GL_EXP, GL_EXP2, GL_LINEAR)

Definition at line 58 of file WTcFog.cpp.

21.26.3.5 void cFog::SetOff ()

Will set the fog effect off.

Definition at line 45 of file WTcFog.cpp.

21.26.3.6 void cFog::SetOn ()

Will set the fog effect on.

Definition at line 34 of file WTcFog.cpp.

21.26.3.7 void cFog::SetRange (float *lStart*, float *lEnd*)

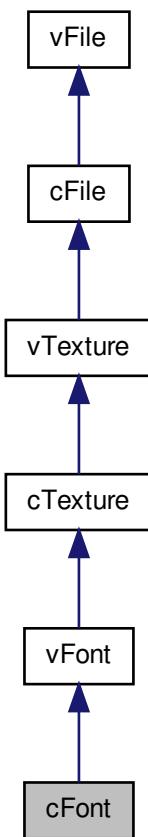
Will set the near and far distances, from the camera, of the fog effect.

Definition at line 71 of file WTcFog.cpp.

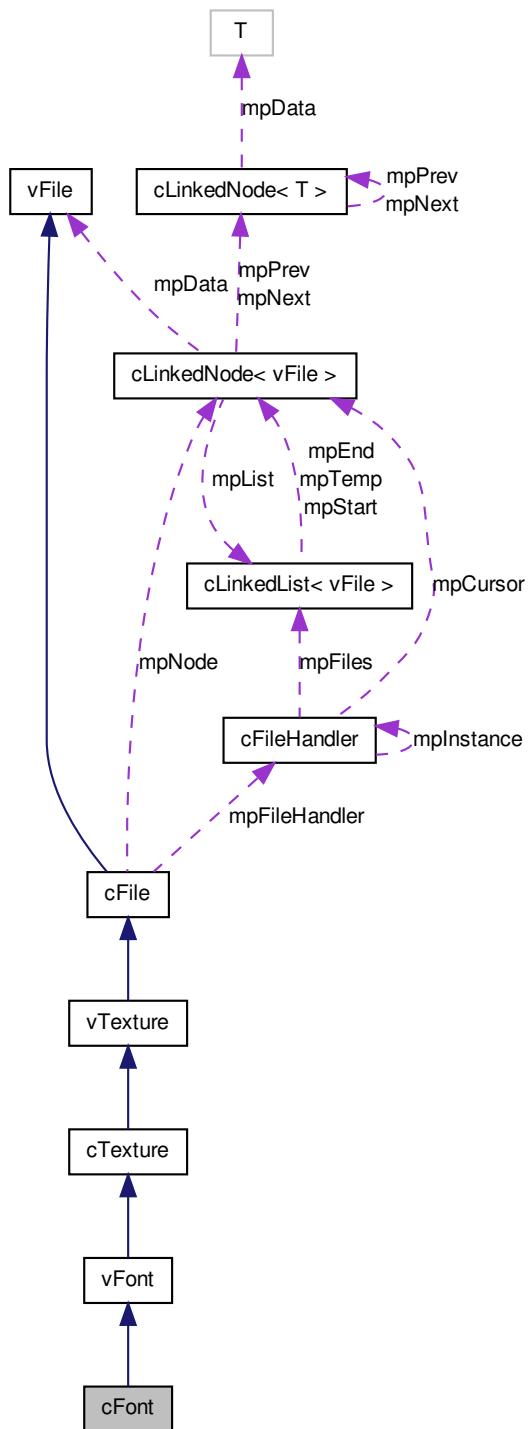
21.27 cFont Class Reference

This class will store the data for a Font ready to be used for rendering [cText](#). This should come from an IMF file and be composed of an image of 64 character images stacked vertically. This is a file class and should be handled entirely by the engine.

Inheritance diagram for cFont:



Collaboration diagram for cFont:



Public Member Functions

- `~cFont()`
- `cFont(cTextureArray *lpData)`
- `uint8 Character(uint8 lcChar)`
- `uint32 Height()`
- `void UpdateTexture()`
- `void BindTexture()`

21.27.1 Detailed Description

This class will store the data for a Font ready to be used for rendering `cText`. This should come from an IMF file and be composed of an image of 64 character images stacked vertically. This is a file class and should be handled entirely by the engine.

Definition at line 8 of file WTcFont.h.

21.27.2 Constructor & Destructor Documentation

21.27.2.1 `cFont::~cFont()`

Definition at line 20 of file WTcFont.cpp.

21.27.2.2 `cFont::cFont(cTextureArray *lpData)`

Definition at line 6 of file WTcFont.cpp.

21.27.3 Member Function Documentation

21.27.3.1 `void cFont::BindTexture()` [virtual]

Reimplemented from `cTexture`.

Definition at line 41 of file WTcFont.cpp.

21.27.3.2 `uint8 cFont::Character(uint8 lcChar)` [virtual]

Implements `vFont`.

Definition at line 49 of file WTcFont.cpp.

21.27.3.3 `uint32 cFont::Height()` [inline, virtual]

Reimplemented from `cTexture`.

Definition at line 16 of file WTcFont.h.

21.27.3.4 void cFont::UpdateTexture() [virtual]

Reimplemented from [cTexture](#).

Definition at line 25 of file WTcFont.cpp.

21.28 cFrameRate Class Reference

This class will store data for controlling the Frame Rate. There are several settings that can be adjusted:

Processes Per Frame:

Frames Per Second:

Frame Time and Process Time:

These are the amount of time that passes every Frame and Process Cycle measured in seconds. Multiplying any distances moved by this will convert them into distance per second. This allows the user to change the running frame rate without affecting the speed the user experiences. If the Frame rate and Process Rate are not going to be changed this can be ignored.

Collaboration diagram for cFrameRate:



Public Member Functions

- void [SetFrameRate](#) (uint8 lfFramesPerSecond)

This is the number of Frames that will be rendered each second. This results in smooth consistent timings for every Process Cycle and Rendered Frame. This is best set to 60 / 70 though in extreme cases it can be set to 30. Rendering is very computationally expensive, so should be maintained at the lowest rate to produce a smooth visual image.

- void [SetProcessesPerFrame](#) (uint8 liPPS)

This is the number of times the Process cycle will be run for every rendered frame. Rendering is very slow and CPU intensive compared to the Processing cycle. The Human eye cannot differentiate between very fast rendering rates, but increased numbers of processing cycles will increase accuracy of collisions and make events smoother.

- float [FrameTime \(\)](#)

this will return the amount of time that passes for every rendered frame.

- float [ProcessTime \(\)](#)

This is the amount of time that passes every Process Cycle measured in seconds. Multiplying any distances moved by this will convert them into distance per second. This allows the user to change the running frame rate without affecting the speed the user experiences. If the Frame rate and Process Rate are not going to be changed this can be ignored.

- uint8 [FramesPerSecond \(\)](#)

This will return the current set number of Frames per Second.

- uint8 [ProcessesPerFrame \(\)](#)

this will return the current set number of Process Cycles per Rendered Frame.

Static Public Member Functions

- static [cFrameRate * Instance \(\)](#)

This will return a pointer to the current [cFrameRate Object](#).

21.28.1 Detailed Description

This class will store data for controlling the Frame Rate. There are several settings that can be adjusted:

Processes Per Frame:

Frames Per Second:

Frame Time and Process Time:

These are the amount of time that passes every Frame and Process Cycle measured in seconds. Multiplying any distances moved by this will convert them into distance per second. This allows the user to change the running frame rate without affecting the speed the user experiences. If the Frame rate and Process Rate are not going to be changed this can be ignored.

Definition at line 13 of file WTcFrameRate.h.

21.28.2 Member Function Documentation

21.28.2.1 uint8 cFrameRate::FramesPerSecond ()

This will return the current set number of Frames per Second.

Definition at line 23 of file WTcFrameRate.cpp.

21.28.2.2 float cFrameRate::FrameTime()

this will return the amount of time that passes for every rendered frame.

Definition at line 21 of file WTcFrameRate.cpp.

21.28.2.3 cFrameRate * cFrameRate::Instance() [static]

This will return a pointer to the current **cFrameRate** Object.

Definition at line 5 of file WTcFrameRate.cpp.

21.28.2.4 uint8 cFrameRate::ProcessesPerFrame()

this will return the current set number of Process Cycles per Rendered Frame.

Definition at line 24 of file WTcFrameRate.cpp.

21.28.2.5 float cFrameRate::ProcessTime()

This is the amount of time that passes every Process Cycle measured in seconds. Multiplying any distances moved by this will convert them into distance per second. This allows the user to change the running frame rate without affecting the speed the user experiences. If the Frame rate and Process Rate are not going to be changed this can be ignored.

Definition at line 22 of file WTcFrameRate.cpp.

21.28.2.6 void cFrameRate::SetFrameRate(uint8 lfFramesPerSecond)

This is the number of Frames that will be rendered each second. This results in smooth consistent timings for every Process Cycle and Rendered Frame. This is best set to 60 / 70 though in extreme cases it can be set to 30. Rendering is very computationally expensive, so should be maintained at the lowest rate to produce a smooth visual image.

Definition at line 19 of file WTcFrameRate.cpp.

21.28.2.7 void cFrameRate::SetProcessesPerFrame(uint8 liPPS)

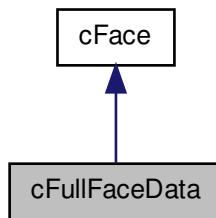
This is the number of times the Process cycle will be run for every rendered frame. Rendering is very slow and CPU intensive compared to the Processing cycle. The Human eye cannot differentiate between very fast rendering rates, but increased numbers of processing cycles will increase accuracy of collisions and make events smoother.

Definition at line 20 of file WTcFrameRate.cpp.

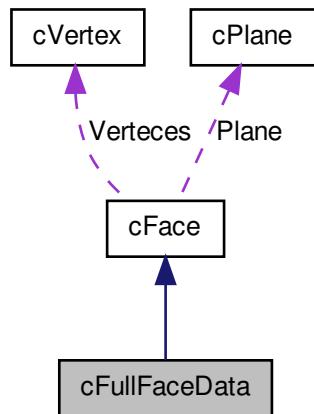
21.29 cFullFaceData Class Reference

This class is a [cFace](#) with expanded Functionality. Allows the user to check information about the face. See list of functions for information.

Inheritance diagram for cFullFaceData:



Collaboration diagram for cFullFaceData:



Public Member Functions

- [cFullFaceData \(\)](#)

- uint32 [FileSize \(\)](#)
 Will Fill the data for this Object from a array of floats representing vertex coordinates (3 floats to a vertex) and an array of 3 uint16s indicating which verteces to use.
- bool [SamePlane \(cPlane &lpOther\)](#)
 Will return 1 if the [cPlane](#) lpOther is in the same plane as this object.
- [cFullFaceData & operator= \(cFullFaceData &lpOther\)](#)
 $= operator$
- bool [IsConcave \(cFullFaceList &lpOther\)](#)
 Will return one if this Face is concave with the first connected cFullFace it is touching.
- bool [IsConcave \(cFace lpOther\)](#)
 Will return 1 if this Face is concave (and shares an edge) with the [cFace](#) lpOther.
- uint8 [NotSharedVertex \(cFace &lpOther\)](#)
 Will return the number of the Vertex owned by lpOther which is not shared by this face (0,1,2). If all faces are shared, will return 3. If no faces are shared will return 2.
- uint8 [SharesVertex \(cVertex *lpOther\)](#)
 Will return the number of verteces that this object shares with the array of 3 [cVertex](#) object pointed to by lpOther (0,1,2,3).
- void [OutputIMFFullFace \(ofstream &FileStream\)](#)
- void [LoadIMFFullFace \(ifstream &FileStream\)](#)
- void [Display \(\)](#)
 Will output this object to the terminal.

21.29.1 Detailed Description

This class is a [cFace](#) with expanded Functionality. Allows the user to check information about the face. See list of functions for information.

Definition at line 7 of file WTcFullFaceData.h.

21.29.2 Constructor & Destructor Documentation

21.29.2.1 [cFullFaceData::cFullFaceData \(\)](#)

Definition at line 174 of file WTcFullFaceData.cpp.

21.29.3 Member Function Documentation

21.29.3.1 void cFullFaceData::Display()

Will output this object to the terminal.

Definition at line 34 of file WTcFullFaceData.cpp.

21.29.3.2 uint32 cFullFaceData::FileSize()

Reimplemented from [cFace](#).

Definition at line 3 of file WTcFullFaceData.cpp.

21.29.3.3 void cFullFaceData::Fill(float *lpVertex, uint16 *mpFace)

Will Fill the data for this Object from a array of floats representing vertex coordinates (3 floats to a vertex) and an array of 3 uint16s indicating which verteces to use.

Definition at line 43 of file WTcFullFaceData.cpp.

21.29.3.4 bool cFullFaceData::IsConcave(cFullFaceList & lpOther)

Will return one if this Face is concave with the first connected cFullFace it is touching.

Definition at line 77 of file WTcFullFaceData.cpp.

21.29.3.5 bool cFullFaceData::IsConcave(cFace lpOther)

Will return 1 if this Face is concave (and shares an edge) with the [cFace](#) lpOther.

Definition at line 109 of file WTcFullFaceData.cpp.

21.29.3.6 void cFullFaceData::LoadIMFFullFace(ifstream & FileStream)

Definition at line 25 of file WTcFullFaceData.cpp.

21.29.3.7 uint8 cFullFaceData::NotSharedVertex(cFace & lpOther)

Will return the number of the Vertex owned by lpOther which is not shared by this face (0,1,2). If all faces are shared, will return 3. If no faces are shared will return 2.

Definition at line 95 of file WTcFullFaceData.cpp.

21.29.3.8 cFullFaceData & cFullFaceData::operator=(cFullFaceData & lpOther)

= operator

Definition at line 68 of file WTcFullFaceData.cpp.

21.29.3.9 void cFullFaceData::OutputIMFFullFace (ostream & FileStream)

Definition at line 16 of file WTcFullFaceData.cpp.

21.29.3.10 bool cFullFaceData::SamePlane (cPlane & lpOther)

Will return 1 if the `cPlane` lpOther is in the same plane as this object.

Definition at line 63 of file WTcFullFaceData.cpp.

21.29.3.11 uint8 cFullFaceData::SharesVertex (cVertex * lpOther)

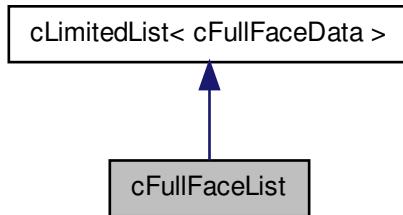
Will return the number of verteces that this object shares with the array of 3 `cVertex` object pointed to by lpOther (0,1,2,3).

Definition at line 53 of file WTcFullFaceData.cpp.

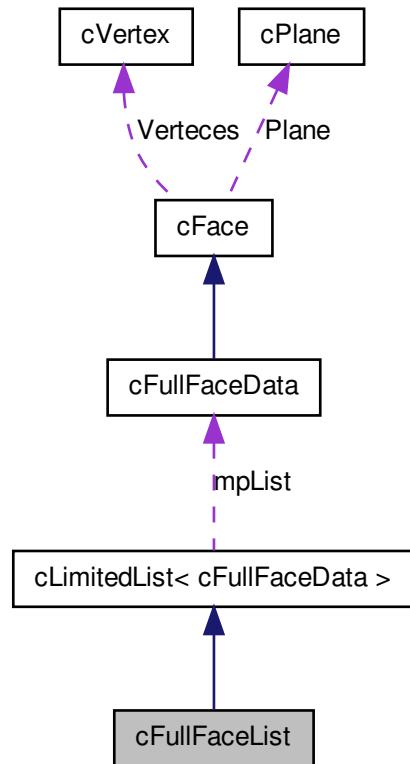
21.30 cFullFaceList Class Reference

A dynamic linked list of `cFullFaceData`. As described. Can load object to and from IMFs.

Inheritance diagram for cFullFaceList:



Collaboration diagram for cFullFaceList:



Public Member Functions

- `cFullFaceList ()`
- `~cFullFaceList ()`
- `void OutputIMFFullFaces (ofstream &FileStream)`
- `void LoadIMFFullFaces (ifstream &FileStream)`
- `uint32 FileSize ()`
- `void Display ()`
- `void Normalise ()`

Will Normalise all Normals of the `cFullFaceData` objects in the list.

21.30.1 Detailed Description

A dynamic linked list of [cFullFaceData](#). As described. Can load object to and from IMFs.

Definition at line 46 of file WTcFullFaceData.h.

21.30.2 Constructor & Destructor Documentation

21.30.2.1 [cFullFaceList::cFullFaceList\(\)](#)

Definition at line 171 of file WTcFullFaceData.cpp.

21.30.2.2 [cFullFaceList::~cFullFaceList\(\)](#)

Definition at line 172 of file WTcFullFaceData.cpp.

21.30.3 Member Function Documentation

21.30.3.1 [void cFullFaceList::Display\(\)](#)

Definition at line 152 of file WTcFullFaceData.cpp.

21.30.3.2 [uint32 cFullFaceList::FileSize\(\)](#)

Definition at line 140 of file WTcFullFaceData.cpp.

21.30.3.3 [void cFullFaceList::LoadIMFFullFaces\(ifstream & FileStream \)](#)

Definition at line 128 of file WTcFullFaceData.cpp.

21.30.3.4 [void cFullFaceList::Normalise\(\)](#)

Will Normalise all Normals of the [cFullFaceData](#) objects in the list.

Definition at line 162 of file WTcFullFaceData.cpp.

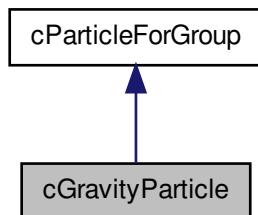
21.30.3.5 [void cFullFaceList::OutputIMFFullFaces\(ofstream & FileStream \)](#)

Definition at line 118 of file WTcFullFaceData.cpp.

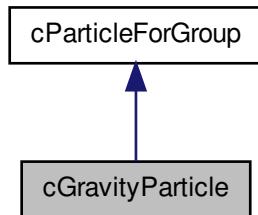
21.31 cGravityParticle Class Reference

cParticles which are affected by Gravity. These Particles have the code to be affected by the variables _GRAVITY_X,_GRAVITY_Y and _GRAVITY_Z. [UpdatePos\(\)](#) will account use the current Gravity settings to calculate the speed and position.

Inheritance diagram for cGravityParticle:



Collaboration diagram for cGravityParticle:



Public Member Functions

- void [UpdatePos \(\)](#)

Friends

- class [cParticleHandler](#)

21.31.1 Detailed Description

cParticles which are affected by Gravity. These Particles have the code to be affected by the variables _GRAVITY_X,_GRAVITY_Y and _GRAVITY_Z. [UpdatePos\(\)](#) will account use the current Gravity settings to calculate the speed and position.

Definition at line 28 of file WTcParticle.h.

21.31.2 Member Function Documentation

21.31.2.1 void cGravityParticle::UpdatePos() [virtual]

Reimplemented from [cParticleForGroup](#).

Definition at line 184 of file WTcParticle.cpp.

21.31.3 Friends And Related Function Documentation

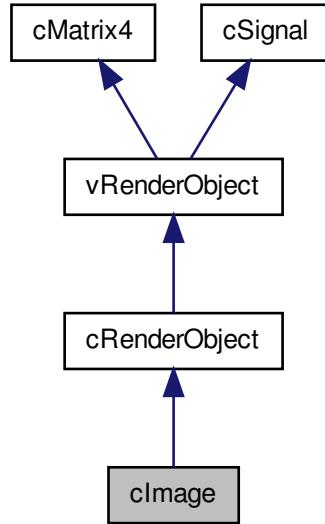
21.31.3.1 friend class cParticleHandler [friend]

Definition at line 36 of file WTcParticle.h.

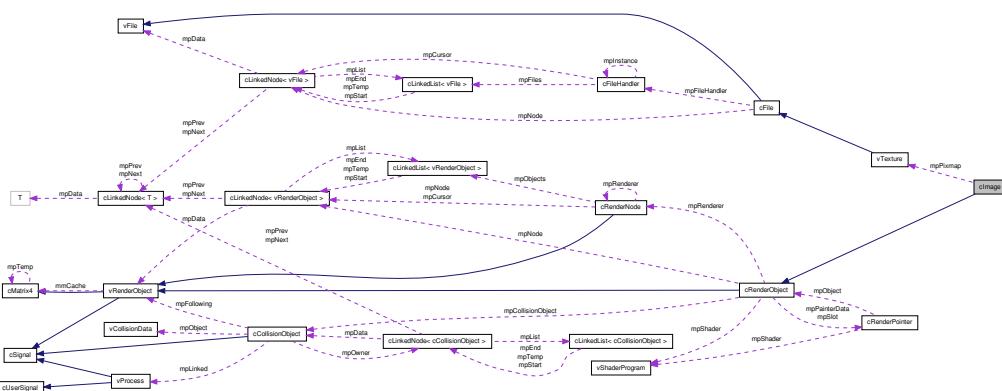
21.32 climage Class Reference

A 2D renderable object.

Inheritance diagram for cImage:



Collaboration diagram for cImage:



Public Member Functions

- **cImage (vTexture *lpTexture)**

Will construct a new [cImage](#) class, and set the [vTexture](#) it will use.

- [cImage \(\)](#)

Constructor for [cImage](#). Will Create an Empty [cImage](#) Object.

- [void SetUp \(\)](#)

- [~cImage \(\)](#)

- [void Texture \(vTexture *lpObject\)](#)

Will set the current [vTexture](#) this object will use.

- [unsigned int TextureNumber \(\)](#)

This returns the Texture ID of the texture this object will use.

- [void RenderPainter \(uint8 liLevel\)](#)

virtual function to allow polymorphism. see [cCamera::RenderPainter\(\)](#);

- [void RenderToPainter \(\)](#)

virtual function to allow polymorphism. see [cCamera::RenderToPainter\(\)](#);

- [void Render \(\)](#)

virtual function to allow polymorphiss. see [cCamera::Render\(\)](#),

- [void Size \(float lfSize\)](#)

Sets the size of the image on screen.

21.32.1 Detailed Description

A 2D renderable object.

Parameters

<i>lpTexture</i>	pointer to the texture to bind to this 2D object This is actually a 3D polygon, which has a texture bound to it. it can be used as an OpenGL accelerated sprite.
------------------	--

Definition at line 9 of file WTcImage.h.

21.32.2 Constructor & Destructor Documentation

21.32.2.1 [cImage::cImage \(vTexture * lpTexture \)](#)

Will construct a new [cImage](#) class, and set the [vTexture](#) it will use.

Constructor for [cImage](#).

Parameters

<i>lpTexture</i>	the vTexture that this object will use.
------------------	---

Definition at line 13 of file WTcImage.cpp.

21.32.2.2 `cImage::cImage()`

Constructor for [cImage](#). Will Create an Empty [cImage](#) Object.

Definition at line 19 of file WTcImage.cpp.

21.32.2.3 `cImage::~cImage()`

Definition at line 11 of file WTcImage.cpp.

21.32.3 Member Function Documentation

21.32.3.1 `void cImage::Render() [virtual]`

virtual function to allow polymorphiss. see [cCamera::Render\(\)](#);

Implements [cRenderObject](#).

Definition at line 65 of file WTcImage.cpp.

21.32.3.2 `void cImage::RenderPainter(uint8 iLevel) [virtual]`

virtual function to allow polymorphism. see [cCamera::RenderPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 50 of file WTcImage.cpp.

21.32.3.3 `void cImage::RenderToPainter() [virtual]`

virtual function to allow polymorphism. see [cCamera::RenderToPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 32 of file WTcImage.cpp.

21.32.3.4 `void cImage::SetUp()`

Definition at line 25 of file WTcImage.cpp.

21.32.3.5 `void cImage::Size(float fSize)`

Sets the size of the image on screen.

Definition at line 124 of file WTcImage.cpp.

21.32.3.6 void cImage::Texture (vTexture * *lpObject*)

Will set the current [vTexture](#) this object will use.

Definition at line 82 of file WTcImage.cpp.

21.32.3.7 unsigned int cImage::TextureNumber ()

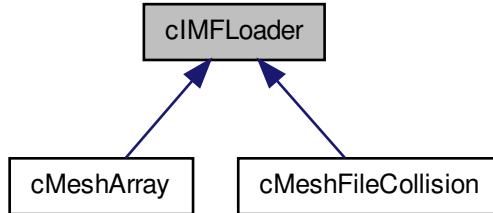
This returns the Texture ID of the texture this object will use.

Reimplemented from [cRenderObject](#).

Definition at line 118 of file WTcImage.cpp.

21.33 cIMFLoader Class Reference

Inheritance diagram for cIMFLoader:



Public Member Functions

- void [LoadIMF](#) (ifstream &FileStream)

21.33.1 Detailed Description

Definition at line 4 of file WTcIMFLoader.h.

21.33.2 Member Function Documentation

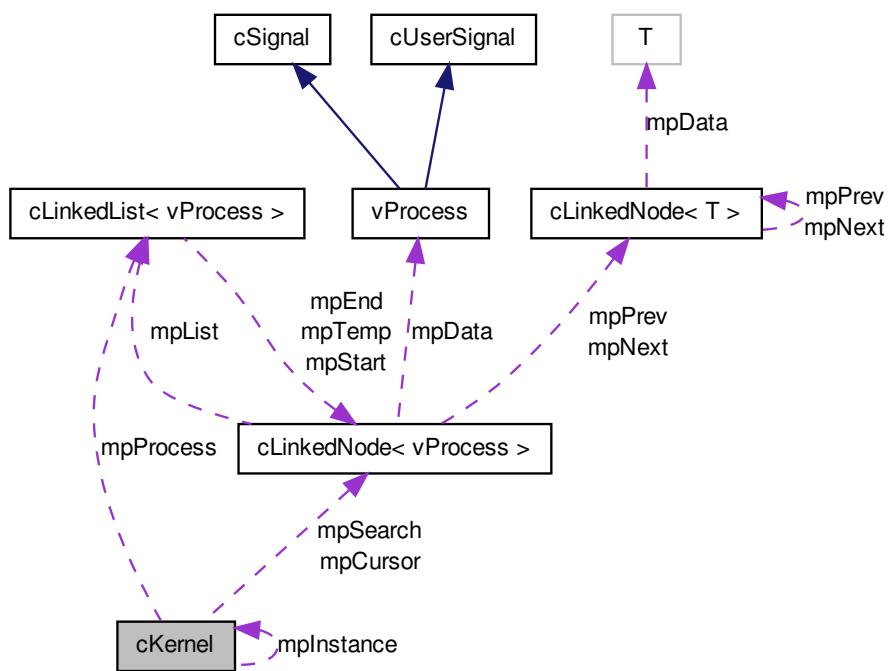
21.33.2.1 void cIMFLoader::LoadIMF (ifstream & FileStream)

Reimplemented in [cMeshFileCollision](#), and [cMeshArray](#).

21.34 cKernel Class Reference

Kernel Object. Handles Processes. Tracks, runs and deletes current processes. Has complete control over every [cProcess](#) object. Will run all awake alive processes every process cycle, will delete dead processes. Also controls the activation of rendering frames and handling interactions with the operating system.

Collaboration diagram for cKernel:



Public Member Functions

- void [KillProgram \(\)](#)
Will kill the entire program. Will delete every process and then exit.
- [~cKernel \(\)](#)
- void [Update \(\)](#)
- [cLinkedNode< vProcess > * Add \(vProcess *lpNew\)](#)
- void [Remove \(cLinkedNode< vProcess > *lpOld\)](#)
- void [DeleteAll \(\)](#)

Will Delete all the processes in the current process list. This will effectively end the program.

- template<class tType >
tType * **FindProcess** ()

This Function will search for a process of either tType OR a type which has uses the type tType as a base type.

- void **ResetFindProcess** ()

*This Function will reset **FindProcess()** to search from the start of the process List.*

Static Public Member Functions

- static cKernel * **Instance** ()

Function which returns current Kernel Instance. Will Create a new instance if one does not already exist.

21.34.1 Detailed Description

Kernel Object. Handles Processes. Tracks, runs and deletes current processes. Has complete control over every **cProcess** object. Will run all awake alive processes every process cycle, will delete dead processes. Also controls the activation of rendering frames and handling interactions with the operating system.

Definition at line 15 of file WTKernel.h.

21.34.2 Constructor & Destructor Documentation

21.34.2.1 cKernel::~cKernel()

Definition at line 94 of file WTKernel.cpp.

21.34.3 Member Function Documentation

21.34.3.1 cLinkedNode< vProcess > * cKernel::Add(vProcess * lpNew)

Definition at line 123 of file WTKernel.cpp.

21.34.3.2 void cKernel::DeleteAll()

Will Delete all the processes in the current process list. This will effectively end the program.

Definition at line 12 of file WTKernel.cpp.

21.34.3.3 template<class tType > tType * cKernel::FindProcess()

This Function will search for a process of either tType OR a type which has uses the type tType as a base type.

Parameters

<i>mpType</i>	is a pointer of the type (or base type) of the desired process.
---------------	---

Returns

Will return a pointer to a process currently in the process list of the correct type.

This Function takes the type of the pointer handed to the function and will search for a process which is of type tType or inherits tType. Can be used to find processes of a certain type or genus. Each call of this function will continue searching from the position stored in mpSearch. Between searches for processes of different types use the function [ResetFindProcess\(\)](#).

```
cGunShip *mpGunShip;
cBattleShip *mpBattleShip;

mpGunShip = FindProcess(mpGunShip);
ResetFindProcess();
mpBattleShip = FindProcess(mpBattleShip);
```

Definition at line 92 of file WTKernel.h.

21.34.3.4 cKernel * cKernel::Instance() [static]

Function which returns current Kernel Instance. Will Create a new instance if one does not already exist.

Definition at line 139 of file WTKernel.cpp.

21.34.3.5 void cKernel::KillProgram()

Will kill the entire program. Will delete every process and then exit.

Definition at line 162 of file WTKernel.cpp.

21.34.3.6 void cKernel::Remove(cLinkedNode< vProcess > * *lpOld*)

Definition at line 134 of file WTKernel.cpp.

21.34.3.7 void cKernel::ResetFindProcess()

This Function will reset [FindProcess\(\)](#) to search from the start of the process List.

Definition at line 160 of file WTKernel.cpp.

21.34.3.8 void cKernel::Update()

Definition at line 18 of file WTKernel.cpp.

21.35 cKeyStore Class Reference

This class stores all the input data for a single keyboard.

Public Member Functions

- bool [operator\[\]](#) (uint16 liKey)
This will return a specific key state, using the relevant key code.
- bool [Array](#) (uint16 liBit, uint16 liKey)
This will create and initialise the keystore object.
- bool * [SetArray](#) (uint16 liBit, uint16 liKey)
- [cKeyStore](#) ()
- bool [operator\[\]](#) (uint16 liKey)
This will return a specific key state, using the relevant key code.
- void [SetKeyState](#) (bool lbState, uint16 liKey)
This will create and initialise the keystore object.
- [cKeyStore](#) ()
- bool [WinKey](#) (uint16 liKey)

Public Attributes

- bool [key](#) [2][256]
This is the array of keystates.

21.35.1 Detailed Description

This class stores all the input data for a single keyboard.

Definition at line 6 of file WTcKeyStore.h.

21.35.2 Constructor & Destructor Documentation

21.35.2.1 [cKeyStore::cKeyStore\(\)](#)

Definition at line 4 of file WTcKeyStore.cpp.

21.35.2.2 **cKeyStore::cKeyStore ()**

21.35.3 Member Function Documentation

21.35.3.1 **bool cKeyStore::Array (uint16 *liBit*, uint16 *liKey*) [inline]**

This will create and initialise the keystore object.

Definition at line 15 of file WTcKeyStore.h.

21.35.3.2 **bool cKeyStore::operator[] (uint16 *liKey*)**

This will return a specific key state, using the relevant key code.

21.35.3.3 **bool cKeyStore::operator[] (uint16 *liKey*) [inline]**

This will return a specific key state, using the relevant key code.

Definition at line 30 of file WTcKeyStore.h.

21.35.3.4 **bool* cKeyStore::SetArray (uint16 *liBit*, uint16 *liKey*) [inline]**

Definition at line 16 of file WTcKeyStore.h.

21.35.3.5 **void cKeyStore::SetKeyState (bool *lbState*, uint16 *liKey*) [inline]**

This will create and initialise the keystore object.

Definition at line 33 of file WTcKeyStore.h.

21.35.3.6 **bool cKeyStore::WinKey (uint16 *liKey*) [inline]**

Definition at line 36 of file WTcKeyStore.h.

21.35.4 Member Data Documentation

21.35.4.1 **bool cKeyStore::key**

This is the array of keystates.

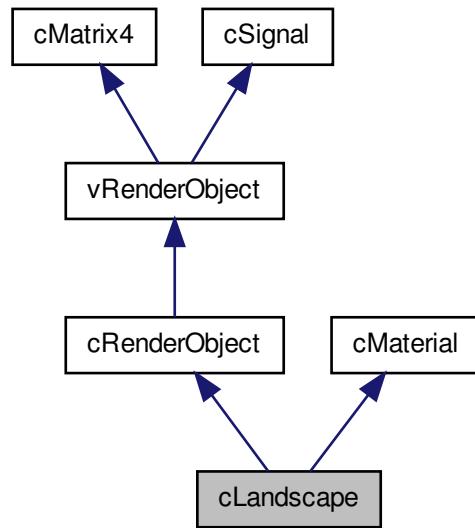
Definition at line 10 of file WTcKeyStore.h.

21.36 cLandscape Class Reference

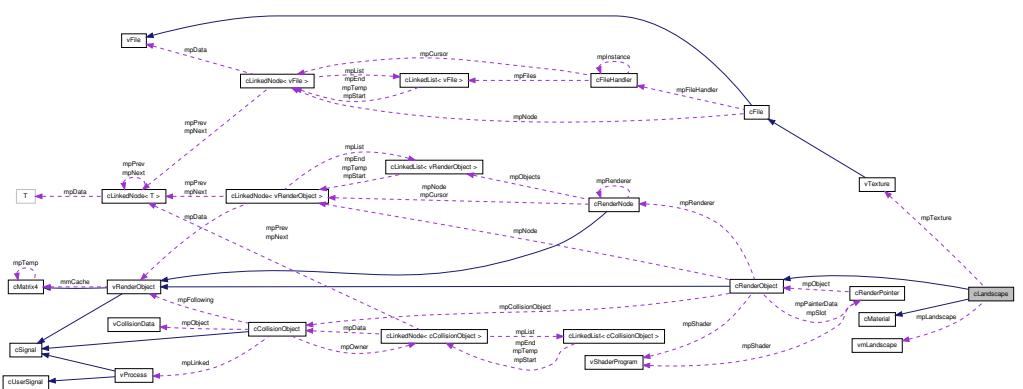
A height map based, matrix structured Landscape. Landscape is composed of a matrix of square polygons. The heights of each vertex is produced using the packaging

software, and is generated from a bitmap.

Inheritance diagram for cLandscape:



Collaboration diagram for `cLandscape`:



Public Member Functions

- void `RenderToPainter ()`

virtual function to allow polymorphism. see cCamera::RenderToPainter();

- void **RenderPainter** (uint8 liLevel)
virtual function to allow polymorphism. see cCamera::RenderPainter();
- void **Render** ()
virtual function to allow polymorphiss. see cCamera::Render();
- **cLandscape** ()
- **cLandscape** (vmLandscape *lpModel, vTexture *lpTexture)
Create a landscape object and set its height map and texture.
- void **Texture** (vTexture *lpTexture)
Set the texture bound to this landscape object.
- void **Landscape** (vmLandscape *lpLandscape)
Set the current height map for this landscape object.
- float **GetHeight** (float lfX, float lfZ)
Will return the height at Global co-ordinates lfX,lfZ.
- float **GetHeightLocal** (float lfX, float lfZ)
Will return the height at the Local position lfX,lfZ (relative to landscapes corner)
- float **GetVertexHeight** (int liX, int liZ)
*Will return the height of the vertex at liX,liZ. (position is based on number of segments
NOT distance)*

21.36.1 Detailed Description

A height map based, matrix structured Landscape. Landscape is composed of a matrix of square polygons. The heights of each vertex is produced using the packaging software, and is generated from a bitmap.

Definition at line 11 of file WTcLandscape.h.

21.36.2 Constructor & Destructor Documentation

21.36.2.1 cLandscape::cLandscape ()

Definition at line 4 of file WTcLandscape.cpp.

21.36.2.2 cLandscape::cLandscape (vmLandscape * lpModel, vTexture * lpTexture)

Create a landscape object and set its height map and texture.

Definition at line 11 of file WTcLandscape.cpp.

21.36.3 Member Function Documentation

21.36.3.1 `float cLandscape::GetHeight (float lfX, float lfZ)`

Will return the height at Global co-ordinates lfX,lfZ.

Definition at line 79 of file WTcLandscape.cpp.

21.36.3.2 `float cLandscape::GetHeightLocal (float lfX, float lfZ)`

Will return the height at the Local position lfX,lfZ (relative to landscapes corner)

Definition at line 81 of file WTcLandscape.cpp.

21.36.3.3 `float cLandscape::GetVertexHeight (int liX, int liZ)`

Will return the height of the vertex at liX,liZ. (position is based on number of segments
NOT distance)

Definition at line 83 of file WTcLandscape.cpp.

21.36.3.4 `void cLandscape::Landscape (vmLandscape * lpLandscape)`

Set the current height map for this landscape object.

Definition at line 77 of file WTcLandscape.cpp.

21.36.3.5 `void cLandscape::Render () [virtual]`

virtual function to allow polymorphiss. see [cCamera::Render\(\)](#);

Implements [cRenderObject](#).

Definition at line 48 of file WTcLandscape.cpp.

21.36.3.6 `void cLandscape::RenderPainter (uint8 liLevel) [virtual]`

virtual function to allow polymorphism. see [cCamera::RenderPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 20 of file WTcLandscape.cpp.

21.36.3.7 `void cLandscape::RenderToPainter () [virtual]`

virtual function to allow polymorphism. see [cCamera::RenderToPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 32 of file WTcLandscape.cpp.

21.36.3.8 void cLandscape::Texture (vTexture * *lpTexture*)

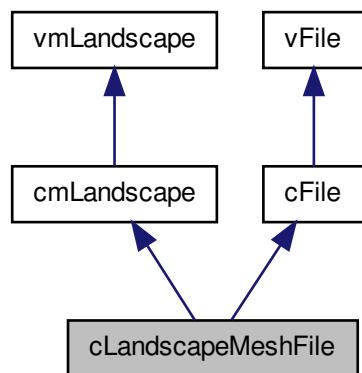
Set the texture bound to this landscape object.

Definition at line 75 of file WTcLandscape.cpp.

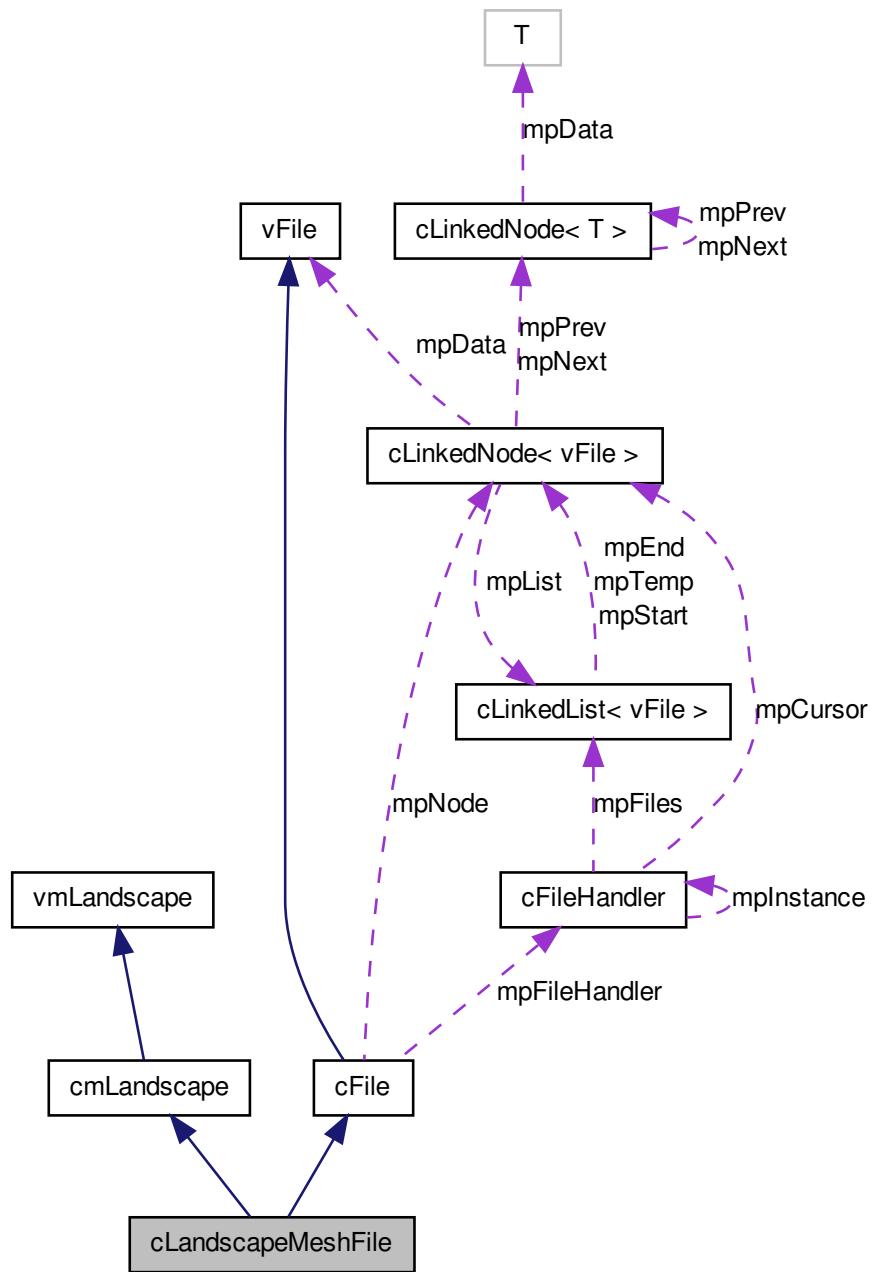
21.37 cLandscapeMeshFile Class Reference

This is the class which the Landscape File is stored in. This can be passed to any of [cmLandscape](#), [cLandscapeMeshIndividual](#) or [cLandscapeMeshRandom](#).

Inheritance diagram for cLandscapeMeshFile:



Collaboration diagram for cLandscapeMeshFile:



Public Member Functions

- [cLandscapeMeshFile \(cmLandscapeArray *lpArray\)](#)

Friends

- class [cmLandscape](#)

21.37.1 Detailed Description

This is the class which the Landscape File is stored in. This can be passed to any of [cmLandscape](#), [cLandscapeMeshIndividual](#) or [cLandscapeMeshRandom](#).

Definition at line 174 of file WTcmLandscape.h.

21.37.2 Constructor & Destructor Documentation

21.37.2.1 [cLandscapeMeshFile::cLandscapeMeshFile \(cmLandscapeArray * lpArray \)](#)

Definition at line 17 of file WTcmLandscape.cpp.

21.37.3 Friends And Related Function Documentation

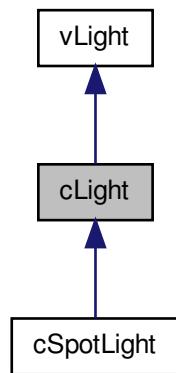
21.37.3.1 [friend class cmLandscape \[friend\]](#)

Definition at line 180 of file WTcmLandscape.h.

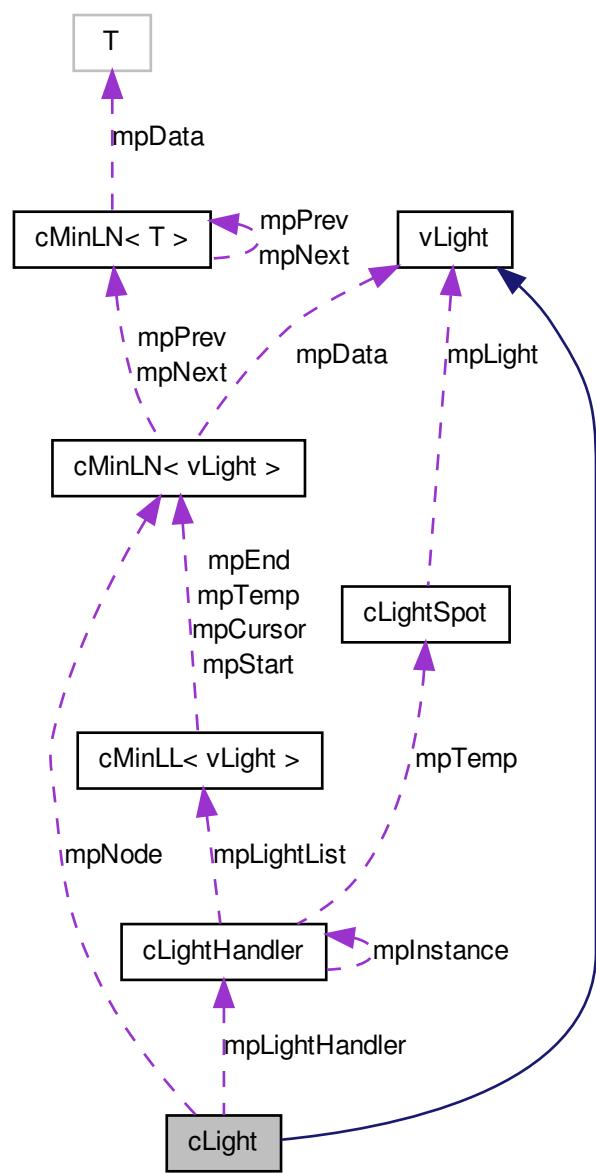
21.38 cLight Class Reference

Creates an OpenGL light effect.

Inheritance diagram for cLight:



Collaboration diagram for cLight:



Public Member Functions

- `cLight ()`
Constructor, will create a new light source.
- `~cLight ()`
- `void Position (float lfX, float lfY, float lfZ)`
Will Move the light to Global position lfX,lfY,lfZ.
- `float * Position ()`
Will Return the Current position of the light.
- `void Ambient (float lfR, float lfG, float lfB, float lfA)`
Will set the Ambient light color for the light source (RGBA).
- `void Diffuse (float lfR, float lfG, float lfB, float lfA)`
Will set the Diffuse light color for the light source (RGBA).
- `void Specular (float lfR, float lfG, float lfB, float lfA)`
Will set the Specular light color for the light source (RGBA).
- `virtual void PrepareLight ()`
Will bind this light source for use in OpenGL. Once all the Lights Variables are set, call this to preapre the light.
- `virtual void PrepareLight (uint32 liLight)`
- `void Attenuation (float liAttenuation)`
Will Set the value for the attenuation factor.
- `void AttenuationType (uint8 liOrder)`
Will Set the order of the attenuation (0=constant, 1=linear or 2=Quadratic).
- `void SetID (uint8 liLightID)`
Will Set the Lights ID (The OpenGL light number this light will render to by default).

Protected Attributes

- `uint8 miLightID`
- `float mpPosition [4]`
- `float mpAmbient [4]`
- `float mpDiffuse [4]`
- `float mpSpecular [4]`
- `float miAttenuation`
- `uint32 miAttenuationType`
- `cMinLN< vLight > * mpNode`

Static Protected Attributes

- static `cLightHandler * mpLightHandler = 0`

21.38.1 Detailed Description

Creates an OpenGL light effect.

Definition at line 4 of file WTcLight.h.

21.38.2 Constructor & Destructor Documentation

21.38.2.1 `cLight::cLight()`

Constructor, will create a new light source.

Definition at line 15 of file WTcLight.cpp.

21.38.2.2 `cLight::~cLight()`

Definition at line 3 of file WTcLight.cpp.

21.38.3 Member Function Documentation

21.38.3.1 `void cLight::Ambient(float lfR, float lfG, float lfB, float lfA)`

Will set the Ambient light color for the light source (RGBA).

Definition at line 38 of file WTcLight.cpp.

21.38.3.2 `void cLight::Attenuation(float liAttenuation)`

Will Set the value for the attenuation factor.

Definition at line 87 of file WTcLight.cpp.

21.38.3.3 `void cLight::AttenuationType(uint8 liOrder)`

Will Set the order of the attenuation (0=constant, 1=linear or 2=Quadratic).

Definition at line 82 of file WTcLight.cpp.

21.38.3.4 `void cLight::Diffuse(float lfR, float lfG, float lfB, float lfA)`

Will set the Diffuse light color for the light source (RGBA).

Definition at line 46 of file WTcLight.cpp.

21.38.3.5 float* cLight::Position() [inline, virtual]

Will Return the Current position of the light.

Implements [vLight](#).

Definition at line 37 of file WTcLight.h.

21.38.3.6 void cLight::Position(float lfX, float lfY, float lfZ)

Will Move the light to Global position lfX,lfY,lfZ.

Definition at line 29 of file WTcLight.cpp.

21.38.3.7 void cLight::PrepareLight(uint32 liLight) [virtual]

Implements [vLight](#).

Reimplemented in [cSpotLight](#).

Definition at line 72 of file WTcLight.cpp.

21.38.3.8 void cLight::PrepareLight() [virtual]

Will bind this light source for use in OpenGL. Once all the Lights Variables are set, call this to preapre the light.

Implements [vLight](#).

Reimplemented in [cSpotLight](#).

Definition at line 63 of file WTcLight.cpp.

21.38.3.9 void cLight::SetID(uint8 liLightID) [virtual]

Will Set the Lights ID (The OpenGL light number this light will render to by default).

Implements [vLight](#).

Definition at line 10 of file WTcLight.cpp.

21.38.3.10 void cLight::Specular(float lfR, float lfG, float lfB, float lfA)

Will set the Specular light color for the light source (RGBA).

Definition at line 54 of file WTcLight.cpp.

21.38.4 Member Data Documentation**21.38.4.1 float cLight::miAttenuation [protected]**

Definition at line 20 of file WTcLight.h.

21.38.4.2 uint32 cLight::miAttenuationType [protected]

Definition at line 22 of file WTcLight.h.

21.38.4.3 uint8 cLight::miLightID [protected]

Definition at line 8 of file WTcLight.h.

21.38.4.4 float cLight::mpAmbient[4] [protected]

Definition at line 13 of file WTcLight.h.

21.38.4.5 float cLight::mpDiffuse[4] [protected]

Definition at line 15 of file WTcLight.h.

21.38.4.6 cLightHandler * cLight::mpLightHandler = 0 [static, protected]

Definition at line 24 of file WTcLight.h.

21.38.4.7 cMinLN<vLight>* cLight::mpNode [protected]

Definition at line 25 of file WTcLight.h.

21.38.4.8 float cLight::mpPosition[4] [protected]

Definition at line 11 of file WTcLight.h.

21.38.4.9 float cLight::mpSpecular[4] [protected]

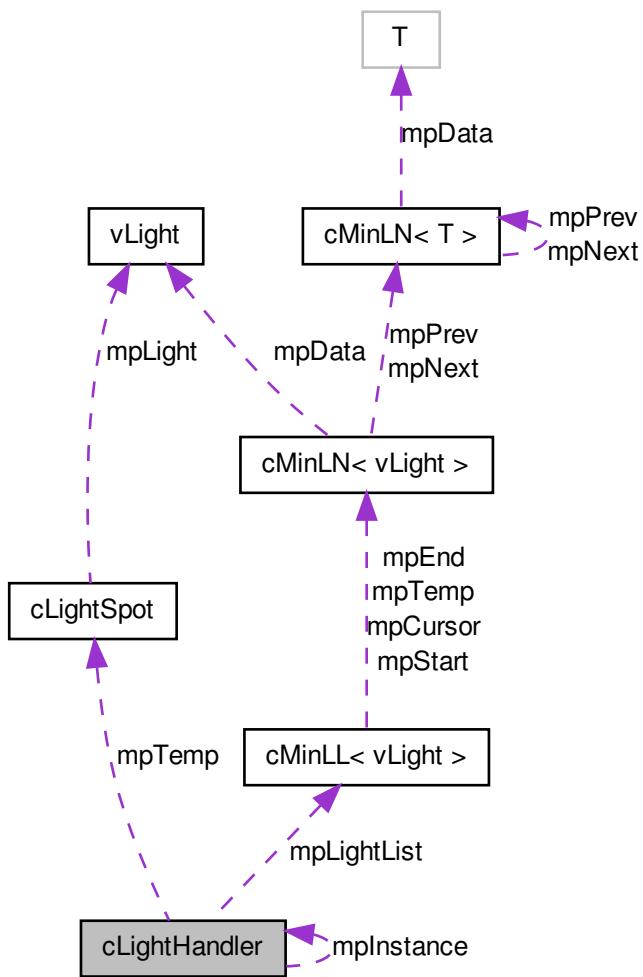
Definition at line 17 of file WTcLight.h.

21.39 cLightHandler Class Reference

[cLightHandler](#) will control the OpenGL Lights. It will turn off lights not required or possible for different renderings to increase speed and circumvent the OpenGL limit

of active lights. OpenGL has a limited number of lights that can be used at any one time. This handler identifies the lights which will have the greatest effect on the current object and prepares the optimal selection of lights for rendering the scene.

Collaboration diagram for cLightHandler:



Public Member Functions

- `~cLightHandler ()`
- `cMinLN< vLight > * Add (vLight *lpNew)`

- void [PrepareLight \(\)](#)
Will Prepare all the Lights for Rendering generally.
- void [PrepareLight \(cMatrix4 *mpObj\)](#)
Will Prepare the Lights for Rendering a specific Object.
- void [Remove \(cMinLN< vLight > *lpOld\)](#)
- void [DeleteAll \(\)](#)
Will Delete all the processes in the current Light list.

Static Public Member Functions

- static void [SetupLights \(\)](#)
- static [cLightHandler * Instance \(\)](#)
This function will return a pointer to the current [cLightHandler](#) Object.

21.39.1 Detailed Description

[cLightHandler](#) will control the OpenGL Lights. It will turn off lights not required or possible for different renderings to increase speed and circumvent the OpenGL limit of active lights. OpenGL has a limited number of lights that can be used at any one time. This handler identifies the lights which will have the greatest effect on the current object and prepares the optimal selection of lights for rendering the scene.

Definition at line 15 of file [WTcLightHandler.h](#).

21.39.2 Constructor & Destructor Documentation

21.39.2.1 [cLightHandler::~cLightHandler \(\)](#)

Definition at line 9 of file [WTcLightHandler.cpp](#).

