# gauge3d Reference Manual

0.3.0

Generated by Doxygen 1.2.4

Wed Jan 31 01:15:27 2001

# Contents

# Chapter 1

# the Grand Unified 3D Game Engine

## 1.1 Concepts

The Grand Unified 3D Game Engine (GAUGE) is more that just a game engine. Many of the underlying classes in GAUGE actually change the way things are programmed at a low level. Before you can dive into the GAUGE class specifications, you should go over these basic concepts.

### 1.1.1 Smart Pointers

GAUGE makes extensive use of smart pointers. These are pointers which automatically detect when an object is no longer in use and delete it for you. These are somewhat similar to Java pointers, but have some key differences that you need to be aware of.

Here's an example:

```
GSmartPointer<int> ptr;

ptr = NEW int;

ptr = NULL;
```

Now, if ptr was just an int∗, the above would be a memory leak. The smart pointer, on the other hand, detects that the int is no longer in use, and deletes it automatically. Other than that, a smart pointer works exactly like a regular pointer.

GSmartPointer works through reference-counting. Every object has an integer reference count associated with it. This integer is stored in the four bytes preceding the object, and is allocated automatically by GAUGE's operator new. The reference count starts at zero. Once you set a GSmartPointer to point at that memory chunk, the reference count is incremented. When the GSmartPointer is set to point at something else, the reference count is decremented. If it hits 0 again, the object is deleted.

There are several things to watch out for when using smart pointers:

- Once an object has a smart pointer pointing at it, it's stuck in the smart pointer system. To keep things organized, you should avoid situations where both a smart pointer and a normal pointer point at the same object.
- Do not set a smart pointer to any address which was not at some point returned by operator new. Obviously, such addresses do not have reference counts and can not be deleted, so the program will crash or memory corruption will occur if you do this.
- Circular links are not detected. So, if you have two objects which contain smart pointers pointing at each other, they will never be deleted, even if no other part of the program points at them. This is different from Java.
- The speed of GSmartPointer has not been measured, but it is definately slower that using regular pointers. I do not believe that the speed difference is very large, but you should still use regular pointers whenever it is reasonable to do so.

The main reason to use smart pointers is that you no longer have to worry about who has to delete what. You can have functions which return pointers to objects without any problems.

For example, GDirectory is a class which represents a directory. One of the functions of GDirectory, OpenFile(), returns a smart pointer to a GFile. Now, if GAUGE did not have smart pointers, the function would have to return a normal pointer to a file. Then, you have a problem: Who's responsibility is it to delete the file?

As it turns out, some subclasses of GDirectory will return the same pointer if you open the same file multiple times. So, if the caller were to delete the file, it would cause other parts of the engine to crash if they attempted to access the same file. So, clearly, GDirectory would have to maintain control over when the file is deleted. This means that it would have to have another function, CloseFile(GAUGE3D::GFile GFile∗ file).

Having to call the GDirectory again to close a file can be a huge problem. For example, say you load up a file, and then pass it on to another class to be processed. Then, you would have to find out from that class when the file needs to be deleted.

The solution to all of this is smart pointers. GDirectory::OpenFile() returns a smart pointer to a GFile. This way, no one has to worry about who has to delete the file. If the directory does not want the file to be deleted, it simply keeps a smart pointer to that file. No CloseFile() function is necessary.

One last note: Writing out ”GSmartPointer<MyObject>” can get annoying very quicly. In GAUGE, we use the convention of typedef'ing type names with a prefix of 'p' as being a smart pointer to that object. So, in the header file that defines "MyObject", you'd see this:

```
typedef GSmartPointer<MyObject> pMyObject;
```

Using pMyObject makes code much more readable and easier to type.

### 1.1.2 Plugins

GAUGE makes extensive use of plugins. A plugin is a program module which is dynamically loaded through an external shared library (DLL). The advantage of using plugins is that you can add functionality to the engine without actually editing or recompiling GAUGE itself.

GAUGE uses plugins to support various file formats. For example, there is a plugin to load 3D Studio model files and another plugin to load Quake2 MD2 models. Even GAUGE's custom model format, GMDL, is loaded (and saved) through a plugin.

GAUGE also uses plugins for many other things. One example is 3D rendering. GAUGE can use either OpenGL or Direct3D for rendering. The engine itself, however, knows nothing about either API. Instead, it uses either the GL or D3D rendering plugin to render images.

For more info about how to load plugins, see the GObjectLoader and GPlugin classes.

### 1.1.3 More to Come!

There are several other interesting concepts that are slated to be implemented in the next version of GAUGE, including cookies, abstract data chunks, a new threading paradigm, and more.

Now, head on over to the modules section and start exploring!

# Chapter 2

# gauge3d Module Index

## 2.1 gauge3d Modules

Here is a list of all modules:

# Chapter 3

# gauge3d Hierarchical Index

## 3.1 gauge3d Class Hierarchy

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

# Chapter 4

# gauge3d Compound Index

## 4.1 gauge3d Compound List

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

# Chapter 5

# gauge3d Module Documentation

## 5.1 Common Classes

Common classes that are widely used throughout the engine.

### Compounds

- class GAUGE3D::GArray
- class GAUGE3D::GStridedArray
- class GAUGE3D::GException
- class GAUGE3D::GString
- class GAUGE3D::GSmartPointer
- class GAUGE3D::GObject

### Support macros for QueryInterface.

- #define GAUGE3D_SET_UID(HIGH,MID,LOW)

  *Set the UID for a class.*

- #define GAUGE3D_DECLARE_INTERFACES(CLASSNAME)

  *Begin interface declaration block for a class.*

- #define GAUGE3D_DECLARE_BASECLASS(BASECLASSNAME)

  *Declare inherited interfaces for a class.*

- #define GAUGE3D_DECLARE_EMBEDDED_INTERFACE(INTERFACE_OBJECT)

  *Declare embedded interfaces for a class.*

- #define GAUGE3D_END_INTERFACES

  *End interface declaration block.*

## PI definitions

To avoid ambiguities in pi constants, we define both PI and M_PI.

- #define M_PI 3.14159265358979323846

  *pi.*

- #define M_PI_2 1.57079632679489661923

  *pi/2.*

- #define M_PI_4 0.78539816339744830962

  *pi/4.*

- #define PI 3.14159265358979323846

  *pi.*

- #define PI_2 1.57079632679489661923

  *pi/2.*

- #define PI_4 0.78539816339744830962

  *pi/4.*

## Standard types

Some of these will be different on some systems.

- typedef unsigned int uint

  *unsigned int.*

- typedef unsigned short ushort

  *unsigned short.*

- typedef unsigned long ulong

  *unsigned long.*

- typedef unsigned char uchar

  *unsigned char.*

- typedef char int8

  *8-bit signed integer.*

- typedef short int16

  *16-bit signed integer.*

- typedef int int32

  *32-bit signed integer.*

- typedef long long int64

  *64-bit signed integer.*

- typedef unsigned char uint8

  *8-bit unsigned integer.*

- typedef unsigned short uint16

  *16-bit unsigned integer.*

- typedef unsigned int uint32

  *32-bit unsigned integer.*

- typedef unsigned long long uint64

  *64-bit unsigned integer.*

- typedef float float32

  *32-bit floating point number.*

- typedef double float64

  *64-bit floating point number.*

- typedef double tTime

  *Time value, measured in seconds.*

## Defines

- #define GAUGE3D_FATAL_ERROR(CAUSE)

  *Abort execution due to a fatal error.*

## Typedefs

- typedef GSmartPointer<GObject> pGObject

  *Smart pointer to a GObject.*

## Functions

- template<class tType> GSmartPointer<tType> QueryInterface (pGObject object)

  *See if an object supports a particular interface.*

### 5.1.1 Detailed Description

Common classes that are widely used throughout the engine.

This module is used by every class in GAUGE. It includes stddefs.h, which must be included by every GAUGE source file. It also includes the memory allocator, GSmartPointer, GObject, GArray, GString, etc.

### 5.1.2 Define Documentation

#### 5.1.2.1 #define GAUGE3D_FATAL_ERROR(CAUSE)

**Initializer:**

```
\
  throw GAUGE3D::GException(CAUSE, __FILE__, __LINE__);
```

Abort execution due to a fatal error.

**Parameters:**

*CAUSE* A human-readable string describing the cause of the error.

This macro will create and throw a GException. It should be used only when a fatal error occurs. The program will quit, possibly printing an error message, when this is used.

#### 5.1.2.2 #define GAUGE3D_SET_UID(HIGH, MID, LOW)

**Initializer:**

```
\
public:                                                                  \
   enum {_UID_HIGH = HIGH, _UID_MID = MID, _UID_LOW = LOW};              \
   virtual pGObject _QueryInterface(uint32 high, uint32 mid, uint32 low);\
private:
```

Set the UID for a class.

GAUGE3D_SET_UID should be used in the definition for any class that is derived from GObject. The numbers are the high, middle, and low 32-bit integers in the 96-bit GAUGE unique identifier for the class. (use gauge3d-uidgen to create them)

#### 5.1.2.3 #define GAUGE3D_DECLARE_INTERFACES(CLASSNAME)

**Initializer:**

```
\
pGObject CLASSNAME::_QueryInterface(uint32 high, uint32 mid, uint32 low)\
{                                                                       \
   pGObject ret;                                                        \
   if((high == _UID_HIGH) && (mid == _UID_MID) && (low  == _UID_LOW))   \
     return this;
```

Begin interface declaration block for a class.

GAUGE3D_DECLARE_INTERFACES should appear in the cpp file for any class that uses GAUGE3D_SET_UID. It should be followed by a GAUGE3D_DECLARE_BASECLASS for each base class (but not base class's base classes, etc.) and a GAUGE3D_DECLARE_EMBEDDED_INTERFACE for every embedded interface. These should preferably be ordered with the most likely requested interface on top. After all of these, GAUGE3D_END_INTERFACES should be used.

#### 5.1.2.4 #define GAUGE3D_DECLARE_BASECLASS(BASECLASSNAME)

**Initializer:**

```
\
   if((ret = BASECLASSNAME::_QueryInterface(high, mid, low)) != NULL)\
     return ret;
```

Declare inherited interfaces for a class.

GAUGE3D_DECLARE_BASECLASS declares direct base classes. Not the base classes of base classes, just direct inherited classes. So, if you are not using multiple inheritance, you should never need more than one of these.

#### 5.1.2.5 #define GAUGE3D_DECLARE_EMBEDDED_INTERFACE(INTERFACE_OBJECT)

**Initializer:**

```
\
   if((ret = (INTERFACE_OBJECT)._QueryInterface(high, mid, low)) != NULL)\
    return ret;
```

Declare embedded interfaces for a class.

GAUGE3D_DECLARE_EMBEDDED_INTERFACE declares interfaces from which this class is not inherited, but which define additional functionality that this class has. The parameter should be an actual instantiated copy of the interface, not the classname of it. This is because that interface's QueryInterface function needs to be called.

#### 5.1.2.6 #define GAUGE3D_END_INTERFACES

**Initializer:**

```
\
   return ret;                                                \
}
```

End interface declaration block.

Place GAUGE3D_END_INTERFACES after all you interface declarations.

### 5.1.3 Typedef Documentation

#### 5.1.3.1 typedef double GAUGE3D::tTime

Time value, measured in seconds.

The time from which this value mesures is completely arbitrary, but will be the same for all tTime values during a particular run of the engine. If you want to get the date, use the ANSI-C functions created for this purpose.

### 5.1.4 Function Documentation

#### 5.1.4.1 template<class tType> GSmartPointer< tType > GAUGE3D::QueryInterface<tType> (pGObject *object*) `[inline]`

See if an object supports a particular interface.

**Parameters:**

***tType*** The type of the interface you want.

***object*** A smart pointer to the object from which you are requesting the interface.

**Returns:**

A smart pointer to the interface you requested, or NULL if the object does not support that interface.

This function determines if the given object supports an interface of type tType. The returned interface may be either an inherited interface or an embedded interface.

For example, if you had an object of type GObject and you wanted to see if it was a GFile, you would do this: QueryInterface<GFile>(myObject);

This function would be a member of GObject, but MSVC doesn't like templated member functions.

## 5.2 3D Math

Classes for performing 3D math calculations.

### Compounds

- class GAUGE3D::GAngles
- class GAUGE3D::GLine
- class GAUGE3D::GPlane
- class GAUGE3D::GPolygon
- class GAUGE3D::GQuaternion
- class GAUGE3D::GSegment
- class GAUGE3D::GTriangle
- class GAUGE3D::GVector

### Vector Constants

- const GVector GVECTOR_I = GVector(1.0, 0.0, 0.0)

  *Unit vector in the positive x direction.*

- const GVector GVECTOR_J = GVector(0.0, 1.0, 0.0)

  *Unit vector in the positive y direction.*

- const GVector GVECTOR_K = GVector(0.0, 0.0, 1.0)

  *Unit vector in the positive z direction.*

- const GVector GVECTOR_0 = GVector(0.0, 0.0, 0.0)

  *Vector with lenght 0.*

- const GVector GVECTOR_FORWARD = GVector(0.0, 1.0, 0.0)

  *Vector pointing in the ”forward” direction.*

- const GVector GVECTOR_RIGHT = GVector(1.0, 0.0, 0.0)

  *Vector pointing in the ”up” direction.*

- const GVector GVECTOR_UP = GVector(0.0, 0.0, 1.0)

  *Vector pointing in the ”right” direction.*

## Typedefs

- typedef float GCoordinate

  *Represents a coordinate.*

## Functions

- bool IsApproxZero (float a)

  *Test if a float is equal to zero with a tolerance of +-0.0001.*

- bool IsApproxZero (double a)

  *Test if a double is equal to zero with a tolerance of +-0.0000001.*

## 5.2.1 Detailed Description

Classes for performing 3D math calculations.

These classes should be helpful for most basic 3D math calculations, from finding dot and cross products to line-triangle intersections.

## 5.2.2 Typedef Documentation

### 5.2.2.1 typedef float GAUGE3D::GCoordinate

Represents a coordinate.

This type should be used in all 3D math functions and any functions that manipulate positions in three dinemtional space.

## 5.2.3 Variable Documentation

### 5.2.3.1 const GVector GAUGE3D::GVECTOR_FORWARD = GVector(0.0, 1.0, 0.0)

Vector pointing in the ”forward” direction.