21.39.3 Member Function Documentation

21.39.3.1 [cMinLN< vLight > * cLightHandler::Add \(vLight * lpNew \)](#)

Definition at line 36 of file [WTcLightHandler.cpp](#).

21.39.3.2 [void cLightHandler::DeleteAll \(\)](#)

Will Delete all the processes in the current Light list.

Definition at line 50 of file [WTcLightHandler.cpp](#).

21.39.3.3 cLightHandler * cLightHandler::Instance() [static]

This function will return a pointer to the current [cLightHandler](#) Object.

Definition at line 18 of file WTcLightHandler.cpp.

21.39.3.4 void cLightHandler::PrepareLight(cMatrix4 * mpObj)

Will Prepare the Lights for Rendering a specific Object.

Definition at line 72 of file WTcLightHandler.cpp.

21.39.3.5 void cLightHandler::PrepareLight()

Will Prepare all the Lights for Rendering generally.

Definition at line 61 of file WTcLightHandler.cpp.

21.39.3.6 void cLightHandler::Remove(cMinLN< vLight > * lpOld)

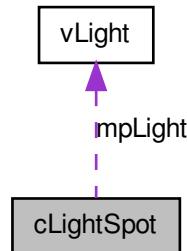
Definition at line 56 of file WTcLightHandler.cpp.

21.39.3.7 void cLightHandler::SetupLights() [static]

Definition at line 24 of file WTcLightHandler.cpp.

21.40 cLightSpot Class Reference

Collaboration diagram for cLightSpot:



Public Attributes

- `vLight * mpLight`
- double `mfDist`

21.40.1 Detailed Description

Definition at line 4 of file WTcLightHandler.h.

21.40.2 Member Data Documentation

21.40.2.1 double cLightSpot::mfDist

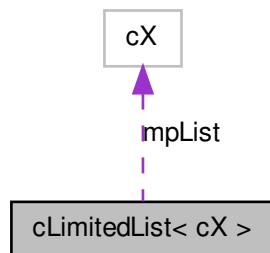
Definition at line 8 of file WTcLightHandler.h.

21.40.2.2 vLight* cLightSpot::mpLight

Definition at line 7 of file WTcLightHandler.h.

21.41 cLimitedList< cX > Class Template Reference

Collaboration diagram for cLimitedList< cX >:



Public Member Functions

- void `ChangeSize` (uint32 liSize)
- `cLimitedList` ()
- void `Init` (uint32 liSpaces)

- `cLimitedList` (`uint32 liSpaces`)
- `~cLimitedList` ()
- `cX & operator[]` (`uint32 liItem`)
- `cLimitedList< cX > & operator= (cLimitedList< cX > &lpOther)`
- `uint32 Items ()`
- `void SetItems (uint32 liItems)`
- `void Add (cX *lpTemp)`
- `void Remove (uint32 liPos)`
- `void SwitchItems (uint32 li1, uint32 li2)`

Public Attributes

- `cX * mpList`
- `uint32 miSpaces`
- `uint32 miItems`

21.41.1 Detailed Description

`template<class cX> class cLimitedList< cX >`

Definition at line 5 of file WTLimitedList.h.

21.41.2 Constructor & Destructor Documentation

21.41.2.1 `template<class cX> cLimitedList< cX >::cLimitedList() [inline]`

Definition at line 29 of file WTLimitedList.h.

21.41.2.2 `template<class cX> cLimitedList< cX >::cLimitedList(uint32 liSpaces) [inline]`

Definition at line 43 of file WTLimitedList.h.

21.41.2.3 `template<class cX> cLimitedList< cX >::~cLimitedList() [inline]`

Definition at line 48 of file WTLimitedList.h.

21.41.3 Member Function Documentation

21.41.3.1 `template<class cX> void cLimitedList< cX >::Add (cX * lpTemp) [inline]`

Definition at line 74 of file WTLimitedList.h.

**21.41.3.2 template<class cX> void cLimitedList< cX >::ChangeSize (uint32 *iSize*)
[inline]**

Definition at line 15 of file WTLimitedList.h.

**21.41.3.3 template<class cX> void cLimitedList< cX >::Init (uint32 *iSpaces*)
[inline]**

Definition at line 31 of file WTLimitedList.h.

21.41.3.4 template<class cX> uint32 cLimitedList< cX >::Items () [inline]

Definition at line 71 of file WTLimitedList.h.

21.41.3.5 template<class cX> cLimitedList<cX>& cLimitedList< cX >::operator= (cLimitedList< cX > & *lpOther*) [inline]

Definition at line 57 of file WTLimitedList.h.

**21.41.3.6 template<class cX> cX& cLimitedList< cX >::operator[] (uint32 *iItem*)
[inline]**

Definition at line 55 of file WTLimitedList.h.

**21.41.3.7 template<class cX> void cLimitedList< cX >::Remove (uint32 *iPos*)
[inline]**

Definition at line 76 of file WTLimitedList.h.

**21.41.3.8 template<class cX> void cLimitedList< cX >::SetItems (uint32 *iItems*)
[inline]**

Definition at line 72 of file WTLimitedList.h.

21.41.3.9 template<class cX> void cLimitedList< cX >::SwitchItems (uint32 *i1*, uint32 *i2*) [inline]

Definition at line 86 of file WTLimitedList.h.

21.41.4 Member Data Documentation

21.41.4.1 template<class cX> uint32 cLimitedList< cX >::miItems

Definition at line 13 of file WTLimitedList.h.

21.41.4.2 template<class cX> uint32 cLimitedList< cX >::miSpaces

Definition at line 12 of file WTLimitedList.h.

21.41.4.3 template<class cX> cX* cLimitedList< cX >::mpList

Definition at line 11 of file WTLimitedList.h.

21.42 cLimitedPointerList< cX > Class Template Reference

Public Member Functions

- [cLimitedPointerList](#) (uint32 liSpaces)
- [~cLimitedPointerList](#) ()
- [cX * operator\[\]](#) (uint32 liItem)
- [cX * Item](#) (uint32 liItem)
- void [Add](#) (cX *lpValue)
- void [Remove](#) (uint32 liPos)

21.42.1 Detailed Description

template<class cX> class cLimitedPointerList< cX >

Definition at line 108 of file WTLimitedList.h.

21.42.2 Constructor & Destructor Documentation

21.42.2.1 template<class cX > cLimitedPointerList< cX >::cLimitedPointerList (uint32 *liSpaces*) [inline]

Definition at line 114 of file WTLimitedList.h.

21.42.2.2 template<class cX > cLimitedPointerList< cX >::~cLimitedPointerList () [inline]

Definition at line 120 of file WTLimitedList.h.

21.42.3 Member Function Documentation

21.42.3.1 template<class cX > void cLimitedPointerList< cX >::Add (cX * *lpValue*) [inline]

Definition at line 130 of file WTLimitedList.h.

```
21.42.3.2 template<class cX> cX* cLimitedPointerList<cX>::Item( uint32 liItem )
[inline]
```

Definition at line 129 of file WTLimitedList.h.

```
21.42.3.3 template<class cX> cX* cLimitedPointerList<cX>::operator[]( uint32 liItem
) [inline]
```

Definition at line 128 of file WTLimitedList.h.

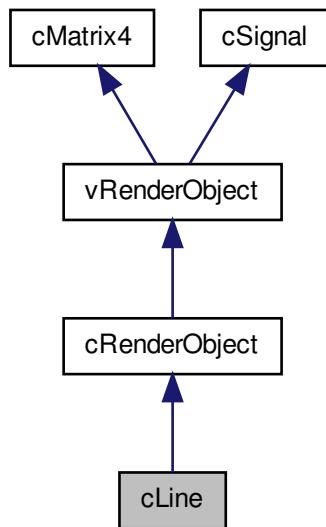
```
21.42.3.4 template<class cX> void cLimitedPointerList<cX>::Remove( uint32 liPos )
[inline]
```

Definition at line 131 of file WTLimitedList.h.

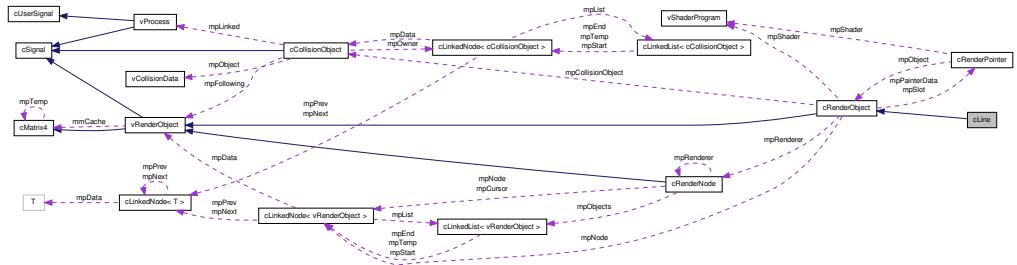
21.43 cLine Class Reference

A standard renderable Line object.

Inheritance diagram for cLine:



Collaboration diagram for cLine:



Public Member Functions

- **cLine ()**
cLine constructor
- **cLine (cRenderNode *lpRenderer)**
cLine constructor. Will be owned by lpRenderer.
- **void RenderPainter (uint8 liLevel)**
virtual function to allow polymorphism. see cCamera::RenderPainter();
- **void RenderToPainter ()**
virtual function to allow polymorphism. see cCamera::RenderToPainter();
- **void Render ()**
virtual function to allow polymorphiss. see cCamera::Render();
- **float * Color ()**
Will return a pointer to the color of this line (RGBA).
- **void Color (float *lfColor)**
Will Set the color of this object to the float array pointed to by lfColor. Expects 4 floats (RGBA).
- **void Color (float lfR, float lfG, float lfB, float lfA)**
Will Set the color of this object to the values specified (RGBA).
- **float * Position ()**
Will return a pointer to the position of this line object (XYZ).
- **void Position (float *lfPos)**
Will set the position of this object to the float array pointed to by lfPos. Expects 3 floats (XYZ).

- void **Position** (float lfX, float lfY, float lfZ)
Will set the position of this object to the values specified (XYZ).
- float * **Vector** ()
Will return a pointer to the vector of this line object (XYZ).
- void **Vector** (float *lfPos)
Will set the vector of this object to the float array pointed to by lfPos. Expects 3 floats (XYZ).
- void **Vector** (float lfX, float lfY, float lfZ)
Will set the vector of this object to the values specified (XYZ).

21.43.1 Detailed Description

A standard renderable Line object.

Definition at line 6 of file WTcLine.h.

21.43.2 Constructor & Destructor Documentation

21.43.2.1 **cLine::cLine()**

cLine constructor

Definition at line 4 of file WTcLine.cpp.

21.43.2.2 **cLine::cLine(cRenderNode * lpRenderer)**

cLine constructor. Will be owned by lpRenderer.

Definition at line 27 of file WTcLine.cpp.

21.43.3 Member Function Documentation

21.43.3.1 **float * cLine::Color()**

Will return a pointer to the color of this line (RGBA).

Definition at line 89 of file WTcLine.cpp.

21.43.3.2 **void cLine::Color(float * lfColor)**

Will Set the color of this object to the float array pointed to by lfColor. Expects 4 floats (RGBA).

Definition at line 84 of file WTcLine.cpp.

21.43.3.3 void cLine::Color (float lfR, float lfG, float lfB, float lfA)

Will Set the color of this object to the values specified (RGBA).

Definition at line 76 of file WTcLine.cpp.

21.43.3.4 void cLine::Position (float * lfPos)

Will set the position of this object to the float array pointed to by lfPos. Expects 3 floats (XYZ).

Reimplemented from [cMatrix4](#).

Definition at line 108 of file WTcLine.cpp.

21.43.3.5 void cLine::Position (float lfX, float lfY, float lfZ)

Will set the position of this object to the values specified (XYZ).

Reimplemented from [cMatrix4](#).

Definition at line 110 of file WTcLine.cpp.

21.43.3.6 float * cLine::Position ()

Will return a pointer to the position of this line object (XYZ).

Reimplemented from [cMatrix4](#).

Definition at line 103 of file WTcLine.cpp.

21.43.3.7 void cLine::Render () [virtual]

virtual function to allow polymorphiss. see [cCamera::Render\(\)](#);

Implements [cRenderObject](#).

Definition at line 61 of file WTcLine.cpp.

21.43.3.8 void cLine::RenderPainter (uint8 liLevel) [virtual]

virtual function to allow polymorphism. see [cCamera::RenderPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 33 of file WTcLine.cpp.

21.43.3.9 void cLine::RenderToPainter() [virtual]

virtual function to allow polymorphism. see cCamera::RenderToPainter();

Implements [cRenderObject](#).

Definition at line 45 of file WTcLine.cpp.

21.43.3.10 void cLine::Vector(float * lfPos)

Will set the vector of this object to the float array pointed to by lfPos. Expects 3 floats (XYZ).

Definition at line 122 of file WTcLine.cpp.

21.43.3.11 float * cLine::Vector()

Will return a pointer to the vector of this line object (XYZ).

Definition at line 117 of file WTcLine.cpp.

21.43.3.12 void cLine::Vector(float lfX, float lfY, float lfZ)

Will set the vector of this object to the values specified (XYZ).

Definition at line 130 of file WTcLine.cpp.

21.44 cLinkedList< T > Class Template Reference

This is the control for a linked list. It controls a linked list of cLinkedNodes which each point to their item in the list. Each [cLinkedNode](#) points to the [cLinkedNode](#)'s either side of themselves and the data they own. [cLinkedList](#) is templated and so can be a linked list of any type.

Public Member Functions

- void [StitchOut](#) ([cLinkedNode< T >](#) *lpNode)
- void [StitchIn](#) ([cLinkedNode< T >](#) *lpNode)
- void [StitchIn](#) ([cLinkedNode< T >](#) *lpNode, [cLinkedNode< T >](#) *lpPos)
- [cLinkedList](#) (T *lpData)

This holds a pointer of the relevant type for the list and will create the list by making the item lpData the first item in the list.

- [cLinkedList](#) ()
- [~cLinkedList](#) ()

This will delete the list and delete every item in the list.

- `cLinkedNode< T > * Start ()`
- `cLinkedNode< T > * End ()`
- `cLinkedNode< T > * Find (T *lpData)`

This is the number of items in the list.

- `cLinkedNode< T > * Insert (T *lpData)`

This will create a `cLinkedNode` give it lpData and add that node to the end of the list.

- void `Delete (cLinkedNode< T > *lpOld)`

This will delete the `cLinkedNode` pointed to by lpOld and remove it from the list including mpData.

- void `Initialise ()`

- void `Move (cLinkedNode< T > *lpFrom, cLinkedNode< T > *lpPosition)`

This will Move the node lpFrom to be before lpPosition.

- void `Move (cLinkedNode< T > *lpFrom, cLinkedList< T > *lpPosition)`

- void `ClearAll ()`

- void `DeleteAll ()`

- void `Display ()`

- void `Remove (cLinkedNode< T > *lpNode)`

This will delete the `cLinkedNode` pointed to by lpOld and remove it from the list, but not delete mpData.

Static Public Attributes

- static `cLinkedNode< T > * mpTemp = 0`

This is a pointer that can be used as a cursor by this linked list.

21.44.1 Detailed Description

`template<class T> class cLinkedList< T >`

This is the control for a linked list. It controls a linked list of cLinkedNodes which each point to their item in the list. Each `cLinkedNode` points to the `cLinkedNode`s either side of themselves and the data they own. `cLinkedList` is templated and so can be a linked list of any type.

Definition at line 50 of file WTLLTemplate.h.

21.44.2 Constructor & Destructor Documentation

21.44.2.1 template<class T> cLinkedList< T >::cLinkedList(T * *lpData*)

This holds a pointer of the relevant type for the list and will create the list by making the item lpData the first item in the list.

Definition at line 276 of file WTLLTemplate.h.

21.44.2.2 template<class T> cLinkedList< T >::cLinkedList()

Definition at line 183 of file WTLLTemplate.h.

21.44.2.3 template<class T> cLinkedList< T >::~cLinkedList() [inline]

This will delete the list and delete every item in the list.

Definition at line 68 of file WTLLTemplate.h.

21.44.3 Member Function Documentation

21.44.3.1 template<class T> void cLinkedList< T >::ClearAll()

Definition at line 189 of file WTLLTemplate.h.

21.44.3.2 template<class T> void cLinkedList< T >::Delete(cLinkedNode< T > * *lpOld*)

This will delete the [cLinkedNode](#) pointed to by lpOld and remove it from the list including mpData.

Definition at line 226 of file WTLLTemplate.h.

21.44.3.3 template<class T> void cLinkedList< T >::DeleteAll()

Definition at line 116 of file WTLLTemplate.h.

21.44.3.4 template<class T> void cLinkedList< T >::Display()

Definition at line 300 of file WTLLTemplate.h.

21.44.3.5 template<class T> cLinkedNode< T >* cLinkedList< T >::End() [inline]

Definition at line 75 of file WTLLTemplate.h.

21.44.3.6 template<class T> cLinkedNode<T> * cLinkedList<T>::Find (T * lpData)

This is the number of items in the list.

This will return the **cLinkedNode** which owns the item lpData in this list.

Definition at line 219 of file WTLLTemplate.h.

21.44.3.7 template<class T> void cLinkedList<T>::Initialise ()

Definition at line 107 of file WTLLTemplate.h.

21.44.3.8 template<class T> cLinkedNode<T> * cLinkedList<T>::Insert (T * lpData)

This will create a **cLinkedNode** give it lpData and add that node to the end of the list.

Definition at line 289 of file WTLLTemplate.h.

21.44.3.9 template<class T> void cLinkedList<T>::Move (cLinkedNode<T> * lpFrom, cLinkedNode<T> * lpPosition)

This will Move the node lpFrom to be before lpPosition.

21.44.3.10 template<class T> void cLinkedList<T>::Move (cLinkedNode<T> * lpFrom, cLinkedList<T> * lpPosition)

Definition at line 208 of file WTLLTemplate.h.

21.44.3.11 template<class T> void cLinkedList<T>::Remove (cLinkedNode<T> * lpNode)

This will delete the **cLinkedNode** pointed to by lpOld and remove it from the list, but not delete mpData.

Definition at line 155 of file WTLLTemplate.h.

21.44.3.12 template<class T> cLinkedNode<T> * cLinkedList<T>::Start () [inline]

Definition at line 73 of file WTLLTemplate.h.

21.44.3.13 template<class T> void cLinkedList<T>::StitchIn (cLinkedNode<T> * lpNode)

Definition at line 235 of file WTLLTemplate.h.

21.44.3.14 template<class T> void cLinkedList< T >::StitchIn (cLinkedNode< T > * *lpNode*, cLinkedNode< T > * *lpPos*)

Definition at line 255 of file WTLLTemplate.h.

21.44.3.15 template<class T> void cLinkedList< T >::StitchOut (cLinkedNode< T > * *lpNode*)

Definition at line 163 of file WTLLTemplate.h.

21.44.4 Member Data Documentation

21.44.4.1 template<class T> cLinkedNode< T > * cLinkedList< T >::mpTemp = 0 [static]

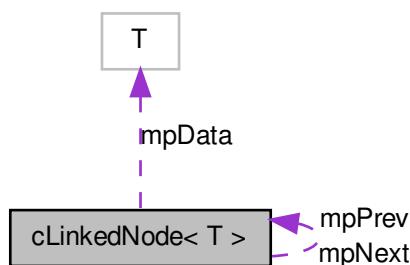
This is a pointer that can be used as a cursor by this linked list.

Definition at line 75 of file WTLLTemplate.h.

21.45 cLinkedNode< T > Class Template Reference

This is a node class to allow templating of the [cLinkedList](#) class. This node will store pointers to the nodes either side of this node in the linked list and a pointer to the object his node owns.

Collaboration diagram for cLinkedNode< T >:



Public Member Functions

- [cLinkedNode< T > * Next \(\)](#)

This will return a pointer to the next node in the linked list. see [cLinkedList](#).

- [cLinkedNode< T >::Previous\(\)](#)

This will return a pointer to the previous node in the linked list. see [cLinkedList](#).

Public Attributes

- [T * mpData](#)

This is a pointer to the data owned by this node.

Friends

- class [cLinkedList< T >](#)

21.45.1 Detailed Description

template<class T> class cLinkedNode< T >

This is a node class to allow templating of the [cLinkedList](#) class. This node will store pointers to the nodes either side of this node in the linked list and a pointer to the object his node owns.

Definition at line 14 of file WTLLTemplate.h.

21.45.2 Member Function Documentation

21.45.2.1 template<class T> cLinkedNode<T>* cLinkedNode< T >::Next() [inline]

This will return a pointer to the next node in the linked list. see [cLinkedList](#).

Definition at line 39 of file WTLLTemplate.h.

21.45.2.2 template<class T> cLinkedNode<T>* cLinkedNode< T >::Previous() [inline]

This will return a pointer to the previous node in the linked list. see [cLinkedList](#).

Definition at line 41 of file WTLLTemplate.h.

21.45.3 Friends And Related Function Documentation

21.45.3.1 template<class T> friend class cLinkedList< T > [friend]

Definition at line 16 of file WTLLTemplate.h.

21.45.4 Member Data Documentation

21.45.4.1 template<class T> T* cLinkedNode< T >::mpData

This is a pointer to the data owned by this node.

Definition at line 33 of file WTLLTemplate.h.

21.46 cMainThread< cX, cS > Class Template Reference

Public Member Functions

- `~cMainThread ()`
- `~cMainThread ()`

Static Public Member Functions

- static uint32 `Start (HINSTANCE hInstance)`
- static uint32 `Start ()`

21.46.1 Detailed Description

`template<class cX, class cS> class cMainThread< cX, cS >`

Definition at line 8 of file WTcBase.h.

21.46.2 Constructor & Destructor Documentation

21.46.2.1 template<class cX , class cS > cMainThread< cX, cS >::~cMainThread() [inline]

Definition at line 12 of file WTcBase.h.

21.46.2.2 template<class cX , class cS > cMainThread< cX, cS >::~cMainThread() [inline]

Definition at line 51 of file WTcBase.h.

21.46.3 Member Function Documentation

21.46.3.1 template<class cX , class cS > static uint32 cMainThread< cX, cS >::Start (HINSTANCE *hInstance*) [inline, static]

Definition at line 13 of file WTcBase.h.

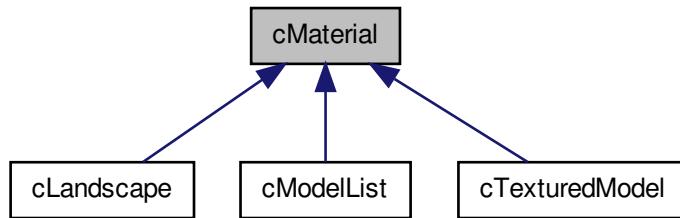
21.46.3.2 template<class cX , class cS > static uint32 cMainThread< cX,cS >::Start() [inline, static]

Definition at line 52 of file WTcBase.h.

21.47 cMaterial Class Reference

A class to store material data for an object. Defines the 'reflectiveness' of the surface.

Inheritance diagram for cMaterial:



Public Member Functions

- [cMaterial \(\)](#)
Will create a new material object.
- void [SetSpecular](#) (float lfRed, float lfGreen, float lfBlue, float lfAlpha)
Will set the RGBA color of Specular reflections of this material (RGBA).
- void [SetShine](#) (float lfShine)
Will set the shininess of this material (0.0f - 1.0f).
- void [PrepareMaterial \(\)](#)
Will bind the material to OpenGL ready to be used. Once all the material variables are set, call this to prepare the material.

21.47.1 Detailed Description

A class to store material data for an object. Defines the 'reflectiveness' of the surface.

Definition at line 5 of file WTcMaterial.h.

21.47.2 Constructor & Destructor Documentation

21.47.2.1 `cMaterial::cMaterial()`

Will create a new material object.

Definition at line 4 of file WTcMaterial.cpp.

21.47.3 Member Function Documentation

21.47.3.1 `void cMaterial::PrepareMaterial()`

Will bind the material to OpenGL ready to be used. Once all the material variables are set, call this to prepare the material.

Definition at line 26 of file WTcMaterial.cpp.

21.47.3.2 `void cMaterial::SetShine(float IfShine)`

Will set the shininess of this material (0.0f - 1.0f).

Definition at line 21 of file WTcMaterial.cpp.

21.47.3.3 `void cMaterial::SetSpecular(float IfRed, float IfGreen, float IfBlue, float IfAlpha)`

Will set the RGBA color of Specular reflections of this material (RGBA).

Definition at line 13 of file WTcMaterial.cpp.

21.48 cMatrix Class Reference

Public Member Functions

- `cMatrix(int liRow, int liColumn)`
- `void operator=(cMatrix &lVal)`
- `void operator=(float &lVal)`
- `void operator=(const float lVal)`
- `cMatrix operator+(cMatrix &lVal)`
- `cMatrix operator+(float &lVal)`
- `cMatrix operator+(const float lVal)`
- `cMatrix operator-(cMatrix &lVal)`
- `cMatrix operator-(float &lVal)`
- `cMatrix operator-(const float lVal)`
- `cMatrix operator*(cMatrix &lVal)`
- `cMatrix operator*(float &lVal)`
- `cMatrix operator*(const float lVal)`
- `cMatrix operator/(cMatrix &lVal)`

- `cMatrix operator/ (float &lVal)`
- `cMatrix operator/ (const float lVal)`
- `bool Equivalent (cMatrix &lVal)`
- `cMatrix Determinant ()`
- `cMatrix Transpose ()`

21.48.1 Detailed Description

Definition at line 4 of file WTMatrixTemplate.h.

21.48.2 Constructor & Destructor Documentation

21.48.2.1 `cMatrix::cMatrix (int liRow, int liColumn)`

Definition at line 277 of file WTMatrixTemplate.h.

21.48.3 Member Function Documentation

21.48.3.1 `cMatrix cMatrix::Determinant ()`

Definition at line 54 of file WTMatrixTemplate.h.

21.48.3.2 `bool cMatrix::Equivalent (cMatrix & lVal) [inline]`

Definition at line 32 of file WTMatrixTemplate.h.

21.48.3.3 `cMatrix cMatrix::operator* (cMatrix & lVal)`

21.48.3.4 `cMatrix cMatrix::operator* (float & lVal)`

21.48.3.5 `cMatrix cMatrix::operator* (const float lVal)`

21.48.3.6 `cMatrix cMatrix::operator+ (cMatrix & lVal)`

Definition at line 88 of file WTMatrixTemplate.h.

21.48.3.7 `cMatrix cMatrix::operator+ (float & lVal)`

Definition at line 103 of file WTMatrixTemplate.h.

21.48.3.8 `cMatrix cMatrix::operator+ (const float lVal)`

Definition at line 114 of file WTMatrixTemplate.h.

21.48.3.9 cMatrix cMatrix::operator- (float & *IVal*)

Definition at line 140 of file WTMatrixTemplate.h.

21.48.3.10 cMatrix cMatrix::operator- (const float *IVal*)

Definition at line 151 of file WTMatrixTemplate.h.

21.48.3.11 cMatrix cMatrix::operator- (cMatrix & *IVal*)

Definition at line 125 of file WTMatrixTemplate.h.

21.48.3.12 cMatrix cMatrix::operator/ (cMatrix & *IVal*)**21.48.3.13 cMatrix cMatrix::operator/ (float & *IVal*)****21.48.3.14 cMatrix cMatrix::operator/ (const float *IVal*)****21.48.3.15 void cMatrix::operator= (float & *IVal*)**

Definition at line 70 of file WTMatrixTemplate.h.

21.48.3.16 void cMatrix::operator= (cMatrix & *IVal*)

Definition at line 59 of file WTMatrixTemplate.h.

21.48.3.17 void cMatrix::operator= (const float *IVal*)

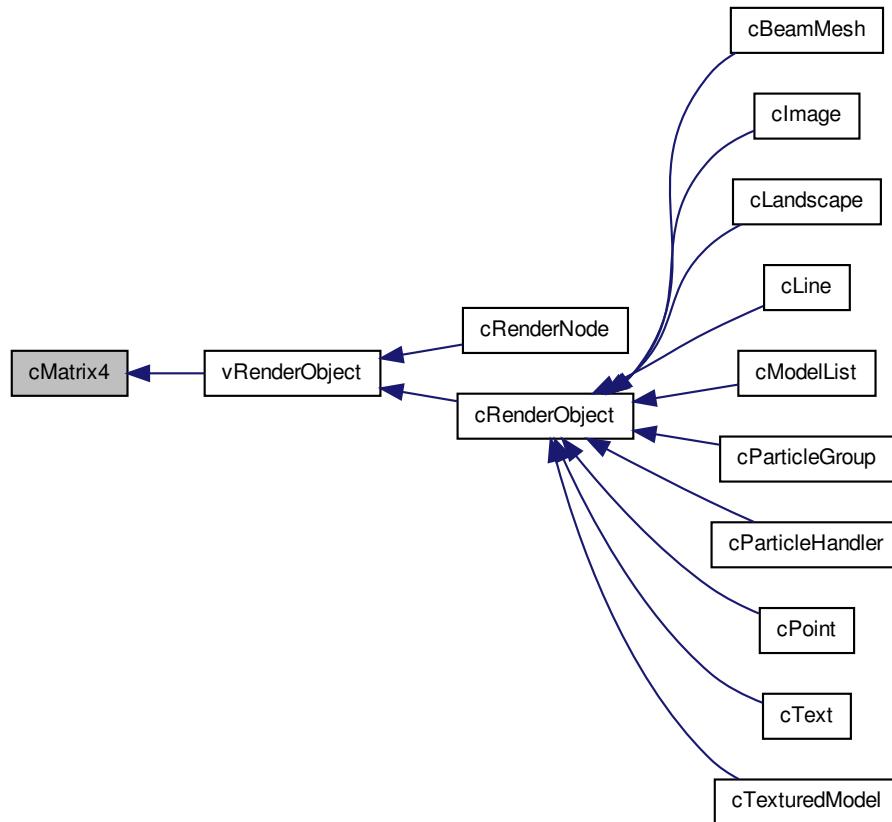
Definition at line 79 of file WTMatrixTemplate.h.

21.48.3.18 cMatrix cMatrix::Transpose ()

Definition at line 39 of file WTMatrixTemplate.h.

21.49 cMatrix4 Class Reference

Inheritance diagram for cMatrix4:



Collaboration diagram for cMatrix4:



Public Member Functions

- void [UpdateMatrix \(\)](#)
This will update the OpenGL matrix on the top of matrix stack to be identical to this matrix.
- void [UpdateMatrix \(float *lpMatrix\)](#)
This appears to multiply this matrix by the matrix lpMatrix and equate this matrix to the result.
- [cMatrix4 \(\)](#)
This will create a [cMatrix4](#) object and will initialise all the static data required.
- float [Determinant \(\)](#)
This will return the determinant of this objects matrix.
- [cMatrix4 InvertRotationMatrix \(\)](#)
*THis will find the inverse of the matrix if it is a standard rotation / translation matrix.
(0,1,2,4,5,6,8,9,10 are orthogonal. 3,7,11 are zero. 12,13,14 are position and 15 is one.)*
- [cMatrix4 Transpose \(\)](#)
This will return the transpose of this objects matrix.
- float * [Matrix \(\)](#)
This will return a pointer to this objects matrix data.
- float * [Matrix \(uint8 lcData\)](#)
This will return a pointer to the float numbered lcData in this objects matrix.
- float * [Position \(\)](#)
This will reutrn a pointer to this objects matrices position vector.
- float [X \(\)](#)
This will return this objects X position value.
- float [Y \(\)](#)
This will return this objects Y position value.
- float [Z \(\)](#)
This will return this objects Z position value.
- float * [XVect \(\)](#)
This will return the vector along this objects local X axis.
- float * [YVect \(\)](#)
This will return the vector along this objects local Y axis.

- `float * ZVect ()`
This will return the vector along this objects local Z axis.
- `void Identity ()`
This will restore this objects matrix to an identity matrix.
- `void Zero ()`
This will set every float in this objects matrix to be equal to zero.
- `cMatrix4 operator* (float &lVal)`
This will multiply every float in this objects matrix by lVal.
- `cMatrix4 operator* (const float lVal)`
This will multiply every float in this objects matrix by lVal.
- `cMatrix4 * operator* (cMatrix4 *lVal)`
This will multiply this objects matrix by the matrix lVal. Returns This.lVal .
- `cMatrix4 operator* (cMatrix4 lVal)`
This will multiply this objects matrix by the matrix lVal. Returns This.lVal .
- `cMatrix4 operator/ (float &lVal)`
This will divide every float in this objects matrix by lVal.
- `cMatrix4 operator/ (const float lVal)`
This will divide every float in this objects matrix by lVal.
- `cMatrix4 operator+ (float &lVal)`
This will add lVal to every float in this objects matrix.
- `cMatrix4 operator+ (const float lVal)`
This will add lVal to every float in this objects matrix.
- `cMatrix4 operator+ (cMatrix4 &lVal)`
This will add this matrix and the matrix lVal.
- `cMatrix4 operator- (float &lVal)`
This will deduct lVal from every float in this objects matrix.
- `cMatrix4 operator- (const float lVal)`
This will deduct lVal from every float in this objects matrix.
- `cMatrix4 operator- (cMatrix4 &lVal)`
This will subtract the matrix lVal from this matrix.

- float & **operator[]** (uint16 liPos)
This will return the float in position liPos in mpData.
- float & **operator()** (uint16 liColumn, uint16 liRow)
This will return the float in position [liColumn,liRow] in mpData.
- float **operator=** (float &lVal)
This will equate every float in mpData to lVal.
- float **operator=** (const float lVal)
This will equate every float in mpData to lVal.
- **cMatrix4 * operator= (cMatrix4 *lVal)**
This will equate the data in mpData to the data in lVal.mpData.
- **cMatrix4 operator= (cMatrix4 lVal)**
This will equate the data in mpData to the data in lVal.mpData.
- float * **operator=** (float *lVal)
- void **Position (c2DVF *lpPosition)**
This will position the current object to the 2D Vector lpPosition. (X,Y)
- void **Position (float lfX, float lfY)**
This will position the current object to lfX,lfY. (X,Y)
- void **Position (c3DVF *lpPosition)**
This will position the current object to the 3D Vector lpPosition. (X,Y,Z)
- void **Position (float lfX, float lfY, float lfZ)**
This will position the current object to lfX,lfY,lfZ. (X,Y,Z)
- void **Position (float *lpPos)**
*This will take 2 or three floats depending if this **cMatrix4** is set to 2D or 3D operations and set the position of the object.*
- void **PositionX (float lfX)**
This will set the objects X position to be equal to lfX.
- void **PositionY (float lfY)**
This will set the objects Y position to be equal to lfX.
- void **PositionZ (float lfZ)**
This will set the objects Z position to be equal to lfX.
- void **AdvanceX (float lfDistance)**
This will advance the object along its local X axis by lfDistance.

- void **AdvanceY** (float lfDistance)
This will advance the object along its local Y axis by lfDistance.
- void **AdvanceZ** (float lfDistance)
This will advance the object along its local Z axis by lfDistance.
- void **Advance** (float lfX, float lfY, float lfZ)
This will advance the object along its local X, Y and Z axis by lfX, lfY and lfZ.
- void **GAdvanceX** (float lfX)
This will advance the object along the global X axis by lfX.
- void **GAdvanceY** (float lfX)
This will advance the object along the global Y axis by lfX.
- void **GAdvanceZ** (float lfX)
This will advance the object along the global Z axis by lfX.
- void **GAdvance** (float lfX, float lfY, float lfZ)
This will advance the object along the global X, Y and Z axis by lfX, lfY and lfZ.
- void **LRotate** (float lfAngle)
This will rotate the object around its local Z axis by lfAngle radians. Suitable for 2D objects.
- void **LRotateX** (float lfAngle)
This will rotate the object around its local X axis by lfAngle radians.
- void **LRotateY** (float lfAngle)
This will rotate the object around its local Y axis by lfAngle radians.
- void **LRotateZ** (float lfAngle)
This will rotate the object around its local Z axis by lfAngle radians.
- void **GRotateX** (float lfAngle)
This will rotate the object around its global X axis by lfAngle radians.
- void **GRotateY** (float lfAngle)
This will rotate the object around its global X axis by lfAngle radians.
- void **GRotateZ** (float lfAngle)
This will rotate the object around its global X axis by lfAngle radians.
- void **GRotateX** (float lfAngle, float lfX, float lfY, float lfZ)
This will rotate the object around the global X axis at point lfX,lfY,lfZ by lfAngle radians.

- void **GRotateY** (float lfAngle, float lfX, float lfY, float lfZ)
This will rotate the object around the global Y axis at point lfX,lfY,lfZ by lfAngle radians.
- void **GRotateZ** (float lfAngle, float lfX, float lfY, float lfZ)
This will rotate the object around the global Z axis at point lfX,lfY,lfZ by lfAngle radians.
- void **GRotateOriginX** (float lfAngle)
This will rotate the object around the global X axis at point 0,0,0 by lfAngle radians.
- void **GRotateOriginY** (float lfAngle)
This will rotate the object around the global Y axis at point 0,0,0 by lfAngle radians.
- void **GRotateOriginZ** (float lfAngle)
This will rotate the object around the global Z axis at point 0,0,0 by lfAngle radians.
- void **Resize** (float lfScale)
This will scale the object by a factor of lfScale.
- void **LResizeX** (float lfScale)
This will scale the object along its local X axis by a factor of lfScale.
- void **LResizeY** (float lfScale)
This will scale the object along its local Y axis by a factor of lfScale.
- void **LResizeZ** (float lfScale)
This will scale the object along its local Z axis by a factor of lfScale.
- void **GResizeX** (float lfScale)
This will scale the object along its global X axis by a factor of lfScale.
- void **GResizeY** (float lfScale)
This will scale the object along its global Y axis by a factor of lfScale.
- void **GResizeZ** (float lfScale)
This will scale the object along its global Z axis by a factor of lfScale.
- float **Distance** (cMatrix4 *lpOther)
This will return the distance between this matrix and the cMatrix4 pointed to by lpOther.
- float **Distance** (float *lpOther)
- double **DistanceSq** (cMatrix4 *lpOther)
This will return the square of the distance between this matrix and the cMatrix4 pointed to by lpOther.

- double **DistanceSq** (**cMatrix4** lpOther)
- double **DistanceSq** (float *lpOther)
- void **Advance** (float lfX, float lfY)

*This will move this matrix along its local X and Y axis by lfX and lfY respectively.
Suitable for 2D objects.*

- void **GAdvance** (float lfX, float lfY)

This will move this matrix along its global X and Y axis by lfX and lfY respectively.

- void **Angle** (float lfAngle)

This will set the angle of rotation about this matrices X axis to lfAngle radians. Suitable for 2D objects.

- void **Rotate** (float lfAngle)

This will rotate this matrix about its X axis by lfAngle radians. Suitable for 2D objects.

- void **GRotateOrigin** (float lfAngle)

This will rotate this matrix in the X axis through 0,0 by lfAngle radians. Suitable for 2D objects.

- void **GRotate** (float lfAngle, float lfX, float lfY)

This will rotate this matrix in the X axis through lfX,lfY by lfAngle radians. Suitable for 2D objects.

- void **Set2D** ()

This will set the current matrix to operate as if it is a 2D object.

- void **Set3D** ()

This will set the current matrix to operate as if it is a 3D object.

- void **Display** ()

This will display the current matrix Textually.

- **cMatrix4** * **Equals** (**cMatrix4** *lpOther)

Static Public Attributes

- static **cMatrix4** mpTemp

*This is a static **cMatrix4** used to store data for operands.*

Protected Attributes

- float **mpData** [16]

This stores a 4x4 matrix of floats representing the current translation of the objet.

- bool [mb3D](#)

This si a boolean flag to define whether the object is 3D or 2D. True is 3D. False is 2D.

Friends

- class [cCameraMatrix4](#)

21.49.1 Detailed Description

Definition at line 12 of file WTcMatrix4.h.

21.49.2 Constructor & Destructor Documentation

21.49.2.1 [cMatrix4::cMatrix4\(\)](#)

This will create a [cMatrix4](#) object and will initialise all the static data required.

Definition at line 7 of file WTcMatrix4.cpp.

21.49.3 Member Function Documentation

21.49.3.1 [void cMatrix4::Advance\(float lfX, float lfY, float lfZ \)](#)

This will advance the object along its local X, Y and Z axis by lfX, lfY and lfZ.

Definition at line 756 of file WTcMatrix4.cpp.

21.49.3.2 [void cMatrix4::Advance\(float lfX, float lfY \)](#)

This will move this matrix along its local X and Y axis by lfX and lfY respectively.
Suitable for 2D objects.

Definition at line 1155 of file WTcMatrix4.cpp.

21.49.3.3 [void cMatrix4::AdvanceX\(float lfDistance \)](#)

This will advance the object along its local X axis by lfDistance.

Definition at line 735 of file WTcMatrix4.cpp.

21.49.3.4 [void cMatrix4::AdvanceY\(float lfDistance \)](#)

This will advance the object along its local Y axis by lfDistance.

Definition at line 742 of file WTcMatrix4.cpp.

21.49.3.5 void cMatrix4::AdvanceZ (float lfDistance)

This will advance the object along its local Z axis by lfDistance.

Definition at line 749 of file WTcMatrix4.cpp.

21.49.3.6 void cMatrix4::Angle (float lfAngle)

This will set the angle of rotation about this matrices X axis to lfAngle radians. Suitable for 2D objects.

Definition at line 1167 of file WTcMatrix4.cpp.

21.49.3.7 float cMatrix4::Determinant ()

This will return the determinant of this objects matrix.

Definition at line 390 of file WTcMatrix4.cpp.

21.49.3.8 void cMatrix4::Display ()

This will display the current matrix Textually.

Definition at line 1242 of file WTcMatrix4.cpp.

21.49.3.9 float cMatrix4::Distance (cMatrix4 * lpOther)

This will return the distance between this matrix and the [cMatrix4](#) pointed to by lpOther.

Definition at line 1145 of file WTcMatrix4.cpp.

21.49.3.10 float cMatrix4::Distance (float * lpOther)

Definition at line 1134 of file WTcMatrix4.cpp.

21.49.3.11 double cMatrix4::DistanceSq (cMatrix4 * lpOther)

This will return the square of the distance between this matrix and the [cMatrix4](#) pointed to by lpOther.

Definition at line 1112 of file WTcMatrix4.cpp.

21.49.3.12 double cMatrix4::DistanceSq (cMatrix4 lpOther)

Definition at line 1100 of file WTcMatrix4.cpp.

21.49.3.13 double cMatrix4::DistanceSq (float * *lpOther*)

Definition at line 1123 of file WTcMatrix4.cpp.

21.49.3.14 cMatrix4* cMatrix4::Equals (cMatrix4 * *lpOther*) [inline]

Definition at line 241 of file WTcMatrix4.h.

21.49.3.15 void cMatrix4::GAdvance (float *lfX*, float *lfY*)

This will move this matrix along its global X and Y axis by lfX and lfY respectively.

Definition at line 1161 of file WTcMatrix4.cpp.

21.49.3.16 void cMatrix4::GAdvance (float *lfX*, float *lfY*, float *lfZ*)

This will advance the object along the global X, Y and Z axis by lfX, lfY and lfZ.

Definition at line 772 of file WTcMatrix4.cpp.

21.49.3.17 void cMatrix4::GAdvanceX (float *lfX*)

This will advance the object along the global X axis by lfX.

Definition at line 763 of file WTcMatrix4.cpp.

21.49.3.18 void cMatrix4::GAdvanceY (float *lfX*)

This will advance the object along the global Y axis by lfX.

Definition at line 766 of file WTcMatrix4.cpp.

21.49.3.19 void cMatrix4::GAdvanceZ (float *lfX*)

This will advance the object along the global Z axis by lfX.

Definition at line 769 of file WTcMatrix4.cpp.

21.49.3.20 void cMatrix4::GResizeX (float *lfScale*)

This will scale the object along its global X axis by a factor of lfScale.

Definition at line 891 of file WTcMatrix4.cpp.

21.49.3.21 void cMatrix4::GResizeY (float *lfScale*)

This will scale the object along its global Y axis by a factor of lfScale.

Definition at line 898 of file WTcMatrix4.cpp.

21.49.3.22 void cMatrix4::GResizeZ (float lfScale)

This will scale the object along its global Z axis by a factor of lfScale.

Definition at line 906 of file WTcMatrix4.cpp.

21.49.3.23 void cMatrix4::GRotate (float lfAngle, float lfX, float lfY)

This will rotate this matrix in the X axis through lfX,lfY by lfAngle radians. Suitable for 2D objects.

Definition at line 1210 of file WTcMatrix4.cpp.

21.49.3.24 void cMatrix4::GRotateOrigin (float lfAngle)

This will rotate this matrix in the X axis through 0,0 by lfAngle radians. Suitable for 2D objects.

Definition at line 1191 of file WTcMatrix4.cpp.

21.49.3.25 void cMatrix4::GRotateOriginX (float lfAngle)

This will rotate the object around the global X axis at point 0,0,0 by lfAngle radians.

Definition at line 915 of file WTcMatrix4.cpp.

21.49.3.26 void cMatrix4::GRotateOriginY (float lfAngle)

This will rotate the object around the global Y axis at point 0,0,0 by lfAngle radians.

Definition at line 944 of file WTcMatrix4.cpp.

21.49.3.27 void cMatrix4::GRotateOriginZ (float lfAngle)

This will rotate the object around the global Z axis at point 0,0,0 by lfAngle radians.

Definition at line 973 of file WTcMatrix4.cpp.

21.49.3.28 void cMatrix4::GRotateX (float lfAngle, float lfX, float lfY, float lfZ)

This will rotate the object around the global X axis at point lfX,lfY,lfZ by lfAngle radians.

Definition at line 1002 of file WTcMatrix4.cpp.

21.49.3.29 void cMatrix4::GRotateX (float lfAngle)

This will rotate the object around its global X axis by lfAngle radians.

Definition at line 779 of file WTcMatrix4.cpp.

21.49.3.30 void cMatrix4::GRotateY (float lfAngle)

This will rotate the object around its global X axis by lfAngle radians.

Definition at line 804 of file WTcMatrix4.cpp.

21.49.3.31 void cMatrix4::GRotateY (float lfAngle, float lfX, float lfY, float lfZ)

This will rotate the object around the global Y axis at point lfX,lfY,lfZ by lfAngle radians.

Definition at line 1035 of file WTcMatrix4.cpp.

21.49.3.32 void cMatrix4::GRotateZ (float lfAngle)

This will rotate the object around its global X axis by lfAngle radians.

Definition at line 828 of file WTcMatrix4.cpp.

21.49.3.33 void cMatrix4::GRotateZ (float lfAngle, float lfX, float lfY, float lfZ)

This will rotate the object around the global Z axis at point lfX,lfY,lfZ by lfAngle radians.

Definition at line 1068 of file WTcMatrix4.cpp.

21.49.3.34 void cMatrix4::Identity ()

This will restore this objects matrix to an identity matrix.

Definition at line 342 of file WTcMatrix4.cpp.

21.49.3.35 cMatrix4 cMatrix4::InvertRotationMatrix ()

THis will find the inverse of the matrix if it is a standard rotation / translation matrix.
(0,1,2,4,5,6,8,9,10 are orthogonal. 3,7,11 are zero. 12,13,14 are position and 15 is one.)

Definition at line 399 of file WTcMatrix4.cpp.

21.49.3.36 void cMatrix4::LResizeX (float lfScale)

This will scale the object along its local X axis by a factor of lfScale.

Definition at line 868 of file WTcMatrix4.cpp.

21.49.3.37 void cMatrix4::LResizeY (float lfScale)

This will scale the object along its local Y axis by a factor of lfScale.

Definition at line 875 of file WTcMatrix4.cpp.

21.49.3.38 void cMatrix4::LResizeZ (float lfScale)

This will scale the object along its local Z axis by a factor of lfScale.

Definition at line 883 of file WTcMatrix4.cpp.

21.49.3.39 void cMatrix4::LRotate (float lfAngle)

This will rotate the object around its local Z axis by lfAngle radians. Suitable for 2D objects.

21.49.3.40 void cMatrix4::LRotateX (float lfAngle)

This will rotate the object around its local X axis by lfAngle radians.

Definition at line 675 of file WTcMatrix4.cpp.

21.49.3.41 void cMatrix4::LRotateY (float lfAngle)

This will rotate the object around its local Y axis by lfAngle radians.

Definition at line 694 of file WTcMatrix4.cpp.

21.49.3.42 void cMatrix4::LRotateZ (float lfAngle)

This will rotate the object around its local Z axis by lfAngle radians.

Definition at line 715 of file WTcMatrix4.cpp.

21.49.3.43 float* cMatrix4::Matrix (uint8 lcData) [inline]

This will return a pointer to the float numbered lcData in this objects matrix.

Definition at line 55 of file WTcMatrix4.h.

21.49.3.44 float* cMatrix4::Matrix() [inline]

This will return a pointer to this objects matrix data.

Definition at line 53 of file WTcMatrix4.h.

21.49.3.45 float & cMatrix4::operator() (uint16 *iColumn*, uint16 *iRow*)

This will return the float in position [*iColumn*,*iRow*] in mpData.

Definition at line 602 of file WTcMatrix4.cpp.

21.49.3.46 cMatrix4 cMatrix4::operator* (float & *IVal*)

This will multiply every float in this objects matrix by *IVal*.

Definition at line 176 of file WTcMatrix4.cpp.

21.49.3.47 cMatrix4 cMatrix4::operator* (const float *IVal*)

This will multiply every float in this objects matrix by *IVal*.

Definition at line 199 of file WTcMatrix4.cpp.

21.49.3.48 cMatrix4 * cMatrix4::operator* (cMatrix4 * *IVal*)

This will multiply this objects matrix by the matrix *IVal*. Returns This.*IVal* .

Definition at line 571 of file WTcMatrix4.cpp.

21.49.3.49 cMatrix4 cMatrix4::operator* (cMatrix4 *IVal*)

This will multiply this objects matrix by the matrix *IVal*. Returns This.*IVal* .

Definition at line 545 of file WTcMatrix4.cpp.

21.49.3.50 cMatrix4 cMatrix4::operator+ (float & *IVal*)

This will add *IVal* to every float in this objects matrix.

Definition at line 89 of file WTcMatrix4.cpp.

21.49.3.51 cMatrix4 cMatrix4::operator+ (const float *IVal*)

This will add *IVal* to every float in this objects matrix.

Definition at line 111 of file WTcMatrix4.cpp.

21.49.3.52 cMatrix4 cMatrix4::operator+ (cMatrix4 & lVal)

This will add this matrix and the matrix lVal.

Definition at line 347 of file WTcMatrix4.cpp.

21.49.3.53 cMatrix4 cMatrix4::operator- (float & lVal)

This will deduct lVal from every float in this objects matrix.

Definition at line 132 of file WTcMatrix4.cpp.

21.49.3.54 cMatrix4 cMatrix4::operator- (const float lVal)

This will deduct lVal from every float in this objects matrix.

Definition at line 155 of file WTcMatrix4.cpp.

21.49.3.55 cMatrix4 cMatrix4::operator- (cMatrix4 & lVal)

This will subtract the matrix lVal from this matrix.

Definition at line 368 of file WTcMatrix4.cpp.

21.49.3.56 cMatrix4 cMatrix4::operator/ (float & lVal)

This will divide every float in this objects matrix by lVal.

Definition at line 221 of file WTcMatrix4.cpp.

21.49.3.57 cMatrix4 cMatrix4::operator/ (const float lVal)

This will divide every float in this objects matrix by lVal.

Definition at line 243 of file WTcMatrix4.cpp.

21.49.3.58 cMatrix4 * cMatrix4::operator= (cMatrix4 * lVal)

This will equate the data in mpData to the data in lVal.mpData.

This will equate the data in mpData to the data in lVal->mpData.

Definition at line 331 of file WTcMatrix4.cpp.

21.49.3.59 float cMatrix4::operator= (const float lVal)

This will equate every float in mpData to lVal.

Definition at line 302 of file WTcMatrix4.cpp.

21.49.3.60 float * cMatrix4::operator= (float * lVal)

Definition at line 266 of file WTcMatrix4.cpp.

21.49.3.61 cMatrix4 cMatrix4::operator= (cMatrix4 lVal)

This will equate the data in mpData to the data in lVal.mpData.

Definition at line 272 of file WTcMatrix4.cpp.

21.49.3.62 float cMatrix4::operator= (float & lVal)

This will equate every float in mpData to lVal.

Definition at line 278 of file WTcMatrix4.cpp.

21.49.3.63 float & cMatrix4::operator[] (uint16 liPos)

This will return the float in position liPos in mpData.

Definition at line 597 of file WTcMatrix4.cpp.

21.49.3.64 void cMatrix4::Position (c2DVf * lpPosition)

This will position the current object to the 2D Vector lpPosition. (X,Y)

Definition at line 626 of file WTcMatrix4.cpp.

21.49.3.65 void cMatrix4::Position (float lfX, float lfY)

This will position the current object to lfX,lfY. (X,Y)

Definition at line 607 of file WTcMatrix4.cpp.

21.49.3.66 void cMatrix4::Position (c3DVf * lpPosition)

This will position the current object to the 3D Vector lpPosition. (X,Y,Z)

Definition at line 631 of file WTcMatrix4.cpp.

21.49.3.67 void cMatrix4::Position (float lfX, float lfY, float lfZ)

This will position the current object to lfX,lfY,lfZ. (X,Y,Z)

Reimplemented in [cLine](#).

Definition at line 613 of file WTcMatrix4.cpp.

21.49.3.68 float* cMatrix4::Position() [inline]

This will return a pointer to this objects matrices position vector.

Reimplemented in [cLine](#).

Definition at line 57 of file WTcMatrix4.h.

21.49.3.69 void cMatrix4::Position(float *lpPos)

This will take 2 or three floats depending if this [cMatrix4](#) is set to 2D or 3D operations and set the position of the object.

Reimplemented in [cLine](#).

Definition at line 620 of file WTcMatrix4.cpp.

21.49.3.70 void cMatrix4::PositionX(float lfX)

This will set the objects X position to be equal to lfX.

Definition at line 636 of file WTcMatrix4.cpp.

21.49.3.71 void cMatrix4::PositionY(float lfY)

This will set the objects Y position to be equal to lfX.

Definition at line 640 of file WTcMatrix4.cpp.

21.49.3.72 void cMatrix4::PositionZ(float lfZ)

This will set the objects Z position to be equal to lfX.

Definition at line 644 of file WTcMatrix4.cpp.

21.49.3.73 void cMatrix4::Resize(float lfScale)

This will scale the object by a factor of lfScale.

Definition at line 853 of file WTcMatrix4.cpp.