In GAUGE, the positive Y axis points forward.

#### 5.2.3.2 const GVector GAUGE3D::GVECTOR_RIGHT = GVector(1.0, 0.0, 0.0)

Vector pointing in the ”up” direction.

In GAUGE, the positive X axis points right.

#### 5.2.3.3 const GVector GAUGE3D::GVECTOR_UP = GVector(0.0, 0.0, 1.0)

Vector pointing in the ”right” direction.

In GAUGE, the positive Z axis points up.

## 5.3 OS Abstraction Layer

Platform independant access to operating system functions.

### Compounds

- class GAUGE3D::GFileSystem
- class GAUGE3D::GLibrary
- class GAUGE3D::GMessageBox
- class GAUGE3D::GThread
- class GAUGE3D::GMutex
- class GAUGE3D::GTimer

### Typedefs

- typedef GSmartPointer< GLibrary > pGLibrary

  *Smart pointer to a GLibrary.*

- typedef GSmartPointer< GMessageBox > pGMessageBox

  *Smart pointer to a GMessageBox.*

- typedef GSmartPointer< GThread > pGThread

  *Smart pointer to a GThread.*

- typedef GSmartPointer< GMutex > pGMutex

  *Smart pointer to a GMutex.*

### 5.3.1 Detailed Description

Platform independant access to operating system functions.

This is a pretty complete abstraction layer for all the back-end functions of your average operating system.

## 5.4 File Manipulation

Classes for manipulating files and data streams abstractly.

### Compounds

- class GAUGE3D::GConfigFile
- class GAUGE3D::GDirectory
- class GAUGE3D::GDiskDirectory
- class GAUGE3D::GDiskFile
- class GAUGE3D::GFile
- class GAUGE3D::GIStream
- class GAUGE3D::GOStream
- class GAUGE3D::GStringFile
- class GAUGE3D::GVirtualDirectory

### Typedefs

- typedef GSmartPointer<GDirectory> pGDirectory

  *Smart pointer to a GDirectory.*

- typedef GSmartPointer<GDiskDirectory> pGDiskDirectory

  *Smart pointer to a GDiskDirectory.*

- typedef GSmartPointer<GDiskFile> pGDiskFile

  *Smart pointer to a GDiskFile.*

- typedef GSmartPointer<GFile> pGFile

  *Smart pointer to a GFile.*

- typedef GSmartPointer<GIStream> pGIStream

  *Smart pointer to a GIStream.*

- typedef GSmartPointer<GOStream> pGOStream

  *Smart pointer to a GOStream.*

- typedef GSmartPointer<GStringFile> pGStringFile

  *Smart pointer to a GStringFile.*

- typedef GSmartPointer<GVirtualDirectory> pGVirtualDirectory

  *Smart pointer to a GVirtualDirectory.*

## Variables

- GAUGE3D_API pGDirectory gWorkingDir

  *The current working directory.*

- GAUGE3D_API pGDirectory gConfigDir

  *The directory in which config files should be placed.*

- GAUGE3D_API pGDirectory gResourceDir

  *The directory which contains program resources.*

### 5.4.1 Detailed Description

Classes for manipulating files and data streams abstractly.

These classes provide completely abstract file and data stream functionality. You can use these classes to do anything from simple disk access to building a virtual filesystem out of data from multiple compressed archives.

### 5.4.2 Variable Documentation

#### 5.4.2.1 GAUGE3D_API pGDirectory GAUGE3D::gWorkingDir

The current working directory.

This is the directory from which the program was run.

#### 5.4.2.2 GAUGE3D_API pGDirectory GAUGE3D::gConfigDir

The directory in which config files should be placed.

All config files for your program should be placed here.

#### 5.4.2.3 GAUGE3D_API pGDirectory GAUGE3D::gResourceDir

The directory which contains program resources.

The data files for your program reside in this directory.

## 5.5 Graphics Description

Classes for describing 3D graphics.

### Compounds

- class GAUGE3D::GBillboardSprite
- class GAUGE3D::GImage
- class GAUGE3D::GModel
- class GAUGE3D::GSphereModel
- class GAUGE3D::GSurface

### Typedefs

- typedef GSmartPointer<GBillboardSprite> pGBillboardSprite

  *Smart pointer to a GBillboardSprite.*

- typedef GSmartPointer<GImage> pGImage

  *Smart pointer to a GImage.*

- typedef GSmartPointer<GModel> pGModel

  *Smart pointer to a GModel.*

- typedef GSmartPointer<GSphereModel> pGSphereModel

  *Smart pointer to a GSphereModel.*

- typedef GSmartPointer<GSuperModel> pGSuperModel

  *Smart pointer to a GSuperModel.*

- typedef GSmartPointer<GSurface> pGSurface

  *Smart pointer to a GSurface.*

### 5.5.1 Detailed Description

Classes for describing 3D graphics.

This module includes classes for describing images, surfaces, 3D models, and 3D worlds. These classes don't actually render anything. They only describe what needs to be rendered. Use the renderer module to render these.

## 5.6 Renderer

Renders 3D scenes.

### Compounds

- class GAUGE3D::GCamera
- class GAUGE3D::GDisplay
- class GAUGE3D::GLight
- class GAUGE3D::GRenderer
- class GAUGE3D::GScene
- class GAUGE3D::GSprite
- class GAUGE3D::GTexture

### Typedefs

- typedef GSmartPointer< GCamera > pGCamera

  *Smart pointer to a GCamera.*

- typedef GSmartPointer< GDisplay > pGDisplay

  *Smart pointer to a GDisplay.*

- typedef GSmartPointer< GLight > pGLight

  *Smart pointer to a GLight.*

- typedef GSmartPointer< GRenderer > pGRenderer

  *Smart pointer to a GRenderer.*

- typedef GSmartPointer< GScene > pGScene

  *Smart pointer to a GScene.*

- typedef GSmartPointer< GSprite > pGSprite

  *Smart pointer to a GSprite.*

- typedef GSmartPointer< GTexture > pGTexture

  *Smart pointer to a GTexture.*

### 5.6.1 Detailed Description

Renders 3D scenes.

This module can render scenes constructed from classes in the graphics description module. GRenderer is a plugin class, and represents a piece of graphics hardware. All the other classes in this module are served from a GRenderer or from an object served from a GRenderer.

## 5.7 Plugin Loader

Loads executable add-ons to handle various functionality or file types. Sort of like a CORBA ORB, except optimized for gaming.

### Compounds

- class GAUGE3D::GFileLoader
- class GAUGE3D::GObjectLoader
- class GAUGE3D::GPlugin

### Defines

- #define GAUGE3D_EXPORT_PLUGIN(CLASSNAME)

  *Export a GPlugin derivative from the library.*

### Typedefs

- typedef GSmartPointer<GFileLoader> pGFileLoader

  *Smart pointer to a GFileLoader.*

- typedef GSmartPointer<GObjectLoader> pGObjectLoader

  *Smart pointer to a GObjectLoader.*

- typedef GSmartPointer<GPlugin> pGPlugin

  *Smart pointer to a GPlugin.*

### Variables

- GAUGE3D_API pGObjectLoader gMainObjectLoader

  *The main GAUGE object loader.*

## 5.7.1 Detailed Description

Loads executable add-ons to handle various functionality or file types. Sort of like a CORBA ORB, except optimized for gaming.

”Plugins” are executable add-ons that increase GAUGE’s functionality. They normally exist on the disk in the form of dynamic libraries (.dll or .so files, depending on your operating system). A plugin may perform some function suck as enabling the use of a particular rendering API, but more often a plugin acts as a loader for a particular type of file. For example, a plugin might load JPEG images, or BSP maps. The core GAUGE library does not know the difference between these formats. It just uses a standard interface to communicate with the plugin and get the info it needs to use the data in the file.

## 5.7.2 Define Documentation

### 5.7.2.1 #define GAUGE3D_EXPORT_PLUGIN(CLASSNAME)

**Initializer:**

```
\
CLASSNAME __gauge3d_plugin;                                   \
                                                              \
 \
extern "C"                                                    \
{                                                             \
void* __gauge3d_plugin_create() {return &__gauge3d_plugin;}   \
}
```

Export a GPlugin derivative from the library.

**Parameters:**
  ***CLASSNAME*** A class derived from GPlugin to export.

This macro should be used once in every plugin. It exports the given class so that GObjectLoader can load that class and access the rest of the plugin through it.

## 5.7.3 Variable Documentation

#### 5.7.3.1 GAUGE3D_API pGObjectLoader GAUGE3D::gMainObjectLoader

The main GAUGE object loader.

In almost all cases, you'll want to use this object loader rather than create your own. Plugins that need to load other plugins will use this object loader, and they will fail if this object loader does not have any plugins in it.

## 5.8 Input

Classes for sending and receiving input messages.

### Compounds

- class GAUGE3D::GInputMap
- class GAUGE3D::GMessage
- class GAUGE3D::GQuitMessage
- class GAUGE3D::GInputMessage
- class GAUGE3D::GKeyboardMessage
- class GAUGE3D::GMouseButtonMessage
- class GAUGE3D::GMouseWheelMessage
- class GAUGE3D::GMouseMotionMessage
- class GAUGE3D::GMessageQueue

### Typedefs

- typedef GSmartPointer<GInputMap> pGInputMap

  *Smart pointer to a GInputMap.*

- typedef GSmartPointer<GMessage> pGMessage

  *Smart pointer to a GMessage.*

- typedef GSmartPointer<GQuitMessage> pGQuitMessage

  *Smart pointer to a GQuitMessage.*

- typedef GSmartPointer<GInputMessage> pGInputMessage

  *Smart pointer to a GInputMessage.*

- typedef GSmartPointer<GKeyboardMessage> pGKeyboardMessage

  *Smart pointer to a GKeyboardMessage.*

- typedef GSmartPointer<GMouseButtonMessage> pGMouseButtonMessage

  *Smart pointer to a GMouseButtonMessage.*

- typedef GSmartPointer<GMouseWheelMessage> pGMouseWheelMessage

  *Smart pointer to a GMouseWheelMessage.*

- typedef GSmartPointer< GMouseMotionMessage > pGMouseMotionMessage

  *Smart pointer to a GMouseMotionMessage.*

- typedef GSmartPointer< GMessageQueue > pGMessageQueue

  *Smart pointer to a GMessageQueue.*

## Variables

- GAUGE3D_API pGMessageQueue gMainMessageQueue

  *The main GAUGE message queue.*

### 5.8.1 Detailed Description

Classes for sending and receiving input messages.

The classes in this module can be used to collect, handle, manipulate, and dispatch input messages.

### 5.8.2 Variable Documentation

#### 5.8.2.1 GAUGE3D_API pGMessageQueue GAUGE3D::gMainMessageQueue

The main GAUGE message queue.

In almost all cases, you'll want to use this message queue rather than create your own. Plugins that supply input will enqueue messages to this queue.

## 5.9 Control

Classes which control the operation and syncronization of the engine.

### Compounds

- class GAUGE3D::GEngine
- class GAUGE3D::GScheduler

### Typedefs

- typedef GSmartPointer<GScheduler> pGScheduler

  *Smart pointer to a GScheduler.*

### Variables

- GAUGE3D_API pGScheduler gMiscScheduler

  *Scheduler for miscellaneous tasks.*

- GAUGE3D_API pGScheduler gInputScheduler

  *Scheduler for input and message handling tasks.*

- GAUGE3D_API pGScheduler gVideoScheduler

  *Scheduler for video and rendering tasks.*

### 5.9.1 Detailed Description

Classes which control the operation and syncronization of the engine.

This includes GScheduler, which syncronizes the engine, and GEngine, which initializes and controls the engine.

### 5.9.2 Variable Documentation

#### 5.9.2.1 GAUGE3D_API pGScheduler GAUGE3D::gMiscScheduler

Scheduler for miscellaneous tasks.

GEngine::MainLoop() will call gMiscScheduler->DoFrame() every frame.

#### 5.9.2.2 GAUGE3D_API pGScheduler GAUGE3D::gInputScheduler

Scheduler for input and message handling tasks.

GEngine::MainLoop() will call gInputScheduler->DoFrame() every frame if the input subsystem has been activated.

#### 5.9.2.3 GAUGE3D_API pGScheduler GAUGE3D::gVideoScheduler

Scheduler for video and rendering tasks.

GEngine::MainLoop() will call gVideoScheduler->DoFrame() every frame if the video subsystem has been activated.

# Chapter 6

# gauge3d Class Documentation

## 6.1 GAUGE3D::GAngles Class Reference

Represents a set of Euler angles (pitch, yaw, and roll).

`#include <gauge3d/3dmath/angles.h>`

### Public Types

- typedef float **tAngle**

### Public Methods

- GAngles ()

  *Leaves pitch, yaw, and roll unititialized.*

- GAngles (tAngle pitch,tAngle yaw,tAngle roll)

  *Sets pitch, yaw, and roll manually.*

- GAngles (const GQuaternion &quat)

  *Converts a quaternion into Euler angles.*

- void Rotate (const GVector in[ ],GVector out[ ],int num)const

  *Rotate a set of vectors by the angles.*

- void MakeVectors (GVector ∗forward,GVector ∗up=NULL,GVector ∗right=NULL)const

  *Finds forward, up, and right vectors for the angles.*

- void SetDirection (const GVector &forward)

  *Sets the angles to point in the given direction.*

- void SetDirection (const GVector &forward,const GVector &up)

  *Sets the angles so that the given forward and up vectors are valid.*

- const GAngles& operator+= (const GAngles &other)

  *Component-wise addition (fast).*

- GAngles operator+ (const GAngles &other)

  *Component-wise addition (fast).*

- const GAngles& operator-= (const GAngles &other)

  *Component-wise subtraction (fast).*

- GAngles operator- (const GAngles &other)

  *Component-wise subtraction (fast).*

- GAngles operator ∗ (GCoordinate a)

  *Component-wise multiplication (fast).*

- const GAngles& operator ∗= (GCoordinate a)

  *Component-wise multiplication (fast).*

- GAngles operator ∗ (const GAngles &other)

  *Combines the angles ”correctly” (slow).*

- const GAngles& operator ∗= (const GAngles &other)

  *Combines the angles ”correctly” (slow).*

- bool operator== (const GAngles &other)const

  *Checks for equality with error tolerance.*

- operator GQuaternion ()const

  *Converts to a GQuaternion.*

## Public Attributes

### Angle Values

*All angles are in radians.*

- tAngle **mPitch**
- tAngle **mYaw**
- tAngle **mRoll**

## Static Public Methods

- tAngle Degrees2Radians (tAngle degrees)

  *Convert degrees to radians.*

- tAngle Radians2Degrees (tAngle radians)

  *Convert radians to degrees.*

## Related Functions

(Note that these are not member functions.)