21.49.3.74 void cMatrix4::Rotate(float lfAngle)

This will rotate this matrix about its X axis by lfAngle radians. Suitable for 2D objects.

Definition at line 1174 of file WTcMatrix4.cpp.

21.49.3.75 void cMatrix4::Set2D()

This will set the current matrix to operate as if it is a 2D object.

Definition at line 1229 of file WTcMatrix4.cpp.

21.49.3.76 void cMatrix4::Set3D()

This will set the current matrix to operate as if it is a 3D object.

Definition at line 1236 of file WTcMatrix4.cpp.

21.49.3.77 cMatrix4 cMatrix4::Transpose()

This will return the transpose of this objects matrix.

Definition at line 493 of file WTcMatrix4.cpp.

21.49.3.78 void cMatrix4::UpdateMatrix()

This will update the OpenGL matrix on the top of matrix stack to be identical to this matrix.

Definition at line 56 of file WTcMatrix4.cpp.

21.49.3.79 void cMatrix4::UpdateMatrix(float * *lpMatrix*)

This appears to multiply this matrix by the matrix *lpMatrix* and equate this matrix to the result.

Definition at line 63 of file WTcMatrix4.cpp.

21.49.3.80 float cMatrix4::X() [inline]

This will return this objects X position value.

Definition at line 59 of file WTcMatrix4.h.

21.49.3.81 float* cMatrix4::XVect() [inline]

This will return the vector along this objects local X axis.

Definition at line 66 of file WTcMatrix4.h.

21.49.3.82 float cMatrix4::Y() [inline]

This will return this objects Y position value.

Definition at line 61 of file WTcMatrix4.h.

21.49.3.83 float* cMatrix4::YVect() [inline]

This will return the vector along this objects local Y axis.

Definition at line 68 of file WTcMatrix4.h.

21.49.3.84 float cMatrix4::Z() [inline]

This will return this objects Z position value.

Definition at line 63 of file WTcMatrix4.h.

21.49.3.85 void cMatrix4::Zero()

This will set every float in this objects matrix to be equal to zero.

Definition at line 337 of file WTcMatrix4.cpp.

21.49.3.86 float* cMatrix4::ZVect() [inline]

This will return the vector along this objects local Z axis.

Definition at line 70 of file WTcMatrix4.h.

21.49.4 Friends And Related Function Documentation

21.49.4.1 friend class cCameraMatrix4 [friend]

Definition at line 14 of file WTcMatrix4.h.

21.49.5 Member Data Documentation

21.49.5.1 bool cMatrix4::mb3D [protected]

This si a boolean flag to define whether the object is 3D or 2D. True is 3D. False is 2D.

Definition at line 30 of file WTcMatrix4.h.

21.49.5.2 float cMatrix4::mpData[16] [protected]

This stores a 4x4 matrix of floats representing the current translation of the objet.

Definition at line 27 of file WTcMatrix4.h.

21.49.5.3 cMatrix4 cMatrix4::mpTemp [static]

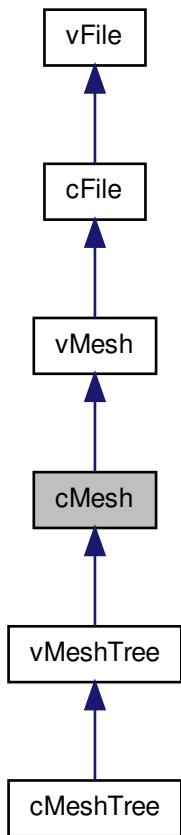
This is a static [cMatrix4](#) used to store data for operands.

Definition at line 39 of file WTcMatrix4.h.

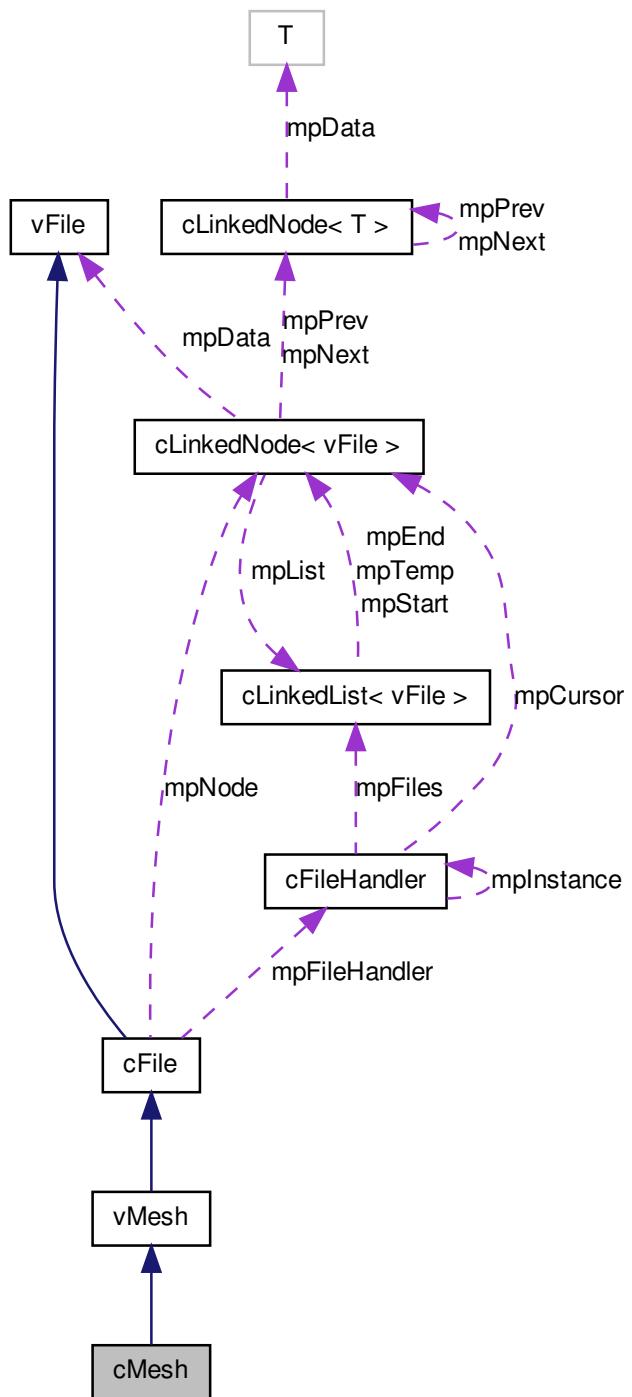
21.50 cMesh Class Reference

This class stores the data for a 3D model.

Inheritance diagram for cMesh:



Collaboration diagram for cMesh:



Public Member Functions

- **cMesh ()**
This will create an empty mesh. No verteces, no faces, nothing.
- **cMesh (cMeshArray *lpMesh)**
This will load the data in lpMesh to be used by this object.
- **~cMesh ()**
- **void SetFormat ()**
This will return the models rendering format.
- **void CreateNormalArray ()**
This will produce an array of vertex normal data for a mesh. This will require the mesh to have vertex and face data and produces normal vectors for all the verteces.
- **uint32 Vertex ()**
This will return the number of verteces in the vertex position array mpVertex.
- **uint32 Faces ()**
This will return the number of faces in the face array mpFaces.
- **float * VertexData ()**
This will return a pointer to the vertex position array.
- **uint16 * FaceData ()**
This will return a pointer to the face array..
- **float * NormalData ()**
This will return a pointer to the array of vertex normals.
- **float * UVData ()**
This will return a pointer to the array of texture co-ordinates.
- **void Position (float lfX, float lfY, float lfZ)**
This will move the objects centre of rotation by lfX,lfY,lfZ.
- **void PositionX (float lfX)**
This will move the objects centre of rotation along its X axis lfX distance.
- **void PositionY (float lfY)**
This will move the objects centre of rotation along its Y axis lfY distance.
- **void PositionZ (float lfZ)**
This will move the objects centre of rotation along its Z axis lfZ distance.
- **void ResetPosition ()**

This will restore the objects original centre of rotation.

- void **BufferMesh ()**

This will move the mesh data to the graphics card memory.

- void **RenderMesh ()**

This will render the mesh directly from graphics memory.

- float **GetSize ()**

*This will return the size of the Mesh. Size being the radius of the sphere required to contain the **cMesh** object.*

- void **FindSize ()**

*This will calculate the size of the Mesh. Size being the radius of the sphere required to contain the **cMesh** object.*

Protected Attributes

- uint8 **miFormat**

*This stores the format which determines what data is used to render the model. The flags in this variable determine what data is present to be used by the renderer * relative to this object. see flags **WT_MESH_FORMAT_VERTEXES**, **WT_MESH_FORMAT_UVS**, **WT_MESH_FORMAT_NORMALS** and **WT_MESH_FORMAT_POSITIVE**.*

- float * **mpVertex**
- float * **mpNormals**
- float * **mpUV**
- uint16 * **mpFaces**
- uint32 **miVertex**
- uint32 **miFaces**
- float **miModelCentre** [3]
- float **mfSize**
- GLuint **mBuffer1**
- GLuint **mBuffer2**

21.50.1 Detailed Description

This class stores the data for a 3D model.

Parameters

<i>lpMesh</i>	This is a pointer to a cMeshArray object which holds the mesh data for this object. This holds the data for a 3D model in a format suitable for rendering. The data is stored in 2 large arrays with the face data stored in the array mpFaces while all the remaining data stored is stored consecutively in mpVertex . mpVertex points to the start of the array while mpNormals and mpUV point to the start of their respective data blocks.
---------------	---

Definition at line 42 of file WTcMesh.h.

21.50.2 Constructor & Destructor Documentation

21.50.2.1 `cMesh::cMesh()`

This will create an empty mesh. No vertexes, no faces, nothing.

Definition at line 237 of file WTcMesh.cpp.

21.50.2.2 `cMesh::cMesh(cMeshArray * lpMesh)`

This will load the data in lpMesh to be used by this object.

Definition at line 58 of file WTcMesh.cpp.

21.50.2.3 `cMesh::~cMesh()`

Definition at line 79 of file WTcMesh.cpp.

21.50.3 Member Function Documentation

21.50.3.1 `void cMesh::BufferMesh() [virtual]`

This will move the mesh data to the graphics card memory.

Implements [vMesh](#).

Reimplemented in [vMeshTree](#).

Definition at line 188 of file WTcMesh.cpp.

21.50.3.2 `void cMesh::CreateNormalArray() [virtual]`

This will produce an array of vertex normal data for a mesh. This will require the mesh to have vertex and face data and produces normal vectors for all the vertexes.

Implements [vMesh](#).

Definition at line 160 of file WTcMesh.cpp.

21.50.3.3 `uint16 * cMesh::FaceData() [virtual]`

This will return a pointer to the face array..

Implements [vMesh](#).

Reimplemented in [vMeshTree](#).

Definition at line 248 of file WTcMesh.cpp.

21.50.3.4 uint32 cMesh::Faces() [virtual]

This will return the number of faces in the face array mpFaces.

Implements [vMesh](#).

Reimplemented in [vMeshTree](#).

Definition at line 244 of file WTcMesh.cpp.

21.50.3.5 void cMesh::FindSize()

This will calculate the size of the Mesh. Size being the radius of the sphere required to contain the [cMesh](#) object.

Reimplemented in [cMeshTree](#).

Definition at line 223 of file WTcMesh.cpp.

21.50.3.6 float cMesh::GetSize() [virtual]

This will return the size of the Mesh. Size being the radius of the sphere required to contain the [cMesh](#) object.

Implements [vMesh](#).

Reimplemented in [cMeshTree](#).

Definition at line 218 of file WTcMesh.cpp.

21.50.3.7 float * cMesh::NormalData() [virtual]

This will return a pointer to the array of vertex normals.

Implements [vMesh](#).

Reimplemented in [vMeshTree](#).

Definition at line 250 of file WTcMesh.cpp.

21.50.3.8 void cMesh::Position(float lfX, float lfY, float lfZ) [virtual]

This will move the objects centre of rotation by lfX,lfY,lfZ.

Implements [vMesh](#).

Reimplemented in [vMeshTree](#).

Definition at line 89 of file WTcMesh.cpp.

21.50.3.9 void cMesh::PositionX(float lfX) [virtual]

This will move the objects centre of rotation along its X axis lfX distance.

Implements [vMesh](#).

Reimplemented in [vMeshTree](#).

Definition at line 105 of file WTcMesh.cpp.

21.50.3.10 void cMesh::PositionY(float lfY) [virtual]

This will move the objects centre of rotation along its Y axis lfY distance.

Implements [vMesh](#).

Reimplemented in [vMeshTree](#).

Definition at line 116 of file WTcMesh.cpp.

21.50.3.11 void cMesh::PositionZ(float lfZ) [virtual]

This will move the objects centre of rotation along its Z axis lfZ distance.

Implements [vMesh](#).

Reimplemented in [vMeshTree](#).

Definition at line 127 of file WTcMesh.cpp.

21.50.3.12 void cMesh::RenderMesh() [virtual]

This will render the mesh directly from graphics memory.

Implements [vMesh](#).

Reimplemented in [vMeshTree](#).

Definition at line 204 of file WTcMesh.cpp.

21.50.3.13 void cMesh::ResetPosition() [virtual]

This will restore the objects original centre of rotation.

Implements [vMesh](#).

Reimplemented in [vMeshTree](#).

Definition at line 138 of file WTcMesh.cpp.

21.50.3.14 void cMesh::SetFormat()

This will return the models rendering format.

Definition at line 150 of file WTcMesh.cpp.

21.50.3.15 float * cMesh::UVData() [virtual]

This will return a pointer to the array of texture co-ordinates.

Implements [vMesh](#).

Reimplemented in [vMeshTree](#).

Definition at line 252 of file WTcMesh.cpp.

21.50.3.16 uint32 cMesh::Vertex() [virtual]

This will return the number of vertices in the vertex position array mpVertex.

Implements [vMesh](#).

Reimplemented in [vMeshTree](#).

Definition at line 242 of file WTcMesh.cpp.

21.50.3.17 float * cMesh::VertexData() [virtual]

This will return a pointer to the vertex position array.

Implements [vMesh](#).

Reimplemented in [vMeshTree](#).

Definition at line 246 of file WTcMesh.cpp.

21.50.4 Member Data Documentation

21.50.4.1 GLuint cMesh::mBuffer1 [protected]

Definition at line 72 of file WTcMesh.h.

21.50.4.2 GLuint cMesh::mBuffer2 [protected]

Definition at line 73 of file WTcMesh.h.

21.50.4.3 float cMesh::mfSize [protected]

Definition at line 70 of file WTcMesh.h.

21.50.4.4 uint32 cMesh::miFaces [protected]

Definition at line 64 of file WTcMesh.h.

21.50.4.5 uint8 cMesh::miFormat [protected]

This stores the format which determines what data is used to render the model. The flags in this variable determine what data is present to be used by the renderer * relative to this object. see flags WT_MESH_FORMAT_VERTEXES, WT_MESH_FORMAT_UVS, WT_MESH_FORMAT_NORMALS and WT_MESH_FORMAT_POSITIVE.

Definition at line 52 of file WTcMesh.h.

21.50.4.6 float cMesh::miModelCentre[3] [protected]

Definition at line 68 of file WTcMesh.h.

21.50.4.7 uint32 cMesh::miVertex [protected]

Definition at line 62 of file WTcMesh.h.

21.50.4.8 uint16* cMesh::mpFaces [protected]

Definition at line 60 of file WTcMesh.h.

21.50.4.9 float* cMesh::mpNormals [protected]

Definition at line 56 of file WTcMesh.h.

21.50.4.10 float* cMesh::mpUV [protected]

Definition at line 58 of file WTcMesh.h.

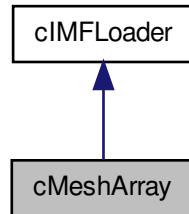
21.50.4.11 float* cMesh::mpVertex [protected]

Definition at line 54 of file WTcMesh.h.

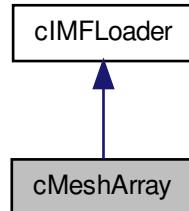
21.51 cMeshArray Class Reference

This is a temporary storage class to ease transformation of data from the hdd to the [cMesh](#) class.

Inheritance diagram for cMeshArray:



Collaboration diagram for cMeshArray:



Public Member Functions

- [cMeshArray \(\)](#)
- void [LoadIMF](#) (ifstream &FileStream)

This will load the data from an IMF File.

- [~cMeshArray \(\)](#)

Public Attributes

- float * [mpVertex](#)
- float * [mpNormals](#)
- float * [mpUV](#)

- uint16 * [mpFaces](#)
- uint32 [miVertex](#)
- uint32 [miFaces](#)
- char * [mpRef](#)

21.51.1 Detailed Description

This is a temporary storage class to ease transformation of data from the hdd to the [cMesh](#) class.

Definition at line 14 of file WTcMesh.h.

21.51.2 Constructor & Destructor Documentation

21.51.2.1 [cMeshArray::cMeshArray\(\)](#)

Definition at line 48 of file WTcMesh.cpp.

21.51.2.2 [cMeshArray::~cMeshArray\(\)](#)

Definition at line 53 of file WTcMesh.cpp.

21.51.3 Member Function Documentation

21.51.3.1 [void cMeshArray::LoadIMF\(ifstream & FileStream \)](#)

This will load the data from an IMF File.

Reimplemented from [cIMFLoader](#).

Definition at line 3 of file WTcMesh.cpp.

21.51.4 Member Data Documentation

21.51.4.1 [uint32 cMeshArray::miFaces](#)

Definition at line 29 of file WTcMesh.h.

21.51.4.2 [uint32 cMeshArray::miVertex](#)

Definition at line 27 of file WTcMesh.h.

21.51.4.3 [uint16* cMeshArray::mpFaces](#)

Definition at line 25 of file WTcMesh.h.

21.51.4.4 float* cMeshArray::mpNormals

Definition at line 21 of file WTcMesh.h.

21.51.4.5 char* cMeshArray::mpRef

Definition at line 31 of file WTcMesh.h.

21.51.4.6 float* cMeshArray::mpUV

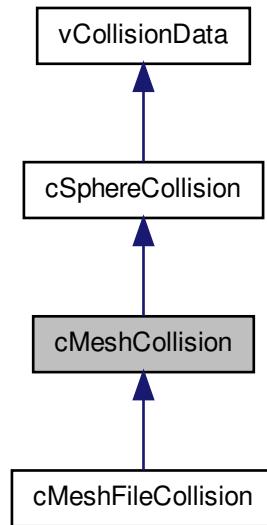
Definition at line 23 of file WTcMesh.h.

21.51.4.7 float* cMeshArray::mpVertex

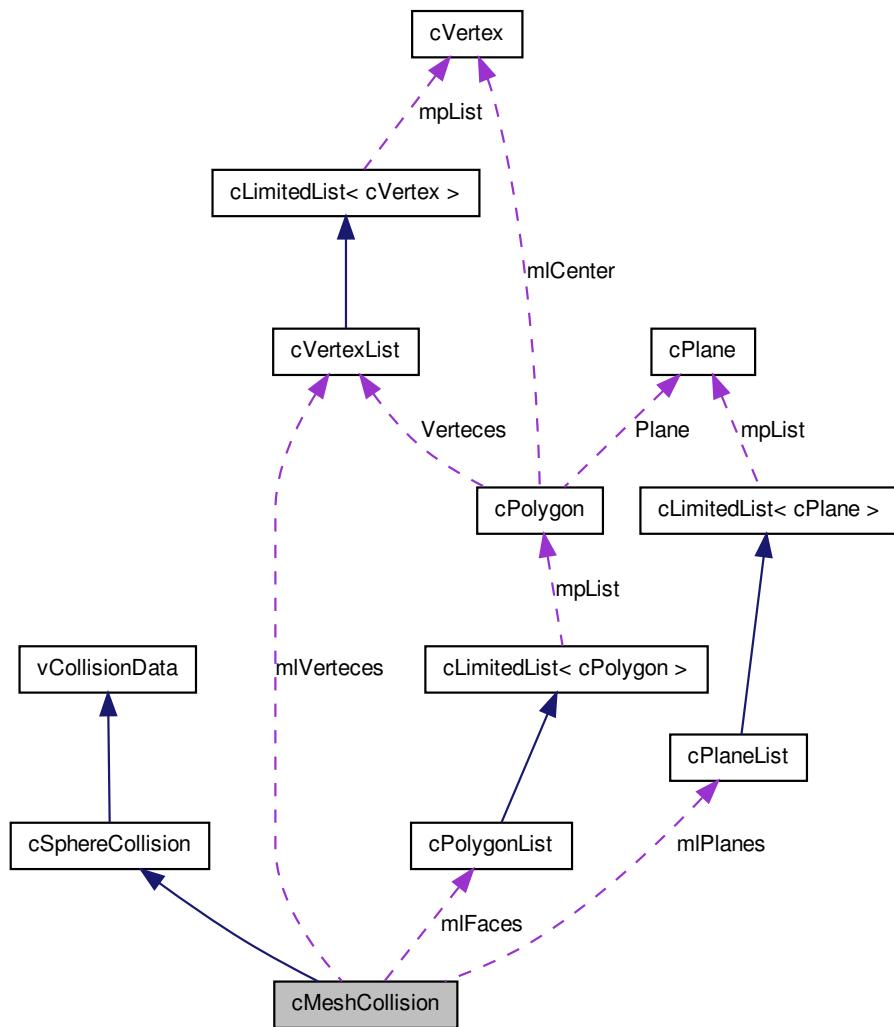
Definition at line 19 of file WTcMesh.h.

21.52 cMeshCollision Class Reference

Inheritance diagram for cMeshCollision:



Collaboration diagram for cMeshCollision:



Public Member Functions

- `cMeshCollision ()`
- `void BuildObject (float *lpRanges)`
- `cMeshCollision * Mesh ()`

Will return a pointer if this object contains a Mesh collision data object. Otherwise returns 0.:

- virtual ~cMeshCollision ()
- uint32 Verteces ()
- float * Vertex (uint32 liPos)
- uint32 Planes ()
- float * Plane (uint32 liPos)
- uint32 Faces ()
- cVertexList FaceVerteces (uint32 liPos)
- cPolygon Polygon (uint32 liPos)
- float * FacePlane (uint32 liPos)
- double GetAngle (float *lpPos, uint32 liPolygon)

*Will return the sum of angles between the point lpPos and every vertex of cPolygon number liPolygon. If this is 2*pi() (or very close) the point lpPos lies on the plane of the polygon inside the outer boundary.*

Public Attributes

- cPlaneList mlPlanes
- cVertexList mlVerteces
- cPolygonList mlFaces

21.52.1 Detailed Description

This is the cCollisionData class for Boxes. Boxes can either be loaded from a file or procedurally generated. These boxes actually work as simple Collision Meshes with 6 planes and 8 verteces. Will Rotate as object moves.

Definition at line 113 of file WTvCollisionData.h.

21.52.2 Constructor & Destructor Documentation

21.52.2.1 cMeshCollision::cMeshCollision () [inline]

Definition at line 123 of file WTvCollisionData.h.

21.52.2.2 cMeshCollision::~cMeshCollision () [virtual]

Definition at line 180 of file WTvCollisionData.cpp.

21.52.3 Member Function Documentation

21.52.3.1 void cMeshCollision::BuildObject (float * lpRanges)

This will build a Box collision. Expects an array of 6 float values representing the length of the normal vector for each plane. (-X, +X, -Y, +Y, -Z, +Z). Floats passed

to this function which are +ve will produce a plane facing away from 0,0,0 (origin is behind the plane). Floats passed to this function which are -ve will produce a plane facing towards 0,0,0 (origin is in front of the plane). This way boxes can be constructed which do not contain the origin.

Definition at line 61 of file WTvCollisionData.cpp.

21.52.3.2 `float * cMeshCollision::FacePlane (uint32 liPos)`

Definition at line 188 of file WTvCollisionData.cpp.

21.52.3.3 `uint32 cMeshCollision::Faces ()`

Definition at line 185 of file WTvCollisionData.cpp.

21.52.3.4 `cVertexList cMeshCollision::FaceVerteces (uint32 liPos)`

Definition at line 186 of file WTvCollisionData.cpp.

21.52.3.5 `double cMeshCollision::GetAngle (float * lpPos, uint32 liPolygon)`

Will return the sum of angles between the point lpPos and every vertex of [cPolygon](#) number liPolygon. If this is 2*pi() (or very close) the point lpPos lies on the plane of the polygon inside the outer boundary.

Definition at line 3 of file WTvCollisionData.cpp.

21.52.3.6 `cMeshCollision * cMeshCollision::Mesh () [virtual]`

Will return a pointer if this object contains a Mesh collision data object. Otherwise returns 0;.

Reimplemented from [cSphereCollision](#).

Definition at line 179 of file WTvCollisionData.cpp.

21.52.3.7 `float * cMeshCollision::Plane (uint32 liPos)`

Definition at line 184 of file WTvCollisionData.cpp.

21.52.3.8 `uint32 cMeshCollision::Planes ()`

Definition at line 183 of file WTvCollisionData.cpp.

21.52.3.9 cPolygon cMeshCollision::Polygon (uint32 *liPos*)

Definition at line 187 of file WTvCollisionData.cpp.

21.52.3.10 uint32 cMeshCollision::Verteces ()

Definition at line 181 of file WTvCollisionData.cpp.

21.52.3.11 float * cMeshCollision::Vertex (uint32 *liPos*)

Definition at line 182 of file WTvCollisionData.cpp.

21.52.4 Member Data Documentation**21.52.4.1 cPolygonList cMeshCollision::mlFaces**

Definition at line 118 of file WTvCollisionData.h.

21.52.4.2 cPlaneList cMeshCollision::mlPlanes

Definition at line 116 of file WTvCollisionData.h.

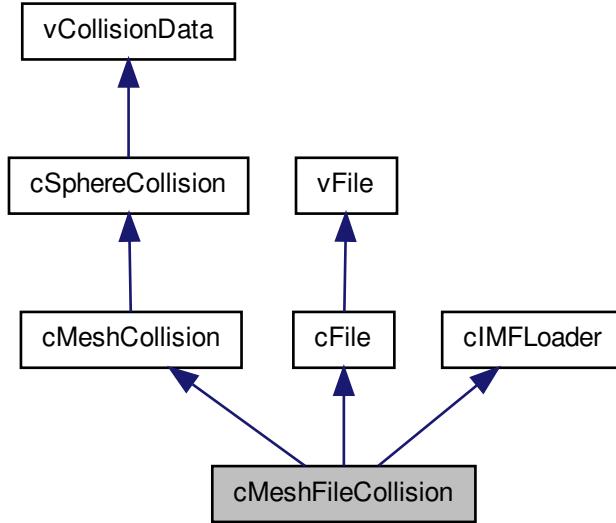
21.52.4.3 cVertexList cMeshCollision::mlVerteces

Definition at line 117 of file WTvCollisionData.h.

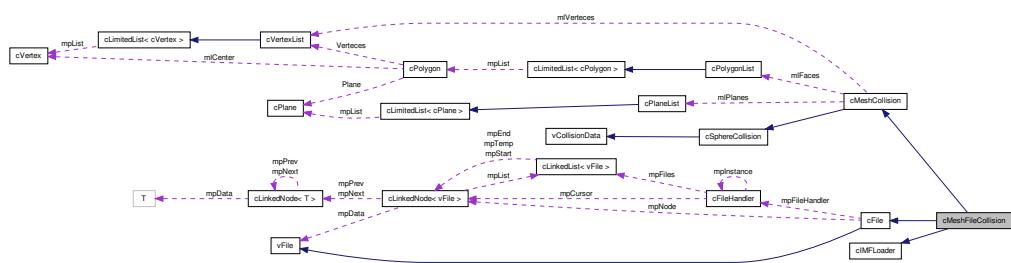
21.53 cMeshFileCollision Class Reference

Mesh Collision Object with functionality to load a Collision Mesh from a file. This inherits [cMeshCollision](#), so uses that code for defining the Mesh Collision Object.

Inheritance diagram for cMeshFileCollision:



Collaboration diagram for cMeshFileCollision:



Public Member Functions

- [cMeshFileCollision \(\)](#)
- void [LoadIMF \(ifstream &FileStream\)](#)

Will Load a Collision Mesh Object from an IMF File.

- virtual [~cMeshFileCollision \(\)](#)

21.53.1 Detailed Description

Mesh Collision Object with functionality to load a Collision Mesh from a file. This inherits [cMeshCollision](#), so uses that code for defining the Mesh Collision Object.

Definition at line 161 of file WTvCollisionData.h.

21.53.2 Constructor & Destructor Documentation

21.53.2.1 [cMeshFileCollision::cMeshFileCollision\(\)](#)

Definition at line 191 of file WTvCollisionData.cpp.

21.53.2.2 [cMeshFileCollision::~cMeshFileCollision\(\) \[virtual\]](#)

Definition at line 192 of file WTvCollisionData.cpp.

21.53.3 Member Function Documentation

21.53.3.1 [void cMeshFileCollision::LoadIMF\(ifstream & FileStream \)](#)

Will Load a Collision Mesh Object from an IMF File.

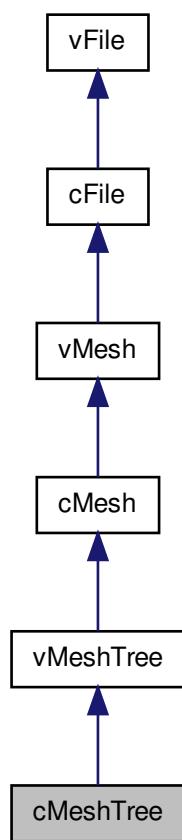
Reimplemented from [cIMFLoader](#).

Definition at line 43 of file WTvCollisionData.cpp.

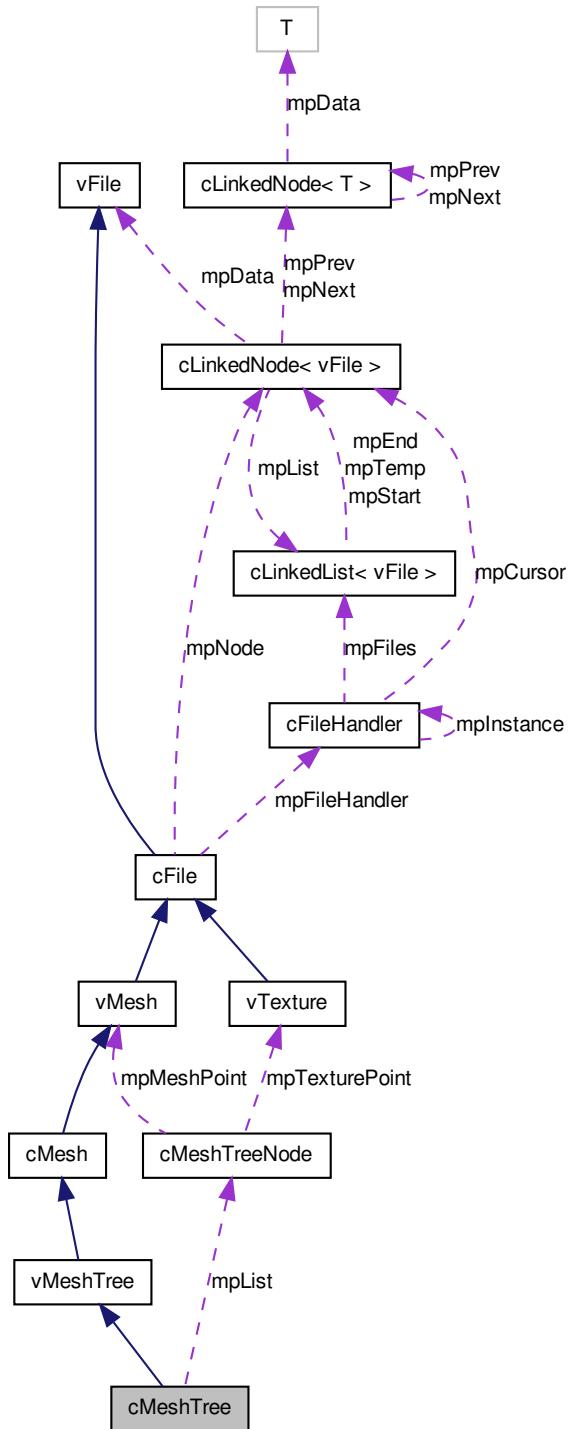
21.54 cMeshTree Class Reference

This will store the data for a cModelList()

Inheritance diagram for cMeshTree:



Collaboration diagram for cMeshTree:



Public Member Functions

- **cMeshTree (cMeshTreeArray *lpArray)**
This will extract the data from the cMeshTreeArray() lpArray.
- **~cMeshTree ()**
- **uint32 Size ()**
This is actually the length of the Tree!!! SOD.
- **cMeshTreeNode * NodeList ()**
This will return a pointer to the first item of mpList.
- **cMeshTreeNode * NodeList (uint32 liCount)**
This will return a pointer to the cMeshTreeNode in position liCount in mpList.
- **float GetSize ()**
This will return the spatial size of this cMeshTree()
- **float FindSize ()**
This will calculate the spatial size of this cMeshTree()

21.54.1 Detailed Description

This will store the data for a cModelList()

Definition at line 34 of file WTcMeshTree.h.

21.54.2 Constructor & Destructor Documentation

21.54.2.1 cMeshTree::cMeshTree (cMeshTreeArray * *lpArray*)

This will extract the data from the cMeshTreeArray() lpArray.

Definition at line 21 of file WTcMeshTree.cpp.

21.54.2.2 cMeshTree::~cMeshTree ()

Definition at line 16 of file WTcMeshTree.cpp.

21.54.3 Member Function Documentation

21.54.3.1 float cMeshTree::FindSize ()

This will calculate the spatial size of this cMeshTree()

Reimplemented from [cMesh](#).

Definition at line 35 of file WTcMeshTree.cpp.

21.54.3.2 float cMeshTree::GetSize() [inline, virtual]

This will return the spatial size of this [cMeshTree\(\)](#)

Reimplemented from [cMesh](#).

Definition at line 54 of file WTcMeshTree.h.

21.54.3.3 cMeshTreeNode* cMeshTree::NodeList() [inline, virtual]

This will return a pointer to the first item of mpList.

Implements [vMeshTree](#).

Definition at line 50 of file WTcMeshTree.h.

21.54.3.4 cMeshTreeNode* cMeshTree::NodeList(uint32 iCount) [inline, virtual]

This will return a pointer to the [cMeshTreeNode](#) in position liCount in mpList.

Implements [vMeshTree](#).

Definition at line 52 of file WTcMeshTree.h.

21.54.3.5 uint32 cMeshTree::Size() [inline, virtual]

This is actually the length of the Tree!!! SOD.

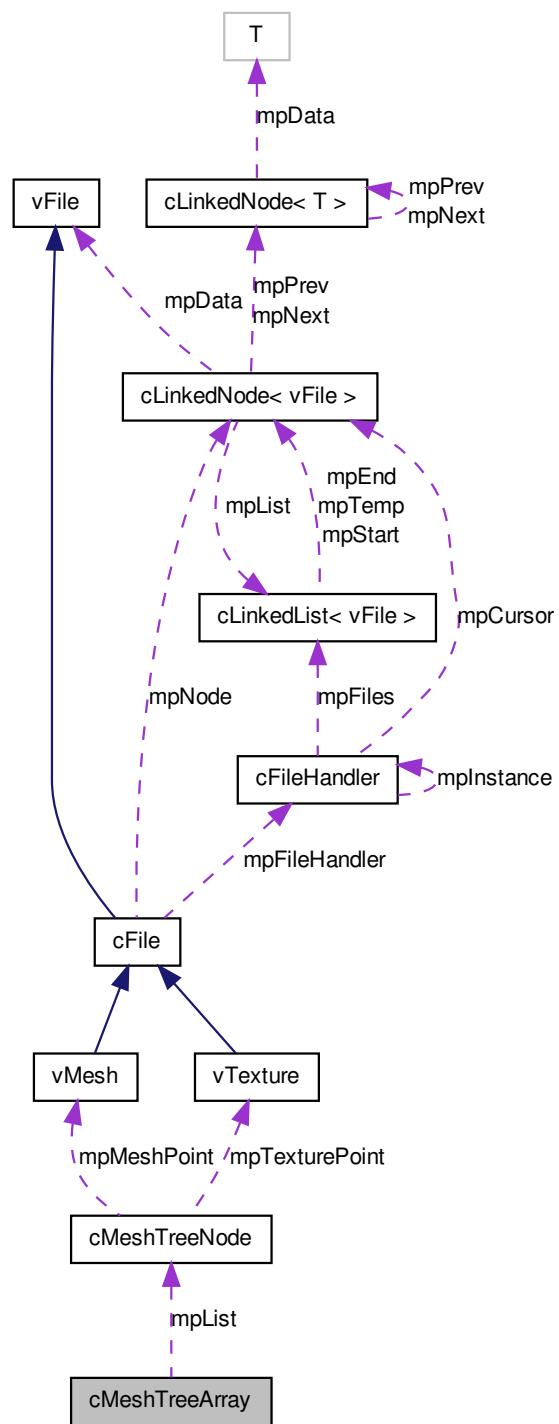
Implements [vMeshTree](#).

Definition at line 48 of file WTcMeshTree.h.

21.55 cMeshTreeArray Class Reference

This will load the information from an IMF file to be handed to a [cMeshTree\(\)](#) This object will store the data for a [cMeshTree\(\)](#) object from an IMF file.

Collaboration diagram for cMeshTreeArray:



Public Member Functions

- [cMeshTreeArray \(\)](#)
This is a public constructor, it will initialise the data for this class.
- [~cMeshTreeArray \(\)](#)
This is a public deconstructor.

Public Attributes

- [cMeshTreeNode * mpList](#)
This will point to an array of cMeshTreeNode() objects.
- [uint32 miTreeLength](#)
This is the number of cMeshTreeNode() objects in the tree.
- [char * mpRef](#)
This will point to an array storing the string reference for this object.

21.55.1 Detailed Description

This will load the information from an IMF file to be handed to a cMeshTree(). This object will store the data for a cMeshTree() object from an IMF file.

Definition at line 16 of file WTcMeshTree.h.

21.55.2 Constructor & Destructor Documentation

21.55.2.1 [cMeshTreeArray::cMeshTreeArray \(\)](#)

This is a public constructor, it will initialise the data for this class.

Definition at line 3 of file WTcMeshTree.cpp.

21.55.2.2 [cMeshTreeArray::~cMeshTreeArray \(\)](#)

This is a public deconstructor.

Definition at line 10 of file WTcMeshTree.cpp.

21.55.3 Member Data Documentation

21.55.3.1 [uint32 cMeshTreeArray::miTreeLength](#)

This is the number of cMeshTreeNode() objects in the tree.

Definition at line 22 of file WTcMeshTree.h.

21.55.3.2 cMeshTreeNode* cMeshTreeArray::mpList

This will point to an array of cMeshTreeNode() objects.

Definition at line 20 of file WTcMeshTree.h.

21.55.3.3 char* cMeshTreeArray::mpRef

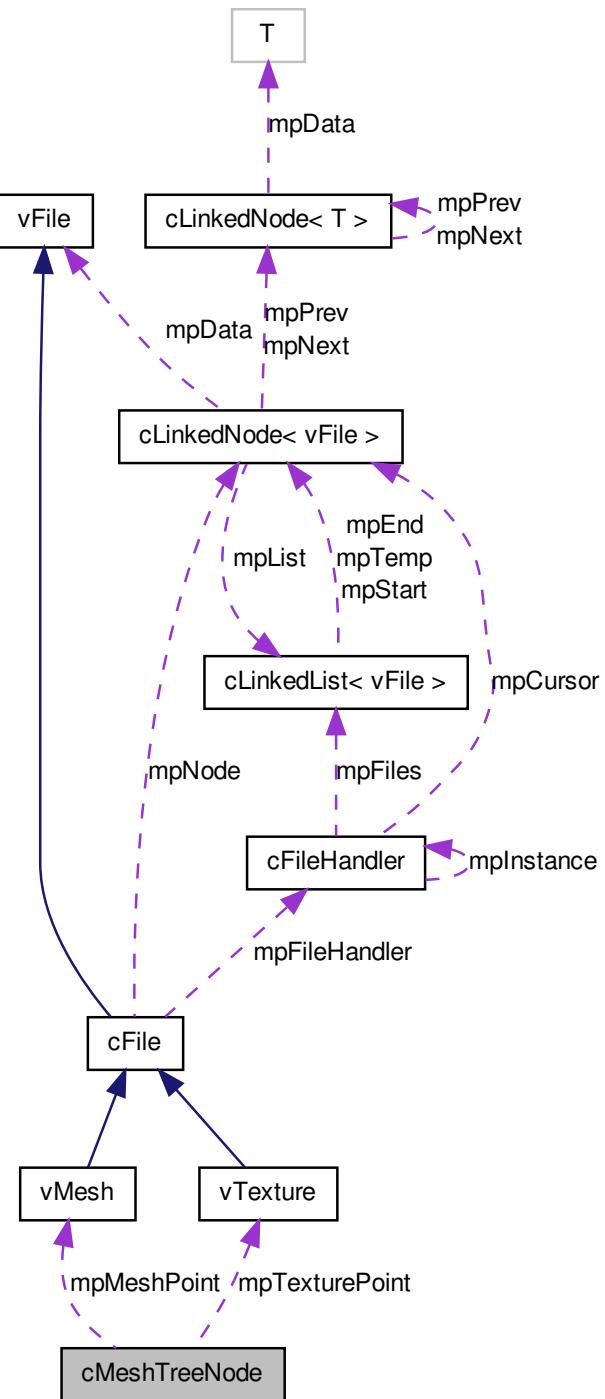
This will point to an array storing the string reference for this object.

Definition at line 24 of file WTcMeshTree.h.

21.56 cMeshTreeNode Class Reference

This object will store the data for a single item in a cMeshTree(). This stores the Position, Rotation, Mesh, Texture, Tree Level, for this object.

Collaboration diagram for cMeshTreeNode:



Public Member Functions

- [cMeshTreeNode \(\)](#)
- [~cMeshTreeNode \(\)](#)
- [void Initialise \(\)](#)

This will Initialise the objects data.

- [void RemoveMesh \(\)](#)

This will Remove and Delete the Mesh.

- [void RemoveTexture \(\)](#)

This will Remove and Delete the Texture.

- [void RemoveStackLevel \(\)](#)

This will reset the Tree Level.

- [void RemovePosition \(\)](#)

This will Reset the Position.

- [void RemoveRotation \(\)](#)

This will Reset the Rotation.

- [void ReadIMF \(ifstream &FileStream\)](#)

This will Load a [cMeshTreeNode\(\)](#) from an IMF File. FileStream should just have reached the start of the Mesh Tree Node.

- [vMesh * Mesh \(\)](#)

This will return the Mesh for this object.

- [vTexture * Texture \(\)](#)

This will return the Texture for this object.

- [float GetSize \(\)](#)

This will return the Spatial Size for this object.

Public Attributes

- uint32 [miFormat](#)
- char * [mpMesh](#)
- vMesh * [mpMeshPoint](#)
- char * [mpTexture](#)
- vTexture * [mpTexturePoint](#)
- uint32 [miLevel](#)
- float * [mpPosition](#)
- float * [mpRotation](#)

21.56.1 Detailed Description

This object will store the data for a single item in a cMeshTree(). This stores the Position, Rotation, Mesh, Texture, Tree Level, for this object.

Definition at line 7 of file WTcMeshTreeNode.h.

21.56.2 Constructor & Destructor Documentation

21.56.2.1 `cMeshTreeNode::cMeshTreeNode()`

Definition at line 16 of file WTcMeshTreeNode.cpp.

21.56.2.2 `cMeshTreeNode::~cMeshTreeNode()`

Definition at line 21 of file WTcMeshTreeNode.cpp.

21.56.3 Member Function Documentation

21.56.3.1 `float cMeshTreeNode::GetSize()`

This will return the Spatial Size for this object.

Definition at line 121 of file WTcMeshTreeNode.cpp.

21.56.3.2 `void cMeshTreeNode::Initialise()`

This will Initialise the objects data.

Definition at line 6 of file WTcMeshTreeNode.cpp.

21.56.3.3 `vMesh * cMeshTreeNode::Mesh()`

This will return the Mesh for this object.

Definition at line 98 of file WTcMeshTreeNode.cpp.

21.56.3.4 `void cMeshTreeNode::ReadIMF(ifstream & FileStream)`

This will Load a [cMeshTreeNode\(\)](#) from an IMF File. FileStream should just have reached the start of the Mesh Tree Node.

Definition at line 48 of file WTcMeshTreeNode.cpp.

21.56.3.5 `void cMeshTreeNode::RemoveMesh()`

This will Remove and Delete the Mesh.

Definition at line 30 of file WTcMeshTreeNode.cpp.

21.56.3.6 void cMeshTreeNode::RemovePosition ()

This will Reset the Position.

Definition at line 41 of file WTcMeshTreeNode.cpp.

21.56.3.7 void cMeshTreeNode::RemoveRotation ()

This will Reset the Rotation.

Definition at line 44 of file WTcMeshTreeNode.cpp.

21.56.3.8 void cMeshTreeNode::RemoveStackLevel ()

This will reset the Tree Level.

Definition at line 36 of file WTcMeshTreeNode.cpp.

21.56.3.9 void cMeshTreeNode::RemoveTexture ()

This will Remove and Delete the Texture.

Definition at line 33 of file WTcMeshTreeNode.cpp.

21.56.3.10 vTexture * cMeshTreeNode::Texture ()

This will return the Texture for this object.

Definition at line 110 of file WTcMeshTreeNode.cpp.

21.56.4 Member Data Documentation

21.56.4.1 uint32 cMeshTreeNode::miFormat

Definition at line 20 of file WTcMeshTreeNode.h.

21.56.4.2 uint32 cMeshTreeNode::miLevel

Definition at line 25 of file WTcMeshTreeNode.h.

21.56.4.3 char* cMeshTreeNode::mpMesh

Definition at line 21 of file WTcMeshTreeNode.h.

21.56.4.4 vMesh* cMeshTreeNode::mpMeshPoint

Definition at line 22 of file WTcMeshTreeNode.h.

21.56.4.5 float* cMeshTreeNode::mpPosition

Definition at line 26 of file WTcMeshTreeNode.h.

21.56.4.6 float* cMeshTreeNode::mpRotation

Definition at line 27 of file WTcMeshTreeNode.h.

21.56.4.7 char* cMeshTreeNode::mpTexture

Definition at line 23 of file WTcMeshTreeNode.h.

21.56.4.8 vTexture* cMeshTreeNode::mpTexturePoint

Definition at line 24 of file WTcMeshTreeNode.h.

21.57 cMinLL< T > Class Template Reference

Linked List Lite. I'm not sure I use this. But quick small and simple templated Linked List.

Public Member Functions

- [cMinLL \(T *lpData\)](#)
- [~cMinLL \(\)](#)
- [uint32 Size \(\)](#)
- [cMinLN< T > * Insert \(int liN, T *lpData\)](#)
- [cMinLN< T > * InsertE \(T *lpData\)](#)
- [cMinLN< T > * InsertB \(T *lpData\)](#)
- [bool Delete \(int liN\)](#)
- [bool Delete \(\[cMinLN< T > *lpOld\]\(#\)\)](#)
- [void Sleep \(\[cMinLN< T > *lpOld\]\(#\)\)](#)
- [void Wake \(int liN, \[cMinLN< T > *lpOld\]\(#\)\)](#)
- [void Output \(\)](#)

Public Attributes

- `cMinLN< T > * mpStart`
- `cMinLN< T > * mpEnd`
- `cMinLN< T > * mpCursor`
- `uint32 miSize`

Static Public Attributes

- static `cMinLN< T > * mpTemp`

21.57.1 Detailed Description

`template<class T> class cMinLL< T >`

Linked List Lite. I'm not sure I use this. But quick small and simple templated Linked List.

Definition at line 21 of file WTMInLinkedList.h.

21.57.2 Constructor & Destructor Documentation

21.57.2.1 `template<class T> cMinLL< T >::cMinLL (T * lpData)`

Definition at line 130 of file WTMInLinkedList.h.

21.57.2.2 `template<class T> cMinLL< T >::~cMinLL () [inline]`

Definition at line 26 of file WTMInLinkedList.h.

21.57.3 Member Function Documentation

21.57.3.1 `template<class T> bool cMinLL< T >::Delete (int liN)`

Definition at line 247 of file WTMInLinkedList.h.

21.57.3.2 `template<class T> bool cMinLL< T >::Delete (cMinLN< T > * lpOld)`

Definition at line 68 of file WTMInLinkedList.h.

21.57.3.3 `template<class T> cMinLN< T > * cMinLL< T >::Insert (int liN, T * lpData)`

Definition at line 181 of file WTMInLinkedList.h.

21.57.3.4 template<class T> cMinLN< T > * cMinLL< T >::InsertB (T * *lpData*)

Definition at line 231 of file WTMInLinkedList.h.

21.57.3.5 template<class T> cMinLN< T > * cMinLL< T >::InsertE (T * *lpData*)

Definition at line 215 of file WTMInLinkedList.h.

21.57.3.6 template<class T> void cMinLL< T >::Output ()

Definition at line 140 of file WTMInLinkedList.h.

21.57.3.7 template<class T> uint32 cMinLL< T >::Size () [inline]

Definition at line 46 of file WTMInLinkedList.h.

21.57.3.8 template<class T> void cMinLL< T >::Sleep (cMinLN< T > * *lpOld*)

Definition at line 85 of file WTMInLinkedList.h.

21.57.3.9 template<class T> void cMinLL< T >::Wake (int *l/N*, cMinLN< T > * *lpOld*)

Definition at line 99 of file WTMInLinkedList.h.

21.57.4 Member Data Documentation

21.57.4.1 template<class T> uint32 cMinLL< T >::miSize

Definition at line 45 of file WTMInLinkedList.h.

21.57.4.2 template<class T> cMinLN< T > * cMinLL< T >::mpCursor

Definition at line 42 of file WTMInLinkedList.h.

21.57.4.3 template<class T> cMinLN< T > * cMinLL< T >::mpEnd

Definition at line 41 of file WTMInLinkedList.h.

21.57.4.4 template<class T> cMinLN< T > * cMinLL< T >::mpStart

Definition at line 38 of file WTMInLinkedList.h.

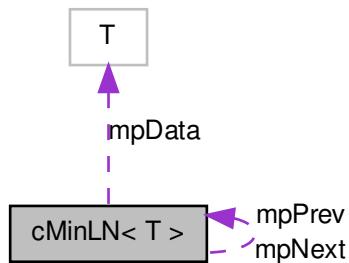
21.57.4.5 `template<class T> cMinLN< T > * cMinLL< T >::mpTemp [static]`

Definition at line 43 of file WTMInLinkedList.h.

21.58 `cMinLN< T >` Class Template Reference

Linked Nodes Lite. I'm not sure I use this. But quick small and simple templated Linked List nodes.

Collaboration diagram for `cMinLN< T >`:



Public Member Functions

- `cMinLN< T > * Next ()`
- `cMinLN< T > * Previous ()`

Public Attributes

- `T * mpData`

Friends

- class `cMinLL< T >`

21.58.1 Detailed Description

```
template<class T> class cMinLN< T >
```

Linked Nodes Lite. I'm not sure I use this. But quick small and simple templated Linked List nodes.

Definition at line 7 of file WTMInLinkedList.h.

21.58.2 Member Function Documentation

```
21.58.2.1 template<class T> cMinLN<T>* cMinLN< T >::Next( ) [inline]
```

Definition at line 16 of file WTMInLinkedList.h.

```
21.58.2.2 template<class T> cMinLN<T>* cMinLN< T >::Previous( ) [inline]
```

Definition at line 17 of file WTMInLinkedList.h.

21.58.3 Friends And Related Function Documentation

```
21.58.3.1 template<class T> friend class cMinLL< T > [friend]
```

Definition at line 9 of file WTMInLinkedList.h.

21.58.4 Member Data Documentation

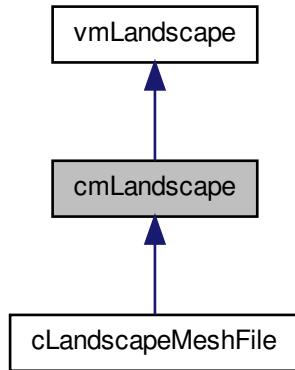
```
21.58.4.1 template<class T> T* cMinLN< T >::mpData
```

Definition at line 15 of file WTMInLinkedList.h.

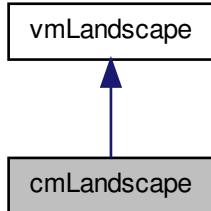
21.59 cmLandscape Class Reference

This will store the data for a landscape mesh. The data can be rendered through [cLandscape](#).

Inheritance diagram for cmLandscape:



Collaboration diagram for cmLandscape:



Public Member Functions

- virtual uint32 [GetNode](#) (uint32 liX, uint32 liZ)
- virtual uint32 [VertexDataSize](#) ()
- uint32 [Vertices](#) ()
- uint16 * [FaceData](#) ()

This will return a pointer to the landscapes face data array.

- `uint32 Faces ()`
This will return the number of quadrilateral faces in the landscape. (actually this appears to be the number of nodes in the face data array.)
- `cmLandscape (cmLandscape *lpArray)`
** This will create a new landscape mesh using the data in lpArray.*
- `cmLandscape (uint32 liXSteps, uint32 liZSteps, float lfXSize, float lfZSize)`
** This will create a new landscape array to the resolution and size specified.*
- `cmLandscape * operator= (cmLandscape *lpNew)`
- `~cmLandscape ()`
- `virtual void PrepareLandscape ()`
This will create all the absent data for the landscape. Vertex Positions, Normals, UVs and face data.
- `virtual void GenerateVertices ()`
- `virtual void GenerateUVs ()`
- `virtual void GenerateNormals ()`
- `virtual void GenerateFaces ()`
- `virtual float GetHeightLocal (float lfX, float lfZ)`
This will return the height of the landscape (using interpolation) at the position lfX,lfZ relative to the landscapes 0,0 corner.
- `float GetVertexHeight (uint32 liX, uint32 liZ)`
This will return the height of the landscapes vertex numbered liX,liZ in the array.
- `virtual void SetHeight (uint32 liX, uint32 liZ, float lfHeight)`
This will set the height of the landscapes vertex numbered liX,liZ in the array.
- `void SetHeightRange (float lfRange)`
Will scale the landscape to fit the new height range.
- `void SetXStep (float lfStep)`
Will scale the landscape step size along the X axis to be lfStep.
- `void SetZStep (float lfStep)`
Will scale the landscape step size along the Z axis to be lfStep.
- `void CreateNormalArray ()`
This will reinitialise the landscapes normal values.
- `void SetXZStep (float lfXStep, float lfZStep)`
This will scale the landscape step size along both the X and Z axis to be lfXStep and lfZStep respectively.
- `void Randomize (float liHeight, uint8 liSize)`

This will randomise the landscape to a maximum height of liHeight and then smooth the heights within liSize of each vertex.