- GAngles operator ∗ (GCoordinate coord,const GAngles &angles)

  *Component-wise subtraction (fast).*

### 6.1.1 Detailed Description

Represents a set of Euler angles (pitch, yaw, and roll).

**See also:**
GQuaternion

Euler angles are more intuitive than quaternions, especially in certain types of games (such as 3D shooters). However, Euler angles have many limitations, and are not very elegant mathematically. Fortunately, you can convert Euler angles into quaternions and vice versa.

A pitch, yaw, and roll of zero points in the direction of GVECTOR_FORWARD. Pitch rotates clockwise around the X axis, yaw rotates clockwise around the Z axis, and roll rotates clockwise around the Y axis. When rotating a vector, roll is applied first, then pitch, then yaw.

### 6.1.2 Member Function Documentation

#### 6.1.2.1 void GAUGE3D::GAngles::Rotate (const GVector *in*[ ], GVector *out*[ ], int *num*) const

Rotate a set of vectors by the angles.

**Parameters:**

***in*** An array of vectors to rotate.

***out*** Location to which to write the rotated vectors. May be the same as in.

***num*** Number of vectors to rotate.

**Remarks:**

Rotating many vectors at once is much faster than rotating vectors individually.

#### 6.1.2.2 void GAUGE3D::GAngles::MakeVectors (GVector ∗ *forward*, GVector ∗ *up* = NULL, GVector ∗ *right* = NULL) const

Finds forward, up, and right vectors for the angles.

**Parameters:**

***forward*** The location to write the forward vector, or NULL if it should not be calculated.

***up*** The location to write the up vector, or NULL if it should not be calculated.

***right*** The location to write the right vector, or NULL if it should not be calculated.

MakeVectors takes an angle and converts it to a vector pointing forward, a vector pointing right, and a vector pointing up, according to the direction the angles are pointing. In other words, it is similar to calling Rotate() with the input being the vectors GVECTOR_FORWARD, GVECTOR_UP, and GVECTOR_RIGHT.

#### 6.1.2.3 void GAUGE3D::GAngles::SetDirection (const GVector & *forward*)

Sets the angles to point in the given direction.

SetDirection takes a vector and sets the pitch and yaw such that that vector points forwards. Roll is set to zero.

#### 6.1.2.4 void GAUGE3D::GAngles::SetDirection (const GVector & *forward*, const GVector & *up*)

Sets the angles so that the given forward and up vectors are valid.

This version sets pitch, yaw, *and roll* so that both the given vectors are equal to what you'd get if you called MakeVectors.

#### 6.1.2.5 GAngles GAUGE3D::GAngles::operator ∗ (const GAngles & *other*) `[inline]`

Combines the angles "correctly" (slow).

Multiplication between GAngles objects is done by converting them to GQuaternion's, multiplying those, and then converting the result back to a GAngles. This results in a more correct combination than simply adding the components does, but it takes longer.

#### 6.1.2.6 const GAngles & GAUGE3D::GAngles::operator ∗= (const GAngles & *other*)

Combines the angles "correctly" (slow).

Multiplication between GAngles objects is done by converting them to GQuaternion's, multiplying those, and then converting the result back to a GAngles. This results in a more correct combination than simply adding the components does, but it takes longer.

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

- gauge3d/3dmath/angles.h

## 6.2 GAUGE3D::GModel::AnimState Class Reference

The animation state for a GModel.

`#include <gauge3d/graphics/model.h>`

Inherits GAUGE3D::GObject.

Inherited by GAUGE3D::GBillboardSprite::BillboardAnimState, GAUGE3D::GSphereModel::SphereModelAnimState, and GAUGE3D::GSuperModel::SuperAnimState.

### Public Methods

- virtual ∼**AnimState** ()

### Public Attributes

- Under Construction

  *This interface is not done yet.*

### 6.2.1 Detailed Description

The animation state for a GModel.

AnimState stores the current position of a model's animation. This allows several objects to use the same model but be in different positions (i.e. one player is walking while another is jumping). Each object would have a different AnimState, but the same GModel.

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

- gauge3d/graphics/model.h

## 6.3 GAUGE3D::GArray Class Template Reference

A garbage-collected array class.

`#include <gauge3d/array.h>`

### Public Methods

- GArray ()

  *Create a null array.*

- GArray (GSmartPointer< tType >array,int size)

  *Create an array from a pointer.*

- GArray (int size)

  *Allocate a new array with the given size.*

- tType& operator[ ] (int index)const

  *Random access to the array.*

- int Size ()const

  *Get the number of elements in the array.*

- GSmartPointer<tType> Data ()

  *Get a smart pointer to the array.*

- const GSmartPointer<tType> Data ()const

  *Get a smart pointer to the array.*

- operator GSmartPointer ()

  *Cast to smart pointer.*

- operator const GSmartPointer ()const

  *Cast to smart pointer.*

- GArray& operator= (const GArray &other)

  *Set the GArray to point to a different data set.*

- bool operator== (const GArray &other)const

*Check if two arrays point at the same data.*

- operator void ∗ ()const

  *Use to test if array is NULL.*

### 6.3.1 Detailed Description

**template<class tType> class GAUGE3D::GArray**

A garbage-collected array class.

GArray is basically the same as GSmartPointer except that it points to an array.

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

- gauge3d/common/array.h

## 6.4 GAUGE3D::GBillboardSprite::BillboardAnimState Class Reference

The animation state for a GBillboardSprite.

`#include <gauge3d/graphics/billboardsprite.h>`

Inherits GAUGE3D::GModel::AnimState.

### Public Methods

- virtual ∼**BillboardAnimState** ()
- Under Construction ()

  *This interface is not done yet.*

### 6.4.1 Detailed Description

The animation state for a GBillboardSprite.

At this time, this class does absolutely nothing, as billboard sprites are not animated.

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

- gauge3d/graphics/billboardsprite.h

## 6.5 GAUGE3D::GBillboardSprite Class Reference

A flat textured rectangle which always faces the camera.

`#include <gauge3d/graphics/billboardsprite.h>`

Inherits GAUGE3D::GModel.

### Public Types

- typedef GSmartPointer< BillboardAnimState > pBillboardAnimState

  *Smart pointer to a BillboardAnimState.*

### Public Methods

- GBillboardSprite (GCoordinate width,GCoordinate height)

  *Construct a billboard sprite of the given dimentions.*

- ∼GBillboardSprite ()

  *Destructor.*

- void SetDimentions (GCoordinate width,GCoordinate height)

  *Set the dimensions of the rectangle.*

- void GetDimentions (GCoordinate ∗width,GCoordinate ∗height)

  *Get the dimensions of the rectangle.*

- void TexCoords (GVector upperLeft,GVector lowerRight)

  *Manually set texture coordinates.*

- virtual pAnimState CreateAnimState ()

  *Generates an AnimState object associated with this model.*

- pRenderInfo CreateRenderInfo ()

  *Create a RenderInfo bound to this model.*

## Friends

- class **BillboardRenderInfo**

### 6.5.1 Detailed Description

A flat textured rectangle which always faces the camera.

### 6.5.2 Member Function Documentation

#### 6.5.2.1 void GAUGE3D::GBillboardSprite::TexCoords (GVector *upperLeft*, GVector *lowerRight*)

Manually set texture coordinates.

**Parameters:**

***upperLeft*** The texture coordinate corresponding to the upper left corner of the sprite.

***lowerRight*** The texture coordinate corresponding to the lower right corner of the sprite.

#### 6.5.2.2 pRenderInfo GAUGE3D::GBillboardSprite::CreateRenderInfo () `[virtual]`

Create a RenderInfo bound to this model.

**Remarks:**

> Obviously, this function uses dynamic memory and should not be called every frame. A renderer should call this once when the model is assigned to a sprite and then use the same RenderInfo object every time the sprite is drawn.

Reimplemented from GAUGE3D::GModel.

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

- gauge3d/graphics/billboardsprite.h

## 6.6 GAUGE3D::GCamera Class Reference

Represents the viewpoint into the 3D scene.

`#include <gauge3d/renderer/camera.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- virtual ~**GCamera** ()
- virtual void Position (const GVector &position)=0

  *Sets the position of the camera in the scene.*
- virtual void Direction (const GQuaternion &direction)=0

  *Sets the direction in which the camera is facing.*
- virtual void Zoom (GCoordinate zoom)=0

  *Sets the magnification of the camera.*

### 6.6.1 Detailed Description

Represents the viewpoint into the 3D scene.

### 6.6.2 Member Function Documentation

#### 6.6.2.1 void GAUGE3D::GCamera::Zoom (GCoordinate *zoom*) `[pure virtual]`

Sets the magnification of the camera.

Zoom! 1.0 is normal, higher is zoomed in. Setting the zoom below 1.0 is probably a bad idea, as this will increase the FOV beyond 90 degrees, which could cause distortion.

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

- gauge3d/renderer/camera.h

## 6.7 GAUGE3D::GConfigFile Class Reference

Access to a section-key-data config file.

`#include <gauge3d/files/configfile.h>`

### Static Public Methods

- pGVirtualDirectory Load (pGFile file)

  *Loads the file into a new virtual directory and returns it.*

- bool Save (pGFile file,pGVirtualDirectory config)

  *Saves the configuration to the given file.*

### 6.7.1 Detailed Description

Access to a section-key-data config file.

GConfigFile is **not** derived from GFile. Heck, GConfigFile is not even an instantiable class (it is fully static). You give it a GFile to use. It treats that file as a config file in a similar format to that of win.ini on Windows systems, but with some enhancements. Each sections starts with a section title in square brackets, and ends with the same section title with a '/' infront of it, again in square brackets. Each line in the section has the format key=data. You look up settings by the key. Also note that sections can be embedded within other sections, and also that the file itself is a section, so if you don't need separate sections, you don't have to use them.

GConfigFile is actually a static class. You should not create an instance of it. Instead, use the functions within GConfigFile to load the configuration into a virtual directory. Then, you can navigate sections of the config file just as you would directories in a filesystem. Each key is a file, and the corresponding data can be retrieved from that file by casting the file to a string. (You can use normal file i/o methods to edit the configuration as well, but casting to a string will be much faster.)

### 6.7.2 Member Function Documentation

#### 6.7.2.1 pGVirtualDirectory GAUGE3D::GConfigFile::Load (pGFile *file*) `[static]`

Loads the file into a new virtual directory and returns it.

**Returns:**

A pointer to the new virtual directory, or NULL if the file could not be read.

#### 6.7.2.2 bool GAUGE3D::GConfigFile::Save (pGFile *file*, pGVirtualDirectory *config*) `[static]`

Saves the configuration to the given file.

**Returns:**

True if successful, or false if the file was not writable.

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

- gauge3d/files/configfile.h

## 6.8 GAUGE3D::GDirectory Class Reference

Abstract interface to a directory.

`#include <gauge3d/files/directory.h>`

Inherits GAUGE3D::GObject.

Inherited by GAUGE3D::GDiskDirectory, and GAUGE3D::GVirtualDirectory.

### Public Methods

- virtual ~**GDirectory** ()
- virtual GArray<GString> ListFiles ()=0

  *Get a list of all the files in the directory.*

- virtual GArray<GString> ListSubdirs ()=0

  *Get a list of all the subdirectories of the directory.*

- virtual pGFile OpenFile (GString name)=0

  *Get a pointer to a file in the directory.*

- virtual pGDirectory OpenSubdir (GString name)=0

  *Get a pointer to a sub directory in the directory.*

- virtual pGFile AddFile (GString name)=0

  *Create a new file in the directory.*

- virtual pGDirectory AddSubdir (GString name)=0

  *Create a new sub directory to the directory.*

- virtual void RemoveFile (GString name)=0

  *Remove a file from the dircetory.*

- virtual void RemoveDirectory (GString name)=0

  *Remove a sub directory from the directory (recursively if necessary).*

### 6.8.1 Detailed Description

Abstract interface to a directory.

This could represents a directory on the hard drive, but it could also represent anything from a compressed archive to an ftp site. Archive plugins should be derived from this.

### 6.8.2 Member Function Documentation

#### 6.8.2.1 pGFile GAUGE3D::GDirectory::OpenFile (GString *name*) `[pure virtual]`

Get a pointer to a file in the directory.

**Parameters:**

***name*** The name of the file to open.

**Remarks:**

Use this to open the contents of a directory. This way, if you have a pointer to the root directory of any filesystem, you can access everything in the system.

**Note:**

It is perfectly legal to say:

- myDirectory.OpenFile(”this/is/a/deeply/nested/file”);

Likewise with OpenSubdir(). You need not manually traverse the directory tree. Always use forward slashes (’/’) to separate directories. Using back slashes (’\’) will not work, even on Windows.

Reimplemented in GAUGE3D::GDiskDirectory, and GAUGE3D::GVirtualDirectory.

#### 6.8.2.2 pGDirectory GAUGE3D::GDirectory::OpenSubdir (GString *name*) `[pure virtual]`

Get a pointer to a sub directory in the directory.

**Parameters:**

***name*** The name of the sub directory to open.

**Remarks:**

Use this to open the contents of a directory. This way, if you have a pointer to the root directory of any filesystem, you can access everything in the system.

**Note:**

It is perfectly legal to say:

- myDirectory.OpenFile("this/is/a/deeply/nested/subdir");

Likewise with OpenSubdir(). You need not manually traverse the directory tree. Always use forward slashes ('/') to separate directories. Using back slashes ('\') will not work, even on Windows.

Reimplemented in GAUGE3D::GDiskDirectory, and GAUGE3D::GVirtualDirectory.

#### 6.8.2.3 pGFile GAUGE3D::GDirectory::AddFile (GString *name*) `[pure virtual]`

Create a new file in the directory.