- void [Randomize](#) (float liHeight)

This will just randomise the landscape up to a maximum height of liHeight.

- void [Randomize](#) (uint32 Lines, float lfHeightRange)

This will randomise the landscape, by laying Lines number of random bisecting lines, which raise the landscape by lfHeightRange each. lfHeightRange is 0.001 by default.

- void [BufferMesh](#) ()

This will move the mesh data to the graphics card memory.

- virtual void [RenderMesh](#) ()

This will render the mesh directly from graphics memory.

- uint32 [Width](#) ()

This will return miXSteps.

- uint32 [Length](#) ()

This will reutrn miZSteps.

- float [GetXSize](#) ()

- float [GetYSize](#) ()

- float [GetZSize](#) ()

- void [GenerateRandomUVs](#) ()

- [cmLandscape](#) * [Duplicate](#) ()

Protected Member Functions

- [cmLandscape](#) ()

Protected Attributes

- float * [mpVertex](#)

- float * [mpNormals](#)

- float * [mpUV](#)

- uint32 [miXSteps](#)

- uint32 [miZSteps](#)

- float [mfXSize](#)

- float [mfZSize](#)

- float [mfHeightRange](#)

- uint16 * [mpQuads](#)

- uint16 [miQuads](#)

- float [mfXSizeI](#)

- float [mfZSizeI](#)

- GLuint * [mpBufferIDs](#)

Static Protected Attributes

- static uint32 `miCustomLandscapes` = 0

Friends

- class `cLandscapeMeshFile`

21.59.1 Detailed Description

This will store the data for a landscape mesh. The data can be rendered through `cLandscape`.

Definition at line 43 of file WTcmLandscape.h.

21.59.2 Constructor & Destructor Documentation

21.59.2.1 `cmLandscape::cmLandscape()` [protected]

Definition at line 835 of file WTcmLandscape.cpp.

21.59.2.2 `cmLandscape::cmLandscape(cmLandscape * lpArray)`

* This will create a new landscape mesh using the data in lpArray.

Parameters

<code>lpArray</code>	This is the <code>cmLandscapeArray</code> holding landscape data for this object.
----------------------	---

Definition at line 99 of file WTcmLandscape.cpp.

21.59.2.3 `cmLandscape::cmLandscape(uint32 liXSteps, uint32 liZSteps, float lfXSize, float lfZSize)`

* This will create a new landscape array to the resolution and size specified.

Parameters

<code>liXSteps</code>	Number of Nodes to form the landscape along the Landscapes X axis.
<code>liZSteps</code>	Number of Nodes to form the landscape along the Landscapes Z axis.
<code>lfXSize</code>	Size of landscape polygons along the landscapes X axis.
<code>lfZSize</code>	Size of landscape polygons along the landscapes Z axis.

Definition at line 72 of file WTcmLandscape.cpp.

21.59.2.4 cmLandscape::~cmLandscape()

Definition at line 130 of file WTcmLandscape.cpp.

21.59.3 Member Function Documentation**21.59.3.1 void cmLandscape::BufferMesh()**

This will move the mesh data to the graphics card memory.

Definition at line 45 of file WTcmLandscape.cpp.

21.59.3.2 void cmLandscape::CreateNormalArray() [virtual]

This will reinitialise the landscapes normal values.

Implements [vmLandscape](#).

Definition at line 345 of file WTcmLandscape.cpp.

21.59.3.3 cmLandscape * cmLandscape::Duplicate()

Definition at line 830 of file WTcmLandscape.cpp.

21.59.3.4 uint16* cmLandscape::FaceData() [inline]

This will return a pointer to the landscapes face data array.

Definition at line 86 of file WTcmLandscape.h.

21.59.3.5 uint32 cmLandscape::Faces() [inline]

This will return the number of quadrilateral faces in the landscape. (actually this appears to be the number of nodes in the face data array).

Definition at line 88 of file WTcmLandscape.h.

21.59.3.6 void cmLandscape::GenerateFaces() [virtual]

Definition at line 782 of file WTcmLandscape.cpp.

21.59.3.7 void cmLandscape::GenerateNormals() [virtual]

Definition at line 724 of file WTcmLandscape.cpp.

21.59.3.8 void cmLandscape::GenerateRandomUVs()

Definition at line 509 of file WTcmLandscape.cpp.

21.59.3.9 void cmLandscape::GenerateUVs() [virtual]

Definition at line 741 of file WTcmLandscape.cpp.

21.59.3.10 void cmLandscape::GenerateVertices() [virtual]

Definition at line 705 of file WTcmLandscape.cpp.

21.59.3.11 float cmLandscape::GetHeightLocal(float lfX, float lfZ) [virtual]

This will return the height of the landscape (using interpolation) at the position lfX,lfZ relative to the landscapes 0,0 corner.

Implements [vmLandscape](#).

Definition at line 139 of file WTcmLandscape.cpp.

21.59.3.12 uint32 cmLandscape::GetNode(uint32 liX, uint32 liZ) [virtual]

Definition at line 64 of file WTcmLandscape.cpp.

21.59.3.13 float cmLandscape::GetVertexHeight(uint32 liX, uint32 liZ) [virtual]

This will return the height of the landscapes vertex numbered liX,liZ in the array.

Implements [vmLandscape](#).

Definition at line 164 of file WTcmLandscape.cpp.

21.59.3.14 float cmLandscape::GetXSize() [inline]

Definition at line 154 of file WTcmLandscape.h.

21.59.3.15 float cmLandscape::GetYSize() [inline]

Definition at line 155 of file WTcmLandscape.h.

21.59.3.16 float cmLandscape::GetZSize() [inline]

Definition at line 156 of file WTcmLandscape.h.

21.59.3.17 uint32 cmLandscape::Length() [inline, virtual]

This will return miZSteps.

Implements [vmLandscape](#).

Definition at line 152 of file WTcmLandscape.h.

21.59.3.18 cmLandscape* cmLandscape::operator=(cmLandscape * lpNew)**21.59.3.19 void cmLandscape::PrepareLandscape() [virtual]**

This will create all the absent data for the landscape. Vertex Positions, Normals, UVs and face data.

Implements [vmLandscape](#).

Definition at line 23 of file WTcmLandscape.cpp.

21.59.3.20 void cmLandscape::Randomize(float liHeight) [virtual]

This will just randomise the landscape up to a maximum height of liHeight.

Implements [vmLandscape](#).

Definition at line 330 of file WTcmLandscape.cpp.

21.59.3.21 void cmLandscape::Randomize(uint32 Lines, float lfHeightRange = 0.001) [virtual]

This will randomise the landscape, by laying Lines number of random bisecting lines, which raise the landscape by lfHeightRange each. lfHeightRange is 0.001 by default.

Implements [vmLandscape](#).

Definition at line 217 of file WTcmLandscape.cpp.

21.59.3.22 void cmLandscape::Randomize(float liHeight, uint8 liSize) [virtual]

This will randomise the landscape to a maximum height of liheight and then smooth the heights within liSize of each vertex.

Implements [vmLandscape](#).

Definition at line 177 of file WTcmLandscape.cpp.

21.59.3.23 void cmLandscape::RenderMesh() [virtual]

This will render the mesh directly from graphics memory.

Implements [vmLandscape](#).

Definition at line 378 of file WTcmLandscape.cpp.

21.59.3.24 void cmLandscape::SetHeight (uint32 *liX*, uint32 *liZ*, float *lfHeight*) [inline, virtual]

This will set the height of the landscapes vertex numbered liX,liZ in the array.

Implements [vmLandscape](#).

Definition at line 169 of file WTcmLandscape.cpp.

21.59.3.25 void cmLandscape::SetHeightRange (float *lfRange*)

Will scale the landscape to fit the new height range.

Definition at line 393 of file WTcmLandscape.cpp.

21.59.3.26 void cmLandscape::SetXStep (float *lfStep*)

Will scale the landscape step size along the X axis to be lfStep.

Definition at line 408 of file WTcmLandscape.cpp.

21.59.3.27 void cmLandscape::SetXZStep (float *lfXStep*, float *lfZStep*)

This will scale the landscape step size along both the X and Z axis to be lfXStep and lfZStep respectively.

Definition at line 443 of file WTcmLandscape.cpp.

21.59.3.28 void cmLandscape::SetZStep (float *lfStep*)

Will scale the landscape step size along the Z axis to be lfStep.

Definition at line 426 of file WTcmLandscape.cpp.

21.59.3.29 uint32 cmLandscape::Vertices () [inline]

Definition at line 84 of file WTcmLandscape.h.

21.59.3.30 virtual uint32 cmLandscape::VertexDataSize () [inline, virtual]

Definition at line 83 of file WTcmLandscape.h.

21.59.3.31 uint32 cmLandscape::Width () [inline, virtual]

This will return miXSteps.

Implements [vmLandscape](#).

Definition at line 150 of file WTcmLandscape.h.

21.59.4 Friends And Related Function Documentation

21.59.4.1 friend class cLandscapeMeshFile [friend]

Definition at line 161 of file WTcmLandscape.h.

21.59.5 Member Data Documentation

21.59.5.1 float cmLandscape::mfHeightRange [protected]

Definition at line 63 of file WTcmLandscape.h.

21.59.5.2 float cmLandscape::mfXSize [protected]

Definition at line 59 of file WTcmLandscape.h.

21.59.5.3 float cmLandscape::mfXSizeI [protected]

Definition at line 71 of file WTcmLandscape.h.

21.59.5.4 float cmLandscape::mfZSize [protected]

Definition at line 61 of file WTcmLandscape.h.

21.59.5.5 float cmLandscape::mfZSizeI [protected]

Definition at line 73 of file WTcmLandscape.h.

21.59.5.6 uint32 cmLandscape::miCustomLandscapes = 0 [static, protected]

Definition at line 47 of file WTcmLandscape.h.

21.59.5.7 uint16 cmLandscape::miQuads [protected]

Definition at line 69 of file WTcmLandscape.h.

21.59.5.8 uint32 cmLandscape::miXSteps [protected]

Definition at line 55 of file WTcmLandscape.h.

21.59.5.9 uint32 cmLandscape::miZSteps [protected]

Definition at line 57 of file WTcmLandscape.h.

21.59.5.10 GLuint* cmLandscape::mpBufferIDs [protected]

Definition at line 76 of file WTcmLandscape.h.

21.59.5.11 float* cmLandscape::mpNormals [protected]

Definition at line 51 of file WTcmLandscape.h.

21.59.5.12 uint16* cmLandscape::mpQuads [protected]

Definition at line 67 of file WTcmLandscape.h.

21.59.5.13 float* cmLandscape::mpUV [protected]

Definition at line 53 of file WTcmLandscape.h.

21.59.5.14 float* cmLandscape::mpVertex [protected]

Definition at line 49 of file WTcmLandscape.h.

21.60 cmLandscapeArray Class Reference

This is a class to transfer data from a hdd file to memory in a format that is quick and easy to render. This is a temporary storage class to ease transformation of data from the hdd to the [cmLandscape](#) class.

Public Member Functions

- [cmLandscapeArray \(\)](#)
- [~cmLandscapeArray \(\)](#)

Public Attributes

- float * [mpVertex](#)
- float * [mpUV](#)
- uint32 [miXSteps](#)
- uint32 [miZSteps](#)
- float [mfXSize](#)

- float [mfZSize](#)
- float [mfHeightRange](#)
- char * [mpRef](#)

21.60.1 Detailed Description

This is a class to transfer data from a hdd file to memory in a format that is quick and easy to render. This is a temporary storage class to ease transformation of data from the hdd to the [cmLandscape](#) class.

Definition at line 13 of file [WTcmLandscape.h](#).

21.60.2 Constructor & Destructor Documentation

21.60.2.1 `cmLandscapeArray::cmLandscapeArray()`

Definition at line 3 of file [WTcmLandscape.cpp](#).

21.60.2.2 `cmLandscapeArray::~cmLandscapeArray()`

Definition at line 11 of file [WTcmLandscape.cpp](#).

21.60.3 Member Data Documentation

21.60.3.1 `float cmLandscapeArray::mfHeightRange`

Definition at line 29 of file [WTcmLandscape.h](#).

21.60.3.2 `float cmLandscapeArray::mfXSize`

Definition at line 25 of file [WTcmLandscape.h](#).

21.60.3.3 `float cmLandscapeArray::mfZSize`

Definition at line 27 of file [WTcmLandscape.h](#).

21.60.3.4 `uint32 cmLandscapeArray::miXSteps`

Definition at line 21 of file [WTcmLandscape.h](#).

21.60.3.5 `uint32 cmLandscapeArray::miZSteps`

Definition at line 23 of file [WTcmLandscape.h](#).

21.60.3.6 char* cmLandscapeArray::mpRef

Definition at line 31 of file WTcmLandscape.h.

21.60.3.7 float* cmLandscapeArray::mpUV

Definition at line 19 of file WTcmLandscape.h.

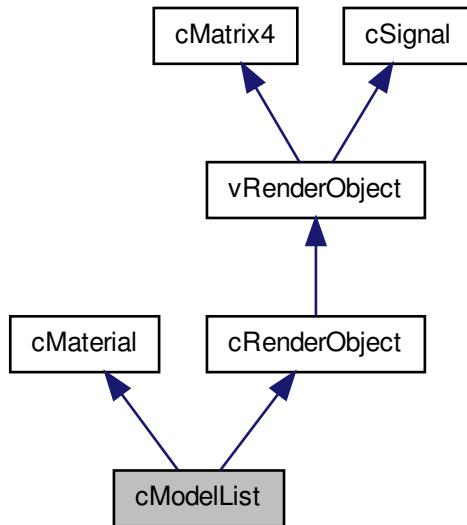
21.60.3.8 float* cmLandscapeArray::mpVertex

Definition at line 17 of file WTcmLandscape.h.

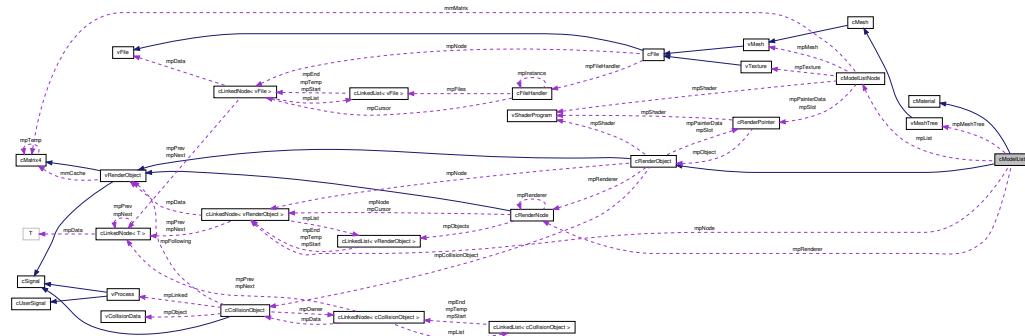
21.61 cModelList Class Reference

Model list is a 'static' render tree holding data for a series of renderable objects. It counts as a single renderable object.

Inheritance diagram for cModelList:



Collaboration diagram for cModelList:



Public Member Functions

- **cModelList (vMeshTree *lpTree)**

This is the constructor for `cModelList`. It will create the list based on the `vMeshTree` file `lpTree`.

- **cModelList (cRenderNode *lpRenderer, vMeshTree *lpTree)**

This is the constructor for `cModelList`. It will create the list based on the `vMeshTree` file `lpTree`, and will be owned by `lpRenderer`.

- **void LoadTree (vMeshTree *lpTree)**

Will re-initialise this `cModelList` using the file `lpTree`. It will have the structure and models of the file.

- **cModelList (uint32 liLength)**

Create an empty `cModelList` of size `liLength`.

- **cModelList (cRenderNode *lpRenderer, uint32 liLength)**

Create an empty `cModelList` of size `liLength`. `lpRenderer` will own this `ModelList`.

- **~cModelList ()**

- **void Initialise (uint32 liLength)**

Will reset this object to be an empty `cModelList` of size `liLength`.

- **uint32 ListLength ()**

Will return the current size of the `cModelList`.

- **cMatrix4 & GetMatrix (uint32 liPos)**

Will return a pointer to the matrix of `model[liPos]`.

- **void RenderToPainter ()**

virtual function to allow polymorphism. see cCamera::RenderToPainter();

- void **RenderNodePainter** (uint8 liListPos)
- void **RenderPainter** (uint8 liLevel)

virtual function to allow polymorphism. see cCamera::RenderPainter();
- void **Render** ()

virtual function to allow polymorphiss. see cCamera::Render();
- void **SetTexture** (uint32 liPos, vTexture *lpTexture)

This will bind a texture for a single model in the model list.
- void **SetShader** (uint32 liPos, vShaderProgram *lpShader)

This will bind a texture for a single model in the model list.
- void **SetMesh** (uint32 liPos, vMesh *lpMesh)

This will set a mesh for a single model in the model list.
- void **SetCommand** (uint32 liPos, uint8 liCom)

This will set a command for a single model in the model list.

21.61.1 Detailed Description

Model list is a 'static' render tree holding data for a series of renderable objects. It counts as a single renderable object.

Parameters

<i>lpTree</i>	a pointer to the vMeshTree object loaded from cIMF::LoadIMF()
<i>lpRenderer</i>	Places this object beneath lpRenderer in the scene graph. This works like a scene graph, The relative depths of the objects is controlled by cModelListNode::miLevel . cModelList holds a static array of cModelListNode . cModelListNode holds all the relative data about each object in the render tree.

Definition at line 36 of file WTcModelList.h.

21.61.2 Constructor & Destructor Documentation

21.61.2.1 **cModelList::cModelList (vMeshTree * lpTree)**

This is the constructor for **cModelList**. It will create the list based on the **vMeshTree** file *lpTree*.

Parameters

<i>lpTree</i>	This is the file containing the data for the structure (and models etc.) of the render tree.
---------------	--

Definition at line 62 of file WTcModelList.cpp.

21.61.2.2 cModelList::cModelList (cRenderNode * *lpRenderer*, vMeshTree * *lpTree*)

This is the constructor for [cModelList](#). It will create the list based on the [vMeshTree](#) file *lpTree*, and will be owned by *lpRenderer*.

Parameters

<i>lpTree</i>	This is the file containing the data for the structure (and models etc.) of the render tree.
<i>lpRenderer</i>	This is the cRenderNode which will own this renderable object.

Definition at line 68 of file WTcModelList.cpp.

21.61.2.3 cModelList::cModelList (uint32 *liLength*)

Create an empty [cModelList](#) of size *liLength*.

Definition at line 54 of file WTcModelList.cpp.

21.61.2.4 cModelList::cModelList (cRenderNode * *lpRenderer*, uint32 *liLength*)

Create an empty [cModelList](#) of size *liLength*. *lpRenderer* will own this ModelList.

Definition at line 59 of file WTcModelList.cpp.

21.61.2.5 cModelList::~cModelList ()

Definition at line 49 of file WTcModelList.cpp.

21.61.3 Member Function Documentation

21.61.3.1 cMatrix4 & cModelList::GetMatrix (uint32 *liPos*)

Will return a pointer to the matrix of model[*liPos*].

Parameters

<i>liPos</i>	The number of the object in the list.
--------------	---------------------------------------

Returns

returns a pointer to the matrix of model[*liPos*]

Definition at line 260 of file WTcModelList.cpp.

21.61.3.2 void cModelList::Initialise (uint32 *liLength*)

Will reset this object to be an empty [cModelList](#) of size liLength.

Definition at line 86 of file WTcModelList.cpp.

21.61.3.3 uint32 cModelList::ListLength ()

Will return the current size of the [cModelList](#).

Definition at line 266 of file WTcModelList.cpp.

21.61.3.4 void cModelList::LoadTree (vMeshTree * *lpTree*)

Will re-initialise this [cModelList](#) using the file lpTree. It will have the structure and models of the file.

Definition at line 74 of file WTcModelList.cpp.

21.61.3.5 void cModelList::Render () [virtual]

virtual function to allow polymorphiss. see [cCamera::Render\(\)](#);

Implements [cRenderObject](#).

Definition at line 192 of file WTcModelList.cpp.

21.61.3.6 void cModelList::RenderNodePainter (uint8 *liListPos*)

Definition at line 169 of file WTcModelList.cpp.

21.61.3.7 void cModelList::RenderPainter (uint8 *liLevel*) [virtual]

virtual function to allow polymorphism. see [cCamera::RenderPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 186 of file WTcModelList.cpp.

21.61.3.8 void cModelList::RenderToPainter () [virtual]

virtual function to allow polymorphism. see [cCamera::RenderToPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 121 of file WTcModelList.cpp.

21.61.3.9 void cModelList::SetCommand (uint32 *liPos*, uint8 *liCom*)

This will set a command for a single model in the model list.

Parameters

<i>liPos</i>	the number of the object in mpList this function will affect.
<i>liCom</i>	The command to be called before rendering model number liPos. The commands determine how the matrix stack is modified before the object is rendered. WT_MODELLIST_NEW_LEVEL will push the current matrix onto the matrix stack. Anything else is the number of matrices that will be popped off the matrix stack.

Definition at line 115 of file WTcModelList.cpp.

21.61.3.10 void cModelList::SetMesh (uint32 *liPos*, vMesh * *lpMesh*)

This will set a mesh for a single model in the model list.

Parameters

<i>liPos</i>	the number of the object in mpList this function will affect.
<i>lpMesh</i>	a pointer to the mesh to use for object number liPos.

Definition at line 110 of file WTcModelList.cpp.

21.61.3.11 void cModelList::SetShader (uint32 *liPos*, vShaderProgram * *lpShader*)

This will bind a texture for a single model in the model list.

Parameters

<i>liPos</i>	the number of the object in mpList this function will affect.
<i>lpShader</i>	a pointer to the Shader for object number liPos to use.

Definition at line 105 of file WTcModelList.cpp.

21.61.3.12 void cModelList::SetTexture (uint32 *liPos*, vTexture * *lpTexture*)

This will bind a texture for a single model in the model list.

Parameters

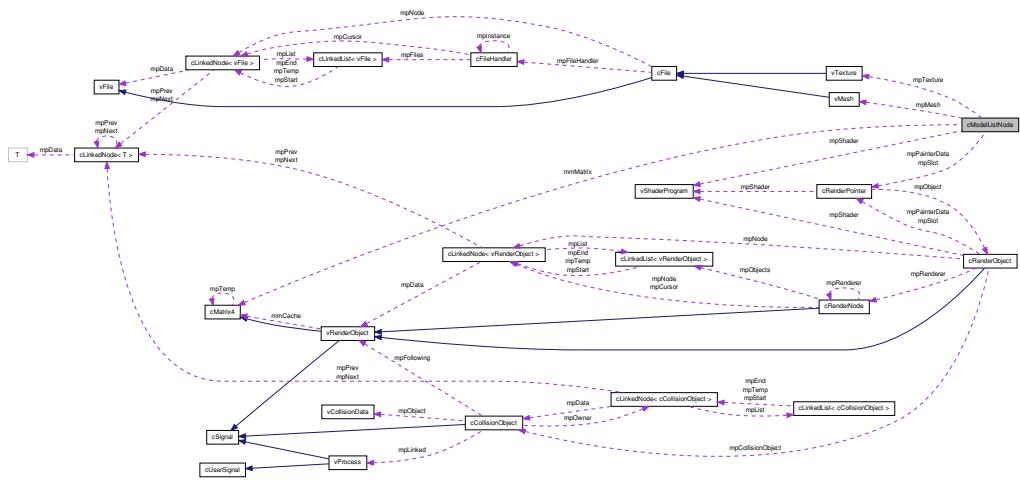
<i>liPos</i>	the number of the object in mpList this function will affect.
<i>lpTexture</i>	a pointer to the Texture to bind to object number liPos.

Definition at line 100 of file WTcModelList.cpp.

21.62 cModelListNode Class Reference

Class for storing data about a single node on in the Model List.

Collaboration diagram for cModelListNode:



Public Member Functions

- [vMesh * Mesh \(\)](#)
- [cModelListNode \(\)](#)
- [~cModelListNode \(\)](#)

Public Attributes

- [cRenderPointer * mpPainterData](#)
- [cRenderPointer ** mpSlot](#)

Friends

- class [cModelList](#)

21.62.1 Detailed Description

Class for storing data about a single node on in the Model List.

Definition at line 8 of file WTCModelList.h.

21.62.2 Constructor & Destructor Documentation

21.62.2.1 `cModelListNode::cModelListNode()`

Definition at line 9 of file WTcModelList.cpp.

21.62.2.2 `cModelListNode::~cModelListNode()`

Definition at line 3 of file WTcModelList.cpp.

21.62.3 Member Function Documentation

21.62.3.1 `vMesh* cModelListNode::Mesh() [inline]`

Definition at line 19 of file WTcModelList.h.

21.62.4 Friends And Related Function Documentation

21.62.4.1 `friend class cModelList [friend]`

Definition at line 26 of file WTcModelList.h.

21.62.5 Member Data Documentation

21.62.5.1 `cRenderPointer* cModelListNode::mpPainterData`

Definition at line 23 of file WTcModelList.h.

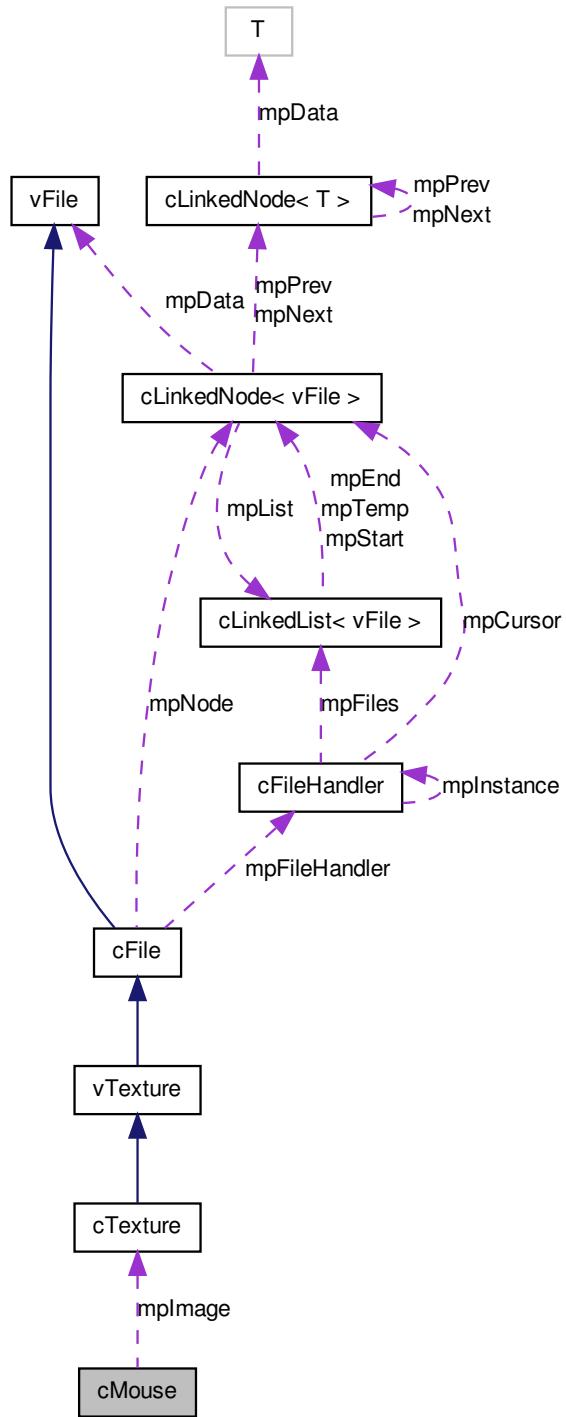
21.62.5.2 `cRenderPointer** cModelListNode::mpSlot`

Definition at line 24 of file WTcModelList.h.

21.63 cMouse Class Reference

This will store all the input data for a single mouse.

Collaboration diagram for cMouse:



Public Member Functions

- `cMouse ()`
- `void Update ()`

This will Update the mouse variable. Should be called every frame by `cKernel`.

- `void Hide ()`

This will hide the mouse cursor.

- `void Show ()`

This will show the mouse cursor.

- `bool Showing ()`

This will return whether the mouse cursor is currently showing.

- `void Lock ()`

This will lock the mouse cursor to the centre of the window (allowing unlimited scrolling).

- `void Unlock ()`

This will unlock the mouse cursor from the centre of the window.

Public Attributes

- `int x`

This stores the current frames X value (in pixels from the windows 0,0).

- `int y`

This stores the current frames Y value (in pixels from the windows 0,0).

- `int z`

This stores the current frames Z value (in pixels from the windows 0,0).

- `bool left`

This stores the mouses current left button state.

- `bool right`

This stores the mouses current right button state.

- `bool middle`

This stores the mouses current middle button state.

- `bool locked`

This stores whether the mouse cursor is locked to the centre of the window.

- int `xs`

This will store the mouses current X speed. (pixels travelled since last frame).

- int `ys`

This will store the mouses current Y speed. (pixels travelled since last frame).

- int `cx`

This is the real mouses X position. It will change as the mouse moves, so will not be consistent through out the frame. Should avoid being used.

- int `cy`

This is the real mouses Y position. It will change as the mouse moves, so will not be consistent through out the frame. Should avoid being used.

- tagPOINT `Pos`

Windows only variable. Windows format for mouse position...?

21.63.1 Detailed Description

This will store all the input data for a single mouse.

Definition at line 4 of file WTcMouse.h.

21.63.2 Constructor & Destructor Documentation

21.63.2.1 `cMouse::cMouse()`

Definition at line 3 of file WTcMouse.cpp.

21.63.3 Member Function Documentation

21.63.3.1 `void cMouse::Hide()`

This will hide the mouse cursor.

Definition at line 63 of file WTcMouse.cpp.

21.63.3.2 `void cMouse::Lock()`

This will lock the mouse cursor to the centre of the window (allowing unlimited scrolling).

Definition at line 42 of file WTcMouse.cpp.

21.63.3.3 void cMouse::Show()

This will show the mouse cursor.

Definition at line 66 of file WTcMouse.cpp.

21.63.3.4 bool cMouse::Showing()

This will return whether the mouse cursor is currently showing.

Definition at line 69 of file WTcMouse.cpp.

21.63.3.5 void cMouse::Unlock()

This will unlock the mouse cursor from the centre of the window.

Definition at line 56 of file WTcMouse.cpp.

21.63.3.6 void cMouse::Update()

This will Update the mouse variable. Should be called every frame by [cKernel](#).

Definition at line 13 of file WTcMouse.cpp.

21.63.4 Member Data Documentation

21.63.4.1 int cMouse::cx

This is the real mouses X position. It will change as the mouse moves, so will not be consistent through out the frame. Should avoid being used.

Definition at line 36 of file WTcMouse.h.

21.63.4.2 int cMouse::cy

This is the real mouses Y position. It will change as the mouse moves, so will not be consistent through out the frame. Should avoid being used.

Definition at line 38 of file WTcMouse.h.

21.63.4.3 bool cMouse::left

This stores the mouses current left button state.

Definition at line 22 of file WTcMouse.h.

21.63.4.4 bool cMouse::locked

This stores whether the mouse cursor is locked to the centre of the window.

Definition at line 28 of file WTcMouse.h.

21.63.4.5 bool cMouse::middle

This stores the mouses current middle button state.

Definition at line 26 of file WTcMouse.h.

21.63.4.6 tagPOINT cMouse::Pos

Windows only variable. Windows format for mouse position...?

Definition at line 42 of file WTcMouse.h.

21.63.4.7 bool cMouse::right

This stores the mouses current right button state.

Definition at line 24 of file WTcMouse.h.

21.63.4.8 int cMouse::x

This stores the current frames X value (in pixels from the windows 0,0).

Definition at line 15 of file WTcMouse.h.

21.63.4.9 int cMouse::xs

This will store the mouses current X speed. (pixels travelled since last frame).

Definition at line 31 of file WTcMouse.h.

21.63.4.10 int cMouse::y

This stores the current frames Y value (in pixels from the windows 0,0).

Definition at line 17 of file WTcMouse.h.

21.63.4.11 int cMouse::ys

This will store the mouses current Y speed. (pixels travelled since last frame).

Definition at line 33 of file WTcMouse.h.

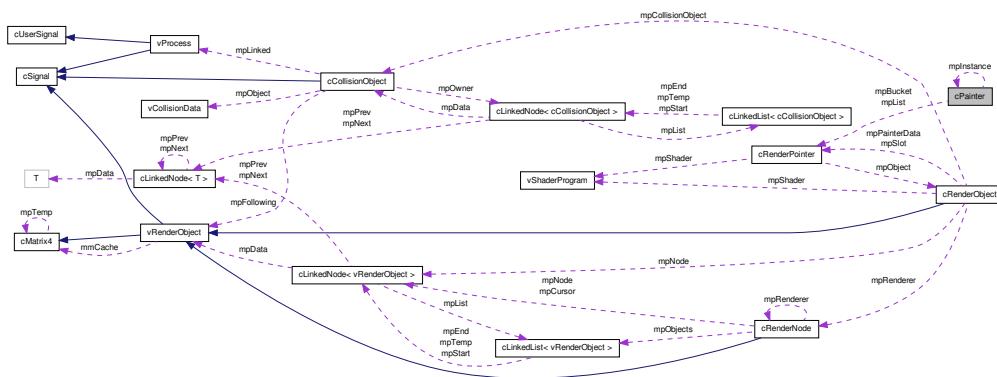
21.63.4.12 int cMouse::z

This stores the current frames Z value (in pixels from the windows 0,0).

Definition at line 19 of file WTcMouse.h.

21.64 cPainter Class Reference

Collaboration diagram for cPainter:



Public Member Functions

- `~cPainter ()`
- `void Add (cRenderPointer *lfData)`
- `void Remove (cRenderPointer *lfSlot)`
- `void Reset ()`
- `void Resize (uint32 liSize)`

This will resize the array of `cRenderPointer` to change the number of `cRenderObjects` that can be held.

- `void SortByShader ()`

This will sort the array `mpList` based on object shader program.

- `void SortByDistance ()`

This will sort the array `mpList` based on object distance from the camera.

- `void SortByTexture ()`

This will sort the array `mpList` based on the texture that each object uses.

- `void SortByAlpha ()`

This will sort the array `mpList` based on the alpha value that each object uses.

- void `TextureState` (uint32 mpCurrent, uint32 mpLast)
- void `ShaderState` (`vShaderProgram` *mpCurrent, `vShaderProgram` *mpLast)
- void `Render` ()

Static Public Member Functions

- static `cPainter` * `Instance` ()

This will return the current instance of the `cPainter` algorithm, if there is no current instance a new one will be created. This should always be used and NOT `cPainter()`;

21.64.1 Detailed Description

When `WT_USE_PAINTER_ALGORITHM` is set to true. `cRenderObjects` will render to this class instead of screen. Once all the `cRenderObjects` have been rendered into this class it will use an optimised Radix sort to sort them to create an efficient rendering order.

Definition at line 11 of file `WTcPainter.h`.

21.64.2 Constructor & Destructor Documentation

21.64.2.1 `cPainter::~cPainter()`

Definition at line 28 of file `WTcPainter.cpp`.

21.64.3 Member Function Documentation

21.64.3.1 `void cPainter::Add(cRenderPointer * IfData)`

Definition at line 52 of file `WTcPainter.cpp`.

21.64.3.2 `cPainter * cPainter::Instance() [static]`

This will return the current instance of the `cPainter` algorithm, if there is no current instance a new one will be created. This should always be used and NOT `cPainter()`;

Definition at line 7 of file `WTcPainter.cpp`.

21.64.3.3 `void cPainter::Remove(cRenderPointer * IfSlot)`

Definition at line 275 of file `WTcPainter.cpp`.

21.64.3.4 void cPainter::Render()

Definition at line 234 of file WTcPainter.cpp.

21.64.3.5 void cPainter::Reset()

Definition at line 290 of file WTcPainter.cpp.

21.64.3.6 void cPainter::Resize(uint32 iSize)

This will resize the array of [cRenderPointer](#) to change the number of [cRenderObjects](#) that can be held.

Definition at line 34 of file WTcPainter.cpp.

21.64.3.7 void cPainter::ShaderState(vShaderProgram * mpCurrent, vShaderProgram * mpLast)

Definition at line 206 of file WTcPainter.cpp.

21.64.3.8 void cPainter::SortByAlpha()

This will sort the array mpList based on the alpha value that each object uses.

Definition at line 191 of file WTcPainter.cpp.

21.64.3.9 void cPainter::SortByDistance()

This will sort the array mpList based on object distance from the camera.

Definition at line 62 of file WTcPainter.cpp.

21.64.3.10 void cPainter::SortByShader()

This will sort the array mpList based on object shader program.

Definition at line 124 of file WTcPainter.cpp.

21.64.3.11 void cPainter::SortByTexture()

This will sort the array mpList based on the texture that each object uses.

Definition at line 157 of file WTcPainter.cpp.

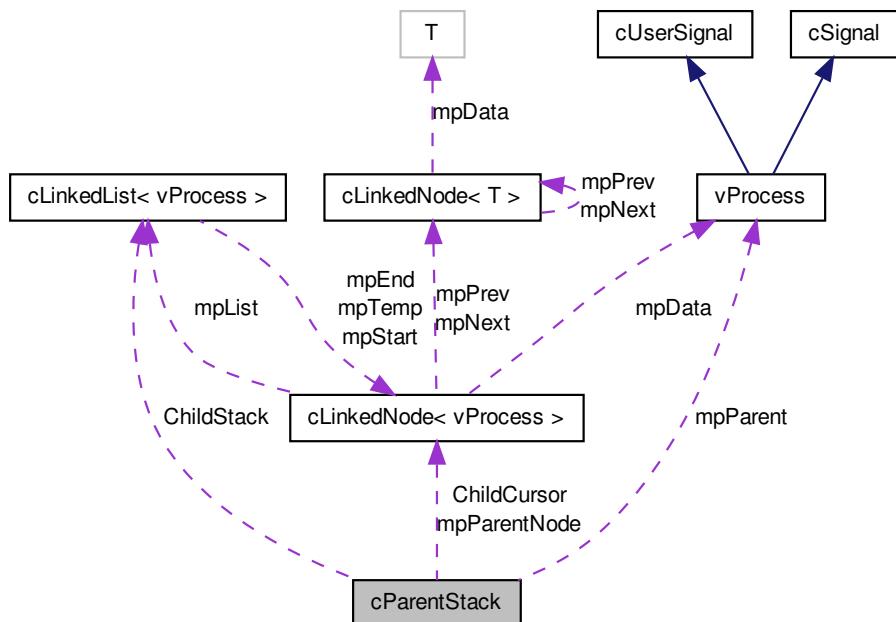
21.64.3.12 void cPainter::TextureState(uint32 mpCurrent, uint32 mpLast)

Definition at line 219 of file WTcPainter.cpp.

21.65 cParentStack Class Reference

This class will automatically track the parents and children of each process. When a new process is created, the process that created is stored as the new processes parent. The new process is added as a child of the creating process. This allows Processes to get their parent (a rather inefficient method). It also allows use of the TREE signal commands to send a signal to a process, all it's children recursively. See [cSignal](#) for more information.

Collaboration diagram for cParentStack:



Public Member Functions

- [cParentStack \(vProcess *lpChild\)](#)
Constructor. The process lpChild is a pointer to the newly created process which will become a child of the current process.
- [~cParentStack \(\)](#)
- [void SetParent \(vProcess *lpPar\)](#)
- [cLinkedNode< vProcess > * AddChild \(vProcess *lpChild\)](#)
- [void RemoveChild \(cLinkedNode< vProcess > *lpNode\)](#)
- [void StripChildrensParent \(\)](#)

- void [ClearParent \(\)](#)
- [vProcess * Parent \(\)](#)

Returns the parent for the process which owns this [cParentStack](#).

- [vProcess * Child \(\)](#)

Returns the next child for the process which owns this [cParentStack](#).

- void [Signal \(SIGNAL lsSignal\)](#)

Once a signal is identified as containing the TREE flag calling this will apply the Signal as appropriate down the tree.

- void [ChildSignal \(SIGNAL lsSignal\)](#)

Once a signal is identified as containing the TREE flag this will apply the Signal to all the Children of this [cParentStack](#).

21.65.1 Detailed Description

This class will automatically track the parents and children of each process. When a new process is created, the process that created is stored as the new processes parent. The new process is added as a child of the creating process. This allows Processes to get their parent (a rather inefficient method). It also allows use of the TREE signal commands to send a signal to a process, all it's children recursively. See [cSignal](#) for more information.

Definition at line 8 of file WTcParentStack.h.

21.65.2 Constructor & Destructor Documentation

21.65.2.1 [cParentStack::cParentStack \(vProcess * lpChild \)](#)

Constructor. The process lpChild is a pointer to the newly created process which will become a child of the current process.

Definition at line 3 of file WTcParentStack.cpp.

21.65.2.2 [cParentStack::~cParentStack \(\)](#)

Definition at line 11 of file WTcParentStack.cpp.

21.65.3 Member Function Documentation

21.65.3.1 [cLinkedNode< vProcess > * cParentStack::AddChild \(vProcess * lpChild \)](#)

Definition at line 70 of file WTcParentStack.cpp.

21.65.3.2 vProcess * cParentStack::Child()

Returns the next child for the process which owns this [cParentStack](#).

Definition at line 44 of file WTcParentStack.cpp.

21.65.3.3 void cParentStack::ChildSignal(SIGNAL IsSignal)

Once a signal is identified as containing the TREE flag this will apply the Signal to all the Children of this [cParentStack](#).

Definition at line 34 of file WTcParentStack.cpp.

21.65.3.4 void cParentStack::ClearParent()

Definition at line 80 of file WTcParentStack.cpp.

21.65.3.5 vProcess * cParentStack::Parent()

Returns the parent for the process which owns this [cParentStack](#).

Definition at line 85 of file WTcParentStack.cpp.

21.65.3.6 void cParentStack::RemoveChild(cLinkedNode< vProcess > * lpNode)

Definition at line 75 of file WTcParentStack.cpp.

21.65.3.7 void cParentStack::SetParent(vProcess * lpPar)

Definition at line 65 of file WTcParentStack.cpp.

21.65.3.8 void cParentStack::Signal(SIGNAL IsSignal)

Once a signal is identified as containing the TREE flag calling this will apply the Signal as appropriate down the tree.

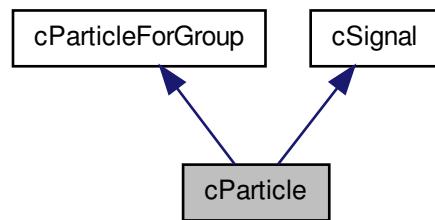
Definition at line 18 of file WTcParentStack.cpp.

21.65.3.9 void cParentStack::StripChildrensParent()

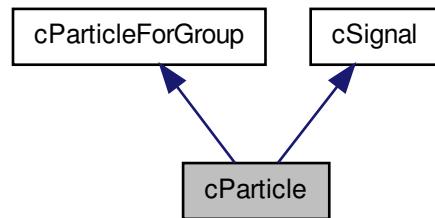
Definition at line 24 of file WTcParentStack.cpp.

21.66 cParticle Class Reference

Inheritance diagram for cParticle:



Collaboration diagram for cParticle:



Public Member Functions

- [cParticle \(\)](#)
- [~cParticle \(\)](#)
- void [UpdateSpeed](#) (float *lpTemp)

This will set a cParticles Speed.

- virtual void [UpdatePos \(\)](#)

This is the function that will update a cParticles Position. if WT_PARTICLE_HANDLER_UPDATE_PARTICLE_POSITIONS is true then [cParticleHandler](#) will do this automatically.

- void [AdditionalKillFunctionality \(\)](#)

Virtual Functions to allow additional commands to be processed when a kill signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

- void [AdditionalSleepFunctionality \(\)](#)

Virtual Functions to allow additional commands to be processed when a sleep signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

- void [AdditionalWakeFunctionality \(\)](#)

Virtual Functions to allow additional commands to be processed when a wake signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

21.66.1 Detailed Description

This Class is fire and Forget Particles. Find Position based on Global matrix as they will not be parented to a RenderNode. Once position is set they will not be affected by changes to the local matrices. These will automatically be grouped and handled by [cParticleHandler](#) for efficiency reasons. Their position can be automatically updated by [cParticleGroup](#) if the flag is set

Definition at line 5 of file WTcParticle.h.

21.66.2 Constructor & Destructor Documentation

21.66.2.1 [cParticle::cParticle \(\)](#)

Definition at line 3 of file WTcParticle.cpp.

21.66.2.2 [cParticle::~cParticle \(\)](#)

Definition at line 9 of file WTcParticle.cpp.

21.66.3 Member Function Documentation

21.66.3.1 [void cParticle::AdditionalKillFunctionality \(\) \[virtual\]](#)

Virtual Functions to allow additional commands to be processed when a kill signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

Reimplemented from [cSignal](#).

Definition at line 14 of file WTcParticle.cpp.

21.66.3.2 void cParticle::AdditionalSleepFunctionality() [virtual]

Virtual Functions to allow additional commands to be processed when a sleep signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

Reimplemented from [cSignal](#).

Definition at line 19 of file WTcParticle.cpp.

21.66.3.3 void cParticle::AdditionalWakeFunctionality() [virtual]

Virtual Functions to allow additional commands to be processed when a wake signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

Reimplemented from [cSignal](#).

Definition at line 24 of file WTcParticle.cpp.

21.66.3.4 void cParticle::UpdatePos() [virtual]

This is the function that will update a cParticles Position. if WT_PARTICLE_HANDLER_-UPDATE_PARTICLE_POSITIONS is true then [cParticleHandler](#) will do this automatically.

Reimplemented from [cParticleForGroup](#).

Definition at line 223 of file WTcParticle.cpp.

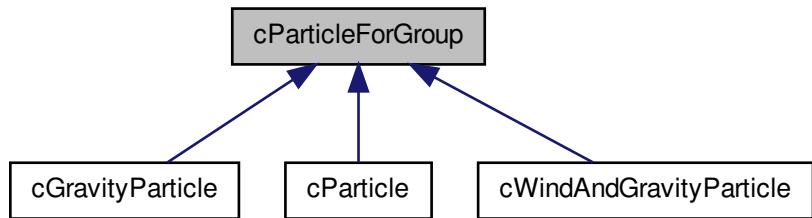
21.66.3.5 void cParticle::UpdateSpeed(float * *lpTemp*)

This will set a cParticles Speed.

Definition at line 215 of file WTcParticle.cpp.

21.67 cParticleForGroup Class Reference

Inheritance diagram for cParticleForGroup:



Public Member Functions

- `~cParticleForGroup ()`
- virtual void `UpdatePos ()`
- void `UpdateFade ()`
- void `Spawn (cParticleSettings &lpData)`
- void `SetColor (float *lpRGB)`
- void `SetSize (float lpSize)`
- void `SetFade (float lpFade)`
- void `SetSpeed (float *lpSpeed)`
- void `SetPosition (float *lpPos)`

Public Attributes

- float `Color [4]`
- float `Position [3]`
- float `Speed [3]`
- float `FadeSpeed`
- float `Life`
- float `Size`

Protected Member Functions

- `cParticleForGroup ()`

Friends

- class [cParticleGroup](#)

21.67.1 Detailed Description

These Particles will be controlled by cParticleGGroup. They should never be created by the user. [cParticle](#) is for User Creation. They should only be generated by a [cParticleGroup](#) Object as this will control them.

Definition at line 32 of file WTcParticleGroup.h.

21.67.2 Constructor & Destructor Documentation**21.67.2.1 `cParticleForGroup::cParticleForGroup() [inline, protected]`**

Definition at line 40 of file WTcParticleGroup.h.

21.67.2.2 `cParticleForGroup::~cParticleForGroup() [inline]`

Definition at line 50 of file WTcParticleGroup.h.

21.67.3 Member Function Documentation**21.67.3.1 `void cParticleForGroup::SetColor(float *lpRGB) [inline]`**

Definition at line 58 of file WTcParticleGroup.h.

21.67.3.2 `void cParticleForGroup::SetFade(float lpFade) [inline]`

Definition at line 60 of file WTcParticleGroup.h.

21.67.3.3 `void cParticleForGroup::SetPosition(float *lpPos) [inline]`

Definition at line 62 of file WTcParticleGroup.h.

21.67.3.4 `void cParticleForGroup::SetSize(float lpSize) [inline]`

Definition at line 59 of file WTcParticleGroup.h.

21.67.3.5 `void cParticleForGroup::SetSpeed(float *lpSpeed) [inline]`

Definition at line 61 of file WTcParticleGroup.h.

21.67.3.6 void cParticleForGroup::Spawn (cParticleSettings & *lpData*)

Definition at line 161 of file WTcParticleGroup.cpp.

21.67.3.7 void cParticleForGroup::UpdateFade () [inline]

Definition at line 54 of file WTcParticleGroup.h.

21.67.3.8 void cParticleForGroup::UpdatePos () [virtual]

Reimplemented in [cParticle](#), [cGravityParticle](#), and [cWindAndGravityParticle](#).

Definition at line 153 of file WTcParticleGroup.cpp.

21.67.4 Friends And Related Function Documentation

21.67.4.1 friend class cParticleGroup [friend]

Definition at line 50 of file WTcParticleGroup.h.

21.67.5 Member Data Documentation

21.67.5.1 float cParticleForGroup::Color[4]

Definition at line 40 of file WTcParticleGroup.h.

21.67.5.2 float cParticleForGroup::FadeSpeed

Definition at line 45 of file WTcParticleGroup.h.

21.67.5.3 float cParticleForGroup::Life

Definition at line 46 of file WTcParticleGroup.h.

21.67.5.4 float cParticleForGroup::Position[3]

Definition at line 43 of file WTcParticleGroup.h.

21.67.5.5 float cParticleForGroup::Size

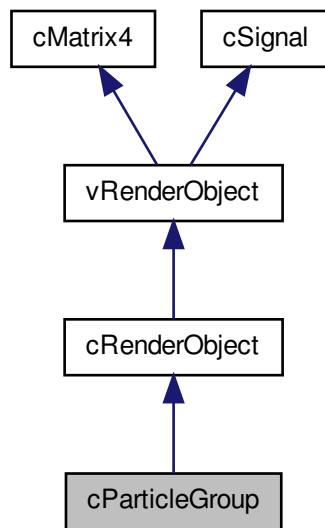
Definition at line 47 of file WTcParticleGroup.h.

21.67.5.6 float cParticleForGroup::Speed[3]

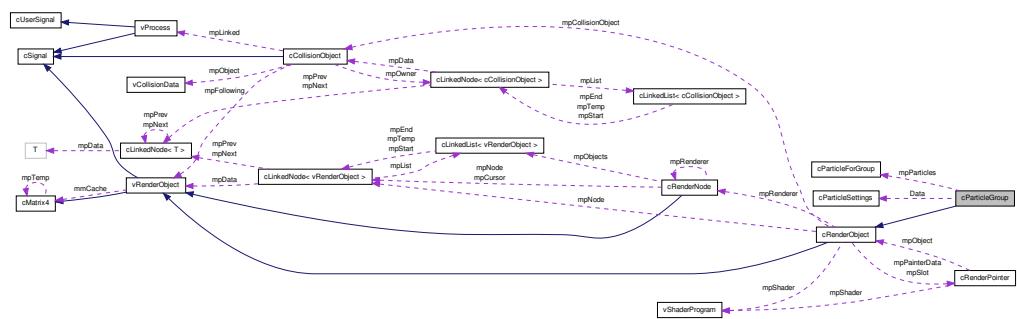
Definition at line 44 of file WTcParticleGroup.h.

21.68 cParticleGroup Class Reference

Inheritance diagram for cParticleGroup:



Collaboration diagram for cParticleGroup:



Public Member Functions

- `cParticleGroup` (`uint32 liParticles, cRenderNode *lpNode`)
 - `cParticleSettings & Settings ()`
 - `~cParticleGroup ()`
 - `void Settings (cParticleSettings &lpOther)`
 - `void RespawnAll ()`
 - `void RespawnOn ()`
 - `void Fade ()`
 - `void Refresh ()`
 - `void Render ()`
- virtual function to allow polymorphiss. see `cCamera::Render()`;*
- `void UseGravity ()`
 - `void NotUseGravity ()`
 - `void RenderPainter ()`
- virtual functions to allow polymorphism. see `cCamera::RenderPainter()`;*
- `void RenderPainter (uint8 liLevel)`
- virtual function to allow polymorphism. see `cCamera::RenderPainter()`;*
- `void RenderToPainter ()`
- virtual function to allow polymorphism. see `cCamera::RenderToPainter()`;*

21.68.1 Detailed Description

Definition at line 70 of file WTcParticleGroup.h.

21.68.2 Constructor & Destructor Documentation

21.68.2.1 `cParticleGroup::cParticleGroup (uint32 liParticles, cRenderNode * lpNode = 0)`

Definition at line 133 of file WTcParticleGroup.cpp.

21.68.2.2 `cParticleGroup::~cParticleGroup ()`

Definition at line 177 of file WTcParticleGroup.cpp.

21.68.3 Member Function Documentation

21.68.3.1 `void cParticleGroup::Fade () [inline]`

Definition at line 86 of file WTcParticleGroup.h.

21.68.3.2 void cParticleGroup::NotUseGravity() [inline]

Definition at line 91 of file WTcParticleGroup.h.

21.68.3.3 void cParticleGroup::Refresh()

Definition at line 4 of file WTcParticleGroup.cpp.

21.68.3.4 void cParticleGroup::Render() [virtual]

virtual function to allow polymorphiss. see [cCamera::Render\(\)](#);

Implements [cRenderObject](#).

Definition at line 98 of file WTcParticleGroup.cpp.

21.68.3.5 void cParticleGroup::RenderPainter() [virtual]

virtual functions to allow polymorphism. see [cCamera::RenderPainter\(\)](#);

Reimplemented from [cRenderObject](#).

Definition at line 38 of file WTcParticleGroup.cpp.

21.68.3.6 void cParticleGroup::RenderPainter(uint8 *liLevel*) [virtual]

virtual function to allow polymorphism. see [cCamera::RenderPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 32 of file WTcParticleGroup.cpp.

21.68.3.7 void cParticleGroup::RenderToPainter() [virtual]

virtual function to allow polymorphism. see [cCamera::RenderToPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 78 of file WTcParticleGroup.cpp.

21.68.3.8 void cParticleGroup::RespawnAll()

Definition at line 141 of file WTcParticleGroup.cpp.