**Parameters:**

***name*** The name of the file to create.

**Returns:**

A pointer to the new file, or NULL if the file already existed or could not be created.

Reimplemented in GAUGE3D::GDiskDirectory, and GAUGE3D::GVirtualDirectory.

#### 6.8.2.4 pGDirectory GAUGE3D::GDirectory::AddSubdir (GString *name*) `[pure virtual]`

Create a new sub directory to the directory.

**Parameters:**

***name*** The name of the sub directory to create.

**Returns:**

A pointer to the new directory, or NULL if the directory already existed or could not be created.

Reimplemented in GAUGE3D::GDiskDirectory, and GAUGE3D::GVirtualDirectory.

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

- gauge3d/files/directory.h

## 6.9 GAUGE3D::GDiskDirectory Class Reference

A directory on a disk.

`#include <gauge3d/files/diskdirectory.h>`

Inherits GAUGE3D::GDirectory.

### Public Methods

- GDiskDirectory ()

  *Create without opening a directory.*

- GDiskDirectory (GString dirname)

  *Create and open the given directory.*

- virtual ∼GDiskDirectory ()

  *Close the directory.*

- bool Open (GString dirname)

  *Opens a particular directory.*

- void Close ()

  *Closes the directory.*

- bool IsOpen ()

  *Returns true if the directory is open.*

- virtual GArray<GString> ListFiles ()

  *Get a list of all the files in the directory.*

- virtual GArray<GString> ListSubdirs ()

  *Get a list of all the subdirectories of the directory.*

- virtual pGFile OpenFile (GString name)

  *Get a pointer to a file in the directory.*

- virtual pGDirectory OpenSubdir (GString name)

  *Get a pointer to a sub directory in the directory.*

- virtual pGFile AddFile (GString name)
  *Create a new file in the directory.*
- virtual pGDirectory AddSubdir (GString name)
  *Create a new sub directory to the directory.*
- virtual void RemoveFile (GString name)
  *Remove a file from the dircetory.*
- virtual void RemoveDirectory (GString name)
  *Remove a sub directory from the directory (recursively if necessary).*

## 6.9.1 Detailed Description

A directory on a disk.

GDiskDirectory is an actual file on the hard drive or other disk.

## 6.9.2 Member Function Documentation

### 6.9.2.1 bool GAUGE3D::GDiskDirectory::Open (GString *dirname*)

Opens a particular directory.

**Parameters:**
***dirname*** The path and name of the directory on disk.

**Returns:**
True on success, false on error.

### 6.9.2.2 virtual pGFile GAUGE3D::GDiskDirectory::OpenFile (GString *name*) `[virtual]`

Get a pointer to a file in the directory.

**Parameters:**
***name*** The name of the file to open.

**Remarks:**

Use this to open the contents of a directory. This way, if you have a pointer to the root directory of any filesystem, you can access everything in the system.

**Note:**

It is perfectly legal to say:

- myDirectory.OpenFile(”this/is/a/deeply/nested/file”);

Likewise with OpenSubdir(). You need not manually traverse the directory tree. Always use forward slashes (’/’) to separate directories. Using back slashes (’\’) will not work, even on Windows.

Reimplemented from GAUGE3D::GDirectory.

#### 6.9.2.3 virtual pGDirectory GAUGE3D::GDiskDirectory::OpenSubdir (GString *name*) `[virtual]`

Get a pointer to a sub directory in the directory.

**Parameters:**

***name*** The name of the sub directory to open.

**Remarks:**

Use this to open the contents of a directory. This way, if you have a pointer to the root directory of any filesystem, you can access everything in the system.

**Note:**

It is perfectly legal to say:

- myDirectory.OpenFile(”this/is/a/deeply/nested/subdir”);

Likewise with OpenSubdir(). You need not manually traverse the directory tree. Always use forward slashes (’/’) to separate directories. Using back slashes (’\’) will not work, even on Windows.

Reimplemented from GAUGE3D::GDirectory.

#### 6.9.2.4 virtual pGFile GAUGE3D::GDiskDirectory::AddFile (GString *name*) `[virtual]`

Create a new file in the directory.

**Parameters:**

***name*** The name of the file to create.

**Returns:**

A pointer to the new file, or NULL if the file already existed or could not be created.

Reimplemented from GAUGE3D::GDirectory.

#### 6.9.2.5 virtual pGDirectory GAUGE3D::GDiskDirectory::AddSubdir (GString *name*) `[virtual]`

Create a new sub directory to the directory.

**Parameters:**

| | |
|---|---|
| ***name*** | The name of the sub directory to create. |

**Returns:**

A pointer to the new directory, or NULL if the directory already existed or could not be created.

Reimplemented from GAUGE3D::GDirectory.

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

- gauge3d/files/diskdirectory.h

## 6.10 GAUGE3D::GDiskFile Class Reference

Access to files on disk.

`#include <gauge3d/files/diskfile.h>`

Inherits GAUGE3D::GFile.

### Public Types

- enum tOpenFlags { NOCREATE = 0, CREATE = 1 << 8, NOREPLACE = 1 << 9, TRUNCATE = 1 << 10 }

  *Flags specified with Open().*

### Public Methods

- GDiskFile ()

  *Create without openning a file.*

- GDiskFile (GString filename,int flags)

  *Create and open a file.*

- virtual ~GDiskFile ()

  *Close the file.*

- bool Open (GString filename,int flags)

  *Create and open a file.*

- void Close ()

  *Close the file.*

- bool IsOpen ()

  *Returns true if the file is open.*

- virtual int Size ()

  *Gets the size of the file in bytes.*

- virtual bool Truncate ()

  *Truncates the file to zero length. Returns false if file was not writable.*

- virtual pGRawIStream GetRawInputStream ()

  *Get a GRawIStream for the file. May be null if the file is not readable.*

- virtual pGRawOStream GetRawOutputStream ()

  *Get a GRawOStream for the file. May be null if the file is not writeable.*

- virtual pFileMap Map (int offset,int size,int flags)

  *Maps a portion of the file to memory.*

- virtual GString MapToDisk ()

  *Copies the file to a temporary file on disk, if it is not already on the disk.*

- virtual GString Name ()

  *Get the name of the file.*

## 6.10.1 Detailed Description

Access to files on disk.

## 6.10.2 Member Enumeration Documentation

### 6.10.2.1 enum GAUGE3D::GDiskFile::tOpenFlags

Flags specified with Open().

**Enumeration values:**

***NOCREATE*** Fail if the file does not exist.

***CREATE*** Create the file if it does not exist.

***NOREPLACE*** Fail if the file already exists.

***TRUNCATE*** Truncate the file to zero length if it exists.

## 6.10.3 Constructor & Destructor Documentation

#### 6.10.3.1 GAUGE3D::GDiskFile::GDiskFile (GString *filename*, int *flags*)

Create and open a file.

**Parameters:**

***filename*** The path and name of the file to open.

***flags*** One or more of tOpenFlags, bitwise or'd.

### 6.10.4 Member Function Documentation

#### 6.10.4.1 bool GAUGE3D::GDiskFile::Open (GString *filename*, int *flags*)

Create and open a file.

**Parameters:**

***filename*** The path and name of the file to open.

***flags*** One or more of tOpenFlags, bitwise or'd.

**Returns:**

True on success, false on error.