21.68.3.9 void cParticleGroup::RespawnOn() [inline]

Definition at line 85 of file WTcParticleGroup.h.

21.68.3.10 cParticleSettings& cParticleGroup::Settings() [inline]

Definition at line 81 of file WTcParticleGroup.h.

21.68.3.11 void cParticleGroup::Settings(cParticleSettings & lpOther) [inline]

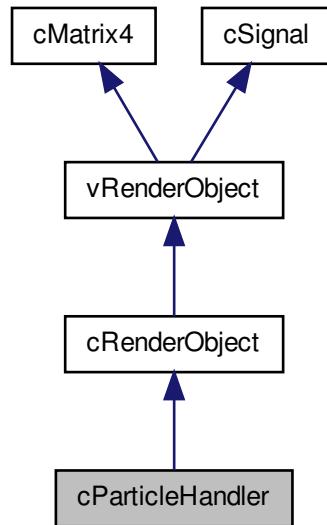
Definition at line 83 of file WTcParticleGroup.h.

21.68.3.12 void cParticleGroup::UseGravity() [inline]

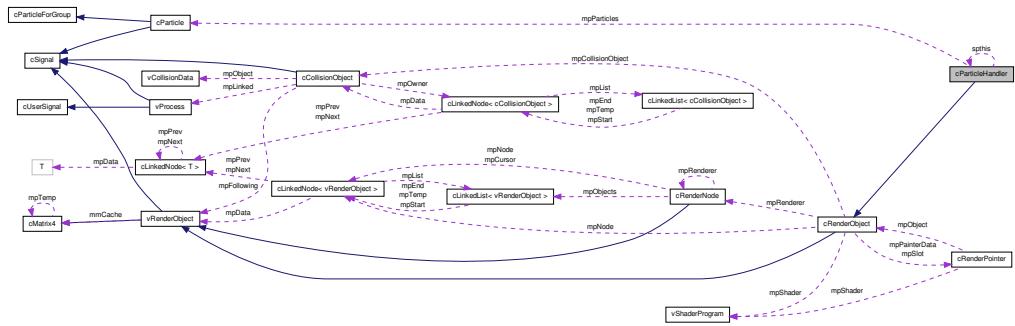
Definition at line 90 of file WTcParticleGroup.h.

21.69 cParticleHandler Class Reference

Inheritance diagram for cParticleHandler:



Collaboration diagram for cParticleHandler:



Public Member Functions

- void **Resize** (uint32 liSize)
- void **Add** (cParticle *lpPart)
- void **Remove** (cParticle *lpPart)
- void **Remove** (uint32 liParticle)
- void **Delete** (cParticle *lpPart)
- void **DeleteAll** ()
- void **Refresh** ()
- void **ForceRefresh** ()
- void **RenderToPainter** ()

virtual function to allow polymorphism. see cCamera::RenderToPainter();

- void **RenderPainter** (uint8 liLevel)

virtual function to allow polymorphism. see cCamera::RenderPainter();

- void **RenderPainter** ()

virtual functions to allow polymorphism. see cCamera::RenderPainter();

- void **Render** ()

virtual function to allow polymorphiss. see cCamera::Render();

Static Public Member Functions

- static cParticleHandler * **Instance** ()

Returns a pointer to the current cParticleHandler.

21.69.1 Detailed Description

This call will take all cParticleFree inheriting Particles (Inc [cGravityParticle](#) and [cWindAndGravityParticle](#)) and will deal with them as a block for efficiency and so the user does not need to track them. All coordinates must be converted to global coordinates when they are created. This will keep the overheads for particles low as they can be handled as a group. Particles can still be controlled by the Toplevel class. Particles will automatically be updated every frame if WT_PARTICLE_HANDLER_UPDATE_PARTICLE_POSITIONS is true. May be issues with time dependancy. Essentially this class will work in teh background and handle objects of types [cParticle](#),[cGravityParticle](#),[cWindAndGravityParticle](#).

Definition at line 59 of file WTcParticle.h.

21.69.2 Member Function Documentation

21.69.2.1 void cParticleHandler::Add ([cParticle](#) * *lpPart*)

Definition at line 68 of file WTcParticle.cpp.

21.69.2.2 void cParticleHandler::Delete ([cParticle](#) * *lpPart*)

Definition at line 75 of file WTcParticle.cpp.

21.69.2.3 void cParticleHandler::DeleteAll ()

Definition at line 56 of file WTcParticle.cpp.

21.69.2.4 void cParticleHandler::ForceRefresh () [inline]

Definition at line 86 of file WTcParticle.h.

21.69.2.5 [cParticleHandler](#) * cParticleHandler::Instance () [static]

Returns a pointer to the current [cParticleHandler](#).

Definition at line 41 of file WTcParticle.cpp.

21.69.2.6 void cParticleHandler::Refresh ()

Definition at line 99 of file WTcParticle.cpp.

21.69.2.7 void cParticleHandler::Remove ([cParticle](#) * *lpPart*)

Definition at line 89 of file WTcParticle.cpp.

21.69.2.8 void cParticleHandler::Remove (uint32 *liParticle*)

Definition at line 80 of file WTcParticle.cpp.

21.69.2.9 void cParticleHandler::Render () [virtual]

virtual function to allow polymorphiss. see [cCamera::Render\(\)](#);

Implements [cRenderObject](#).

Definition at line 163 of file WTcParticle.cpp.

21.69.2.10 void cParticleHandler::RenderPainter (uint8 *liLevel*) [virtual]

virtual function to allow polymorphism. see [cCamera::RenderPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 142 of file WTcParticle.cpp.

21.69.2.11 void cParticleHandler::RenderPainter () [virtual]

virtual functions to allow polymorphism. see [cCamera::RenderPainter\(\)](#);

Reimplemented from [cRenderObject](#).

Definition at line 122 of file WTcParticle.cpp.

21.69.2.12 void cParticleHandler::RenderToPainter () [virtual]

virtual function to allow polymorphism. see [cCamera::RenderToPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 149 of file WTcParticle.cpp.

21.69.2.13 void cParticleHandler::Resize (uint32 *liSize*)

Definition at line 48 of file WTcParticle.cpp.

21.70 cParticleSettings Class Reference

Public Member Functions

- [cParticleSettings & operator= \(cParticleSettings &lpOther\)](#)
- [void SetColours \(float *lpRGB\)](#)
- [void SetSizes \(float *lpSize\)](#)
- [void SetFades \(float *lpFade\)](#)

- void [SetSpeeds](#) (float *lpSpeed)
- void [SetPositions](#) (float *lpPos)

Public Attributes

- float [RGB](#) [8]
- float [Speed](#) [6]
- float [Fade](#) [2]
- float [Position](#) [6]
- float [Size](#) [2]

21.70.1 Detailed Description

Definition at line 4 of file WTcParticleGroup.h.

21.70.2 Member Function Documentation

21.70.2.1 `cParticleSettings& cParticleSettings::operator= (cParticleSettings & lpOther)` [inline]

Definition at line 13 of file WTcParticleGroup.h.

21.70.2.2 `void cParticleSettings::SetColours (float * lpRGB)` [inline]

Definition at line 23 of file WTcParticleGroup.h.

21.70.2.3 `void cParticleSettings::SetFades (float * lpFade)` [inline]

Definition at line 25 of file WTcParticleGroup.h.

21.70.2.4 `void cParticleSettings::SetPositions (float * lpPos)` [inline]

Definition at line 27 of file WTcParticleGroup.h.

21.70.2.5 `void cParticleSettings::SetSizes (float * lpSize)` [inline]

Definition at line 24 of file WTcParticleGroup.h.

21.70.2.6 `void cParticleSettings::SetSpeeds (float * lpSpeed)` [inline]

Definition at line 26 of file WTcParticleGroup.h.

21.70.3 Member Data Documentation

21.70.3.1 float cParticleSettings::Fade[2]

Definition at line 9 of file WTcParticleGroup.h.

21.70.3.2 float cParticleSettings::Position[6]

Definition at line 10 of file WTcParticleGroup.h.

21.70.3.3 float cParticleSettings::RGB[8]

Definition at line 7 of file WTcParticleGroup.h.

21.70.3.4 float cParticleSettings::Size[2]

Definition at line 11 of file WTcParticleGroup.h.

21.70.3.5 float cParticleSettings::Speed[6]

Definition at line 8 of file WTcParticleGroup.h.

21.71 cPlane Class Reference

A class for handling plane data for 3D mesh objects. Is composed of 4 floats, X,Y,Z,N. X,Y,Z components of the Normalised normal vector and distance above the origin along the line of the Normal.

Public Member Functions

- float **X** ()
Returns the X component of the Planes Normal.
- float **Y** ()
Returns the Y component of the Planes Normal.
- float **Z** ()
Returns the Z component of the Planes Normal.
- float **Dist** ()
Return the length of the Planes Normal.
- void **SetPlane** (float x, float y, float z, float dist)

Sets the values for the plane. The Normal vector should be normalised.

- float * **Planes** ()

Will return a pointer to the float array (Normalvector and length, X,Y,Z,L) which stores the data for this object.
- **cPlane** ()

cPlane ()
- void **Display** ()

Will Display this object to the terminal.
- void **GeneratePlane** (cVertex *lpVerteces)

GeneratePlane (cVertex *lpVerteces)
- uint32 **FileSize** ()

FileSize ()
- bool **operator==** (cPlane &lpOther)

operator== (cPlane &lpOther)
- **cPlane & operator=** (cPlane &lpOther)

*Will set this object equal to the **cPlane** lpOther.*
- **cFullFaceData & operator=** (cFullFaceData &lpOther)

*Will set this plane equal to the Plane of **cFullFaceData**.*
- void **Normalise** ()

Will Normalise the Normal Vector of this object. This ignores the Length set.
- bool **Similar** (cPlane &lpOther, float lfRange)

*Will return true if all values (X,Y,Z,L) of this **cPlane** and the **cPlane** lpOther are within +/- the range lfRange.*
- double **DotProduct** (cVertex &lpOther)

*Will Dot prod<cPolygon>uct the vector **cVertex** with the Normal Vector of this plane.*
- bool **AbovePlane** (cVertex &lpOther)

*Will return true if the 3D Position lpOther is in front of the facing of this **cPlane**.*
- float **Distance** ()

Will Return the Distance this Plane sits from the origin. (Can be +ve or -ve)
- double **AbsoluteDistance** (cPlane &lpOther)

Returns the distance between the points on the two planes that lie on their normal vectors from the origin.
- void **OutputIMFPlane** (ofstream &FileStream)

Will Output this Object to ofstream in the IMF format.
- void **LoadIMFPlane** (ifstream &FileStream)

*Will Load a **cPlane** from ifstream FileStream in the IMF format.*

21.71.1 Detailed Description

A class for handling plane data for 3D mesh objects. Is composed of 4 floats, X,Y,Z,N. X,Y,Z components of the Normalised normal vector and distance above the origin along the line of the Normal.

Definition at line 7 of file WTcPlane.h.

21.71.2 Constructor & Destructor Documentation

21.71.2.1 `cPlane::cPlane()`

Definition at line 129 of file WTcPlane.cpp.

21.71.3 Member Function Documentation

21.71.3.1 `bool cPlane::AbovePlane(cVertex & lpOther)`

Will return true if the 3D Position lpOther is in front of the faceing of this `cPlane`.

Definition at line 161 of file WTcPlane.cpp.

21.71.3.2 `double cPlane::AbsoluteDistance(cPlane & lpOther)`

Returns the distance between the points on the two planes that lie on their normal vectors from the origin.

Definition at line 50 of file WTcPlane.cpp.

21.71.3.3 `void cPlane::Display()`

Will Display this object to the terminal.

Definition at line 131 of file WTcPlane.cpp.

21.71.3.4 `float cPlane::Dist()`

Return the length of the Planes Normal.

Definition at line 123 of file WTcPlane.cpp.

21.71.3.5 `float cPlane::Distance()`

Will Return the Distance this Plane sits from the origin. (Can be +ve or -ve)

Definition at line 167 of file WTcPlane.cpp.

21.71.3.6 double cPlane::DotProduct (cVertex & lpOther)

Will Dot product the vector `cVertex` with the Normal Vector of this plane.
Definition at line 156 of file WTcPlane.cpp.

21.71.3.7 uint32 cPlane::FileSize ()

Definition at line 136 of file WTcPlane.cpp.

21.71.3.8 void cPlane::GeneratePlane (cVertex * lpVertices)

Definition at line 4 of file WTcPlane.cpp.

21.71.3.9 void cPlane::LoadIMFPlane (ifstream & FileStream)

Will Load a `cPlane` from ifstream FileStream in the IMF format.
Definition at line 176 of file WTcPlane.cpp.

21.71.3.10 void cPlane::Normalise ()

Will Normalise the Normal Vector of this object. This ignores the Length set.
Definition at line 31 of file WTcPlane.cpp.

21.71.3.11 cPlane & cPlane::operator= (cPlane & lpOther)

Will set this object equal to the `cPlane` lpOther.
Definition at line 146 of file WTcPlane.cpp.

21.71.3.12 cFullFaceData & cPlane::operator= (cFullFaceData & lpOther)

Will set this plane equal to the Plane of `cFullFaceData`.
Definition at line 107 of file WTcPlane.cpp.

21.71.3.13 bool cPlane::operator== (cPlane & lpOther)

Definition at line 141 of file WTcPlane.cpp.

21.71.3.14 void cPlane::OutputIMFPlane (ofstream & FileStream)

Will Output this Object to ofstream in the IMF format.
Definition at line 171 of file WTcPlane.cpp.

21.71.3.15 float * cPlane::Planes ()

Will return a pointer to the float array (Normalvector and length, X,Y,Z,L) which stores the data for this object.

Definition at line 127 of file WTcPlane.cpp.

21.71.3.16 void cPlane::SetPlane (float x, float y, float z, float dist)

Sets the values for the plane. The Normal vector should be normalised.

Definition at line 125 of file WTcPlane.cpp.

21.71.3.17 bool cPlane::Similar (cPlane & lpOther, float lfRange)

Will return true if all values (X,Y,Z,L) of this [cPlane](#) and the [cPlane](#) lpOther are within +/- the range lfRange.

Definition at line 41 of file WTcPlane.cpp.

21.71.3.18 float cPlane::X ()

Returns the X component of the Planes Normal.

Definition at line 120 of file WTcPlane.cpp.

21.71.3.19 float cPlane::Y ()

Returns the Y component of the Planes Normal.

Definition at line 121 of file WTcPlane.cpp.

21.71.3.20 float cPlane::Z ()

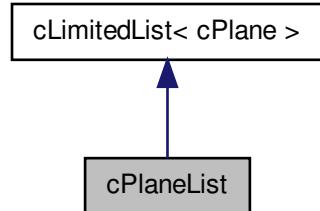
Returns the Z component of the Planes Normal.

Definition at line 122 of file WTcPlane.cpp.

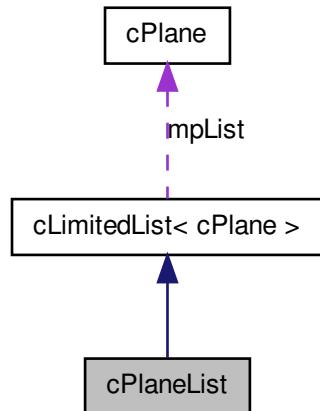
21.72 cPlaneList Class Reference

A resizable array of [cPlane](#) Objects. Stores and Handles a list of [cPlane](#) objects using [cLimitedList](#).

Inheritance diagram for cPlaneList:



Collaboration diagram for cPlaneList:



Public Member Functions

- [cPlaneList \(\)](#)
- [~cPlaneList \(\)](#)
- [void Display \(\)](#)

Display this object to the Terminal.

- void [OutputIMFPlanes](#) (ofstream &FileStream)
Will Output the list of planes to ofstream FileStream in the IMF format.
- void [LoadIMFPlanes](#) (ifstream &FileStream)
Will Load the list of planes from ifstream FileStream in the IMF format.
- uint32 [FileSize](#) ()
Will return the size of this object in Bytes when written in IMF format.

21.72.1 Detailed Description

A resizable array of [cPlane](#) Objects. Stores and Handles a list of [cPlane](#) objects using [cLimitedList](#).

Definition at line 72 of file WTcPlane.h.

21.72.2 Constructor & Destructor Documentation

21.72.2.1 [cPlaneList::cPlaneList\(\)](#) [inline]

Definition at line 75 of file WTcPlane.h.

21.72.2.2 [cPlaneList::~cPlaneList\(\)](#) [inline]

Definition at line 76 of file WTcPlane.h.

21.72.3 Member Function Documentation

21.72.3.1 [void cPlaneList::Display\(\)](#)

Display this object to the Terminal.

Definition at line 64 of file WTcPlane.cpp.

21.72.3.2 [uint32 cPlaneList::FileSize\(\)](#)

Will return the size of this object in Bytes when written in IMF format.

Definition at line 95 of file WTcPlane.cpp.

21.72.3.3 [void cPlaneList::LoadIMFPlanes\(ifstream & FileStream \)](#)

Will Load the list of planes from ifstream FileStream in the IMF format.

Definition at line 84 of file WTcPlane.cpp.

21.72.3.4 void cPlaneList::OutputIMFPlanes (ostream & *FileStream*)

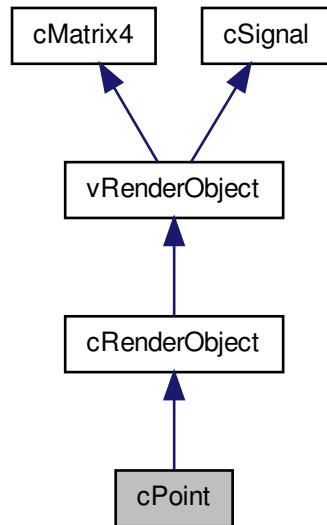
Will Output the list of planes to ofstream FileStream in the IMF format.

Definition at line 73 of file WTcPlane.cpp.

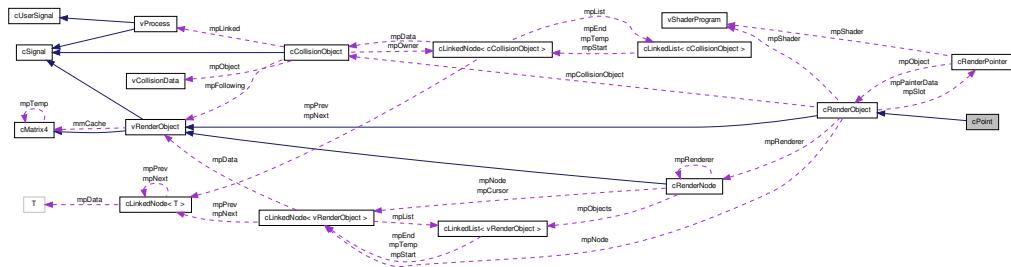
21.73 cPoint Class Reference

A Renderable object for rendering single points.

Inheritance diagram for cPoint:



Collaboration diagram for cPoint:



Public Member Functions

- **cPoint ()**
cPoint constructor
- **cPoint (cRenderNode *lpRenderer)**
cPoint constructor. Will be owned by lpRenderer.
- **void RenderPainter (uint8 liLevel)**
virtual function to allow polymorphism. see cCamera::RenderPainter();
- **void RenderToPainter ()**
virtual function to allow polymorphism. see cCamera::RenderToPainter();
- **void Render ()**
virtual function to allow polymorphiss. see cCamera::Render();
- **float * Color ()**
Will return the color of this point object.
- **void Color (float *lfColor)**
Will set the color of this point object. Expects 4 floats (RGBA).
- **void Color (float lfR, float lfG, float lfB, float lfA)**
Will set the color of this point object. (RGBA).
- **float PointSize ()**
Will return the size of this point in pixels (Distance does not affect size without using an appropriate cShaderProgram).
- **void PointSize (float lfPointSize)**
Will set the size of this point in pixels.

21.73.1 Detailed Description

A Renderable object for rendering single points.

Definition at line 6 of file WTcPoint.h.

21.73.2 Constructor & Destructor Documentation

21.73.2.1 `cPoint::cPoint()`

`cPoint` constructor

Definition at line 4 of file WTcPoint.cpp.

21.73.2.2 `cPoint::cPoint(cRenderNode * lpRenderer)`

`cPoint` constructor. Will be owned by lpRenderer.

Definition at line 13 of file WTcPoint.cpp.

21.73.3 Member Function Documentation

21.73.3.1 `float * cPoint::Color()`

Will return the color of this point object.

Definition at line 76 of file WTcPoint.cpp.

21.73.3.2 `void cPoint::Color(float * lfColor)`

Will set the color of this point object. Expects 4 floats (RGBA).

Definition at line 71 of file WTcPoint.cpp.

21.73.3.3 `void cPoint::Color(float lfR, float lfG, float lfB, float lfA)`

Will set the color of this point object. (RGBA).

Definition at line 63 of file WTcPoint.cpp.

21.73.3.4 `void cPoint::PointSize(float lfPointSize)`

Will set the size of this point in pixels.

Definition at line 81 of file WTcPoint.cpp.

21.73.3.5 float cPoint::PointSize() [inline]

Will return the size of this point in pixels (Distance does not affect size without using an appropriate [cShaderProgram](#)).

Definition at line 31 of file WTcPoint.h.

21.73.3.6 void cPoint::Render() [virtual]

virtual function to allow polymorphiss. see [cCamera::Render\(\)](#);

Implements [cRenderObject](#).

Definition at line 46 of file WTcPoint.cpp.

21.73.3.7 void cPoint::RenderPainter(uint8 liLevel) [virtual]

virtual function to allow polymorphism. see [cCamera::RenderPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 19 of file WTcPoint.cpp.

21.73.3.8 void cPoint::RenderToPainter() [virtual]

virtual function to allow polymorphism. see [cCamera::RenderToPainter\(\)](#);

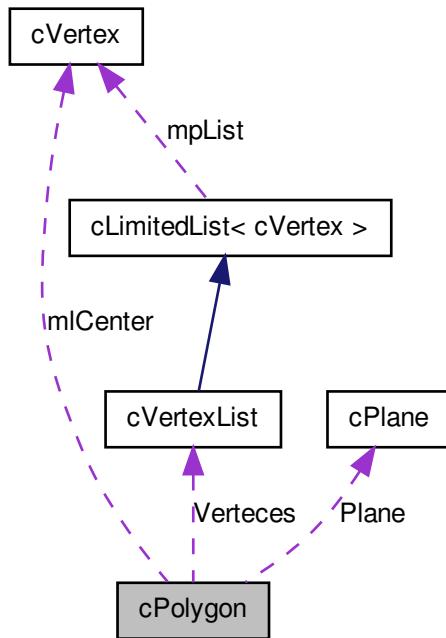
Implements [cRenderObject](#).

Definition at line 32 of file WTcPoint.cpp.

21.74 cPolygon Class Reference

This class stores Polygon Data for 3D Meshes. A Polygon is a single convex boundaried Face in a single Plane with any number of verteces above 3. cFaces which are sharing an edge and in the same plane can be combined into a single [cPolygon](#). This allows models to be represented with less Planes for the purposes of calculating Collisions. Uses [cPlane](#) and [cVertex](#) Objects.

Collaboration diagram for cPolygon:



Public Member Functions

- `cPolygon ()`
- `cPolygon (uint32 liSpaces)`

Constructor, creates a `cPolygon` with enough spaces for liSpaces cVertices.

- `~cPolygon ()`
- `uint32 FileSize ()`
- `cVertex & operator[] (uint8 liPos)`

Will Return `cVertex` number liPos.

- `cPolygon & operator= (cPolygon &lpOther)`

Will make this equal to another `cPolygon` object lpOther.

- `cPlane PlaneData ()`

Will return the `cPlane` object storing the plane for this object.

- void [Init](#) (uint32 liSize)
- void [GeneratePolygon](#) ([cFullFaceData](#) &lpFace)
- void [AddVertex](#) ([cVertex](#) *lpOther)
- void [RemoveVertex](#) (uint32 liPos)
- uint8 [SharesVertex](#) ([cFullFaceData](#) &lpOther)

Returns the number of Verteces that are shared with the [cFullFaceData](#) object lpOther.

- uint32 [SharesVertex](#) ([cPolygon](#) &lpOther)

Returns the number of Verteces that are shared with the [cPolygon](#) object lpOther.

- bool [SharesVertex](#) ([cVertex](#) &lpOther)

Returns the true if [cVertex](#) lpOther is the same as any vertex in this object.

- uint8 [NotSharedVertex](#) ([cFullFaceData](#) &lpOther)

Returns the position (0,1,2) of the Vertex which is NOT shared with this object.

- void [OrderClockwise](#) ()

- float [Area](#) ()

Will return the Area of this object.

- void [OutputIMFPolygon](#) (ofstream &FileStream)

Will Output this object to the ofstream FileStream in IMF format.

- uint32 [GetSmallestAngle](#) (uint32 liStart)

- double [GetAngleSum](#) (float *lpPos)

*Will return the sum of angles from the point lpPos (Array of 3 float 3D Position, X,Y,Z).
if this is 2*pi or very close the point is on the plane and within the boundaries of the [cPolygon](#).*

- void [CalculateCenter](#) ()

Will Calculate the Center point from all the [cVertex](#) objects in the List.

- [cVertex](#) [Center](#) ()

Will return the [cVertex](#) storing the Central point of the polygon. This will be on the plane.

- void [LoadIMFPolygon](#) (ifstream &FileStream)

Will Load this object in IMF format from the ifstream FileStream.

- void [Display](#) ()

Will Display this object top the screen.

Public Attributes

- [cPlane](#) [Plane](#)
- [cVertexList](#) [Verteces](#)
- [cVertex](#) [mCenter](#)

21.74.1 Detailed Description

This class stores Polygon Data for 3D Meshes. A Polygon is a single convex bound-
aried Face in a single Plane with any number of vertices above 3. cFaces which are
sharing an edge and in the same plane can be combined into a single [cPolygon](#). This
allows models to be represented with less Planes for the purposes of calculating Colli-
sions. Uses [cPlane](#) and [cVertex](#) Objects.

Definition at line 8 of file WTcPolygon.h.

21.74.2 Constructor & Destructor Documentation

21.74.2.1 [cPolygon::cPolygon\(\)](#)

Definition at line 427 of file WTcPolygon.cpp.

21.74.2.2 [cPolygon::cPolygon\(uint32 liSpaces \)](#)

Constructor, creates a [cPolygon](#) with enough spaces for liSpaces cVertices.

Definition at line 429 of file WTcPolygon.cpp.

21.74.2.3 [cPolygon::~cPolygon\(\)](#)

Definition at line 434 of file WTcPolygon.cpp.

21.74.3 Member Function Documentation

21.74.3.1 [void cPolygon::AddVertex\(cVertex * lpOther \)](#)

Definition at line 423 of file WTcPolygon.cpp.

21.74.3.2 [float cPolygon::Area\(\)](#)

Will return the Area of this object.

Definition at line 20 of file WTcPolygon.cpp.

21.74.3.3 [void cPolygon::CalculateCenter\(\)](#)

Will Calculate the Center point from all the [cVertex](#) objects in the List.

Definition at line 239 of file WTcPolygon.cpp.

21.74.3.4 cVertex cPolygon::Center ()

Will return the [cVertex](#) storing the Central point of the polygon. This will be on the plane.

Definition at line 409 of file WTcPolygon.cpp.

21.74.3.5 void cPolygon::Display ()

Will Display this object top the screen.

Definition at line 417 of file WTcPolygon.cpp.

21.74.3.6 uint32 cPolygon::FileSize ()

Definition at line 3 of file WTcPolygon.cpp.

21.74.3.7 void cPolygon::GeneratePolygon (cFullFaceData & lpFace)

Definition at line 182 of file WTcPolygon.cpp.

21.74.3.8 double cPolygon::GetAngleSum (float * lpPos)

Will return the sum of angles from the point lpPos (Array of 3 float 3D Position, X,Y,Z). if this is 2*pi or very close the point is on the plane and within the boundaries of the [cPolygon](#).

Definition at line 110 of file WTcPolygon.cpp.

21.74.3.9 uint32 cPolygon::GetSmallestAngle (uint32 liStart)

Definition at line 73 of file WTcPolygon.cpp.

21.74.3.10 void cPolygon::Init (uint32 liSize)

Definition at line 176 of file WTcPolygon.cpp.

21.74.3.11 void cPolygon::LoadIMFPolygon (ifstream & FileStream)

Will Load this object in IMF format from the ifstream FileStream.

Definition at line 411 of file WTcPolygon.cpp.

21.74.3.12 uint8 cPolygon::NotSharedVertex (cFullFaceData & lpOther)

Returns the position (0,1,2) of the Vertex which is NOT shared with this object.

Definition at line 220 of file WTcPolygon.cpp.

21.74.3.13 `cPolygon & cPolygon::operator= (cPolygon & lpOther)`

Will make this equal to another `cPolygon` object lpOther.

Definition at line 12 of file WTcPolygon.cpp.

21.74.3.14 `cVertex & cPolygon::operator[] (uint8 liPos)`

Will Return `cVertex` number liPos.

Definition at line 426 of file WTcPolygon.cpp.

21.74.3.15 `void cPolygon::OrderClockwise ()`

Definition at line 325 of file WTcPolygon.cpp.

21.74.3.16 `void cPolygon::OutputIMFPolygon (ostream & FileStream)`

Will Output this object to the ofstream FileStream in IMF format.

Definition at line 67 of file WTcPolygon.cpp.

21.74.3.17 `cPlane cPolygon::PlaneData ()`

Will return the `cPlane` object storing the plane for this object.

Definition at line 425 of file WTcPolygon.cpp.

21.74.3.18 `void cPolygon::RemoveVertex (uint32 liPos)`

Definition at line 424 of file WTcPolygon.cpp.

21.74.3.19 `uint32 cPolygon::SharesVertex (cPolygon & lpOther)`

Returns the number of Verteces that are shared with the `cPolygon` object lpOther.

Definition at line 206 of file WTcPolygon.cpp.

21.74.3.20 `uint8 cPolygon::SharesVertex (cFullFaceData & lpOther)`

Returns the number of Verteces that are shared with the `cFullFaceData` object lpOther.

Definition at line 193 of file WTcPolygon.cpp.

21.74.3.21 bool cPolygon::SharesVertex (cVertex & lpOther)

Returns the true if `cVertex` lpOther is the same as any vertex in this object.

Definition at line 152 of file WTcPolygon.cpp.

21.74.4 Member Data Documentation**21.74.4.1 cVertex cPolygon::mlCenter**

Definition at line 14 of file WTcPolygon.h.

21.74.4.2 cPlane cPolygon::Plane

Definition at line 12 of file WTcPolygon.h.

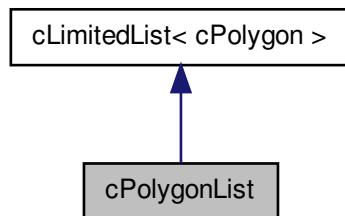
21.74.4.3 cVertexList cPolygon::Verteces

Definition at line 13 of file WTcPolygon.h.

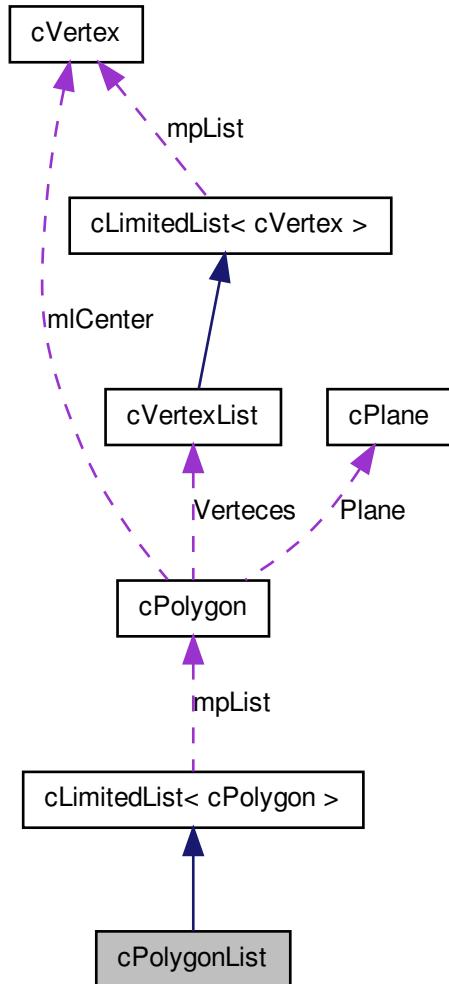
21.75 cPolygonList Class Reference

An array of `cPolygon` Objects. Creates an array of `cPolygon` objects using the class `cLimitedList`.

Inheritance diagram for cPolygonList:



Collaboration diagram for cPolygonList:



Public Member Functions

- `cPolygonList ()`
- `~cPolygonList ()`
- `void GeneratePolygonList (cFullFaceList &lpFaces)`
- `void OptimiseRays ()`

Will Optimise the order of the Polygons for collisions. First it will [Combine\(\)](#) objects to reduce the number of planes. It will sort the Verteces into CW order and then Call [OrderArea\(\)](#) to optimise for Beam / Ray collisions.

- void [OptimiseMeshes \(\)](#)

Will Optimise the order of the Polygons for collisions. First it will [Combine\(\)](#) objects to reduce the number of planes. It will sort the Verteces into CW order and then Call [OrderDistance\(\)](#) to optimise for Mesh Collsions.

- void [OrderDistance \(\)](#)
- void [OrderArea \(\)](#)
- void [OrderVerteces \(\)](#)
- void [Combine \(\)](#)
- void [OutputIMFPolygons \(ostream &FileStream\)](#)

Will output all the cPolygons in the list to the ostream FileStream.

- void [LoadIMFPolygons \(ifstream &FileStream\)](#)

Will Load a [cPolygon](#) list from the ifstream FileStream.

- uint32 [FileSize \(\)](#)
- void [Display \(\)](#)

Will Display the list of cPolygons to the terminal.

21.75.1 Detailed Description

An array of [cPolygon](#) Objects. Creates an array of [cPolygon](#) objects using the class [cLimitedList](#).

Definition at line 86 of file WTcPolygon.h.

21.75.2 Constructor & Destructor Documentation

21.75.2.1 [cPolygonList::cPolygonList \(\)](#)

Definition at line 440 of file WTcPolygon.cpp.

21.75.2.2 [cPolygonList::~cPolygonList \(\)](#)

Definition at line 441 of file WTcPolygon.cpp.

21.75.3 Member Function Documentation

21.75.3.1 [void cPolygonList::Combine \(\)](#)

Definition at line 263 of file WTcPolygon.cpp.

21.75.3.2 void cPolygonList::Display()

Will Display the list of cPolygons to the terminal.

Definition at line 399 of file WTcPolygon.cpp.

21.75.3.3 uint32 cPolygonList::FileSize()

Definition at line 387 of file WTcPolygon.cpp.

21.75.3.4 void cPolygonList::GeneratePolygonList(cFullFaceList & lpFaces)

Definition at line 160 of file WTcPolygon.cpp.

21.75.3.5 void cPolygonList::LoadIMFPolygons(ifstream & FileStream)

Will Load a [cPolygon](#) list from the ifstream FileStream.

Definition at line 375 of file WTcPolygon.cpp.

21.75.3.6 void cPolygonList::OptimiseMeshes()

Will Optimise the order of the Polygons for collisions. First it will [Combine\(\)](#) objects to reduce the number of planes. It will sort the Verteces into CW order and then Call [OrderDistance\(\)](#) to optimise for Mesh Collsions.

Definition at line 345 of file WTcPolygon.cpp.

21.75.3.7 void cPolygonList::OptimiseRays()

Will Optimise the order of the Polygons for collisions. First it will [Combine\(\)](#) objects to reduce the number of planes. It will sort the Verteces into CW order and then Call [OrderArea\(\)](#) to optimise for Beam / Ray collisions.

Definition at line 338 of file WTcPolygon.cpp.

21.75.3.8 void cPolygonList::OrderArea()

Definition at line 307 of file WTcPolygon.cpp.

21.75.3.9 void cPolygonList::OrderDistance()

Definition at line 291 of file WTcPolygon.cpp.

21.75.3.10 void cPolygonList::OrderVerteces()

Definition at line 354 of file WTcPolygon.cpp.

21.75.3.11 void cPolygonList::OutputIMFPolygons(ostream & FileStream)

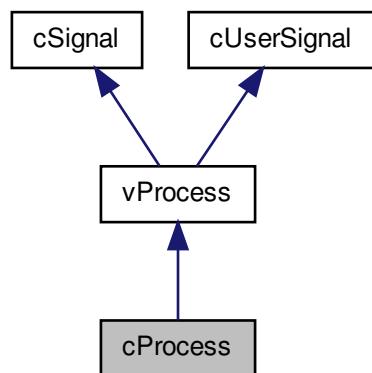
Will output all the cPolygons in the list to the ostream FileStream.

Definition at line 365 of file WTcPolygon.cpp.

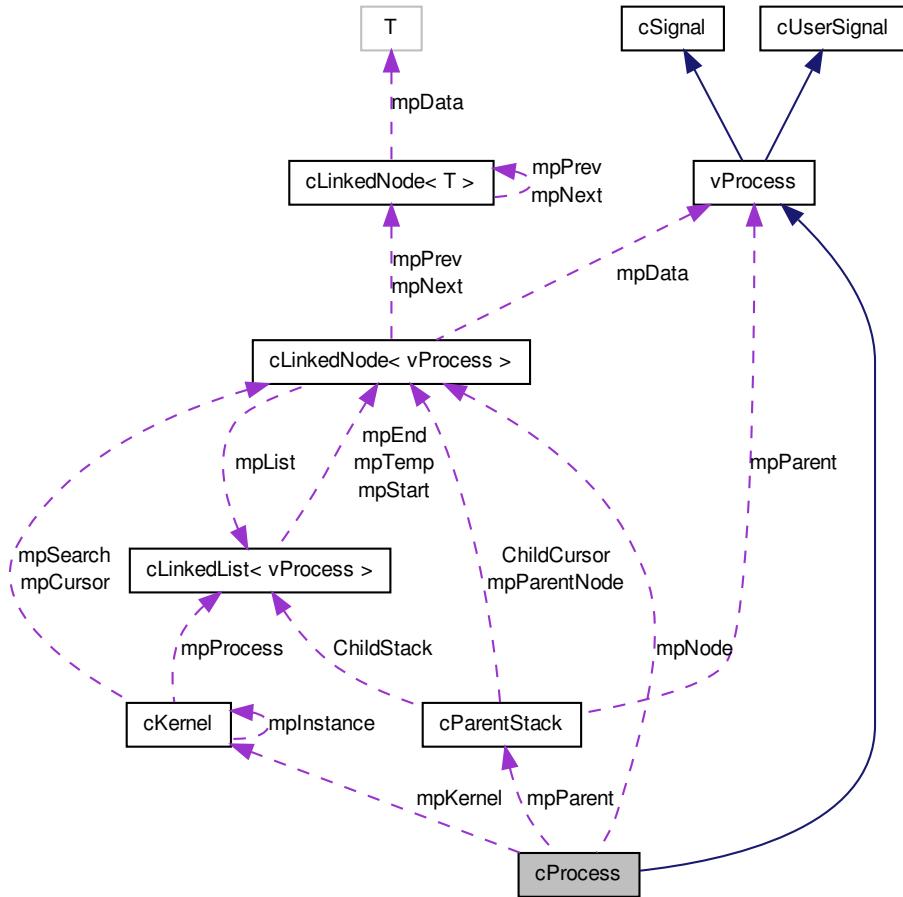
21.76 cProcess Class Reference

This is the base code for a process. This will automatically create a new process. It will hand itself to [cKernel](#) to be processed every frame. Any Processes created by the user should inherit this type to be handled by [cKernel](#) automatically. Initialisation code should go in the constructor of the user type. Linking to [cKernel](#) is performed automatically by [cProcess](#). Update code should go in the function [Run\(\)](#). Destruction code should go in the destructor of the user type. Code to handle interaction of two processes should go in either processes [UserSignal\(\)](#) function.

Inheritance diagram for cProcess:



Collaboration diagram for cProcess:



Public Member Functions

- virtual void `Run ()=0`

This Function stores the code which will update this process by a single frame. This user defined function is the single most important function for any `cProcess` object. Every object which inherits `cProcess` MUST define this function. This Function will be called on every currently running process every frame. This should contain the code which tells the Process how to update every frame. It should not contain initialisation or deconstruction code. It is likely to direct the `cRenderableObjects` which represent this process. It should contain AI. I should process the state of the object. It should control generation of `cSignal` calls and search for collisions as appropriate.

- void [Signal \(SIGNAL lbSignal\)](#)

This is an engine defined inter-Process Signal. It will change the run state of the current [cProcess](#) by lbSignal.

- void [KillAll \(\)](#)

This Function will kill all currently running [cProcess](#) and start the end of the program.

- virtual void [UserSignal \(SIGNAL lsSignal, void *lpData\)](#)

This is a user defined function allowing the user to add Process specific inter-process signals. This Function should include the code for processing the signal as there is no signal buffer. It is useful to ensure a single detection and activation of process interactions. One Process shoudl detect the interaction and signal the other to respond. This way the order of Process evaluation is unimportant.

- void [AdditionalKillFunctionality \(\)](#)

Virtual Functions to allow additional commands to be processed when a kill signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

- void [AdditionalSleepFunctionality \(\)](#)

Virtual Functions to allow additional commands to be processed when a sleep signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

- void [AdditionalWakeFunctionality \(\)](#)

Virtual Functions to allow additional commands to be processed when a wake signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

- [cProcess \(\)](#)

cProcess Constructor. Will Initialise all generic [cProcess](#) variables and add the current process to the process list held by [cKernel](#).

- [cParentStack * ParentStack \(\)](#)

Will return a pointer to the [cParentStack](#) owned by this process.

Protected Member Functions

- [~cProcess \(\)](#)

cProcess destructor. Should not be called by User. Will delete all generic [cProcess](#) variables and remove the current process from the process list held by ckernel. This should only be called by [cKernel](#). Instead use _KILL() or Signal(_S_KILL).

Protected Attributes

- [cKernel * mpKernel](#)

Friends

- class [cKernel](#)

21.76.1 Detailed Description

This is the base code for a process. This will automatically create a new process. It will hand itself to [cKernel](#) to be processed every frame. Any Processes created by the user should inherit this type to be handled by [cKernel](#) automatically. Initialisation code should go in the constructor of the user type. Linking to [cKernel](#) is performed automatically by [cProcess](#). Update code should go in the function [Run\(\)](#). Destruction code should go in the destructor of the user type. Code to handle interaction of two processes should go in either processes [UserSignal\(\)](#) function.

```
class cUserProcess : cProcess
{
    cTexturedModel *Model;
    int32 LifeSpan;
public:

    cUserProcess()
    {
        //Initialise the Process LifeSpan
        LifeSpan=100;
        //Create a Model for this process.
        Model=new cTexturedModel;
        //Give the Model some media to use.
        Model->Mesh(_GET_FILE(vMesh*, "UserMesh"));
        Model->Texture(_GET_FILE(vTexture*, "UserTexture"));
        Model->Shader(_GET_SHADER("UserShaderProgram"));
    };

    Run()
    {
        //Move the Model in the direction it is facing.
        Model->Advance(0.1f);
        //Reduce the remaining LifeSpan of this object.
        --LifeSpan;
        //If the object is out of Lifespan it is dead.
        if(!LifeSpan) _KILL(this);
    };

    ~cUserProcess()
    {
        //Clean Up the Object.
        _KILL(MODEL);
    };
}
```

Definition at line 49 of file [WTcProcess.h](#).

21.76.2 Constructor & Destructor Documentation

21.76.2.1 `cProcess::cProcess()`

`cProcess` Constructor. Will Initialise all generic `cProcess` variables and add the current process to the process list held by `cKernel`.

Definition at line 7 of file WTcProcess.cpp.

21.76.2.2 `cProcess::~cProcess() [protected]`

`cProcess` destructor. Should not be called by User. Will delete all generic `cProcess` variables and remove the current process from the process list held by `cKernel`. This should only be called by `cKernel`. Instead use `_KILL()` or `Signal(_S_KILL)`.

Definition at line 20 of file WTcProcess.cpp.

21.76.3 Member Function Documentation

21.76.3.1 `void cProcess::AdditionalKillFunctionality() [inline, virtual]`

Virtual Functions to allow additional commands to be processed when a kill signal is received by an object. This can be user modified for classes inheriting `cProcess`.

Implements `vProcess`.

Definition at line 87 of file WTcProcess.h.

21.76.3.2 `void cProcess::AdditionalSleepFunctionality() [inline, virtual]`

Virtual Functions to allow additional commands to be processed when a sleep signal is received by an object. This can be user modified for classes inheriting `cProcess`.

Implements `vProcess`.

Definition at line 88 of file WTcProcess.h.

21.76.3.3 `void cProcess::AdditionalWakeFunctionality() [inline, virtual]`

Virtual Functions to allow additional commands to be processed when a wake signal is received by an object. This can be user modified for classes inheriting `cProcess`.

Implements `vProcess`.

Definition at line 89 of file WTcProcess.h.

21.76.3.4 `void cProcess::KillAll()`

This Function will kill all currently running `cProcess` and start the end of the program.

Definition at line 26 of file WTcProcess.cpp.

21.76.3.5 cParentStack * cProcess::ParentStack() [virtual]

Will return a pointer to the [cParentStack](#) owned by this process.

Implements [vProcess](#).

Definition at line 46 of file WTcProcess.cpp.

21.76.3.6 virtual void cProcess::Run() [pure virtual]

This Function stores the code which will update this process by a single frame. This user defined function is the single most important function for any [cProcess](#) object. Every object which inherits [cProcess](#) MUST define this function. This Function will be called on every currently running process every frame. This should contain the code which tells the Process how to update every frame. It should not contain initialisation or deconstruction code. It is likely to direct the [cRenderableObjects](#) which represent this process. It should contain AI. I should process the state of the object. It should control generation of [cSignal](#) calls and search for collisions as appropriate.

Implements [vProcess](#).

21.76.3.7 void cProcess::Signal (SIGNAL lbSignal) [virtual]

This is an engine defined inter-Process Signal. It will change the run state of the current [cProcess](#) by lbSignal.

Parameters

<i>lbSignal</i>	lbSignal hands the type of signal to send. Applicable signal types are listed in WTcFlags.h . This function will change the run state of the current cProcess . It allows other processes to force this process to _S_SLEEP or _S_WAKE or _S_KILL. Any System State signal will be processed in this function.
-----------------	--

Reimplemented from [cSignal](#).

Definition at line 28 of file WTcProcess.cpp.

21.76.3.8 void cProcess::UserSignal (SIGNAL lsSignal, void * lpData) [virtual]

This is a user defined function allowing the user to add Process specific inter-process signals. This Function should include the code for processing the signal as there is no signal buffer. It is useful to ensure a single detection and activation of process interactions. One Process shoudl detect the interaction and signal the other to respond. This way the order of Process evaluation is unimportant.

Reimplemented from [cUserSignal](#).

Definition at line 39 of file WTcProcess.cpp.

21.76.4 Friends And Related Function Documentation

21.76.4.1 **friend class cKernel** [friend]

Definition at line 52 of file WTcProcess.h.

21.76.5 Member Data Documentation

21.76.5.1 **cKernel* cProcess::mpKernel** [protected]

Definition at line 97 of file WTcProcess.h.

21.77 cPushPopStack< cX > Class Template Reference

Public Member Functions

- [cPushPopStack \(\)](#)
- [~cPushPopStack \(\)](#)
- [cX Push \(cX lcTemp\)](#)
- [void Pop \(\)](#)
- [cX Top \(\)](#)

21.77.1 Detailed Description

template<class cX> class cPushPopStack< cX >

Definition at line 4 of file WTcPushPopStack.h.

21.77.2 Constructor & Destructor Documentation

21.77.2.1 **template<class cX > cPushPopStack< cX >::cPushPopStack ()**
[inline]

Definition at line 10 of file WTcPushPopStack.h.

21.77.2.2 **template<class cX > cPushPopStack< cX >::~cPushPopStack ()**
[inline]

Definition at line 11 of file WTcPushPopStack.h.

21.77.3 Member Function Documentation

21.77.3.1 `template<class cX> void cPushPopStack<cX>::Pop() [inline]`

Definition at line 14 of file WTcPushPopStack.h.

21.77.3.2 `template<class cX> cX cPushPopStack<cX>::Push(cX lctemp) [inline]`

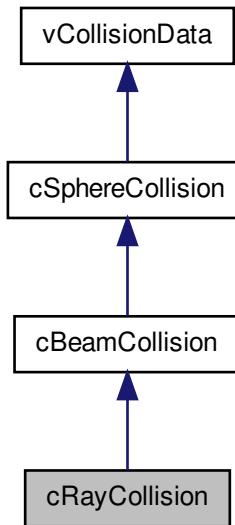
Definition at line 13 of file WTcPushPopStack.h.

21.77.3.3 `template<class cX> cX cPushPopStack<cX>::Top() [inline]`

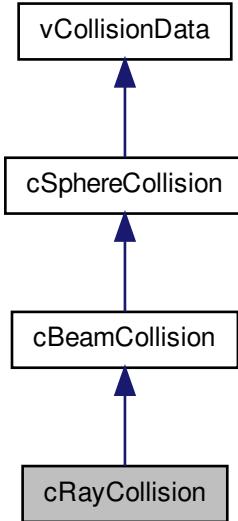
Definition at line 15 of file WTcPushPopStack.h.

21.78 cRayCollision Class Reference

Inheritance diagram for cRayCollision:



Collaboration diagram for cRayCollision:



Public Member Functions

- [cRayCollision \(\)](#)
- virtual [~cRayCollision \(\)](#)
- [cRayCollision * Ray \(\)](#)

Will return a pointer if this object contains a Ray collision data object. Otherwise returns 0;

- void [Update \(cMatrix4 &New\)](#)
- bool [CreatedThisFrame \(\)](#)

21.78.1 Detailed Description

This class is for representing objects moving very fast. The system will assume that the object is a sphere of radius equal to the value set in SetSize(float *lfSize). It will move the object in a straight line from its previous position to the position it is in at the start of this frame. This is a fast and perfect (after the assumption that the object is a sphere) way of colliding fast moving objects. Best used for projectiles. Note: Drastic camera manipulations can interfere with this class.

Definition at line 93 of file WTvCollisionData.h.

21.78.2 Constructor & Destructor Documentation

21.78.2.1 `cRayCollision::cRayCollision()`

Definition at line 174 of file WTvCollisionData.cpp.

21.78.2.2 `cRayCollision::~cRayCollision() [virtual]`

Definition at line 175 of file WTvCollisionData.cpp.

21.78.3 Member Function Documentation

21.78.3.1 `bool cRayCollision::CreatedThisFrame()`

Definition at line 177 of file WTvCollisionData.cpp.

21.78.3.2 `cRayCollision * cRayCollision::Ray() [virtual]`

Will return a pointer if this object contains a Ray collision data object. Otherwise returns 0;

Reimplemented from [cBeamCollision](#).

Definition at line 176 of file WTvCollisionData.cpp.

21.78.3.3 `void cRayCollision::Update(cMatrix4 & New) [virtual]`

Reimplemented from [cBeamCollision](#).

Definition at line 25 of file WTvCollisionData.cpp.

21.79 cReferenceList Class Reference

This will Load from an IMF and store a list of string type references. This will load a list of string type references from an IMF filestream. The References can be accessed as if they were an array.

Public Member Functions

- `cReferenceList()`

This is a public constructor and will initialise the function.

- `~cReferenceList()`

This is a public deconstructor and will delete the all the references and mpReferenceList.

- void [LoadIMF](#) (ifstream &FileStream)

This will load a Reference List from an IMF file. FileStream should have just reached the start of a Reference List.

- int8 * [Reference](#) (uint32 liID)

This will return a pointer to the Reference at position liID in the array mpReferenceList.

- uint32 [Size](#) ()

This will return the number of references stored by this reference list.

21.79.1 Detailed Description

This will Load from an IMF and store a list of string type references. This will load a list of string type references from an IMF filestream. The References can be accessed as if they were an array.

Definition at line 9 of file WTcReferenceList.h.

21.79.2 Constructor & Destructor Documentation

21.79.2.1 cReferenceList::cReferenceList() [inline]

This is a public constructor and will initialise the function.

Definition at line 17 of file WTcReferenceList.h.

21.79.2.2 cReferenceList::~cReferenceList()

This is a public deconstructor and will delete the all the references and mpReferenceList.

Definition at line 26 of file WTcReferenceList.cpp.

21.79.3 Member Function Documentation

21.79.3.1 void cReferenceList::LoadIMF (ifstream & FileStream)

This will load a Reference List from an IMF file. FileStream should have just reached the start of a Reference List.

Definition at line 3 of file WTcReferenceList.cpp.

21.79.3.2 `int8 * cReferenceList::Reference (uint32 iID)`

This will return a pointer to the Reference at position *iID* in the array mpReferenceList.

Definition at line 36 of file WTcReferenceList.cpp.

21.79.3.3 `uint32 cReferenceList::Size ()`

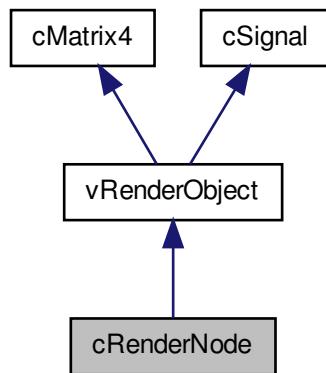
This will return the number of references stored by this reference list.

Definition at line 41 of file WTcReferenceList.cpp.

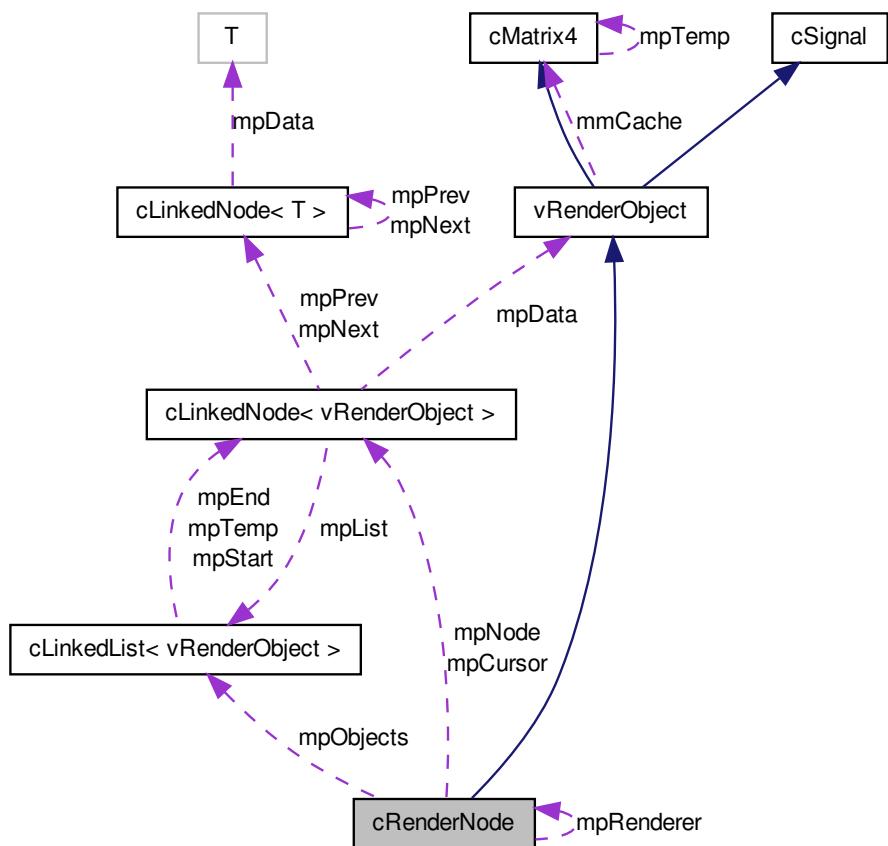
21.80 cRenderNode Class Reference

This is a dynamic render tree branch. This class stores a dynamic list of cRenderObjects called mpObjects. [cRenderNode](#) inherits [cRenderObject](#) and so can be stored in mpObjects of other cRenderNodes. Any translations applied to this [cRenderNode](#) will modify the base coordinates of any objects stored in mpObjects. This allows objects to be grouped for the purposes of translations. A [cRenderNode](#) volume encompasses all objects beneath it in the render tree, this increases the speed of collision searches as a [cRenderNode](#) can remove all it's sub objects from the search.

Inheritance diagram for cRenderNode:



Collaboration diagram for cRenderNode:



Public Member Functions

- `cRenderNode * Renderer ()`
Returns the `cRenderNode` which owns this object.
- `cRenderNode ()`
Constructor for `cRenderNode`.
- `cRenderNode (cRenderNode *lpRenderer)`
Constructs a new `cRenderNode` which is owned by `lpRenderer`.
- `~cRenderNode ()`

- void **Initialise** ()
- void **Remove** (`cLinkedNode< vRenderObject > *lpOld`)
- `cLinkedNode< vRenderObject > * Add (vRenderObject *lpNew)`
- `cLinkedList< vRenderObject > * RenderList ()`
- void **DeleteAll** ()
- void **RenderToPainter** ()
Renders this object to the `cPainter` render list.
- void **RenderPainter** (uint8 liLevel)
- void **RenderPainter** ()
Renders this object from the `cPainter` render list to the screen.
- void **Render** ()
Renders this object to the screen.
- void **LinkCollisionObject** (`cCollisionObject *lpObj`)
Links The `cCollisionObject` lpObj to this Renderable Object.
- void **AdditionalRenderFunctions** ()
*This will store any additional functions to be performed as the object is rendered.
Updating Caches, etc.*
- void **UpdateCache** ()
This will update the cache matrix.
- float * **GetPos** ()
This will return the Local position of the selected object.
- float * **GetGlobalPos** ()
*This will return the global position of the object as rendered at the end of last frame.
Note, this will contain the camera matrix.*

Public Attributes

- `cLinkedNode< vRenderObject > * mpCursor`

21.80.1 Detailed Description

This is a dynamic render tree branch. This class stores a dynamic list of cRenderObjects called mpObjects. `cRenderNode` inherits `cRenderObject` and so can be stored in mpObjects of other cRenderNodes. Any translations applied to this `cRenderNode` will modify the base coordinates of any objects stored in mpObjects. This allows objects to be grouped for the purposes of translations. A `cRenderNode` volume encompasses all objects beneath it in the render tree, this increases the speed of collision searches as a `cRenderNode` can remove all it's sub objects from the search.

Definition at line 12 of file WTCRenderNode.h.

21.80.2 Constructor & Destructor Documentation

21.80.2.1 cRenderNode::cRenderNode()

Constructor for [cRenderNode](#).

Definition at line 12 of file WTcRenderNode.cpp.

21.80.2.2 cRenderNode::cRenderNode(cRenderNode * *lpRenderer*)

Constructs a new [cRenderNode](#) which is owned by *lpRenderer*.

Definition at line 21 of file WTcRenderNode.cpp.

21.80.2.3 cRenderNode::~cRenderNode()

Definition at line 39 of file WTcRenderNode.cpp.

21.80.3 Member Function Documentation

21.80.3.1 cLinkedNode< vRenderObject > * cRenderNode::Add(vRenderObject * *lpNew*)

Definition at line 54 of file WTcRenderNode.cpp.

21.80.3.2 void cRenderNode::AdditionalRenderFunctions() [virtual]

This will store any additional functions to be performed as the object is rendered. Updating Caches, etc.

Reimplemented from [vRenderObject](#).

Definition at line 161 of file WTcRenderNode.cpp.

21.80.3.3 void cRenderNode::DeleteAll()

Definition at line 45 of file WTcRenderNode.cpp.

21.80.3.4 float * cRenderNode::GetGlobalPos() [virtual]

This will return the global position of the object as rendered at the end of last frame.
Note, this will contain the camera matrix.

Implements [vRenderObject](#).

Definition at line 200 of file WTcRenderNode.cpp.

21.80.3.5 float * cRenderNode::GetPos() [virtual]

This will return the Local position of the selected object.

Implements [vRenderObject](#).

Definition at line 194 of file WTcRenderNode.cpp.

21.80.3.6 void cRenderNode::Initialise()

Definition at line 3 of file WTcRenderNode.cpp.

21.80.3.7 void cRenderNode::LinkCollisionObject (cCollisionObject * *lpObj*) [virtual]

Links The [cCollisionObject](#) lpObj to this Renderable Object.

Reimplemented from [vRenderObject](#).

Definition at line 182 of file WTcRenderNode.cpp.

21.80.3.8 void cRenderNode::Remove (cLinkedNode< vRenderObject > * *lpOld*)

Definition at line 60 of file WTcRenderNode.cpp.

21.80.3.9 void cRenderNode::Render() [virtual]

Renders this object to the screen.

Implements [vRenderObject](#).

Definition at line 115 of file WTcRenderNode.cpp.

21.80.3.10 cRenderNode * cRenderNode::Renderer() [virtual]

Returns the [cRenderNode](#) which owns this object.

Implements [vRenderObject](#).

Definition at line 176 of file WTcRenderNode.cpp.

21.80.3.11 cLinkedList< vRenderObject > * cRenderNode::RenderList()

Definition at line 170 of file WTcRenderNode.cpp.

21.80.3.12 void cRenderNode::RenderPainter (uint8 *liLevel*)

Definition at line 166 of file WTcRenderNode.cpp.

21.80.3.13 void cRenderNode::RenderPainter() [virtual]

Renders this object from the [cPainter](#) render list to the screen.

Implements [vRenderObject](#).

Definition at line 167 of file WTcRenderNode.cpp.

21.80.3.14 void cRenderNode::RenderToPainter() [virtual]

Renders this object to the [cPainter](#) render list.

Implements [vRenderObject](#).

Definition at line 66 of file WTcRenderNode.cpp.

21.80.3.15 void cRenderNode::UpdateCache() [virtual]

This will update the cache matrix.

Implements [vRenderObject](#).

Definition at line 188 of file WTcRenderNode.cpp.

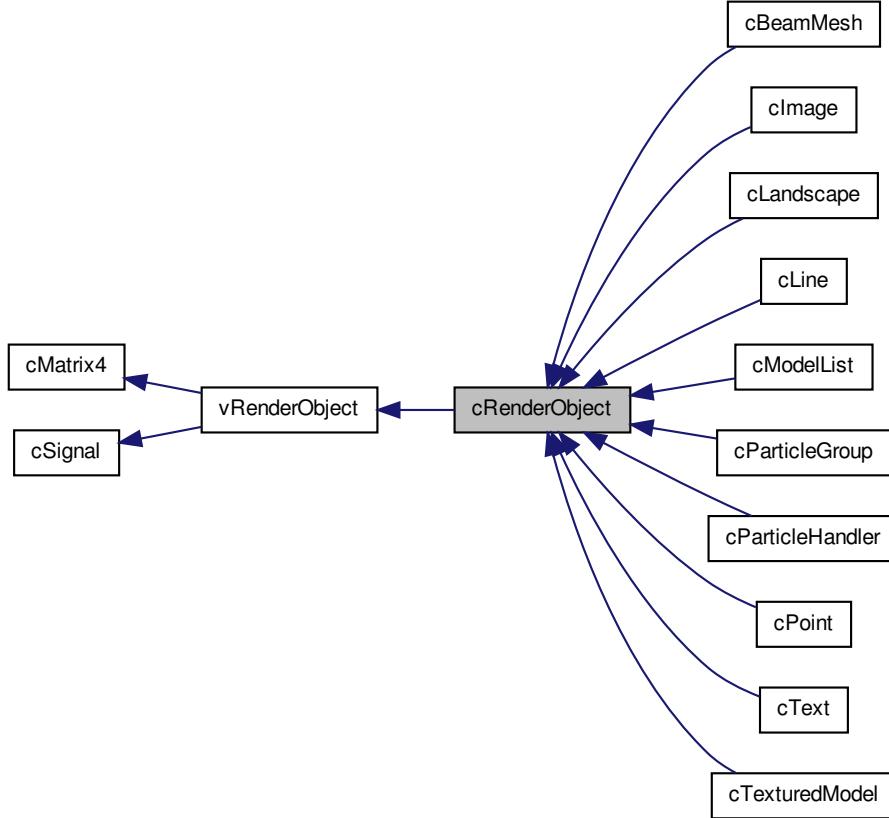
21.80.4 Member Data Documentation**21.80.4.1 cLinkedNode<vRenderObject>* cRenderNode::mpCursor**

Definition at line 28 of file WTcRenderNode.h.

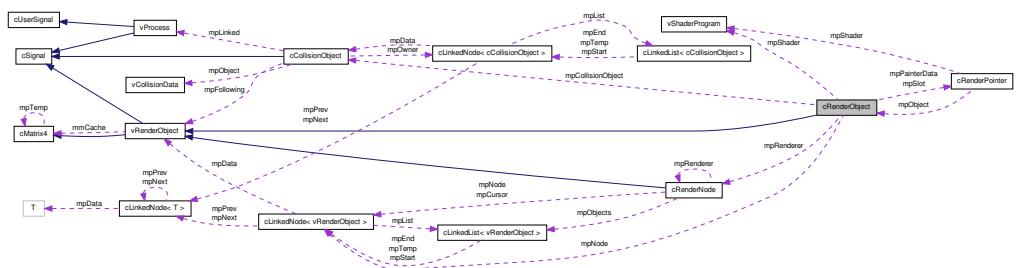
21.81 cRenderObject Class Reference

This class contains the base code for all renderable objects. Any renderable object should inherit this class.

Inheritance diagram for cRenderObject:



Collaboration diagram for `cRenderObject`:



Public Member Functions

- void [Shader \(vShaderProgram *lpShader\)](#)
Will set the shader this object will use.
- [vShaderProgram * Shader \(\)](#)
Will return a pointer to the Shader Program that is currently bound to this model.
- [cRenderObject \(\)](#)
Constructor for [cRenderObject](#). Creates a new render object and adds itself to [cCamera::mpRenderList](#).
- [cRenderObject \(cRenderNode *lpNode\)](#)
Constructor for [cRenderObject](#). Creates a new render object and adds itself to [lpNode](#).
- [~cRenderObject \(\)](#)
- void [LinkCollisionObject \(cCollisionObject *lpObj\)](#)
This will set the Collision Object that is linked to this renderable object. This should be called automatically when a [cCollisionObject](#) is created.
- [cRenderNode * Renderer \(\)](#)
Returns the [cRenderNode](#) which owns this renderable process.
- virtual void [RenderPainter \(\)](#)
virtual functions to allow polymorphism. see [cCamera::RenderPainter\(\)](#);
- virtual void [RenderPainter \(uint8 liLevel\)=0](#)
virtual function to allow polymorphism. see [cCamera::RenderPainter\(\)](#);
- virtual void [RenderToPainter \(\)=0](#)
virtual function to allow polymorphism. see [cCamera::RenderToPainter\(\)](#);
- virtual void [Render \(\)=0](#)
virtual function to allow polymorphiss. see [cCamera::Render\(\)](#);
- void [Delete \(\)](#)
Will remove this object from the render list owned by mpRenderer.
- [cLinkedNode< vRenderObject > * SetRenderNode \(cRenderNode *lpNode\)](#)
Will return a pointer to the Shader Program that is currently bound to this model.
- unsigned int [TextureNumber \(\)](#)
virtual function for polymorphism. Should return the texture ID of this renderable objects Texture.
- void [AdditionalRenderFunctions \(\)](#)

This will contain additional functions to perform when an object renders.

- virtual void [AdditionalKillFunctionality \(\)](#)

Virtual Functions to allow additional commands to be processed when a kill signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

- virtual void [AdditionalSleepFunctionality \(\)](#)

Virtual Functions to allow additional commands to be processed when a sleep signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

- virtual void [AdditionalWakeFunctionality \(\)](#)

Virtual Functions to allow additional commands to be processed when a wake signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

- void [UpdateCache \(\)](#)

This function will update the object cache. For consistency all Renderable Objects must update the cache as they are rendered. This will be the global position of the object and is used to get consistent positions throughout a frame for collision detection.

- float * [GetPos \(\)](#)

This will return the Local position of the selected object.

- float * [GetGlobalPos \(\)](#)

This will return the global position of the object as rendered at the end of last frame. Note, this will contain the camera matrix.

Protected Attributes

- [cRenderPointer * mpPainterData](#)

A Pointer to the [cRenderPointer](#) Object for this Renderable Object. Used when [cPainter](#) is enabled.

- [cRenderPointer ** mpSlot](#)

A Pointer to the position in the list that the [cRenderPointer](#) is at in [cPainter](#).

- [vShaderProgram * mpShader](#)

The current shader that this program will use.

21.81.1 Detailed Description

This class contains the base code for all renderable objects. Any renderable object should inherit this class.

Parameters

<i>lpNode</i>	<p><i>lpNode</i> is not required, but will add the renderable object to the cRenderNode pointed to by <i>lpNode</i>. This class should be inherited by any renderable object. If it is handed no parameters then it will add itself to cCamera::mpRenderList, which is owned by cCamera. If you wish to produce a scene graph, the new renderable object can be handed a pointer to a cRenderNode. Any translations applied to <i>lpNode</i> will change the co-ordinate system for this renderable object.</p> <pre>cRenderNode mpNode=new cRenderNode; cTexturedModel mpObject=new cTexturedModel(mpNode); mpNode.LAdvance(1.0f,0.0f,0.0f); mpObject.LAdvance(1.0f,0.0f,0.0f);</pre> <p>After this code <i>mpObject</i> will be at 2.0f,0.0f,0.0. This Function should only be used for Creating New Renderable Object Types. It should be virtual.</p>
---------------	---

Definition at line 22 of file WTcRenderObject.h.

21.81.2 Constructor & Destructor Documentation

21.81.2.1 [cRenderObject::cRenderObject\(\)](#)

Constructor for [cRenderObject](#). Creates a new render object and adds itself to [cCamera::mpRenderList](#).

Definition at line 16 of file WTcRenderObject.cpp.

21.81.2.2 [cRenderObject::cRenderObject\(cRenderNode * lpNode \)](#)

Constructor for [cRenderObject](#). Creates a new render object and adds itself to *lpNode*.

Definition at line 24 of file WTcRenderObject.cpp.

21.81.2.3 [cRenderObject::~cRenderObject\(\)](#)

Definition at line 3 of file WTcRenderObject.cpp.

21.81.3 Member Function Documentation

21.81.3.1 [void cRenderObject::AdditionalKillFunctionality\(\)](#) [virtual]

Virtual Functions to allow additional commands to be processed when a kill signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

Reimplemented from [cSignal](#).

Definition at line 82 of file WTcRenderObject.cpp.

21.81.3.2 void cRenderObject::AdditionalRenderFunctions() [virtual]

This will contain additional functions to perform when an object renders.

Reimplemented from [vRenderObject](#).

Definition at line 77 of file WTcRenderObject.cpp.

21.81.3.3 void cRenderObject::AdditionalSleepFunctionality() [virtual]

Virtual Functions to allow additional commands to be processed when a sleep signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

Reimplemented from [cSignal](#).

Definition at line 89 of file WTcRenderObject.cpp.

21.81.3.4 void cRenderObject::AdditionalWakeFunctionality() [virtual]

Virtual Functions to allow additional commands to be processed when a wake signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

Reimplemented from [cSignal](#).

Definition at line 103 of file WTcRenderObject.cpp.

21.81.3.5 void cRenderObject::Delete()

Will remove this object from the render list owned by mpRenderer.

Definition at line 48 of file WTcRenderObject.cpp.

21.81.3.6 float * cRenderObject::GetGlobalPos() [virtual]

This will return the global position of the object as rendered at the end of last frame.
Note, this will contain the camera matrix.

Implements [vRenderObject](#).

Definition at line 159 of file WTcRenderObject.cpp.

21.81.3.7 float * cRenderObject::GetPos() [virtual]

This will return the Local position of the selected object.

Implements [vRenderObject](#).

Definition at line 153 of file WTcRenderObject.cpp.

21.81.3.8 void cRenderObject::LinkCollisionObject (cCollisionObject * *lpObj*) [virtual]

This will set the Collision Object that is linked to this renderable object. This should be called automatically when a [cCollisionObject](#) is created.

Reimplemented from [vRenderObject](#).

Definition at line 10 of file WTcRenderObject.cpp.

21.81.3.9 virtual void cRenderObject::Render () [pure virtual]

virtual function to allow polymorphiss. see [cCamera::Render\(\)](#);

Implements [vRenderObject](#).

Implemented in [cBeamMesh](#), [cImage](#), [cLandscape](#), [cLine](#), [cModelList](#), [cParticleHandler](#), [cParticleGroup](#), [cPoint](#), [cTexturedModel](#), and [cText](#).

21.81.3.10 cRenderNode * cRenderObject::Renderer () [virtual]

Returns the [cRenderNode](#) which owns this renderable process.

Implements [vRenderObject](#).

Definition at line 146 of file WTcRenderObject.cpp.

21.81.3.11 virtual void cRenderObject::RenderPainter () [inline, virtual]

virtual functions to allow polymorphism. see [cCamera::RenderPainter\(\)](#);

Implements [vRenderObject](#).

Reimplemented in [cParticleHandler](#), and [cParticleGroup](#).

Definition at line 65 of file WTcRenderObject.h.

21.81.3.12 virtual void cRenderObject::RenderPainter (uint8 *liLevel*) [pure virtual]

virtual function to allow polymorphism. see [cCamera::RenderPainter\(\)](#);

Implemented in [cBeamMesh](#), [cImage](#), [cLandscape](#), [cLine](#), [cModelList](#), [cParticleHandler](#), [cParticleGroup](#), [cPoint](#), [cTexturedModel](#), and [cText](#).

21.81.3.13 virtual void cRenderObject::RenderToPainter () [pure virtual]

virtual function to allow polymorphism. see [cCamera::RenderToPainter\(\)](#);

Implements [vRenderObject](#).

Implemented in [cBeamMesh](#), [cImage](#), [cLandscape](#), [cLine](#), [cModelList](#), [cParticleHandler](#), [cParticleGroup](#), [cPoint](#), [cTexturedModel](#), and [cText](#).

21.81.3.14 `cLinkedNode< vRenderObject > * cRenderObject::SetRenderNode (cRenderNode * lpNode)`

Will return a pointer to the Shader Program that is currently bound to this model.

Will move the current renderable object from the current `cRenderNode` to `lpNode`.

Parameters

<code>lpNode</code>	The <code>cRenderNode</code> which will now own this renderable object.
---------------------	---

Returns

This function returns the new `cLinkedNode` which now owns this renderable object.

Definition at line 67 of file WTcRenderObject.cpp.

21.81.3.15 `vShaderProgram * cRenderObject::Shader ()`

Will return a pointer to the Shader Program that is currently bound to this model.

Definition at line 140 of file WTcRenderObject.cpp.

21.81.3.16 `void cRenderObject::Shader (vShaderProgram * lpShader)`

Will set the shader this object will use.

Definition at line 134 of file WTcRenderObject.cpp.

21.81.3.17 `unsigned int cRenderObject::TextureNumber () [inline]`

virtual function for polymorphism. Should return the texture ID of this renderable objects Texture.

Reimplemented in `cImage`, `cTexturedModel`, and `cText`.

Definition at line 83 of file WTcRenderObject.h.

21.81.3.18 `void cRenderObject::UpdateCache () [virtual]`

This function will update the object cache. For consistency all Renderable Objects must update the cache as they are rendered. This will be the global position of the object and is used to get consistent positions throughout a frame for collision detection.

Implements `vRenderObject`.

Definition at line 115 of file WTcRenderObject.cpp.

21.81.4 Member Data Documentation

21.81.4.1 cRenderPointer* cRenderObject::mpPainterData [protected]

A Pointer to the [cRenderPointer](#) Object for this Renderable Object. Used when [cPainter](#) is enabled.

Definition at line 36 of file WTcRenderObject.h.

21.81.4.2 vShaderProgram* cRenderObject::mpShader [protected]

The current shader that this program will use.

Definition at line 41 of file WTcRenderObject.h.

21.81.4.3 cRenderPointer** cRenderObject::mpSlot [protected]

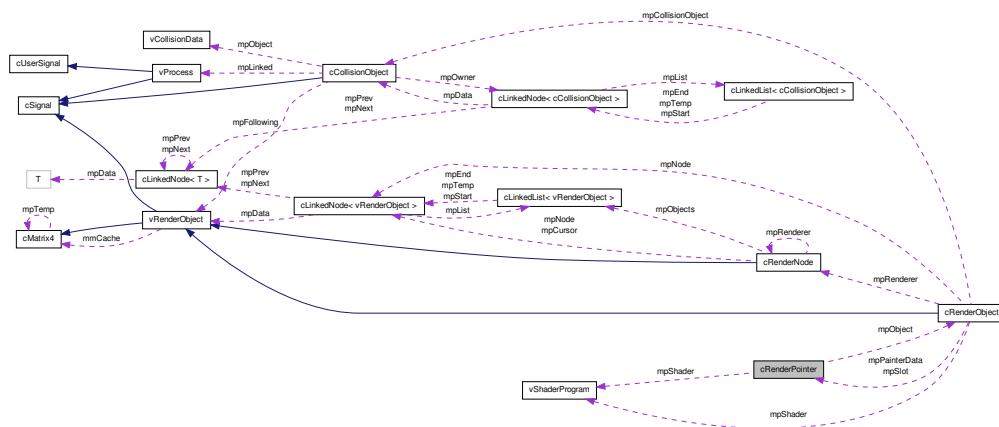
A Pointer to the position in the list that the [cRenderPointer](#) is at in [cPainter](#).

Definition at line 38 of file WTcRenderObject.h.

21.82 cRenderPointer Class Reference

This class is a temporary store for a [cRenderObjects](#) data. This is the complete render matrix and all other data required to render the object.

Collaboration diagram for [cRenderPointer](#):



Public Member Functions

- [cRenderPointer \(\)](#)

- `cRenderPointer operator= (cRenderPointer lcOther)`
- `void SetObject (cRenderObject *lpObject)`
Will Set the `cRenderObject` that this will link to.
- `void SetTexture (uint32 liTexture)`
This will set the texture to be used.
- `void SetShader (vShaderProgram *lpShader)`
This will set the shader to be used by the object.
- `void SetLevel (uint32 liLevel)`
This will set an additional parameter which is returned to `cRenderObject` when it is finally rendered. This gives it information about how to render.
- `void SetAlpha (bool lbAlpha)`
This will set the Alpha of the object.
- `void SetAll (cRenderObject *lpObject, uint32 liTexture=0, vShaderProgram *lpShader=0, bool lbAlpha=0, uint32 liLevel=0)`
This will allow the user to set some / all of the parameters for this object.
- `void RenderAgain ()`
This tells `cPainter` that this object will require rendering in the painter algorithm to allow it to be displayed on the screen.

Protected Attributes

- `cRenderObject * mpObject`
- `uint32 miDist`
- `uint32 mpTexture`
- `bool mbAlpha`
- `vShaderProgram * mpShader`
- `uint8 miLevel`
- `bool mbReRender`

Friends

- class `cPainter`

21.82.1 Detailed Description

This class is a temporary store for a cRenderObjects data. This is the complete render matrix and all other data required to render the object. This is a storage class for the `cPainter` class. It stores all the relevant data about a Renderable Object to allow it to be processed by `cPainter`.

Definition at line 12 of file WTcRenderPointer.h.

21.82.2 Constructor & Destructor Documentation

21.82.2.1 cRenderPointer::cRenderPointer()

Definition at line 9 of file WTcRenderPointer.cpp.

21.82.3 Member Function Documentation

21.82.3.1 cRenderPointer cRenderPointer::operator= (cRenderPointer *lcOther*)

Definition at line 20 of file WTcRenderPointer.cpp.

21.82.3.2 void cRenderPointer::RenderAgain()

This tells [cPainter](#) that this object will require rendering in the painter algorithm to allow it to be displayed on the screen.

Definition at line 3 of file WTcRenderPointer.cpp.

21.82.3.3 void cRenderPointer::SetAll (cRenderObject * *lpObject*, uint32 *liTexture* = 0, vShaderProgram * *lpShader* = 0, bool *lbAlpha* = 0, uint32 *liLevel* = 0) [inline]

This will all the user to set some / all of the parameters for this object.

Definition at line 37 of file WTcRenderPointer.cpp.

21.82.3.4 void cRenderPointer::SetAlpha (bool *lbAlpha*)

This will set the Alpha of the object.

Definition at line 35 of file WTcRenderPointer.cpp.

21.82.3.5 void cRenderPointer::SetLevel (uint32 *liLevel*)

This will set an additional paramter which is returned to [cRenderObject](#) when it is finally rendered. This gives it information about how to render.

Definition at line 34 of file WTcRenderPointer.cpp.

21.82.3.6 void cRenderPointer::SetObject (cRenderObject * *lpObject*)

Will Set the [cRenderObject](#) that this will link to.

Definition at line 31 of file WTcRenderPointer.cpp.

21.82.3.7 void cRenderPointer::SetShader (vShaderProgram * *lpShader*)

This will set the shader to be used by the object.

Definition at line 33 of file WTcRenderPointer.cpp.

21.82.3.8 void cRenderPointer::SetTexture (uint32 *liTexture*)

This will set the texture to be used.

Definition at line 32 of file WTcRenderPointer.cpp.

21.82.4 Friends And Related Function Documentation

21.82.4.1 friend class cPainter [friend]

Definition at line 42 of file WTcRenderPointer.h.

21.82.5 Member Data Documentation

21.82.5.1 bool cRenderPointer::mbAlpha [protected]

Definition at line 18 of file WTcRenderPointer.h.

21.82.5.2 bool cRenderPointer::mbReRender [protected]

Definition at line 21 of file WTcRenderPointer.h.

21.82.5.3 uint32 cRenderPointer::miDist [protected]

Definition at line 16 of file WTcRenderPointer.h.

21.82.5.4 uint8 cRenderPointer::miLevel [protected]

Definition at line 20 of file WTcRenderPointer.h.

21.82.5.5 cRenderObject* cRenderPointer::mpObject [protected]

Definition at line 15 of file WTcRenderPointer.h.

21.82.5.6 vShaderProgram* cRenderPointer::mpShader [protected]

Definition at line 19 of file WTcRenderPointer.h.

21.82.5.7 uint32 cRenderPointer::mpTexture [protected]

Definition at line 17 of file WTcRenderPointer.h.

21.83 cSettings Class Reference

Public Member Functions

- [cSettings \(\)](#)
- [~cSettings \(\)](#)
- [virtual void Settings \(\)](#)

Friends

- class [cMainThread](#)

21.83.1 Detailed Description

Definition at line 4 of file WTSettings.h.

21.83.2 Constructor & Destructor Documentation

21.83.2.1 cSettings::cSettings() [inline]

Definition at line 9 of file WTSettings.h.

21.83.2.2 cSettings::~cSettings() [inline]

Definition at line 10 of file WTSettings.h.

21.83.3 Member Function Documentation

21.83.3.1 virtual void cSettings::Settings() [inline, virtual]

Definition at line 11 of file WTSettings.h.

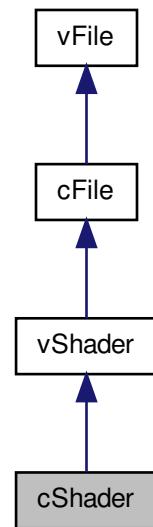
21.83.4 Friends And Related Function Documentation

21.83.4.1 friend class cMainThread [friend]

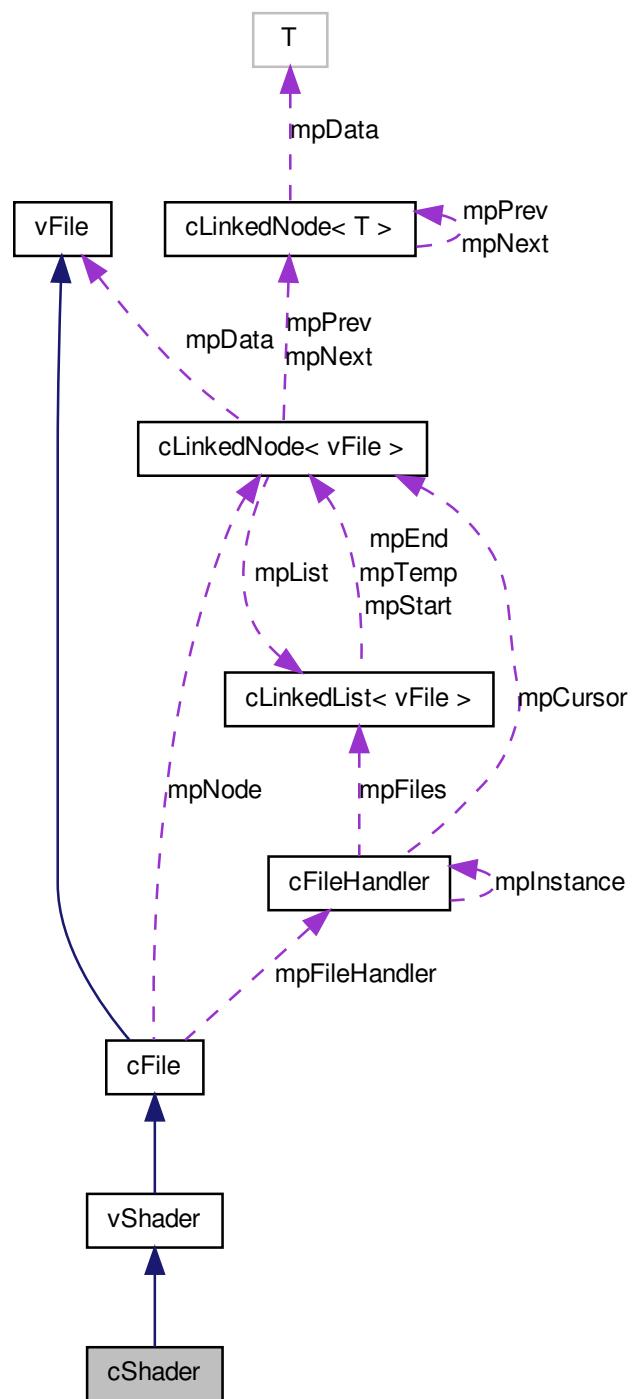
Definition at line 11 of file WTSettings.h.

21.84 cShader Class Reference

Inheritance diagram for cShader:



Collaboration diagram for cShader:



Public Member Functions

- [cShader \(\)](#)
- [cShader \(cShaderArray *lpData\)](#)

This will create a [cShader\(\)](#) object from the data in cShaderArray().

- [~cShader \(\)](#)

Destructor. Will delete mpShaderText and all arrays storing code.

- [uint32 ID \(\)](#)

This will return the OpenGL Shader ID.

Protected Attributes

- [uint32 miCharCount](#)

This is the total number of characters in this shader code.

- [const char ** mpShaderText](#)

This will point to an array of strings for each line of code.

- [uint8 miShaderType](#)

This is the type of Shader object for this object. It can be IMF_SHADER_TYPE_NULL or IMF_SHADER_TYPE_VERTEX or IMF_SHADER_TYPE_FRAGMENT or IMF_SHADER_TYPE_GEOMETRY.

- [uint32 miShaderID](#)

This is the OpenGL ID for this object.

- [uint32 miLines](#)

This is the number of lines in the Shader code.

21.84.1 Detailed Description

Definition at line 42 of file WTcShader.h.

21.84.2 Constructor & Destructor Documentation

21.84.2.1 [cShader::cShader\(\) \[inline\]](#)

Definition at line 56 of file WTcShader.h.

21.84.2.2 cShader::cShader (cShaderArray * *lpData*)

This will create a [cShader\(\)](#) object from the data in cShaderArray().

Definition at line 63 of file WTcShader.cpp.

21.84.2.3 cShader::~cShader ()

Destructor. Will delete mpShaderText and all arrays storing code.

Definition at line 57 of file WTcShader.cpp.

21.84.3 Member Function Documentation**21.84.3.1 uint32 cShader::ID () [inline, virtual]**

This will return the OpenGL Shader ID.

Implements [vShader](#).

Definition at line 62 of file WTcShader.h.

21.84.4 Member Data Documentation**21.84.4.1 uint32 cShader::miCharCount [protected]**

This is the total number of characters in this shader code.

Definition at line 46 of file WTcShader.h.

21.84.4.2 uint32 cShader::miLines [protected]

This is the number of lines in the Shader code.

Definition at line 54 of file WTcShader.h.

21.84.4.3 uint32 cShader::miShaderID [protected]

This is the OpenGL ID for this object.

Definition at line 52 of file WTcShader.h.

21.84.4.4 uint8 cShader::miShaderType [protected]

This is the type of Shader object for this object. It can be IMF_SHADER_TYPE_NULL or IMF_SHADER_TYPE_VERTEX or IMF_SHADER_TYPE_FRAGMENT or IMF_SHADER_TYPE_GEOMETRY.

Definition at line 50 of file WTcShader.h.

21.84.4.5 const char** cShader::mpShaderText [protected]

This will point to an array of strings for each line of code.

Definition at line 48 of file WTcShader.h.

21.85 cShaderArray Class Reference

This is the transfer class to transfer shader data from raw data to a [cShader](#) class. This will collect raw data from an IMF class and process it ready to be handed to a [cShader\(\)](#) object.

Public Member Functions

- [cShaderArray \(\)](#)
- [~cShaderArray \(\)](#)
- [uint8 Type \(\)](#)
- [void SetSize \(uint32 liCharCount\)](#)

This will set the total number of characters in the entire shader program. (Every line).

- [void SetType \(uint8 liShaderType\)](#)

This will set the type of this [cShaderArray\(\)](#) object.

- [void SetText \(char *lpShaderText\)](#)

This will set the character array storing the text for the Shader code to be the array pointed at by lpShaderText.

- [void SetLines \(uint32 liLines\)](#)

This will set the number of lines in this Shader Program.

- [void SetLineLengths \(uint32 *lpLines\)](#)

This will set the array of line lengths to be the array pointed to by lpLines.

- [void LoadIMF \(ifstream &FileStream\)](#)

This will load the Shader code from the IMF file. FileStream should just be at the start of a Shader object in an IMF file.

Public Attributes

- [uint32 miCharCount](#)
- [char * mpShaderText](#)
- [uint8 miShaderType](#)
- [char * mpRef](#)
- [uint32 miLines](#)
- [uint32 * mpLines](#)

21.85.1 Detailed Description

This is the transfer class to transfer shader data from raw data to a [cShader](#) class. This will collect raw data from an IMF class and process it ready to be handed to a [cShader\(\)](#) object.

Definition at line 10 of file [WTcShader.h](#).

21.85.2 Constructor & Destructor Documentation

21.85.2.1 [cShaderArray::cShaderArray\(\)](#)

Definition at line 41 of file [WTcShader.cpp](#).

21.85.2.2 [cShaderArray::~cShaderArray\(\)](#)

Definition at line 51 of file [WTcShader.cpp](#).

21.85.3 Member Function Documentation

21.85.3.1 [void cShaderArray::LoadIMF\(ifstream & FileStream \)](#)

This will load the Shader code from the IMF file. FileStream should just be at the start of a Shader object in an IMF file.

Definition at line 3 of file [WTcShader.cpp](#).

21.85.3.2 [void cShaderArray::SetLineLengths\(uint32 * lpLines \) \[inline\]](#)

This will set the array of line lengths to be the array pointed to by lpLines.

Definition at line 35 of file [WTcShader.h](#).

21.85.3.3 [void cShaderArray::SetLines\(uint32 liLines \) \[inline\]](#)

This will set the number of lines in this Shader Program.

Definition at line 33 of file [WTcShader.h](#).

21.85.3.4 [void cShaderArray::SetSize\(uint32 liCharCount \) \[inline\]](#)

This will set the total number of characters in the entire shader program. (Every line).

Definition at line 27 of file [WTcShader.h](#).

21.85.3.5 void cShaderArray::SetText (char * *lpShaderText*) [inline]

This will set the character array storing the text for the Shader code to be the array pointed at by lpShaderText.

Definition at line 31 of file WTcShader.h.

21.85.3.6 void cShaderArray::SetType (uint8 *liShaderType*) [inline]

This will set the type of this [cShaderArray\(\)](#) object.

Definition at line 29 of file WTcShader.h.

21.85.3.7 uint8 cShaderArray::Type () [inline]

Definition at line 25 of file WTcShader.h.

21.85.4 Member Data Documentation

21.85.4.1 uint32 cShaderArray::miCharCount

Definition at line 14 of file WTcShader.h.

21.85.4.2 uint32 cShaderArray::miLines

Definition at line 18 of file WTcShader.h.

21.85.4.3 uint8 cShaderArray::miShaderType

Definition at line 16 of file WTcShader.h.

21.85.4.4 uint32* cShaderArray::mpLines

Definition at line 19 of file WTcShader.h.

21.85.4.5 char* cShaderArray::mpRef

Definition at line 17 of file WTcShader.h.

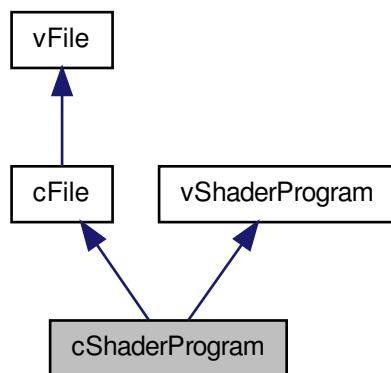
21.85.4.6 char* cShaderArray::mpShaderText

Definition at line 15 of file WTcShader.h.

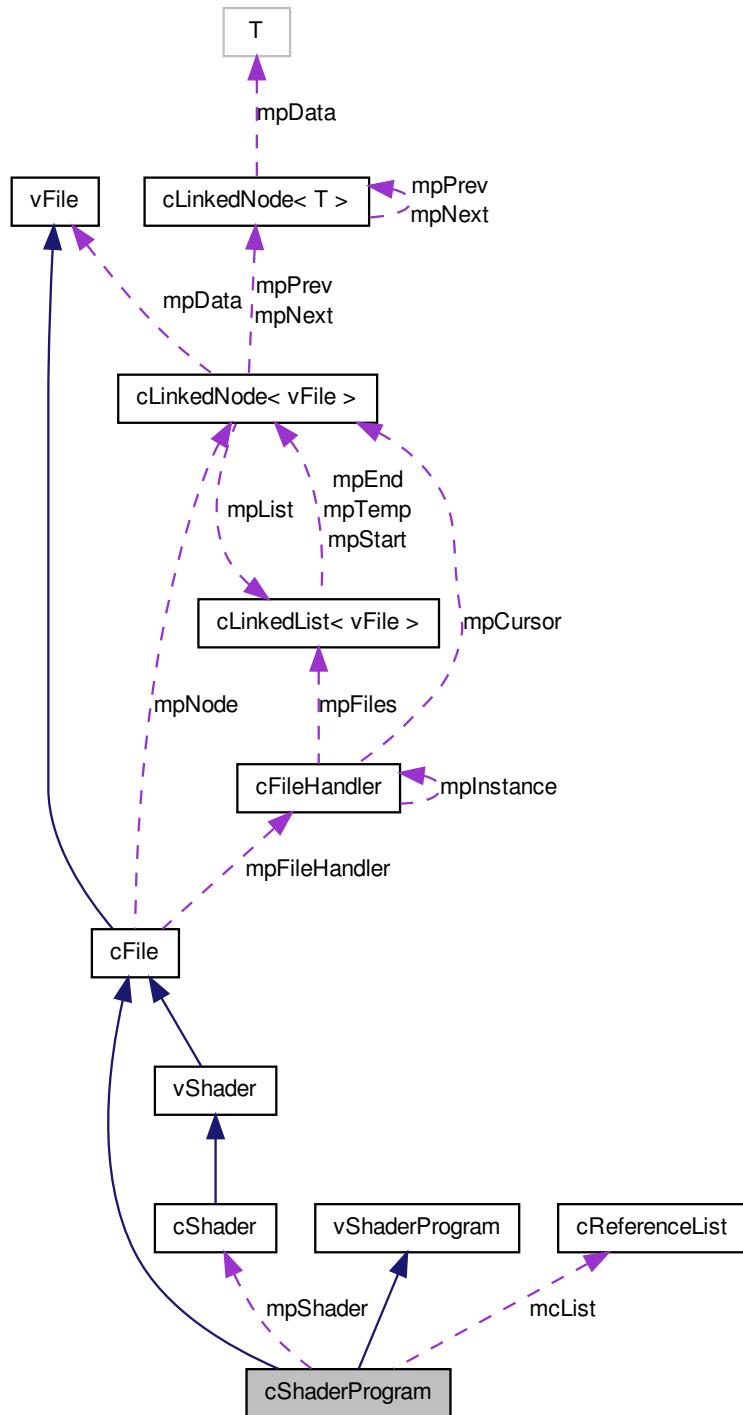
21.86 cShaderProgram Class Reference

A [cShaderProgram\(\)](#) is a series of [cShader\(\)](#) objects compiled into a program. [cShaderProgram](#) is a [cFile\(\)](#) object. The [cShaderProgram\(\)](#) finds a pointer to all the Shaders that compose it and compile them into a program. A [cShader\(\)](#) object can be used in many different programs. The [cShaderProgram\(\)](#) can be produced manually using the [AttachShader\(\)](#) and [Link\(\)](#) functions. The [cShaderProgram\(\)](#) can be turned on with the [use](#) function.

Inheritance diagram for [cShaderProgram](#):



Collaboration diagram for cShaderProgram:



Public Member Functions

- **cShaderProgram ()**
Public Constructor.
- **~cShaderProgram ()**
Public Destructor.
- **uint32 ID ()**
This will return the OpenGL Shader Program ID for this [cShaderProgram\(\)](#) object.
- **void AttachShader (cShader *lpShader)**
This will manually Attach the [cShader\(\)](#) object lpShader to the OpenGL Shader Program miProgramID.
- **void DetachShader (cShader *lpShader)**
This will manually remove the [cShader\(\)](#) object lpShader from the OpenGL Shader Program miProgramID.
- **void Link ()**
This will [Link\(\)](#) the compiled [cShader\(\)](#) objects together.
- **void Use ()**
This will turn on shaders and make this shader the current shader system.
- **uint32 Size ()**
This will return the number of [cShader\(\)](#) objects used by this system.
- **cShader * operator[] (uint32 liCount)**
This will return a pointer to the [cShader\(\)](#) object in position liCount of the [cShader\(\)](#) List;.
- **void LoadIMF (ifstream &FileStream)**
This will allow the [cShaderProgram\(\)](#) to extract a list of string references for Shaders. It will then find pointers to the [cShader\(\)](#) objects and compile the program using them.

Static Public Member Functions

- **static void UseFixedFunction ()**
This will turn off the ususage of shaders and return the system to using the Fixed Function Pipeline.

21.86.1 Detailed Description

A [cShaderProgram\(\)](#) is a series of [cShader\(\)](#) objects compiled into a program. [cShaderProgram](#) is a [cFile\(\)](#) object. The [cShaderProgram\(\)](#) finds a pointer to all the Shaders that compose it and compile them into a program. A [cShader\(\)](#) object can be used in many different programs. The [cShaderProgram\(\)](#) can be produced manually using the [AttachShader\(\)](#) and [Link\(\)](#) functions. The [cShaderProgram\(\)](#) can be turned on with the [use](#) function.

Definition at line 11 of file [WTcShaderProgram.h](#).

21.86.2 Constructor & Destructor Documentation

21.86.2.1 [cShaderProgram::cShaderProgram\(\)](#)

Public Constructor.

Definition at line 32 of file [WTcShaderProgram.cpp](#).

21.86.2.2 [cShaderProgram::~cShaderProgram\(\)](#)

Public Destructor.

Definition at line 39 of file [WTcShaderProgram.cpp](#).

21.86.3 Member Function Documentation

21.86.3.1 [void cShaderProgram::AttachShader\(cShader * lpShader \)](#)

This will manually Attach the [cShader\(\)](#) object *lpShader* to the OpenGL Shader Program *miProgramID*.

Definition at line 46 of file [WTcShaderProgram.cpp](#).

21.86.3.2 [void cShaderProgram::DetachShader\(cShader * lpShader \)](#)

This will manually remove the [cShader\(\)](#) object *lpShader* from the OpenGL Shader Program *miProgramID*.

Definition at line 52 of file [WTcShaderProgram.cpp](#).

21.86.3.3 [uint32 cShaderProgram::ID\(\) \[inline, virtual\]](#)

This will return the OpenGL Shader Program ID for this [cShaderProgram\(\)](#) object.

Implements [vShaderProgram](#).

Definition at line 25 of file [WTcShaderProgram.h](#).

21.86.3.4 void cShaderProgram::Link()

This will [Link\(\)](#) the compiled cShader() objects together.

Definition at line 57 of file WTcShaderProgram.cpp.

21.86.3.5 void cShaderProgram::LoadIMF(ifstream & FileStream)

This will allow the [cShaderProgram\(\)](#) to extract a list of string references for Shaders. It will then find pointers to the cShader() objects and compile the program using them.

Definition at line 3 of file WTcShaderProgram.cpp.

21.86.3.6 cShader* cShaderProgram::operator[](uint32 liCount) [inline]

This will return a pointer to the cShader() object in position liCount of the cShader() List;

Definition at line 40 of file WTcShaderProgram.h.

21.86.3.7 uint32 cShaderProgram::Size() [inline]

This will return the number of cShader() objects used by this system.

Definition at line 35 of file WTcShaderProgram.h.

21.86.3.8 void cShaderProgram::Use() [virtual]

This will turn on shaders and make this shader the current shader system.

Implements [vShaderProgram](#).

Definition at line 66 of file WTcShaderProgram.cpp.

21.86.3.9 static void cShaderProgram::UseFixedFunction() [inline, static]

This will turn off the usage of shaders and return the system to using the Fixed Function Pipeline.

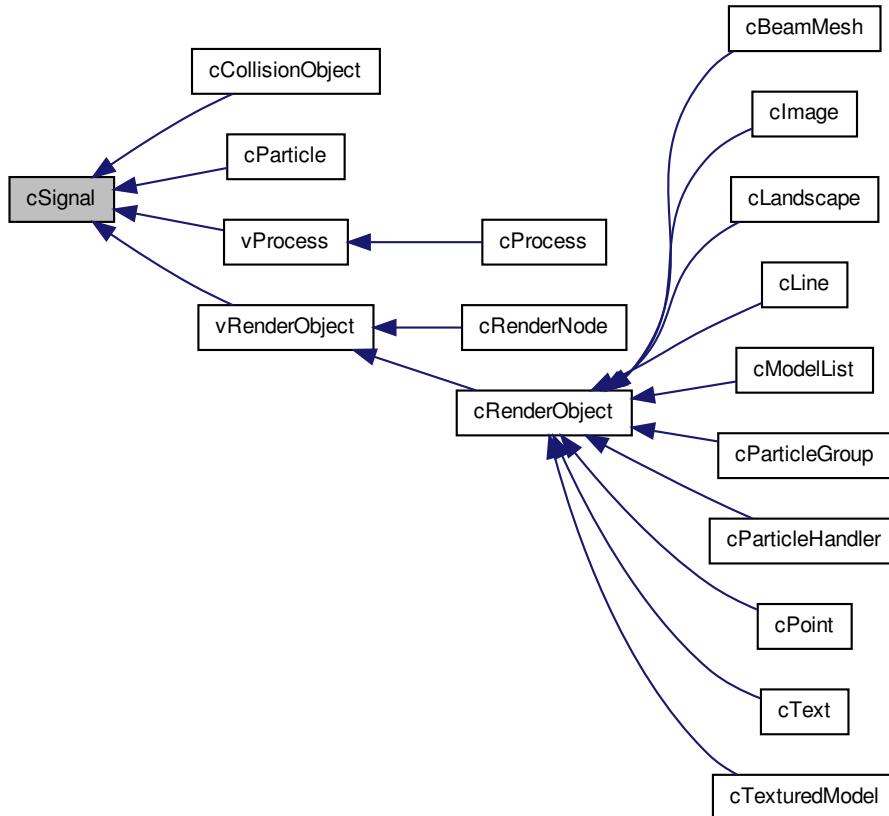
Definition at line 37 of file WTcShaderProgram.h.

21.87 cSignal Class Reference

Class for handling Signals sent between objects ([cProcess](#), [cRenderObject](#), [cCollisionObject](#)). Allows the user to wake, sleep and kill objects. For [cProcess](#) (while [cParentStack](#) is enabled) also allows signals to be sent to a process that will recursively affect all the children of that process. Possible signals to be passed in are _S_-

SLEEP,_S_WAKE,_S_KILL,_S_SLEEP_TREE, _S_WAKE_TREE,_S_KILL_TREE
User Specified Signals are controlled by the class [cUserSignal](#).

Inheritance diagram for cSignal:



Public Member Functions

- virtual void [Signal](#) (SIGNAL lsSignal)

This is the function that will handle a system signal. Possible signals to be passed in are _S_SLEEP,_S_WAKE,_S_KILL,_S_SLEEP_TREE, _S_WAKE_TREE,_S_KILL_TREE.

- bool [Awake](#) ()

This will return true if the object is Awake.

- bool [Asleep](#) ()

This will return true if the object is Asleep.

- bool [Alive \(\)](#)

This will return true if the object is Alive.

- bool [Dead \(\)](#)

This will return true if the object is Dead.

- virtual void [AdditionalKillFunctionality \(\)](#)

Virtual Functions to allow additional commands to be processed when a kill signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

- virtual void [AdditionalSleepFunctionality \(\)](#)

Virtual Functions to allow additional commands to be processed when a sleep signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

- virtual void [AdditionalWakeFunctionality \(\)](#)

Virtual Functions to allow additional commands to be processed when a wake signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

Protected Attributes

- bool [mbAwake](#)
- bool [mbAlive](#)

21.87.1 Detailed Description

Class for handling Signals sent between objects ([cProcess](#), [cRenderObject](#), [cCollisionObject](#)). Allows the user to wake, sleep and kill objects. For [cProcess](#) (while [cParentStack](#) is enabled) also allows signals to be sent to a process that will recursively affect all the children of that process. Possible signals to be passed in are `_S_SLEEP,_S_WAKE,_S_KILL,_S_SLEEP_TREE, _S_WAKE_TREE,_S_KILL_TREE`. User Specified Signals are controlled by the class [cUserSignal](#).

Definition at line 11 of file `WTcSignal.h`.

21.87.2 Member Function Documentation

21.87.2.1 virtual void [cSignal::AdditionalKillFunctionality \(\)](#) [inline, virtual]

Virtual Functions to allow additional commands to be processed when a kill signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

Reimplemented in [cParticle](#), [cRenderObject](#), [cProcess](#), and [vProcess](#).

Definition at line 28 of file `WTcSignal.h`.

21.87.2.2 virtual void cSignal::AdditionalSleepFunctionality() [inline, virtual]

Virtual Functions to allow additional commands to be processed when a sleep signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

Reimplemented in [cParticle](#), [cRenderObject](#), [cProcess](#), and [vProcess](#).

Definition at line 30 of file WTcSignal.h.

21.87.2.3 virtual void cSignal::AdditionalWakeFunctionality() [inline, virtual]

Virtual Functions to allow additional commands to be processed when a wake signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

Reimplemented in [cParticle](#), [cRenderObject](#), [cProcess](#), and [vProcess](#).

Definition at line 32 of file WTcSignal.h.

21.87.2.4 bool cSignal::Alive()

This will return true if the object is Alive.

Definition at line 16 of file WTcSignal.cpp.

21.87.2.5 bool cSignal::Asleep()

This will return true if the object is Asleep.

Definition at line 15 of file WTcSignal.cpp.

21.87.2.6 bool cSignal::Awake()

This will return true if the object is Awake.

Definition at line 14 of file WTcSignal.cpp.

21.87.2.7 bool cSignal::Dead()

This will return true if the object is Dead.

Definition at line 17 of file WTcSignal.cpp.

21.87.2.8 void cSignal::Signal(SIGNAL IsSignal) [virtual]

This is the function that will handle a system signal. Possible signals to be passed in are _S_SLEEP,_S_WAKE,_S_KILL,_S_SLEEP_TREE, _S_WAKE_TREE,_S_KILL_TREE.

Reimplemented in [cProcess](#), and [cCollisionObject](#).

Definition at line 3 of file WTcSignal.cpp.

21.87.3 Member Data Documentation

21.87.3.1 bool cSignal::mbAlive [protected]

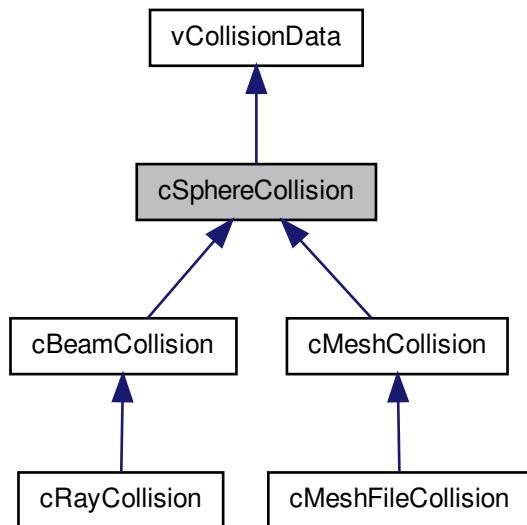
Definition at line 15 of file WTcSignal.h.

21.87.3.2 bool cSignal::mbAwake [protected]

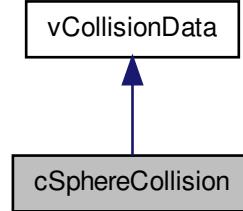
Definition at line 14 of file WTcSignal.h.

21.88 cSphereCollision Class Reference

Inheritance diagram for cSphereCollision:



Collaboration diagram for cSphereCollision:



Public Member Functions

- [cSphereCollision \(\)](#)
Initialisation of the `cSphereCollision` Object. Will initialise the size of the collision to 0.
- virtual [~cSphereCollision \(\)](#)
- void [SetSize \(float IfSize\)](#)
Will Set the Size of the Collision (For the Sphere aspect of collisions.) This should be the radius of the collision Sphere.
- float [CollisionSize \(\)](#)
Will return the Collision Size Value, which is the radius of the Collision Sphere squared.
- [cSphereCollision * Sphere \(\)](#)
Will return a pointer if this object contains a sphere collision data object. Otherwise returns 0;
- virtual [cBeamCollision * Beam \(\)](#)
Will return a pointer if this object contains a Beam collision data object. Otherwise returns 0;.
- virtual [cMeshCollision * Mesh \(\)](#)
Will return a pointer if this object contains a Mesh collision data object. Otherwise returns 0;.
- virtual [cRayCollision * Ray \(\)](#)
Will return a pointer if this object contains a Ray collision data object. Otherwise returns 0;.

- virtual void [Update \(cMatrix4 &New\)](#)

Protected Attributes

- float [mfSize](#)

21.88.1 Detailed Description

This is the class for storing the data for Sphere collisions. All [vCollisionData](#) inherits [cSphereCollision](#). This is because everyobject will attempt a sphere collision before refining the collisions to the appropriate type (for speed). See [vCollisionData](#) for more information.

Definition at line 38 of file WTvCollisionData.h.

21.88.2 Constructor & Destructor Documentation

21.88.2.1 [cSphereCollision::cSphereCollision \(\)](#)

Initialisation of the [cSphereCollision](#) Object. Will initialise the size of the collision to 0.

Definition at line 164 of file WTvCollisionData.cpp.

21.88.2.2 [cSphereCollision::~cSphereCollision \(\) \[virtual\]](#)

Definition at line 165 of file WTvCollisionData.cpp.

21.88.3 Member Function Documentation

21.88.3.1 [cBeamCollision * cSphereCollision::Beam \(\) \[virtual\]](#)

Will return a pointer if this object contains a Beam collision data object. Otherwise returns 0;.

Implements [vCollisionData](#).

Reimplemented in [cBeamCollision](#).

Definition at line 169 of file WTvCollisionData.cpp.

21.88.3.2 [float cSphereCollision::CollisionSize \(\) \[virtual\]](#)

Will return the Collision Size Value, which is the radius of the Collision Sphere squared.

Implements [vCollisionData](#).

Definition at line 167 of file WTvCollisionData.cpp.

21.88.3.3 cMeshCollision * cSphereCollision::Mesh() [virtual]

Will return a pointer if this object contains a Mesh collision data object. Otherwise returns 0;.

Implements [vCollisionData](#).

Reimplemented in [cMeshCollision](#).

Definition at line 170 of file WTvCollisionData.cpp.

21.88.3.4 cRayCollision * cSphereCollision::Ray() [virtual]

Will return a pointer if this object contains a Ray collision data object. Otherwise returns 0;.

Implements [vCollisionData](#).

Reimplemented in [cBeamCollision](#), and [cRayCollision](#).

Definition at line 171 of file WTvCollisionData.cpp.

21.88.3.5 void cSphereCollision::SetSize(float lFSize) [virtual]

Will Set the Size of the Collision (For the Sphere aspect of collisions.) This should be the radius of the collision Sphere.

Implements [vCollisionData](#).

Definition at line 166 of file WTvCollisionData.cpp.

21.88.3.6 cSphereCollision * cSphereCollision::Sphere() [virtual]

Will return a pointer if this object contains a sphere collision data object. Otherwise returns 0;.

Implements [vCollisionData](#).

Definition at line 168 of file WTvCollisionData.cpp.

21.88.3.7 void cSphereCollision::Update(cMatrix4 & New) [virtual]

Implements [vCollisionData](#).

Reimplemented in [cBeamCollision](#), and [cRayCollision](#).

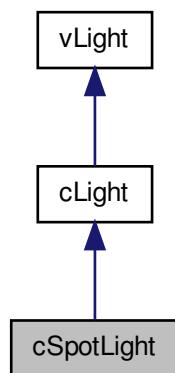
Definition at line 172 of file WTvCollisionData.cpp.

21.88.4 Member Data Documentation**21.88.4.1 float cSphereCollision::mfSize [protected]**

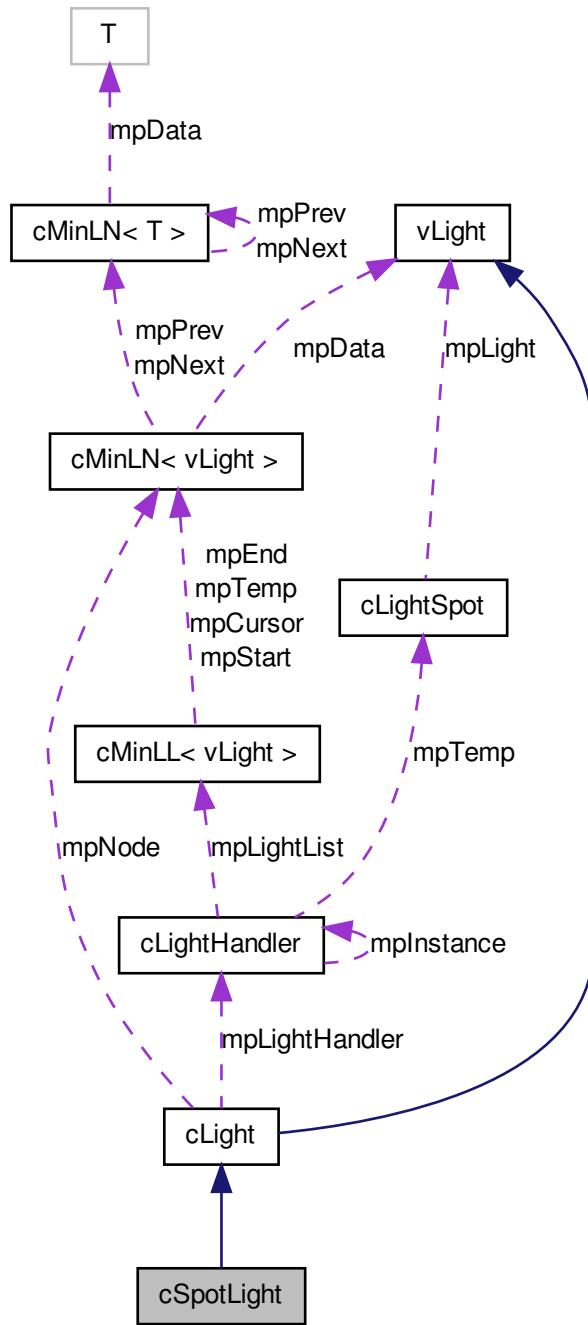
Definition at line 41 of file WTvCollisionData.h.

21.89 cSpotLight Class Reference

Inheritance diagram for cSpotLight:



Collaboration diagram for cSpotLight:



Public Member Functions

- void [Direction](#) (float lfX, float lfY, float lfZ, float lfA)

This will set the Global Direction the Spot light will point in. X,Y,Z,Cutoff.

- void [PrepareLight](#) ()

Will bind this light source for use in OpenGL. Once all the Lights Variables are set, call this to preapre the light.

- void [PrepareLight](#) (uint32 liLight)

21.89.1 Detailed Description

Definition at line 4 of file WTcSpotLight.h.

21.89.2 Member Function Documentation

21.89.2.1 void cSpotLight::Direction (float lfX, float lfY, float lfZ, float lfA)

This will set the Global Direction the Spot light will point in. X,Y,Z,Cutoff.

Definition at line 3 of file WTcSpotLight.cpp.

21.89.2.2 void cSpotLight::PrepareLight (uint32 liLight) [virtual]

Reimplemented from [cLight](#).

Definition at line 23 of file WTcSpotLight.cpp.

21.89.2.3 void cSpotLight::PrepareLight () [virtual]

Will bind this light source for use in OpenGL. Once all the Lights Variables are set, call this to preapre the light.

Reimplemented from [cLight](#).

Definition at line 12 of file WTcSpotLight.cpp.

21.90 cSync Class Reference

This is the timer class. It allows the user to time events and pause the system.

Collaboration diagram for cSync:



Public Member Functions

- [cSync \(\)](#)
- [~cSync \(\)](#)
- void [Tick \(\)](#)

This will tick the system and recalculate the current system speed.

- void [SleepWrap \(uint32 lMS\)](#)

This will will sleep wrap the system by lMS miliseconds.

- float [GetTimeMod \(\)](#)

This will return mfTimeMod.

- float [GetTimeAcc \(\)](#)

This will reutrn mfTimeAcc.

- float [GetCPS \(\)](#)

This will return mfCPS.

21.90.1 Detailed Description

This is the timer class. It allows the user to time events and pause the system.

Definition at line 46 of file WTcSync.h.

21.90.2 Constructor & Destructor Documentation

21.90.2.1 [cSync::cSync \(\)](#)

Definition at line 56 of file WTcSync.cpp.

21.90.2.2 `cSync::~cSync()`

Definition at line 73 of file WTcSync.cpp.

21.90.3 Member Function Documentation**21.90.3.1 `float cSync::GetCPS() [inline]`**

This will return mfCPS.

Definition at line 88 of file WTcSync.h.

21.90.3.2 `float cSync::GetTimeAcc() [inline]`

This will reutrn mfTimeAcc.

Definition at line 86 of file WTcSync.h.

21.90.3.3 `float cSync::GetTimeMod() [inline]`

This will return mfTimeMod.

Definition at line 84 of file WTcSync.h.

21.90.3.4 `void cSync::SleepWrap(uint32 IMS)`

This will will sleep wrap the system by liMS miliseconds.

Definition at line 44 of file WTcSync.cpp.

21.90.3.5 `void cSync::Tick()`

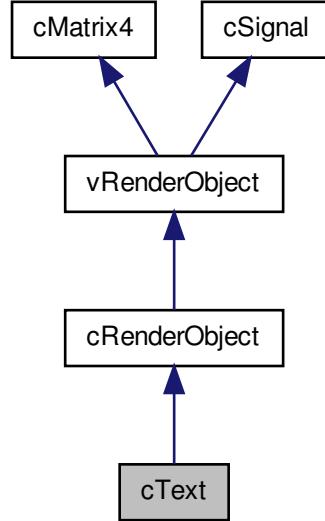
This will tick the system and recalculate the current system speed.

Definition at line 83 of file WTcSync.cpp.

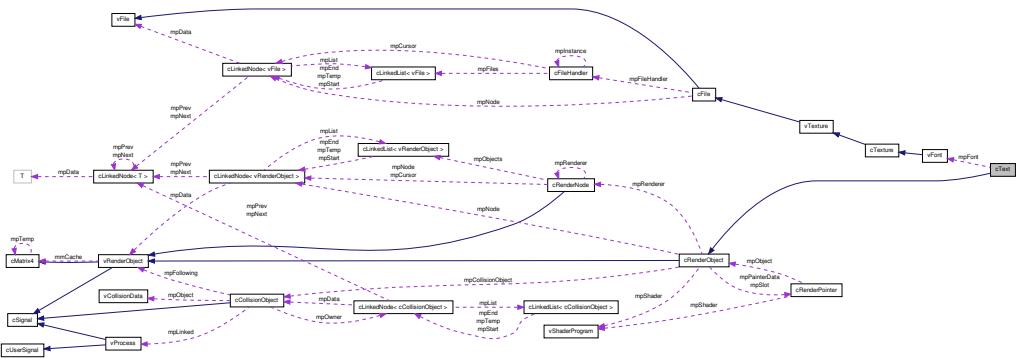
21.91 cText Class Reference

This class is a text renderable object.

Inheritance diagram for cText:



Collaboration diagram for cText:



Public Member Functions

- [cText \(const char *lsText\)](#)

Creates a text object and gives it a text string to use.

- [cText \(\)](#)

Creates an empty text object with no text string.

- [void Setup \(\)](#)

- [void Font \(vFont *lpFont\)](#)

Will set the font texture the text will use.

- [void Size \(float liSize\)](#)

Will set the size of the characters.

- [void Text \(char *lsText\)](#)

Will set the text string the [cText](#) object will render.

- [template<class T >](#)

- [void Value \(const T &t\)](#)

Will accept a generic data type to render to the screen (will convert to a string).

- [unsigned int TextureNumber \(\)](#)

virtual function for polymorphism. Should return the texture ID of this renderable objects Texture.

- [void RenderPainter \(uint8 liLevel\)](#)

virtual function to allow polymorphism. see [cCamera::RenderPainter\(\)](#);

- [void RenderToPainter \(\)](#)

virtual function to allow polymorphism. see [cCamera::RenderToPainter\(\)](#);

- [void Render \(\)](#)

virtual function to allow polymorphiss. see [cCamera::Render\(\)](#);

21.91.1 Detailed Description

This class is a text renderable object.

Definition at line 6 of file WTcTextureText.h.

21.91.2 Constructor & Destructor Documentation

21.91.2.1 [cText::cText \(const char * lsText \)](#)

Creates a text object and gives it a text string to use.

Definition at line 99 of file WTcTextureText.cpp.

21.91.2.2 cText::cText()

Creates an empty text object with no text string.

Definition at line 108 of file WTcTextureText.cpp.

21.91.3 Member Function Documentation**21.91.3.1 void cText::Font(vFont * lpFont)**

Will set the font texture the text will use.

Definition at line 114 of file WTcTextureText.cpp.

21.91.3.2 void cText::Render() [virtual]

virtual function to allow polymorphiss. see [cCamera::Render\(\)](#);

Implements [cRenderObject](#).

Definition at line 53 of file WTcTextureText.cpp.

21.91.3.3 void cText::RenderPainter(uint8 liLevel) [virtual]

virtual function to allow polymorphism. see [cCamera::RenderPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 4 of file WTcTextureText.cpp.

21.91.3.4 void cText::RenderToPainter() [virtual]

virtual function to allow polymorphism. see [cCamera::RenderToPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 34 of file WTcTextureText.cpp.

21.91.3.5 void cText::Setup()

Definition at line 94 of file WTcTextureText.cpp.

21.91.3.6 void cText::Size(float liSize)

Will set the size of the characters.

Definition at line 120 of file WTcTextureText.cpp.

21.91.3.7 void cText::Text (char * *IsText*)

Will set the text string the [cText](#) object will render.

Definition at line 87 of file WTcTextureText.cpp.

21.91.3.8 unsigned int cText::TextureNumber () [inline]

virtual function for polymorphism. Should return the texture ID of this renderable objects Texture.

Reimplemented from [cRenderObject](#).

Definition at line 38 of file WTcTextureText.h.

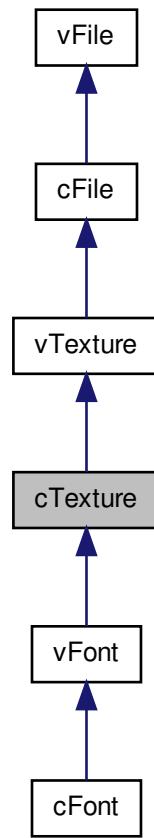
21.91.3.9 template<class T> void cText::Value (const T & *t*) [inline]

Will accept a generic data type to render to the screen (will convert to a string).

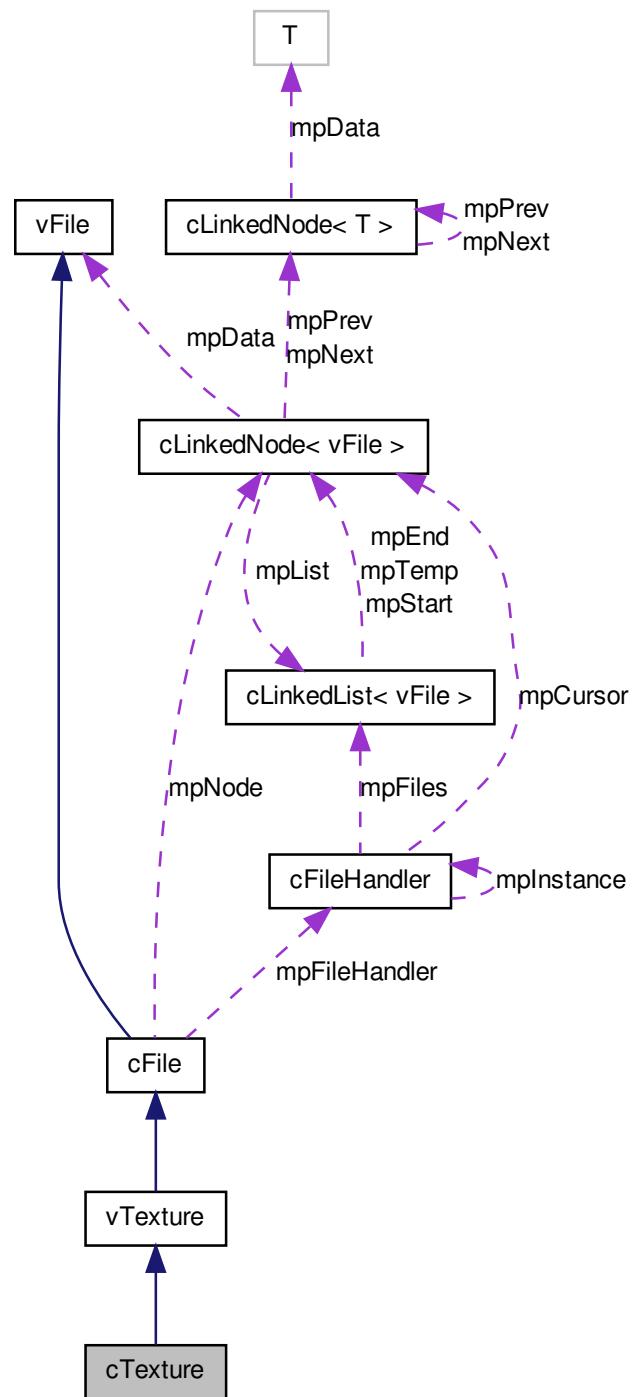
Definition at line 51 of file WTcTextureText.h.

21.92 cTexture Class Reference

Inheritance diagram for cTexture:



Collaboration diagram for cTexture:



Public Member Functions

- `uint8 * Data ()`
- `cTexture ()`
- `cTexture (cTextureArray *lpArray)`
- `~cTexture ()`
- `uint32 Width ()`
- `uint32 Height ()`
- `uint32 Depth ()`
- `void UpdateTexture ()`
- `void BindTexture ()`
- `uint32 TextureNumber ()`

Protected Attributes

- `int miWidth`
- `int miHeight`
- `int miDepth`
- `unsigned int miTexture`
- `uint8 * mpData`

21.92.1 Detailed Description

Definition at line 14 of file WTcTexture.h.

21.92.2 Constructor & Destructor Documentation

21.92.2.1 `cTexture::cTexture() [inline]`

Definition at line 18 of file WTcTexture.h.

21.92.2.2 `cTexture::cTexture (cTextureArray * lpArray)`

Definition at line 17 of file WTcTexture.cpp.

21.92.2.3 `cTexture::~cTexture() [inline]`

Definition at line 20 of file WTcTexture.h.

21.92.3 Member Function Documentation

21.92.3.1 void cTexture::BindTexture() [virtual]

Implements [vTexture](#).

Reimplemented in [cFont](#).

Definition at line 53 of file WTcTexture.cpp.

21.92.3.2 uint8 * cTexture::Data() [virtual]

Implements [vTexture](#).

Definition at line 61 of file WTcTexture.cpp.

21.92.3.3 uint32 cTexture::Depth() [inline, virtual]

Implements [vTexture](#).

Definition at line 23 of file WTcTexture.h.

21.92.3.4 uint32 cTexture::Height() [inline, virtual]

Implements [vTexture](#).

Reimplemented in [cFont](#).

Definition at line 22 of file WTcTexture.h.

21.92.3.5 uint32 cTexture::TextureNumber() [inline, virtual]

Implements [vTexture](#).

Definition at line 28 of file WTcTexture.h.

21.92.3.6 void cTexture::UpdateTexture() [virtual]

Implements [vTexture](#).

Reimplemented in [cFont](#).

Definition at line 31 of file WTcTexture.cpp.

21.92.3.7 uint32 cTexture::Width() [inline, virtual]

Implements [vTexture](#).

Definition at line 21 of file WTcTexture.h.

21.92.4 Member Data Documentation

21.92.4.1 int cTexture::miDepth [protected]

Definition at line 33 of file WTcTexture.h.

21.92.4.2 int cTexture::miHeight [protected]

Definition at line 33 of file WTcTexture.h.

21.92.4.3 unsigned int cTexture::miTexture [protected]

Definition at line 34 of file WTcTexture.h.

21.92.4.4 int cTexture::miWidth [protected]

Definition at line 33 of file WTcTexture.h.

21.92.4.5 uint8* cTexture::mpData [protected]

Definition at line 35 of file WTcTexture.h.

21.93 cTextureArray Class Reference

Public Member Functions

- [cTextureArray \(\)](#)
- [~cTextureArray \(\)](#)

Public Attributes

- uint32 [miWidth](#)
- uint32 [miHeight](#)
- uint32 [miDepth](#)
- uint8 * [mpData](#)
- int8 * [mpRef](#)

21.93.1 Detailed Description

Definition at line 4 of file WTcTexture.h.

21.93.2 Constructor & Destructor Documentation**21.93.2.1 cTextureArray::cTextureArray()**

Definition at line 5 of file WTcTexture.cpp.

21.93.2.2 cTextureArray::~cTextureArray()

Definition at line 12 of file WTcTexture.cpp.

21.93.3 Member Data Documentation**21.93.3.1 uint32 cTextureArray::miDepth**

Definition at line 8 of file WTcTexture.h.

21.93.3.2 uint32 cTextureArray::miHeight

Definition at line 8 of file WTcTexture.h.

21.93.3.3 uint32 cTextureArray::miWidth

Definition at line 8 of file WTcTexture.h.

21.93.3.4 uint8* cTextureArray::mpData

Definition at line 9 of file WTcTexture.h.

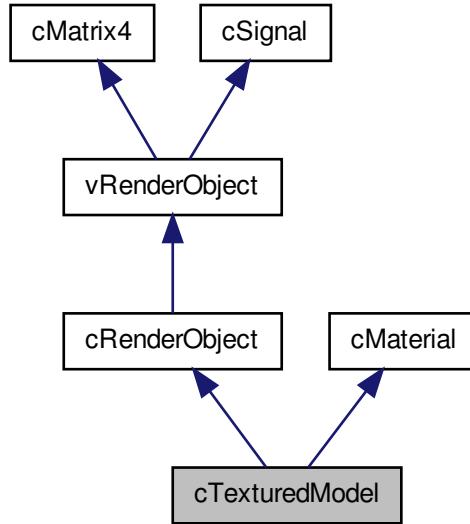
21.93.3.5 int8* cTextureArray::mpRef

Definition at line 10 of file WTcTexture.h.

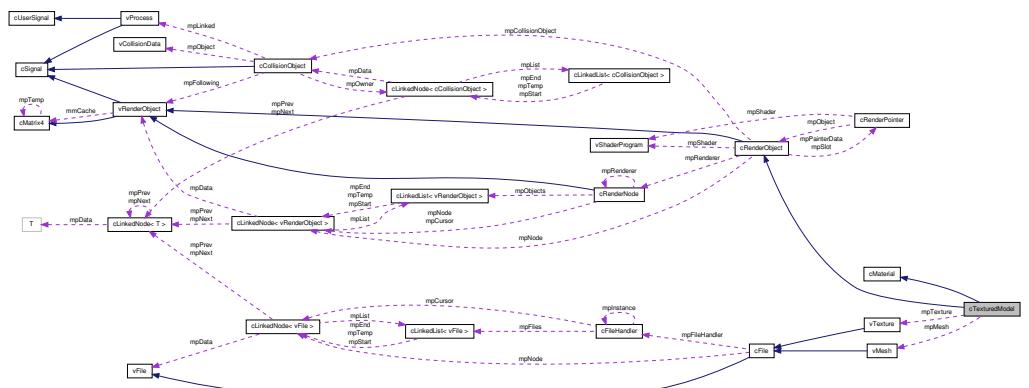
21.94 cTexturedModel Class Reference

A standard Textured Model renderable object.

Inheritance diagram for cTexturedModel:



Collaboration diagram for cTexturedModel:



Public Member Functions

- `cTexturedModel ()`
cTexturedModel constructor

- **cTexturedModel (cRenderNode *lpRenderer)**
cTexturedModel constructor. Will be owned by lpRenderer.
- **void Mesh (vMesh *lpObject)**
Will set the mesh the model will use.
- **unsigned int TextureNumber ()**
virtual function for polymorphism. Should return the texture ID of this renderable objects Texture.
- **void Texture (vTexture *lpTexture)**
Will set the texture bound to this model.
- **vTexture * Texture ()**
Will return the pointer to the *vTexture* that is currently bound to this model.
- **void RenderPainter (uint8 liLevel)**
virtual function to allow polymorphism. see *cCamera::RenderPainter()*;
- **void RenderToPainter ()**
virtual function to allow polymorphism. see *cCamera::RenderToPainter()*;
- **void Render ()**
virtual function to allow polymorphiss. see *cCamera::Render()*;

21.94.1 Detailed Description

A standard Textured Model renderable object.

Definition at line 6 of file WTcTexturedModel.h.

21.94.2 Constructor & Destructor Documentation

21.94.2.1 cTexturedModel::cTexturedModel ()

cTexturedModel constructor

Definition at line 4 of file WTcTexturedModel.cpp.

21.94.2.2 cTexturedModel::cTexturedModel (cRenderNode * lpRenderer)

cTexturedModel constructor. Will be owned by lpRenderer.

Definition at line 11 of file WTcTexturedModel.cpp.

21.94.3 Member Function Documentation

21.94.3.1 void cTexturedModel::Mesh (*vMesh * lpObject*)

Will set the mesh the model will use.

Definition at line 23 of file WTcTexturedModel.cpp.

21.94.3.2 void cTexturedModel::Render () [virtual]

virtual function to allow polymorphiss. see [cCamera::Render\(\)](#);

Implements [cRenderObject](#).

Definition at line 58 of file WTcTexturedModel.cpp.

21.94.3.3 void cTexturedModel::RenderPainter (*uint8 liLevel*) [virtual]

virtual function to allow polymorphism. see [cCamera::RenderPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 29 of file WTcTexturedModel.cpp.

21.94.3.4 void cTexturedModel::RenderToPainter () [virtual]

virtual function to allow polymorphism. see [cCamera::RenderToPainter\(\)](#);

Implements [cRenderObject](#).

Definition at line 41 of file WTcTexturedModel.cpp.

21.94.3.5 void cTexturedModel::Texture (*vTexture * lpTexture*)

Will set the texture bound to this model.

Definition at line 18 of file WTcTexturedModel.cpp.

21.94.3.6 *vTexture * cTexturedModel::Texture ()*

Will return the pointer to the [vTexture](#) that is currently bound to this model.

Definition at line 86 of file WTcTexturedModel.cpp.

21.94.3.7 unsigned int cTexturedModel::TextureNumber ()

virtual function for polymorphism. Should return the texture ID of this renderable objects Texture.

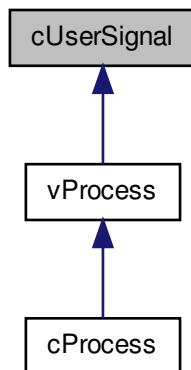
Reimplemented from [cRenderObject](#).

Definition at line 80 of file WTcTexturedModel.cpp.

21.95 cUserSignal Class Reference

Class for handling user specified signals sent between classes inheriting [cProcess](#). The function UserSignal should be defined for each object. The signals each class has defined can be independant. The code for processing the signal should be defined in UserSignal as there is no signal buffer. This should be used for dealing with Process interactions, to make a single point of detection of interaction and allowing both processes to handle the interaction. System signals (sleep, wake and kill) should be handled through [cSignal](#).

Inheritance diagram for cUserSignal:



Public Member Functions

- virtual void [UserSignal](#) (SIGNAL liSignal, void *lpData)

Function to handle user specified signals. SIGNAL is an unsigned integer and lpData allows additional data to be passed to the function.

21.95.1 Detailed Description

Class for handling user specified signals sent between classes inheriting [cProcess](#). The function UserSignal should be defined for each object. The signals each class has defined can be independant. The code for processing the signal should be defined in

UserSignal as there is no signal buffer. This should be used for dealing with Process interactions, to make a single point of detection of interaction and allowing both processes to handle the interaction. System signals (sleep, wake and kill) should be handled through [cSignal](#).

Definition at line 41 of file WTcSignal.h.

21.95.2 Member Function Documentation

21.95.2.1 virtual void cUserSignal::UserSignal (SIGNAL *liSignal*, void * *lpData*) [inline, virtual]

Function to handle user specified signals. SIGNAL is an unsigned integer and lpData allows additional data to be passed to the function.

Reimplemented in [cProcess](#).

Definition at line 46 of file WTcSignal.h.

21.96 cVertex Class Reference

Public Member Functions

- [cVertex \(\)](#)
- [cVertex \(float *lp1\)](#)
- void [SetVertex](#) (float x, float y, float z)
- float * [Verteces \(\)](#)
- float [X \(\)](#)
- float [Y \(\)](#)
- float [Z \(\)](#)
- void [X \(float lfX\)](#)
- void [Y \(float lfY\)](#)
- void [Z \(float lfZ\)](#)
- float & [operator\[\] \(uint8 liPos\)](#)
- bool [operator== \(cVertex &lpOther\)](#)
- bool [operator== \(float *lpOther\)](#)
- [cVertex & operator= \(cVertex &lpOther\)](#)
- float * [operator= \(float *lpOther\)](#)
- float * [Data \(\)](#)
- bool [Similar \(cVertex lpOther\)](#)
- void [Display \(\)](#)
- uint32 [FileSize \(\)](#)
- double [SqrDistance \(\)](#)
- double [Distance \(\)](#)
- void [OutputIMFVertex \(ofstream &FileStream\)](#)
- void [LoadIMFVertex \(ifstream &FileStream\)](#)
- double [AbsoluteDistance \(cVertex &lpOther\)](#)

Public Attributes

- float `mpData` [3]

21.96.1 Detailed Description

Definition at line 10 of file WTcVertex.h.

21.96.2 Constructor & Destructor Documentation

21.96.2.1 `cVertex::cVertex()` [inline]

Definition at line 15 of file WTcVertex.h.

21.96.2.2 `cVertex::cVertex(float *lp1)` [inline]

Definition at line 16 of file WTcVertex.h.

21.96.3 Member Function Documentation

21.96.3.1 `double cVertex::AbsoluteDistance(cVertex & lpOther)`

Definition at line 10 of file WTcVertex.cpp.

21.96.3.2 `float* cVertex::Data()` [inline]

Definition at line 33 of file WTcVertex.h.

21.96.3.3 `void cVertex::Display()` [inline]

Definition at line 38 of file WTcVertex.h.

21.96.3.4 `double cVertex::Distance()` [inline]

Definition at line 49 of file WTcVertex.h.

21.96.3.5 `uint32 cVertex::FileSize()` [inline]

Definition at line 43 of file WTcVertex.h.

21.96.3.6 `void cVertex::LoadIMFVertex(ifstream & FileStream)` [inline]

Definition at line 56 of file WTcVertex.h.

21.96.3.7 `cVertex& cVertex::operator=(cVertex & lpOther) [inline]`

Definition at line 31 of file WTcVertex.h.

21.96.3.8 `float* cVertex::operator=(float * lpOther) [inline]`

Definition at line 32 of file WTcVertex.h.

21.96.3.9 `bool cVertex::operator==(float * lpOther) [inline]`

Definition at line 30 of file WTcVertex.h.

21.96.3.10 `bool cVertex::operator==(cVertex & lpOther) [inline]`

Definition at line 29 of file WTcVertex.h.

21.96.3.11 `float& cVertex::operator[](uint8 liPos) [inline]`

Definition at line 28 of file WTcVertex.h.

21.96.3.12 `void cVertex::OutputIMFVertex(ostream & FileStream) [inline]`

Definition at line 51 of file WTcVertex.h.

21.96.3.13 `void cVertex::SetVertex(float x, float y, float z) [inline]`

Definition at line 18 of file WTcVertex.h.

21.96.3.14 `bool cVertex::Similar(cVertex lpOther)`

Definition at line 3 of file WTcVertex.cpp.

21.96.3.15 `double cVertex::SqrDistance() [inline]`

Definition at line 48 of file WTcVertex.h.

21.96.3.16 `float* cVertex::Verteces() [inline]`

Definition at line 19 of file WTcVertex.h.

21.96.3.17 `float cVertex::X() [inline]`

Definition at line 20 of file WTcVertex.h.

21.96.3.18 void cVertex::X(float *fX*) [inline]

Definition at line 24 of file WTcVertex.h.

21.96.3.19 void cVertex::Y(float *fY*) [inline]

Definition at line 25 of file WTcVertex.h.

21.96.3.20 float cVertex::Y() [inline]

Definition at line 21 of file WTcVertex.h.

21.96.3.21 void cVertex::Z(float *fZ*) [inline]

Definition at line 26 of file WTcVertex.h.

21.96.3.22 float cVertex::Z() [inline]

Definition at line 22 of file WTcVertex.h.

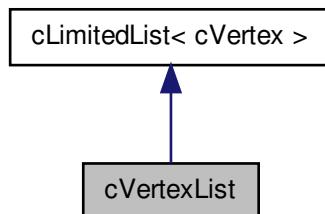
21.96.4 Member Data Documentation

21.96.4.1 float cVertex::mpData[3]

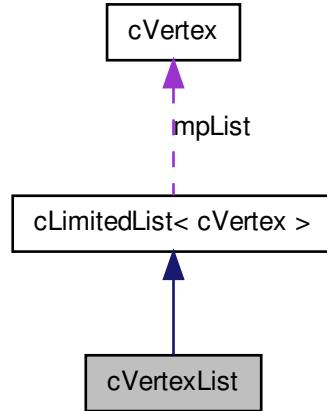
Definition at line 14 of file WTcVertex.h.

21.97 cVertexList Class Reference

Inheritance diagram for cVertexList:



Collaboration diagram for cVertexList:



Public Member Functions

- `cVertexList ()`
- `~cVertexList ()`
- `void GenerateVerteces (float *lpVerteces, uint32 liVerteces)`
- `void Display ()`
- `void Optimise ()`
- `void OrderDistance ()`
- `void Strip ()`
- `void LoadIMFVerteces (ifstream &FileStream)`
- `void OutputIMFVerteces (ofstream &FileStream)`
- `uint32 FileSize ()`

21.97.1 Detailed Description

Definition at line 65 of file WTCVertex.h.

21.97.2 Constructor & Destructor Documentation

21.97.2.1 `cVertexList::cVertexList () [inline]`

Definition at line 68 of file WTCVertex.h.

21.97.2.2 `cVertexList::~cVertexList()` [inline]

Definition at line 70 of file WTcVertex.h.

21.97.3 Member Function Documentation

21.97.3.1 `void cVertexList::Display()` [inline]

Definition at line 73 of file WTcVertex.h.

21.97.3.2 `uint32 cVertexList::FileSize()`

Definition at line 49 of file WTcVertex.cpp.

21.97.3.3 `void cVertexList::GenerateVertices(float *lpVertices, uint32 liVertices)`

Definition at line 61 of file WTcVertex.cpp.

21.97.3.4 `void cVertexList::LoadIMFVertices(ifstream & FileStream)`

Definition at line 23 of file WTcVertex.cpp.

21.97.3.5 `void cVertexList::Optimise()` [inline]

Definition at line 82 of file WTcVertex.h.

21.97.3.6 `void cVertexList::OrderDistance()`

Definition at line 81 of file WTcVertex.cpp.

21.97.3.7 `void cVertexList::OutputIMFVertices(ofstream & FileStream)`

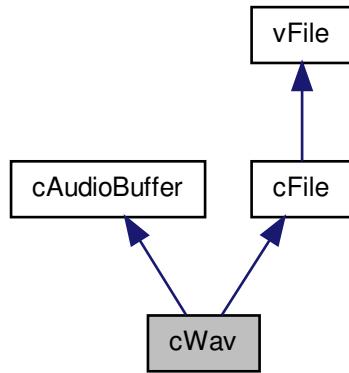
Definition at line 38 of file WTcVertex.cpp.

21.97.3.8 `void cVertexList::Strip()`

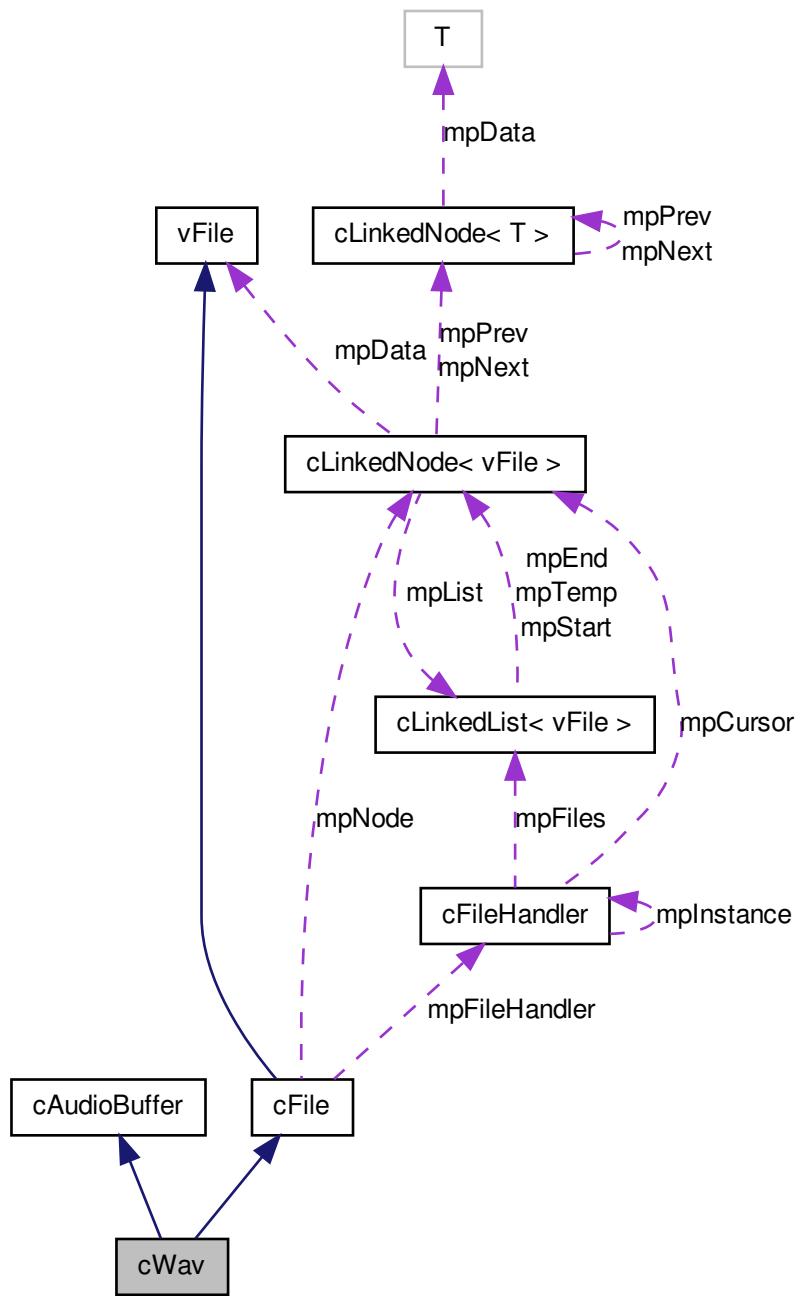
Definition at line 115 of file WTcVertex.cpp.

21.98 cWav Class Reference

Inheritance diagram for cWav:



Collaboration diagram for cWav:



Public Member Functions

- [cWav \(char *lpPath\)](#)

This constructor will load the wav file from the file lpPath.

- [~cWav \(\)](#)

This will free the wav file from memory and destroy the OpenAL buffer.

21.98.1 Detailed Description

This will load and store a wav file into memory. It is itself an OpenAL buffer, so does not need to be added. It can be played by linking to an [cAudioObject](#) and calling [Play\(\)](#) Definition at line 6 of file WTcWav.h.

21.98.2 Constructor & Destructor Documentation

21.98.2.1 cWav::cWav (char * lpPath)

This constructor will load the wav file from the file lpPath.

Parameters

<i>lpPath</i>	This is the file path of the wav file. The file lpPath will be loaded into an OpenAL buffer.
---------------	--

Definition at line 5 of file WTcWav.cpp.

21.98.2.2 cWav::~cWav ()

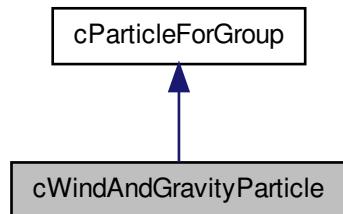
This will free the wav file from memory and destroy the OpenAL buffer.

Definition at line 69 of file WTcWav.cpp.

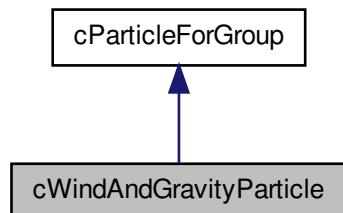
21.99 cWindAndGravityParticle Class Reference

cGravityParticles which are also affected by Wind. These Particles have the code to be affected by the variables _WIND_X,_WIND_Y and _WIND_Z. [UpdatePos\(\)](#) will account use the current Wind and Gravity settings to calculate the speed and position of each particle.

Inheritance diagram for cWindAndGravityParticle:



Collaboration diagram for cWindAndGravityParticle:



Public Member Functions

- void [UpdatePos \(\)](#)

Friends

- class [cParticleHandler](#)

21.99.1 Detailed Description

cGravityParticles which are also affected by Wind. These Particles have the code to be affected by the variables _WIND_X,_WIND_Y and _WIND_Z. [UpdatePos\(\)](#) will

account use the current Wind and Gravity settings to calculate the speed and position of each particle.

Definition at line 40 of file WTcParticle.h.

21.99.2 Member Function Documentation

21.99.2.1 void cWindAndGravityParticle::UpdatePos() [virtual]

Reimplemented from [cParticleForGroup](#).

Definition at line 195 of file WTcParticle.cpp.

21.99.3 Friends And Related Function Documentation

21.99.3.1 friend class cParticleHandler [friend]

Definition at line 48 of file WTcParticle.h.

21.100 cWindow Class Reference

This class will create control and destroy desktop windows. This is ugly and disgusting. It will create new windows, link the window with OpenGL and do basic event handling. This is the users access to interact with the OS. Note it is all very OS specific.

Public Member Functions

- [cWindow \(HINSTANCE lphInstance\)](#)

*This will create a new [cWindow](#) class. Create a desktop window and [InitialiseOpenGL\(\)](#).
This is Windows OS only.*

- [cWindow \(\)](#)

*This will create a new [cWindow](#) class. Create a desktop window and [InitialiseOpenGL\(\)](#).
This is Linux OS only.*

- void [MovePointer \(uint32 liX, uint32 liY\)](#)

This will move the mouse cursor to the position liX,liY in the current window. see _MOUSE.

- void [GetMouseSpeed \(\)](#)

This will update the current mouse speeds. see _MOUSE.

- [~cWindow \(\)](#)

- float [Ratio \(\)](#)

This will return mfRatio.

- void [StartWindow \(\)](#)
This will create the window object and initialise it.
- void [EnableOpenGL \(\)](#)
This will create the OpenGL context and enable OpenGL.
- void [InitialiseOpenGL \(\)](#)
This will prepare the Rendering matrix and other OpenGL initialisations.
- void [EnableInput \(\)](#)
This will do NOTHING.
- void [DisableInput \(\)](#)
This will do NOTHING.
- void [Move \(short liX, short liY\)](#)
This will Move the window to position liX,liY (in pixels) on the desktop.
- void [Resize \(short liX, short liY\)](#)
This will resize the window to size liX,liY (in pixels).
- void [MakeCurrent \(\)](#)
This will make this context the current one (to receive rendering data).
- void [HandleMessages \(\)](#)
This is an interrupt function that will handle messages from the OS.
- void [HandleChanges \(\)](#)
This will handle changes to the window (from the OS).

Public Attributes

- WNDCLASS [wc](#)
- HWND [hWnd](#)
- HDC [hDC](#)
- HGLRC [hRC](#)
- MSG [msg](#)
- Display * [lpDisplay](#)
- XVisualInfo * [VisualInfo](#)
- GLXContext [glContext](#)
- XSetWindowAttributes [WindowAttributes](#)
- XWindowAttributes [gwa](#)
- Window [IWindow](#)
- Window [IRoot](#)
- XEvent [Event](#)

- bool `mbQuit`

This is a flag telling the system if it is quitting or should quit.

- int `X`

This is the window current X position on the desktop in pixels.

- int `Y`

This is the windows current Y position on the desktop in pixels.

- int `Width`

This is the windows current width in pixels.

- int `Height`

This is the windows current height in pixels.

- bool `Resized`

This is a flag telling the system if the window has been resized and so if the context needs updating.

- bool `Moved`

This is a flag telling the system if the window has been moved.

- bool `Hidden`

This is a flag telling the system if the window is hidden or not.

- bool `Repaint`

This is a flag telling the system that the window requires repainting (updating)

- float `mfRatio`

This is a float storing the ratio of the window to avoid distorting the view.

Static Public Attributes

- static int `DisplayAttributes` [5] = { `GLX_RGBA`, `GLX_DEPTH_SIZE`, 24, `GLX_DOUBLEBUFFER`, `None` }

21.100.1 Detailed Description

This class will create control and destroy desktop windows. This is ugly and disgusting. It will create new windows, link the window with OpenGL and do basic event handling. This is the users access to interact with the OS. Note it is all very OS specific.

Definition at line 11 of file `WTcWindow.h`.

21.100.2 Constructor & Destructor Documentation

21.100.2.1 cWindow::cWindow (HINSTANCE *lphInstance*)

This will create a new [cWindow](#) class. Create a desktop window and [InitialiseOpenGL\(\)](#).
This is Windows OS only.

21.100.2.2 cWindow::cWindow ()

This will create a new [cWindow](#) class. Create a desktop window and [InitialiseOpenGL\(\)](#).
This is Linux OS only.

Definition at line 236 of file WTcWindow.cpp.

21.100.2.3 cWindow::~cWindow ()

Definition at line 262 of file WTcWindow.cpp.

21.100.3 Member Function Documentation

21.100.3.1 void cWindow::DisableInput ()

This will do NOTHING.

Definition at line 294 of file WTcWindow.cpp.

21.100.3.2 void cWindow::EnableInput ()

This will do NOTHING.

Definition at line 293 of file WTcWindow.cpp.

21.100.3.3 void cWindow::EnableOpenGL ()

This will create the OpenGL context and enable OpenGL.

Definition at line 286 of file WTcWindow.cpp.

21.100.3.4 void cWindow::GetMouseSpeed ()

This will update the current mouse speeds. see [_MOUSE](#).

Definition at line 383 of file WTcWindow.cpp.

21.100.3.5 void cWindow::HandleChanges ()

This will handle changes to the window (from the OS).

Definition at line 370 of file WTcWindow.cpp.

21.100.3.6 void cWindow::HandleMessages()

This is an interrupt function that will handle messages from the OS.

Definition at line 304 of file WTcWindow.cpp.

21.100.3.7 void cWindow::InitialiseOpenGL()

This will prepare the Rendering matrix and other OpenGL initialisations.

Definition at line 3 of file WTcWindow.cpp.

21.100.3.8 void cWindow::MakeCurrent()

This will make this context the current one (to receive rendering data).

Definition at line 299 of file WTcWindow.cpp.

21.100.3.9 void cWindow::Move(short liX, short liY)

This will Move the window to position liX,liY (in pixels) on the desktop.

Definition at line 296 of file WTcWindow.cpp.

21.100.3.10 void cWindow::MovePointer(uint32 liX, uint32 liY)

This will move the mouse cursor to the position liX,liY in the current window. see
_MOUSE.

Definition at line 378 of file WTcWindow.cpp.

21.100.3.11 float cWindow::Ratio()

This will return mfRatio.

Definition at line 230 of file WTcWindow.cpp.

21.100.3.12 void cWindow::Resize(short liX, short liY)

This will resize the window to size liX,liY (in pixels).

Definition at line 297 of file WTcWindow.cpp.

21.100.3.13 void cWindow::StartWindow()

This will create the window object and initialise it.

Definition at line 276 of file WTcWindow.cpp.

21.100.4 Member Data Documentation

21.100.4.1 int cWindow::DisplayAttributes = { GLX_RGBA, GLX_DEPTH_SIZE, 24, GLX_DOUBLEBUFFER, None } [static]

Definition at line 26 of file WTcWindow.h.

21.100.4.2 XEvent cWindow::Event

Definition at line 32 of file WTcWindow.h.

21.100.4.3 GLXContext cWindow::glContext

Definition at line 28 of file WTcWindow.h.

21.100.4.4 XWindowAttributes cWindow::gwa

Definition at line 30 of file WTcWindow.h.

21.100.4.5 HDC cWindow::hDC

Definition at line 17 of file WTcWindow.h.

21.100.4.6 int cWindow::Height

This is the windows current height in pixels.

Definition at line 50 of file WTcWindow.h.

21.100.4.7 bool cWindow::Hidden

This is a flag telling the system if the window is hidden or not.

Definition at line 56 of file WTcWindow.h.

21.100.4.8 HGLRC cWindow::hRC

Definition at line 18 of file WTcWindow.h.

21.100.4.9 HWND cWindow::hWnd

Definition at line 16 of file WTcWindow.h.

21.100.4.10 Display* cWindow::lpDisplay

Definition at line 25 of file WTcWindow.h.

21.100.4.11 Window cWindow::lRoot

Definition at line 31 of file WTcWindow.h.

21.100.4.12 Window cWindow::lWindow

Definition at line 31 of file WTcWindow.h.

21.100.4.13 bool cWindow::mbQuit

This is a flag telling the system if it is quitting or should quit.

Definition at line 42 of file WTcWindow.h.

21.100.4.14 float cWindow::mfRatio

This is a float storing the ratio of the window to avoid distorting the view.

Definition at line 60 of file WTcWindow.h.

21.100.4.15 bool cWindow::Moved

This is a flag telling the system if the window has been moved.

Definition at line 54 of file WTcWindow.h.

21.100.4.16 MSG cWindow::msg

Definition at line 19 of file WTcWindow.h.

21.100.4.17 bool cWindow::Repaint

This is a flag telling the system that the window requires repainting (updating)

Definition at line 58 of file WTcWindow.h.

21.100.4.18 bool cWindow::Resized

This is a flag telling the system if the window has been resized and so if the context needs updating.

Definition at line 52 of file WTcWindow.h.

21.100.4.19 XVisualInfo* cWindow::VisualInfo

Definition at line 27 of file WTcWindow.h.

21.100.4.20 WNDCLASS cWindow::wc

Definition at line 15 of file WTcWindow.h.

21.100.4.21 int cWindow::Width

This is the windows current width in pixels.

Definition at line 48 of file WTcWindow.h.

21.100.4.22 XSetWindowAttributes cWindow::WindowAttributes

Definition at line 29 of file WTcWindow.h.

21.100.4.23 int cWindow::X

This is the window current X position on the desktop in pixels.

Definition at line 44 of file WTcWindow.h.

21.100.4.24 int cWindow::Y

This is the windows current Y position on the desktop in pixels.

Definition at line 46 of file WTcWindow.h.

21.101 dwLog Class Reference

Public Member Functions

- [dwLog \(\)](#)
- void [addSink](#) (string name, stream_ptr sink)
- void [addSource](#) (string name)
- bool [bindSourceToSink](#) (const string &sourceName, const string &sinkName)
- bool [printToSource](#) (const string &sourceName, const string &msg)
- stream_ptr [getStream](#) (string name)

21.101.1 Detailed Description

Definition at line 88 of file log.h.

21.101.2 Constructor & Destructor Documentation

21.101.2.1 `dwLog::dwLog() [inline]`

Definition at line 90 of file log.h.

21.101.3 Member Function Documentation

21.101.3.1 `void dwLog::addSink(string name, stream_ptr sink) [inline]`

Definition at line 94 of file log.h.

21.101.3.2 `void dwLog::addSource(string name) [inline]`

Definition at line 98 of file log.h.

21.101.3.3 `bool dwLog::bindSourceToSink(const string & sourceName, const string & sinkName) [inline]`

Definition at line 102 of file log.h.

21.101.3.4 `stream_ptr dwLog::getStream(string name) [inline]`

Definition at line 142 of file log.h.

21.101.3.5 `bool dwLog::printToSource(const string & sourceName, const string & msg) [inline]`

Definition at line 120 of file log.h.

21.102 v2DPolygon Class Reference

This stores mesh data for a single square quadrilateral. This is used for cTextureText.

Static Public Member Functions

- static void `SizeArrays(float lfSize)`

This will scale the polygon to lfsize.

- static void `SetTextCoords(float liRange)`

This will position the texture co-ordinates for a single character in a 1x64 character font texture.

- static void [ResetTextCoords \(\)](#)

This will reset the texture co-ordinates to use the entire texture.

Static Public Attributes

- static float [mpVertex \[12\]](#)

This stores the vertex position array for this polygon.

- static uint16 [mpFaces \[6\]](#)

This stores the face array for this polygon.

- static float [mpTextCoords \[8\]](#)

This stores the vertex texture co-ordinates for this polygon.

- static float [mpNormals \[12\]](#)

This stores the vertex normals data for this polygon.

21.102.1 Detailed Description

This stores mesh data for a single square quadrilateral. This is used for cTextureText.
Definition at line 4 of file WTv2DPolygon.h.

21.102.2 Member Function Documentation

21.102.2.1 void v2DPolygon::ResetTextCoords () [static]

This will reset the texture co-ordinates to use the entire texture.
Definition at line 36 of file WTv2DPolygon.cpp.

21.102.2.2 void v2DPolygon::SetTextCoords (float *lRange*) [static]

This will position the texture co-ordinates for a single character in a 1x64 character
font texture.
Definition at line 30 of file WTv2DPolygon.cpp.

21.102.2.3 void v2DPolygon::SizeArrays (float *lSize*) [static]

This will scale the polygon to lsize.
Definition at line 42 of file WTv2DPolygon.cpp.

21.102.3 Member Data Documentation

21.102.3.1 uint16 v2DPolygon::mpFaces [static]

Initial value:

```
{
    0, 1, 2, 0, 2, 3
}
```

This stores the face array for this polygon.

Definition at line 10 of file WTv2DPolygon.h.

21.102.3.2 float v2DPolygon::mpNormals [static]

Initial value:

```
{
    0.0f, 0.0f, 1.0f,
    0.0f, 0.0f, 1.0f,
    0.0f, 0.0f, 1.0f,
    0.0f, 0.0f, 1.0f
}
```

This stores the vertex normals data for this polygon.

Definition at line 14 of file WTv2DPolygon.h.

21.102.3.3 float v2DPolygon::mpTextCoords [static]

Initial value:

```
{
    1.0f, 0.0f,
    1.0f, 1.0f,
    0.0f, 1.0f,
    0.0f, 0.0f
}
```

This stores the vertex texture co-ordinates for this polygon.

Definition at line 12 of file WTv2DPolygon.h.

21.102.3.4 float v2DPolygon::mpVertex [static]

Initial value:

```
{
    1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f,
    0.0f, 1.0f, 0.0f,
    0.0f, 0.0f, 0.0f
}
```

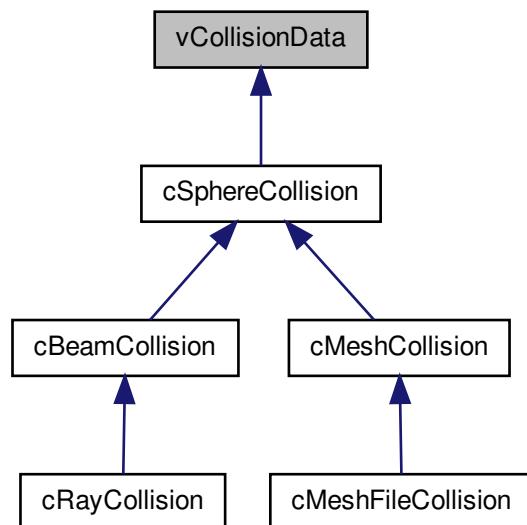
This stores the vertex position array for this polygon.

Definition at line 8 of file WTv2DPolygon.h.

21.103 vCollisionData Class Reference

Virtual Class so [cCollisionObject](#) can access the Collision data object it needs.

Inheritance diagram for vCollisionData:



Public Member Functions

- [vCollisionData \(\)](#)
- virtual [~vCollisionData \(\)](#)
- virtual void [SetSize \(float lfSize\)=0](#)

Will Set the Size of the Collision (For the Sphere aspect of collisions.) This should be the radius of the collision Sphere.

- virtual float [CollisionSize \(\)=0](#)

Will return the Collision Size Value, which is the radius of the Collision Sphere squared.

- virtual [cSphereCollision * Sphere \(\)=0](#)

Will return a pointer if this object contains a sphere collision data object. Otherwise returns 0;.

- virtual `cBeamCollision * Beam ()=0`

Will return a pointer if this object contains a Beam collision data object. Otherwise returns 0;.

- virtual `cMeshCollision * Mesh ()=0`

Will return a pointer if this object contains a Mesh collision data object. Otherwise returns 0;.

- virtual `cRayCollision * Ray ()=0`

Will return a pointer if this object contains a Ray collision data object. Otherwise returns 0;.

- virtual void `Update (cMatrix4 &New)=0`

21.103.1 Detailed Description

Virtual Class so `cCollisionObject` can access the Collision data object it needs.

Definition at line 11 of file WTvCollisionData.h.

21.103.2 Constructor & Destructor Documentation

21.103.2.1 `vCollisionData::vCollisionData() [inline]`

Definition at line 15 of file WTvCollisionData.h.

21.103.2.2 `virtual vCollisionData::~vCollisionData() [inline, virtual]`

Definition at line 16 of file WTvCollisionData.h.

21.103.3 Member Function Documentation

21.103.3.1 `virtual cBeamCollision* vCollisionData::Beam() [pure virtual]`

Will return a pointer if this object contains a Beam collision data object. Otherwise returns 0;.

Implemented in `cSphereCollision`, and `cBeamCollision`.

21.103.3.2 `virtual float vCollisionData::CollisionSize() [pure virtual]`

Will return the Collision Size Value, which is the radius of the Collision Sphere squared.

Implemented in `cSphereCollision`.

21.103.3.3 virtual cMeshCollision* vCollisionData::Mesh() [pure virtual]

Will return a pointer if this object contains a Mesh collision data object. Otherwise returns 0;.

Implemented in [cSphereCollision](#), and [cMeshCollision](#).

21.103.3.4 virtual cRayCollision* vCollisionData::Ray() [pure virtual]

Will return a pointer if this object contains a Ray collision data object. Otherwise returns 0;.

Implemented in [cSphereCollision](#), [cBeamCollision](#), and [cRayCollision](#).

21.103.3.5 virtual void vCollisionData::SetSize(float IfSize) [pure virtual]

Will Set the Size of the Collision (For the Sphere aspect of collisions.) This should be the radius of the collision Sphere.

Implemented in [cSphereCollision](#).

21.103.3.6 virtual cSphereCollision* vCollisionData::Sphere() [pure virtual]

Will return a pointer if this object contains a sphere collision data object. Otherwise returns 0;.

Implemented in [cSphereCollision](#).

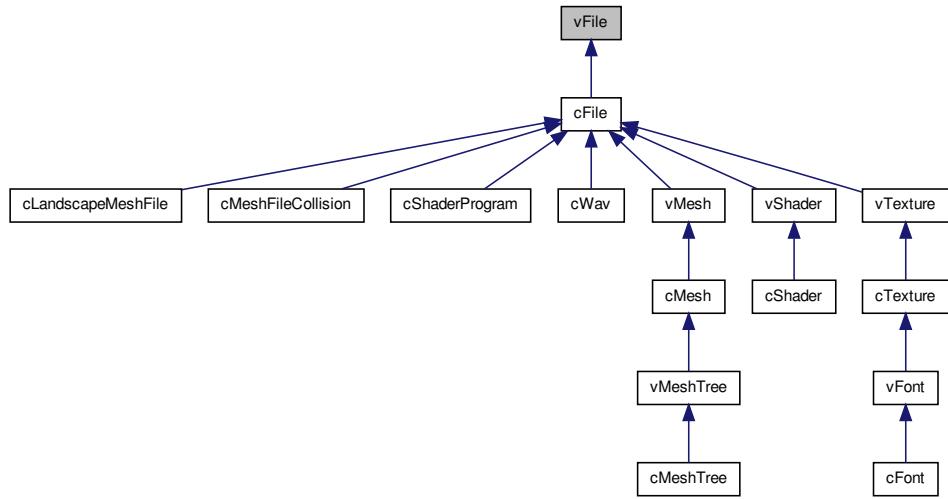
21.103.3.7 virtual void vCollisionData::Update(cMatrix4 & New) [pure virtual]

Implemented in [cSphereCollision](#), [cBeamCollision](#), and [cRayCollision](#).

21.104 vFile Class Reference

This is the virtual file for [cFile](#). It is a virtual representation of the base code for files loaded from a hdd.

Inheritance diagram for vFile:



Public Member Functions

- virtual ~vFile ()
- virtual char * [FileName](#) ()=0

This will return the filename or file reference of this file.

- virtual void [Load](#) ()=0

This will load the file from a hdd into memory.

21.104.1 Detailed Description

This is the virtual file for [cFile](#). It is a virtual representation of the base code for files loaded from a hdd.

Definition at line 5 of file WTvFile.h.

21.104.2 Constructor & Destructor Documentation

21.104.2.1 virtual vFile::~vFile() [inline, virtual]

Definition at line 9 of file WTvFile.h.

21.104.3 Member Function Documentation

21.104.3.1 virtual char* vFile::fileName() [pure virtual]

This will return the filename or file reference of this file.

Implemented in [cFile](#).

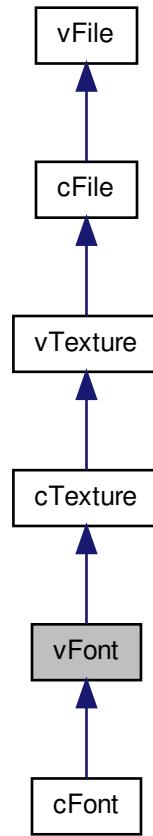
21.104.3.2 virtual void vFile::Load() [pure virtual]

This will load the file from a hdd into memory.

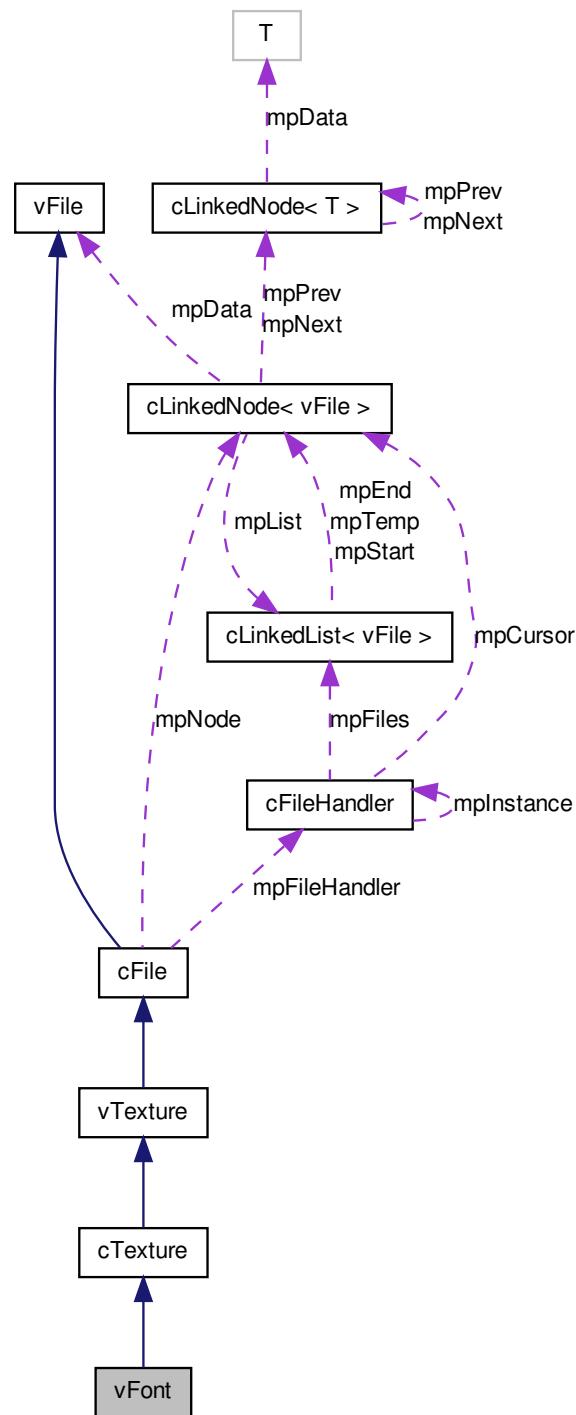
Implemented in [cFile](#).

21.105 vFont Class Reference

Inheritance diagram for vFont:



Collaboration diagram for vFont:



Public Member Functions

- virtual uint8 [Character](#) (uint8 lcChar)=0

21.105.1 Detailed Description

Definition at line 4 of file WTvFont.h.

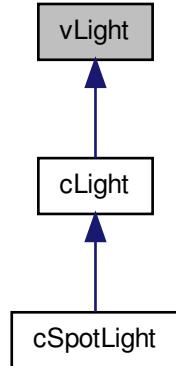
21.105.2 Member Function Documentation

21.105.2.1 virtual uint8 vFont::Character (uint8 *lcChar*) [pure virtual]

Implemented in [cFont](#).

21.106 vLight Class Reference

Inheritance diagram for vLight:



Public Member Functions

- [~vLight](#) ()
- virtual void [PrepareLight](#) ()=0
- virtual void [PrepareLight](#) (uint32 liLight)=0
- virtual float * [Position](#) ()=0
- virtual void [SetID](#) (uint8 liLightID)=0

21.106.1 Detailed Description

A virtual class for [cLight](#)

Definition at line 6 of file WTvLight.h.

21.106.2 Constructor & Destructor Documentation

21.106.2.1 `vLight::~vLight() [inline]`

Definition at line 13 of file WTvLight.h.

21.106.3 Member Function Documentation

21.106.3.1 `virtual float* vLight::Position() [pure virtual]`

Implemented in [cLight](#).

21.106.3.2 `virtual void vLight::PrepareLight(uint32 liLight) [pure virtual]`

Implemented in [cLight](#), and [cSpotLight](#).

21.106.3.3 `virtual void vLight::PrepareLight() [pure virtual]`

Implemented in [cLight](#), and [cSpotLight](#).

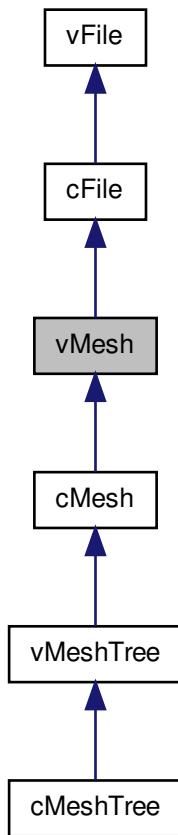
21.106.3.4 `virtual void vLight::SetID(uint8 liLightID) [pure virtual]`

Implemented in [cLight](#).

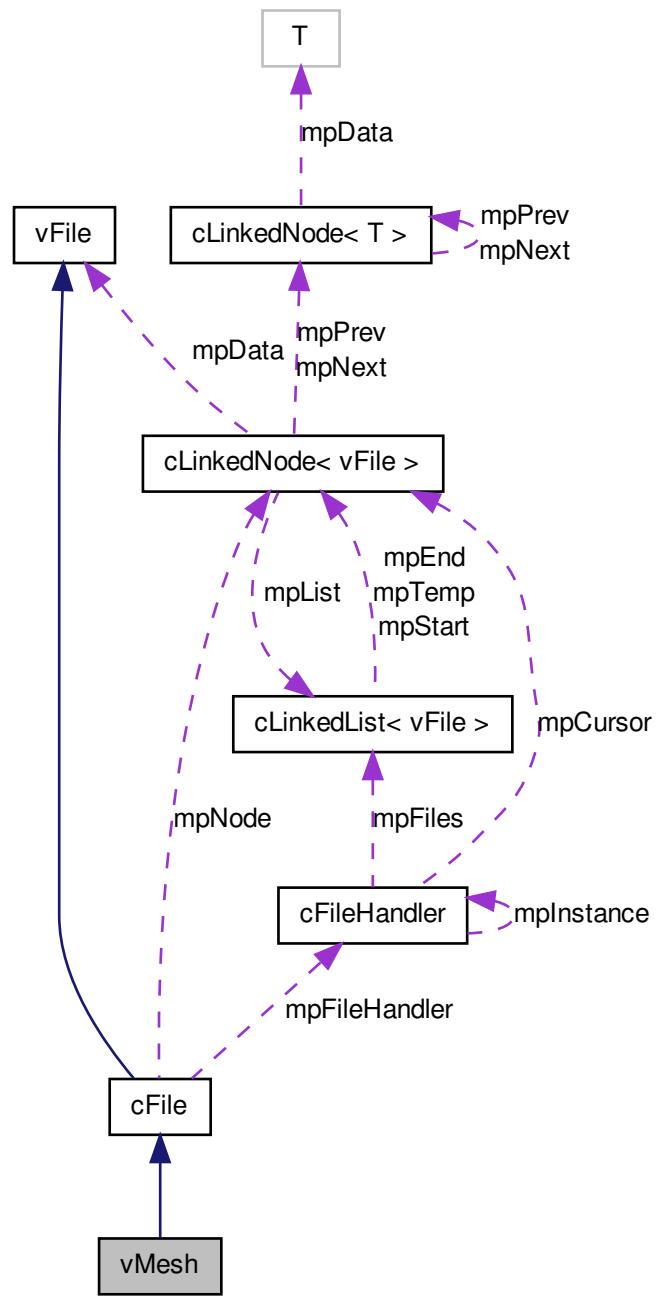
21.107 vMesh Class Reference

This is the virtual base class for a [cMesh](#) object.

Inheritance diagram for vMesh:



Collaboration diagram for vMesh:



Public Member Functions

- virtual ~vMesh ()
- virtual uint32 **Vertex** ()=0

Will return the number of verteces in the mesh.
- virtual uint32 **Faces** ()=0

Will return the number of Faces in the mesh.
- virtual float * **VertexData** ()=0

Will return a pointer to the vertex position data array.
- virtual uint16 * **FaceData** ()=0

Will return a pointer to the face array.
- virtual float * **NormalData** ()=0

Will return a pointer to the vertex normal array.
- virtual float * **UVData** ()=0

Will return a pointer to the texture co-ordinate array.
- virtual void **Position** (float lfX, float lfY, float Z)=0

Will move the meshes centre of rotation by lfX,lfY,lfZ.
- virtual void **PositionX** (float lfX)=0

Will move the meshes centre of rotation by lfX along its X axis.
- virtual void **PositionY** (float lfX)=0

Will move the meshes centre of rotation by lfX along its Y axis.
- virtual void **PositionZ** (float lfX)=0

Will move the meshes centre of rotation by lfX along its Z axis.
- virtual void **BufferMesh** ()=0

This will move the mesh data to the graphics card memory.
- virtual void **RenderMesh** ()=0

This will render the mesh directly from graphics memory.
- virtual void **ResetPosition** ()=0

Will reset the meshes centre of rotation to its original position.
- virtual float **GetSize** ()=0

Will return the Size of the Mesh.
- virtual void **CreateNormalArray** ()=0

Will Generate the normal array from faces and verteces.

21.107.1 Detailed Description

This is the virtual base class for a [cMesh](#) object.

Definition at line 5 of file [WTvMesh.h](#).

21.107.2 Constructor & Destructor Documentation

21.107.2.1 `virtual vMesh::~vMesh() [inline, virtual]`

Definition at line 8 of file [WTvMesh.h](#).

21.107.3 Member Function Documentation

21.107.3.1 `virtual void vMesh::BufferMesh() [pure virtual]`

This will move the mesh data to the graphics card memory.

Implemented in [cMesh](#), and [vMeshTree](#).

21.107.3.2 `virtual void vMesh::CreateNormalArray() [pure virtual]`

Will Generate the normal array from faces and vertexes.

Implemented in [cMesh](#).

21.107.3.3 `virtual uint16* vMesh::FaceData() [pure virtual]`

Will return a pointer to the face array.

Implemented in [cMesh](#), and [vMeshTree](#).

21.107.3.4 `virtual uint32 vMesh::Faces() [pure virtual]`

Will return the number of Faces in the mesh.

Implemented in [cMesh](#), and [vMeshTree](#).

21.107.3.5 `virtual float vMesh::GetSize() [pure virtual]`

Will return the Size of the Mesh.

Implemented in [cMesh](#), and [cMeshTree](#).

21.107.3.6 `virtual float* vMesh::NormalData() [pure virtual]`

Will return a pointer to the vertex normal array.

Implemented in [cMesh](#), and [vMeshTree](#).

21.107.3.7 virtual void vMesh::Position (float lfX, float lfY, float Z) [pure virtual]

Will move the meshes centre of rotation by lfX,lfY,lfZ.

Implemented in [cMesh](#), and [vMeshTree](#).

21.107.3.8 virtual void vMesh::PositionX (float lfX) [pure virtual]

Will move the meshes centre of rotation by lfX along its X axis.

Implemented in [cMesh](#), and [vMeshTree](#).

21.107.3.9 virtual void vMesh::PositionY (float lfX) [pure virtual]

Will move the meshes centre of rotation by lfX along its Y axis.

Implemented in [cMesh](#), and [vMeshTree](#).

21.107.3.10 virtual void vMesh::PositionZ (float lfX) [pure virtual]

Will move the meshes centre of rotation by lfX along its Z axis.

Implemented in [cMesh](#), and [vMeshTree](#).

21.107.3.11 virtual void vMesh::RenderMesh () [pure virtual]

This will render the mesh directly from graphics memory.

Implemented in [cMesh](#), and [vMeshTree](#).

21.107.3.12 virtual void vMesh::ResetPosition () [pure virtual]

Will reset the meshes centre of rotation to its original position.

Implemented in [cMesh](#), and [vMeshTree](#).

21.107.3.13 virtual float* vMesh::UVData () [pure virtual]

Will return a pointer to the texture co-ordinate array.

Implemented in [cMesh](#), and [vMeshTree](#).

21.107.3.14 virtual uint32 vMesh::Vertex () [pure virtual]

Will return the number of vertices in the mesh.

Implemented in [cMesh](#), and [vMeshTree](#).

21.107.3.15 virtual float* vMesh::VertexData() [pure virtual]

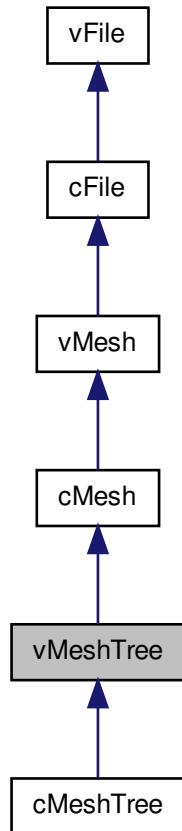
Will return a pointer to the vertex position data array.

Implemented in [cMesh](#), and [vMeshTree](#).

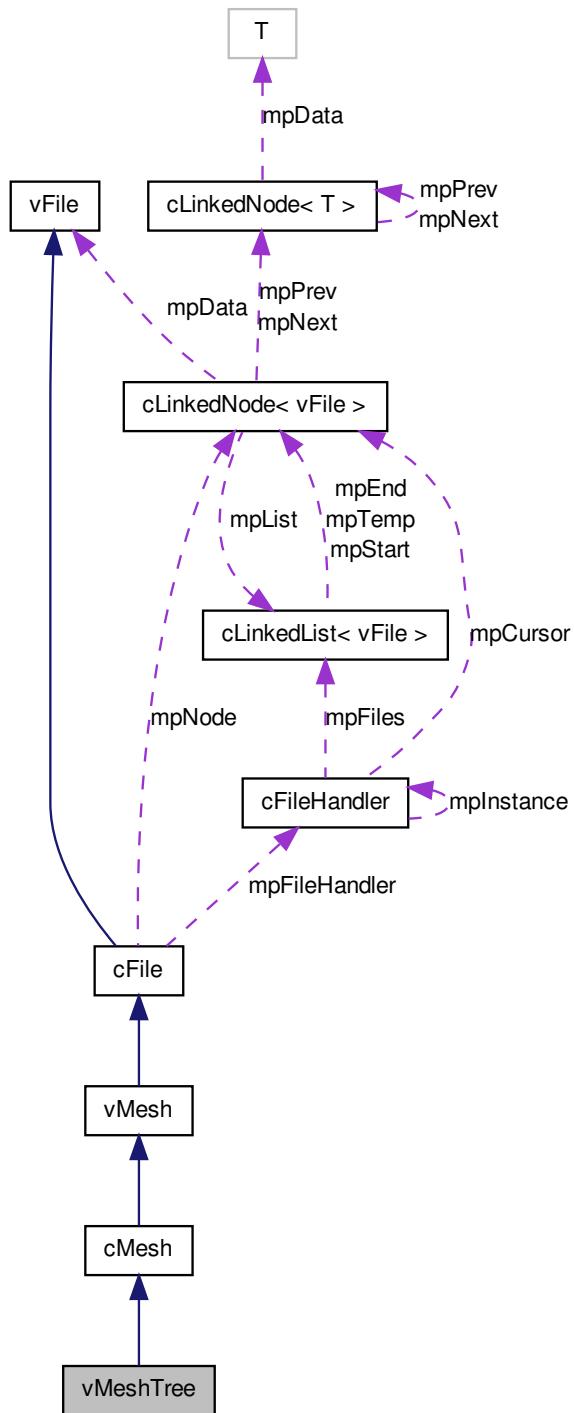
21.108 vMeshTree Class Reference

This is a virtual class for inherited by cMeshTree().

Inheritance diagram for vMeshTree:



Collaboration diagram for vMeshTree:



Public Member Functions

- virtual `~vMeshTree ()`
- virtual `uint32 Size ()=0`
- virtual `cMeshTreeNode * NodeList ()=0`
- virtual `cMeshTreeNode * NodeList (uint32 liCount)=0`
- virtual `uint32 Vertex ()`

This will return the number of verteces in the vertex position array mpVertex.

- virtual `uint32 Faces ()`

This will return the number of faces in the face array mpFaces.

- virtual `float * VertexData ()`

This will return a pointer to the vertex position array.

- virtual `uint16 * FaceData ()`

This will return a pointer to the face array..

- virtual `float * NormalData ()`

This will return a pointer to the array of vertex normals.

- virtual `float * UVData ()`

This will return a pointer to the array of texture co-ordinates.

- virtual `void Position (float lfX, float lfY, float lfZ)`

This will move the objects centre of rotation by lfX,lfY,lfZ.

- virtual `void PositionX (float lfX)`

This will move the objects centre of rotation along its X axis lfX distance.

- virtual `void PositionY (float lfY)`

This will move the objects centre of rotation along its Y axis lfY distance.

- virtual `void PositionZ (float lfZ)`

This will move the objects centre of rotation along its Z axis lfZ distance.

- virtual `void BufferMesh ()`

This will move the mesh data to the graphics card memory.

- virtual `void RenderMesh ()`

This will render the mesh directly from graphics memory.

- virtual `void ResetPosition ()`

This will restore the objects original centre of rotation.

21.108.1 Detailed Description

This is a virtual class for inherited by cMeshTree().

Definition at line 8 of file WTvMeshTree.h.

21.108.2 Constructor & Destructor Documentation

21.108.2.1 `virtual vMeshTree::~vMeshTree() [inline, virtual]`

Definition at line 11 of file WTvMeshTree.h.

21.108.3 Member Function Documentation

21.108.3.1 `virtual void vMeshTree::BufferMesh() [inline, virtual]`

This will move the mesh data to the graphics card memory.

Reimplemented from [cMesh](#).

Definition at line 30 of file WTvMeshTree.h.

21.108.3.2 `virtual uint16* vMeshTree::FaceData() [inline, virtual]`

This will return a pointer to the face array..

Reimplemented from [cMesh](#).

Definition at line 21 of file WTvMeshTree.h.

21.108.3.3 `virtual uint32 vMeshTree::Faces() [inline, virtual]`

This will return the number of faces in the face array mpFaces.

Reimplemented from [cMesh](#).

Definition at line 19 of file WTvMeshTree.h.

21.108.3.4 `virtual cMeshTreeNode* vMeshTree::NodeList() [pure virtual]`

Implemented in [cMeshTree](#).

21.108.3.5 `virtual cMeshTreeNode* vMeshTree::NodeList(uint32 liCount) [pure virtual]`

Implemented in [cMeshTree](#).

21.108.3.6 virtual float* vMeshTree::NormalData() [inline, virtual]

This will return a pointer to the array of vertex normals.

Reimplemented from [cMesh](#).

Definition at line 22 of file WTvMeshTree.h.

21.108.3.7 virtual void vMeshTree::Position(float lfX, float lfY, float lfZ) [inline, virtual]

This will move the objects centre of rotation by lfX,lfY,lfZ.

Reimplemented from [cMesh](#).

Definition at line 25 of file WTvMeshTree.h.

21.108.3.8 virtual void vMeshTree::PositionX(float lfX) [inline, virtual]

This will move the objects centre of rotation along its X axis lfX distance.

Reimplemented from [cMesh](#).

Definition at line 26 of file WTvMeshTree.h.

21.108.3.9 virtual void vMeshTree::PositionY(float lfY) [inline, virtual]

This will move the objects centre of rotation along its Y axis lfY distance.

Reimplemented from [cMesh](#).

Definition at line 27 of file WTvMeshTree.h.

21.108.3.10 virtual void vMeshTree::PositionZ(float lfZ) [inline, virtual]

This will move the objects centre of rotation along its Z axis lfZ distance.

Reimplemented from [cMesh](#).

Definition at line 28 of file WTvMeshTree.h.

21.108.3.11 virtual void vMeshTree::RenderMesh() [inline, virtual]

This will render the mesh directly from graphics memory.

Reimplemented from [cMesh](#).

Definition at line 31 of file WTvMeshTree.h.

21.108.3.12 virtual void vMeshTree::ResetPosition() [inline, virtual]

This will restore the objects original centre of rotation.

Reimplemented from [cMesh](#).

Definition at line 33 of file WTvMeshTree.h.

21.108.3.13 virtual uint32 vMeshTree::Size() [pure virtual]

Implemented in [cMeshTree](#).

21.108.3.14 virtual float* vMeshTree::UVData() [inline, virtual]

This will return a pointer to the array of texture co-ordinates.

Reimplemented from [cMesh](#).

Definition at line 23 of file WTvMeshTree.h.

21.108.3.15 virtual uint32 vMeshTree::Vertex() [inline, virtual]

This will return the number of verteces in the vertex position array mpVertex.

Reimplemented from [cMesh](#).

Definition at line 18 of file WTvMeshTree.h.

21.108.3.16 virtual float* vMeshTree::VertexData() [inline, virtual]

This will return a pointer to the vertex position array.

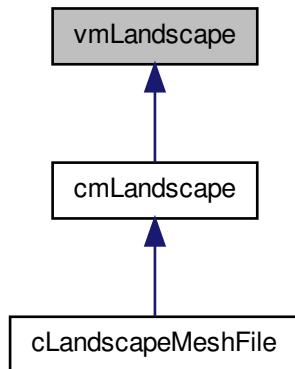
Reimplemented from [cMesh](#).

Definition at line 20 of file WTvMeshTree.h.

21.109 vmLandscape Class Reference

This is the virtual base class for [cmLandscape](#).

Inheritance diagram for vmLandscape:



Public Member Functions

- [vmLandscape \(\)](#)
- virtual void [RenderMesh \(\)=0](#)
- virtual void [PrepareLandscape \(\)=0](#)

This will prepare absent data for the landscape. Normal, UV and Face data.

- virtual float [GetHeightLocal \(float lfX, float lfZ\)=0](#)

This will return the interpolated height at the local position lfX,lfZ relative to the landscapes 0,0 corner.

- virtual float [GetVertexHeight \(uint32 liX, uint32 liZ\)=0](#)

This will return the height of the vertex numbered liX,liZ in the vertex position array.

- virtual void [SetHeight \(uint32 liX, uint32 liZ, float lfHeight\)=0](#)

This will set the height of the vertex numbered liX,liZ in the vertex position array.

- virtual void [CreateNormalArray \(\)=0](#)

This will update the vertex normal array based on the current vertex positions.

- virtual void [Randomize \(float liHeight, uint8 liSize\)=0](#)

This will randomise the landscape to a maximum height of liHeight. Then each vertex will be smoothed based on the heights of all verteces within liSize segments.

- virtual void [Randomize \(float liHeight\)=0](#)

This will randomise the landscape to a maximum height of lfheight.

- virtual void **Randomize** (uint32 Lines, float lfLandscapeHeight)=0

*This will randomise the landscape by randomly bisecting the landscape Lines times.
Each bisection raises the land on one side by lfLandscapeHeight.*

- virtual uint32 **Width** ()=0

This will return miXSize.

- virtual uint32 **Length** ()=0

This will return miZSize.

21.109.1 Detailed Description

This is the virtual base class for [cmLandscape](#).

Definition at line 4 of file WTvmLandscape.h.

21.109.2 Constructor & Destructor Documentation

21.109.2.1 **vmLandscape::vmLandscape()** [inline]

Definition at line 10 of file WTvmLandscape.h.

21.109.3 Member Function Documentation

21.109.3.1 virtual void **vmLandscape::CreateNormalArray()** [pure virtual]

This will update the vertex normal array based on the current vertex positions.

Implemented in [cmLandscape](#).

21.109.3.2 virtual float **vmLandscape::GetHeightLocal(float lfX, float lfZ)** [pure virtual]

This will return the interpolated height at the local position lfX,lfZ relative to the landscapes 0,0 corner.

Implemented in [cmLandscape](#).

21.109.3.3 virtual float **vmLandscape::GetVertexHeight(uint32 liX, uint32 liZ)** [pure virtual]

This will return the height of the vertex numbered liX,liZ in the vertex position array.

Implemented in [cmLandscape](#).

21.109.3.4 virtual uint32 vmLandscape::Length() [pure virtual]

This will return miZSize.

Implemented in [cmLandscape](#).

21.109.3.5 virtual void vmLandscape::PrepareLandscape() [pure virtual]

This will prepare absent data for the landscape. Normal, UV and Face data.

Implemented in [cmLandscape](#).

21.109.3.6 virtual void vmLandscape::Randomize(uint32 Lines, float lfLandscapeHeight) [pure virtual]

This will randomise the landscape by randomly bisecting the landscape Lines times.
Each bisection raises the land on one side by lfLandscapeHeight.

Implemented in [cmLandscape](#).

21.109.3.7 virtual void vmLandscape::Randomize(float liHeight) [pure virtual]

This will randomise the landscape to a maximum height of liheight.

Implemented in [cmLandscape](#).

21.109.3.8 virtual void vmLandscape::Randomize(float liHeight, uint8 liSize) [pure virtual]

This will randomise the landscape to a maximum height of liHeight. Then each vertex
will be smoothed based on the heights of all verteces within liSize segments.

Implemented in [cmLandscape](#).

21.109.3.9 virtual void vmLandscape::RenderMesh() [pure virtual]

Implemented in [cmLandscape](#).

21.109.3.10 virtual void vmLandscape::SetHeight(uint32 liX, uint32 liZ, float lfHeight) [pure virtual]

This will set the height of the vertex numbered liX,liZ in the vertex position array.

Implemented in [cmLandscape](#).

21.109.3.11 virtual uint32 vmLandscape::Width() [pure virtual]

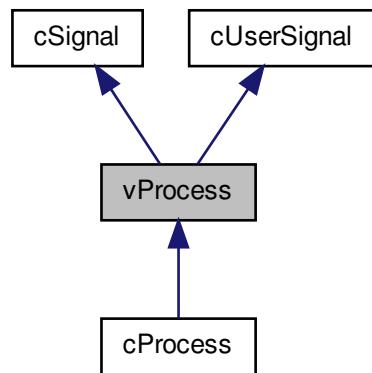
This will return miXSize.

Implemented in [cmLandscape](#).

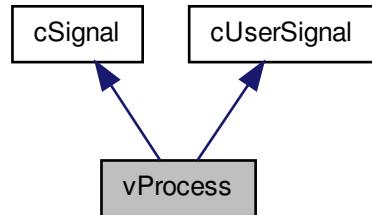
21.110 vProcess Class Reference

This is the virtual base class for a system process, see [cProcess](#).

Inheritance diagram for vProcess:



Collaboration diagram for vProcess:



Public Member Functions

- virtual [~vProcess \(\)](#)
- virtual void [Run \(\)=0](#)

Virtual function linking code to update (or run) the process by single frame.
- virtual void [AdditionalKillFunctionality \(\)=0](#)

Virtual Functions to allow additional commands to be processed when a kill signal is received by an object. This can be user modified for classes inheriting [cProcess](#).
- virtual void [AdditionalSleepFunctionality \(\)=0](#)

Virtual Functions to allow additional commands to be processed when a sleep signal is received by an object. This can be user modified for classes inheriting [cProcess](#).
- virtual void [AdditionalWakeFunctionality \(\)=0](#)

Virtual Functions to allow additional commands to be processed when a wake signal is received by an object. This can be user modified for classes inheriting [cProcess](#).
- virtual [cParentStack * ParentStack \(\)=0](#)

Virtual Function for:

21.110.1 Detailed Description

This is the virtual base class for a system process, see [cProcess](#).

Definition at line 8 of file WTvProcess.h.

21.110.2 Constructor & Destructor Documentation

21.110.2.1 virtual vProcess::~vProcess () [inline, virtual]

Definition at line 11 of file WTvProcess.h.

21.110.3 Member Function Documentation

21.110.3.1 virtual void vProcess::AdditionalKillFunctionality () [pure virtual]

Virtual Functions to allow additional commands to be processed when a kill signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

Reimplemented from [cSignal](#).

Implemented in [cProcess](#).

21.110.3.2 virtual void vProcess::AdditionalSleepFunctionality() [pure virtual]

Virtual Functions to allow additional commands to be processed when a sleep signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

Reimplemented from [cSignal](#).

Implemented in [cProcess](#).

21.110.3.3 virtual void vProcess::AdditionalWakeFunctionality() [pure virtual]

Virtual Functions to allow additional commands to be processed when a wake signal is received by an object. This can be user modified for classes inheriting [cProcess](#).

Reimplemented from [cSignal](#).

Implemented in [cProcess](#).

21.110.3.4 virtual cParentStack* vProcess::ParentStack() [pure virtual]

Virtual Function for.

Implemented in [cProcess](#).

21.110.3.5 virtual void vProcess::Run() [pure virtual]

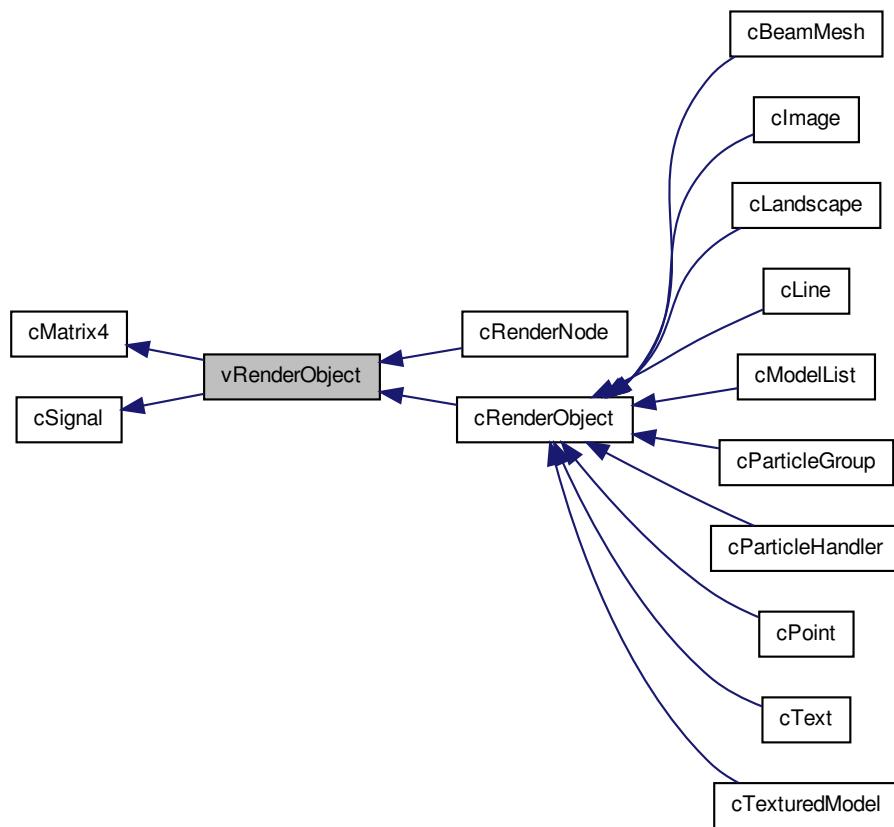
Virtual function linking code to update (or run) the process by single frame.

Implemented in [cProcess](#).

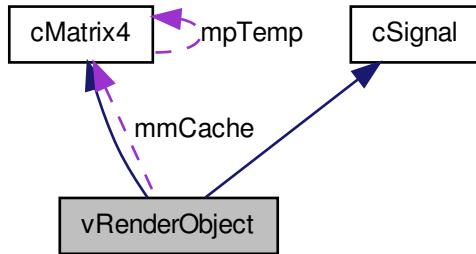
21.111 vRenderObject Class Reference

virtual function for producing renderable objects. see [cRenderObject](#).

Inheritance diagram for vRenderObject:



Collaboration diagram for vRenderObject:



Public Member Functions

- virtual `~vRenderObject ()`
- virtual `cRenderNode * Renderer ()=0`
Returns the `cRenderNode` which owns this object.
- virtual void `LinkCollisionObject (cCollisionObject *lpObj)`
Links The `cCollisionObject` lpObj to this Renderable Object.
- virtual void `RenderPainter ()=0`
Renders this object from the `cPainter` render list to the screen.
- virtual void `RenderToPainter ()=0`
Renders this object to the `cPainter` render list.
- virtual void `Render ()=0`
Renders this object to the screen.
- virtual void `AdditionalRenderFunctions ()`
*This will store any additional functions to be performed as the object is rendered.
Updating Caches, etc.*
- virtual void `UpdateCache ()=0`
This will update the cache matrix.
- virtual float * `GetPos ()=0`
This will return the position of the selected object.
- virtual float * `GetGlobalPos ()=0`

*This will return the global position of the object as rendered at the end of last frame.
Note, this will contain the camera matrix.*

Public Attributes

- **cMatrix4 mmCache**

*Matrix which Stores the final global position matrix of the object from the last frame.
This is used to make collisions consistent and for finding object Global Positions.
Note the Camera MAtrix will be included in this.*

21.111.1 Detailed Description

virtual function for producing renderable objects. see [cRenderObject](#).

Definition at line 10 of file WTvRenderObject.h.

21.111.2 Constructor & Destructor Documentation

21.111.2.1 virtual vRenderObject::~vRenderObject() [inline, virtual]

Definition at line 19 of file WTvRenderObject.h.

21.111.3 Member Function Documentation

21.111.3.1 virtual void vRenderObject::AdditionalRenderFunctions() [inline, virtual]

This will store any additional functions to be performed as the object is rendered. Updating Caches, etc.

Reimplemented in [cRenderNode](#), and [cRenderObject](#).

Definition at line 37 of file WTvRenderObject.h.

21.111.3.2 virtual float* vRenderObject::GetGlobalPos() [pure virtual]

This will return the global position of the object as rendered at the end of last frame.
Note, this will contain the camera matrix.

Implemented in [cRenderNode](#), and [cRenderObject](#).

21.111.3.3 virtual float* vRenderObject::GetPos() [pure virtual]

This will return the position of the selected object.

Implemented in [cRenderNode](#), and [cRenderObject](#).

21.111.3.4 virtual void vRenderObject::LinkCollisionObject (cCollisionObject * *lpObj*)
[inline, virtual]

Links The [cCollisionObject](#) lpObj to this Renderable Object.

Reimplemented in [cRenderNode](#), and [cRenderObject](#).

Definition at line 23 of file WTvRenderObject.h.

21.111.3.5 virtual void vRenderObject::Render () [pure virtual]

Renders this object to the screen.

Implemented in [cRenderNode](#), [cBeamMesh](#), [cImage](#), [cLandscape](#), [cLine](#), [cModelList](#), [cParticleHandler](#), [cParticleGroup](#), [cPoint](#), [cTexturedModel](#), [cText](#), and [cRenderObject](#).

21.111.3.6 virtual cRenderNode* vRenderObject::Renderer () [pure virtual]

Returns the [cRenderNode](#) which owns this object.

Implemented in [cRenderNode](#), and [cRenderObject](#).

21.111.3.7 virtual void vRenderObject::RenderPainter () [pure virtual]

Renders this object from the [cPainter](#) render list to the screen.

Implemented in [cRenderNode](#), [cParticleHandler](#), [cParticleGroup](#), and [cRenderObject](#).

21.111.3.8 virtual void vRenderObject::RenderToPainter () [pure virtual]

Renders this object to the [cPainter](#) render list.

Implemented in [cRenderNode](#), [cBeamMesh](#), [cImage](#), [cLandscape](#), [cLine](#), [cModelList](#), [cParticleHandler](#), [cParticleGroup](#), [cPoint](#), [cTexturedModel](#), [cText](#), and [cRenderObject](#).

21.111.3.9 virtual void vRenderObject::UpdateCache () [pure virtual]

This will update the cache matrix.

Implemented in [cRenderNode](#), and [cRenderObject](#).

21.111.4 Member Data Documentation

21.111.4.1 cMatrix4 vRenderObject::mmCache

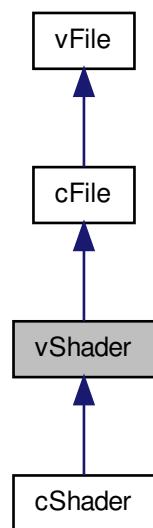
Matrix which Stores the final global position matrix of the object from the last frame. This is used to make collisions consistent and for finding object Global Positions. Note the Camera MAtrix will be included in this.

Definition at line 17 of file WTvRenderObject.h.

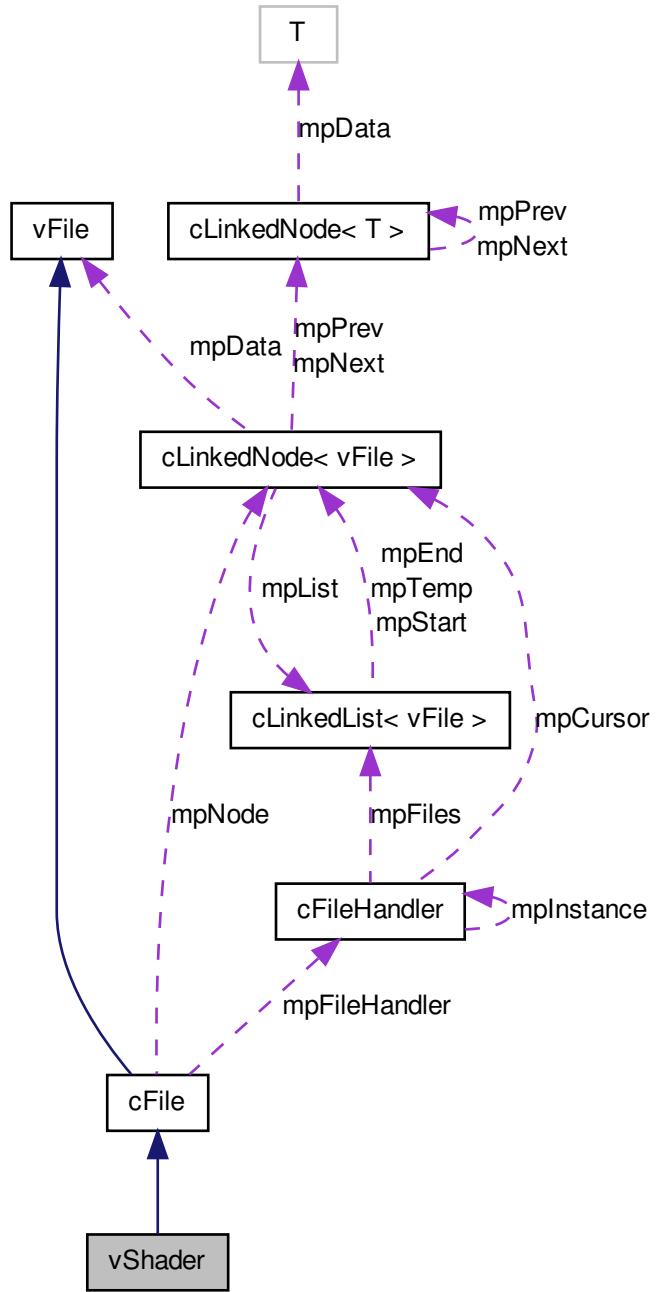
21.112 vShader Class Reference

This is the virtual base class for the Shader Files - both Vertex and Fragment.

Inheritance diagram for vShader:



Collaboration diagram for vShader:



Public Member Functions

- virtual `~vShader()`

Public Destructor.

- virtual uint32 `ID()=0`

Will return the Shaders ID number.

21.112.1 Detailed Description

This is the virutal base class for the Shader Files - both Vertex and Fragment.

Definition at line 5 of file WTvShader.h.

21.112.2 Constructor & Destructor Documentation

21.112.2.1 virtual vShader::~vShader() [inline, virtual]

Public Destructor.

Definition at line 9 of file WTvShader.h.

21.112.3 Member Function Documentation

21.112.3.1 virtual uint32 vShader::ID() [pure virtual]

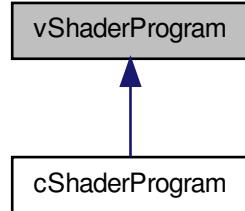
Will return the Shaders ID number.

Implemented in [cShader](#).

21.113 vShaderProgram Class Reference

This is a virtual class for a Shader Program. A shader program is a series of cShader() objects to be compiled into a single Shader Program.

Inheritance diagram for vShaderProgram:



Public Member Functions

- `~vShaderProgram ()`
- virtual void `Use ()=0`

This will make the system Use this Shader Program, until another is set or a shader program is turned off.

- virtual uint32 `ID ()=0`

This will return the Program ID of this Shader Program.

21.113.1 Detailed Description

This is a virtual class for a Shader Program. A shader program is a series of cShader() objects to be compiled into a single Shader Program.

Definition at line 7 of file WTvShaderProgram.h.

21.113.2 Constructor & Destructor Documentation

21.113.2.1 vShaderProgram::~vShaderProgram () [inline]

Definition at line 11 of file WTvShaderProgram.h.

21.113.3 Member Function Documentation

21.113.3.1 virtual uint32 vShaderProgram::ID () [pure virtual]

This will return the Program ID of this Shader Program.

Implemented in [cShaderProgram](#).

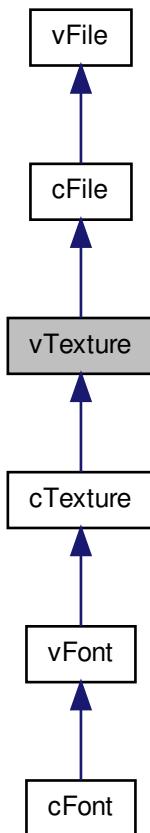
21.113.3.2 virtual void vShaderProgram::Use() [pure virtual]

This will make the system Use this Shader Program, until another is set or a shader program is turned off.

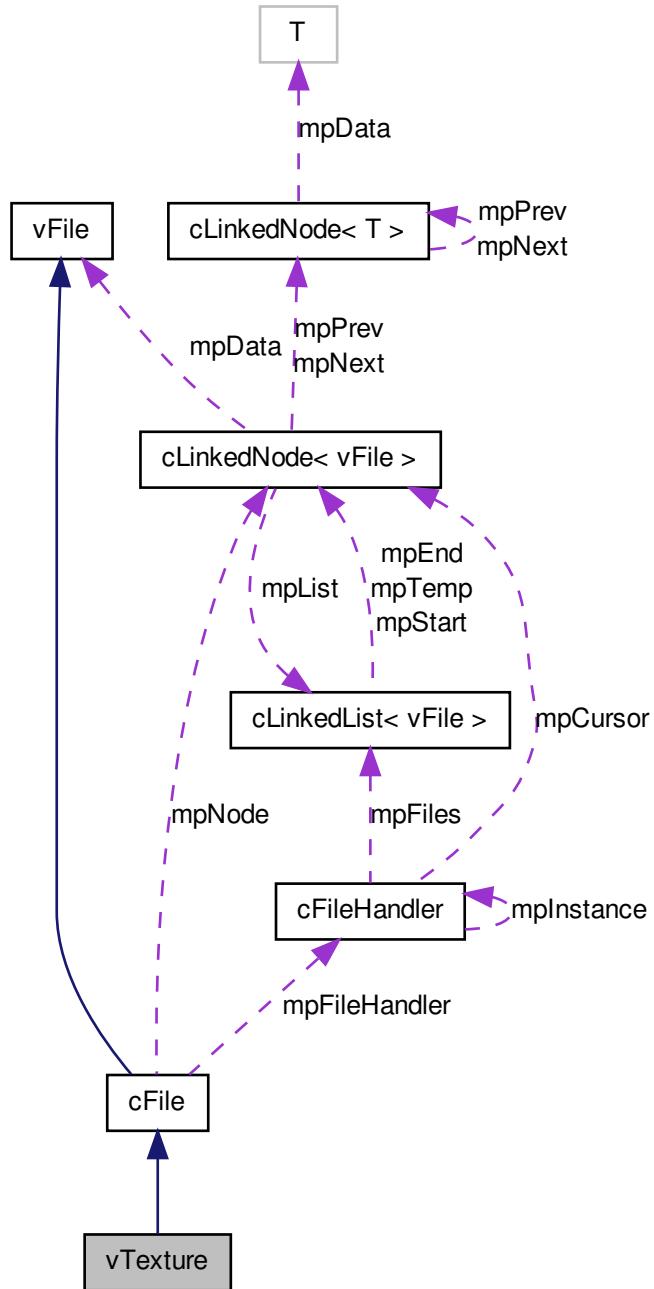
Implemented in [cShaderProgram](#).

21.114 vTexture Class Reference

Inheritance diagram for vTexture:



Collaboration diagram for vTexture:



Public Member Functions

- virtual [~vTexture \(\)](#)
- virtual uint32 [Width \(\)=0](#)
- virtual uint32 [Height \(\)=0](#)
- virtual uint32 [Depth \(\)=0](#)
- virtual void [UpdateTexture \(\)=0](#)
- virtual void [BindTexture \(\)=0](#)
- virtual uint8 * [Data \(\)=0](#)
- virtual uint32 [TextureNumber \(\)=0](#)

21.114.1 Detailed Description

Definition at line 4 of file WTvTexture.h.

21.114.2 Constructor & Destructor Documentation

21.114.2.1 virtual vTexture::~vTexture () [inline, virtual]

Definition at line 7 of file WTvTexture.h.

21.114.3 Member Function Documentation

21.114.3.1 virtual void vTexture::BindTexture () [pure virtual]

Implemented in [cFont](#), and [cTexture](#).

21.114.3.2 virtual uint8* vTexture::Data () [pure virtual]

Implemented in [cTexture](#).

21.114.3.3 virtual uint32 vTexture::Depth () [pure virtual]

Implemented in [cFont](#), and [cTexture](#).

21.114.3.4 virtual uint32 vTexture::Height () [pure virtual]

Implemented in [cFont](#), and [cTexture](#).

21.114.3.5 virtual uint32 vTexture::TextureNumber () [pure virtual]

Implemented in [cTexture](#).

21.114.3.6 virtual void vTexture::UpdateTexture() [pure virtual]

Implemented in [cFont](#), and [cTexture](#).

21.114.3.7 virtual uint32 vTexture::Width() [pure virtual]

Implemented in [cTexture](#).