#### 6.10.4.2 virtual pFileMap GAUGE3D::GDiskFile::Map (int *offset*, int *size*, int *flags*) `[virtual]`

Maps a portion of the file to memory.

**Parameters:**

***offset*** Where to start the map relative to the beginning of the file.

***size*** How many bytes to map to memory.

***flags*** One or more of tMapFlags, bitwise or'd.

**Returns:**

A pointer to a FileMap class describing the map, or NULL if the file could not be mapped.

Reimplemented from GAUGE3D::GFile.

#### 6.10.4.3 virtual GString GAUGE3D::GDiskFile::MapToDisk () `[virtual]`

Copies the file to a temporary file on disk, if it is not already on the disk.

**Returns:**

The path and name of the file on disk.

You should not write to the file on disk as other parts of the engine may be reading from it as well. The file on disk will be deleted when the GFile that created it is deleted (unless the file was on the disk to begin with).

**Remarks:**

Do not use this if you can avoid it. It is mainly meant for use with GLibrary, which must load shared libraries from disk due to the limitations of the library loading functions of every operating system I know of.

Reimplemented from GAUGE3D::GFile.

#### 6.10.4.4 virtual GString GAUGE3D::GDiskFile::Name () `[virtual]`

Get the name of the file.

**Note:**

This is not always the same as the name of the directory entry that points to the file, especially if the file is in a virtual filesystem.

Reimplemented from GAUGE3D::GFile.

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

- gauge3d/files/diskfile.h

## 6.11 GAUGE3D::GDisplay Class Reference

Represents a two dimentional surface to which scenes can be rendered.

`#include <gauge3d/renderer/camera.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- virtual ∼**GDisplay** ()
- virtual bool Open (int width,int height,int depth,bool fullScreen)=0

  *Resets the display resolution and color depth.*
- virtual void Camera (pGCamera camera)=0

  *Sets the viewpoint from which to render the display.*
- virtual void CaptureMouse (bool enable)=0

  *Set whether the display should capture mouse input.*

### 6.11.1 Detailed Description

Represents a two dimentional surface to which scenes can be rendered.

This is basically the frame buffer of the graphics card, though in some more interesting cases it might represent only a portion of the framebuffer (e.g. for a split-screen effect) or even a texture (render-to-texture).

### 6.11.2 Member Function Documentation

#### 6.11.2.1 bool GAUGE3D::GDisplay::Open (int *width*, int *height*, int *depth*, bool *fullScreen*) `[pure virtual]`

Resets the display resolution and color depth.

**Parameters:**

***width*** The width, in pixels, of the desired display.

***height*** The height, in pixels, of the desired display.

***depth*** The number of bits per pixel of the desired display.

***fullScreen*** True if the display should be fullscreen. False otherwise.

This function must be called once before any rendering can be done. If the display could not be openned, false is returned. Passing zero for colorDepth will cause the current display depth to be used.

**Note:**

This function will be removed before GAUGE v1.0. By that time, the user will be able to configure the display type themselves through the rendering plugin's configuration dialog.

#### 6.11.2.2 void GAUGE3D::GDisplay::Camera (pGCamera *camera*) `[pure virtual]`

Sets the viewpoint from which to render the display.

**Remarks:**

Obviously, the camera must have come from a GScene object which came from the same renderer as this GDisplay object. If you don't follow this rule, the renderer is free to crash your program. :)

#### 6.11.2.3 void GAUGE3D::GDisplay::CaptureMouse (bool *enable*) `[pure virtual]`

Set whether the display should capture mouse input.

This controls whether the mouse is treated as controling a pointer on the screen or if the mouse input is captured raw. When mouse capturing is enabled, mouse input messages will no longer contain absolute display coordinates, but the mouse can move infinately in any direction without hitting the side of the display or leaving the display window. When off, the mouse just controls a pointer that moves about the screen.

By default, the mouse is not captured.

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

- gauge3d/renderer/display.h

## 6.12 GAUGE3D::GEngine Class Reference

Used to initialize the engine.

`#include <gauge3d/engine.h>`

### Public Types

- enum tSubSystems { INPUT = 1 << 0, VIDEO = 1 << 1, FILESYSTEM = 1 << 2, OBJECT_LOADER = 1 << 3, ALL_SUBSYSTEMS = 0x7FFFFFFF }

  *Bits for subsystems.*

### Static Public Methods

- void CommandLine (int argc,char ∗argv[ ])

  *Call this when your program starts to tell GAUGE the command-line arguments used.*

- void CommandLine (char ∗args)

  *Call this when your program starts to tell GAUGE the command-line arguments used.*

- void CommandLine (int argc,GString argv[ ])

  *Call this when your program starts to tell GAUGE the command-line arguments used.*

- void CommandLine (GArray< GString >args)

  *Call this when your program starts to tell GAUGE the command-line arguments used.*

- GArray<GString> CommandLine ()

  *Get the command-line arguments.*

- void ProgName (GString name)
- GString ProgName ()

  *Get the program’s name.*

- void Init (int subsystems)
- int InitializedSubSystems ()

  *Get the initialized subsystems.*

- void ShutdownSubSystem (int subsystems)

  *Shutdown one or more subsystems.*

- void Shutdown ()

  *Shutdown the engine and all active subsystems.*

- void MainLoop ()

  *Execute main loop.*

- void Quit ()

  *Stop the main loop.*

## 6.12.1 Detailed Description

Used to initialize the engine.

All the functions in this class are static.

## 6.12.2 Member Enumeration Documentation

### 6.12.2.1 enum GAUGE3D::GEngine::tSubSystems

Bits for subsystems.

**Enumeration values:**

| | |
|---|---|
| ***INPUT*** | Input subsystem (message handling). |
| ***VIDEO*** | Video and 3D rendering subsystem. |
| ***FILESYSTEM*** | File I/O and data stream subsystem. |
| ***OBJECT_LOADER*** | Object/Plugin loader subsystem. |
| ***ALL_SUBSYSTEMS*** | All subsystems. |

## 6.12.3 Member Function Documentation

#### 6.12.3.1 void GAUGE3D::GEngine::ProgName (GString *name*) `[static]`

\breif Set the program's name.

**ameter name This should be a short name for the program consisting of only** lower-case letters, numerals, hyphens, and underscores. Depending on the OS, this name may be used for file and directory names. For example, on unix-like OS's, the user's config directory will be "∼/.progname", where "progname" is the name you specify.

#### 6.12.3.2 void GAUGE3D::GEngine::Init (int *subsystems*) `[static]`

\breif Initialize the engine and one or more sub-systems.

**Parameters:**

***subsystems*** A bitwise OR of one or more of the values of tSubSystems.

You may call Init() multiple times, or initialize everything at once.

#### 6.12.3.3 void GAUGE3D::GEngine::MainLoop () `[static]`

Execute main loop.

While running the main loop, GAUGE will repeatedly call the DoFrame() functions of each subsystem's scheduler. See GScheduler for more.

#### 6.12.3.4 void GAUGE3D::GEngine::Quit () `[static]`

Stop the main loop.

MainLoop() will return on the frame after Quit() is called.

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

- gauge3d/engine.h

## 6.13 GAUGE3D::GException Class Reference

Thrown on fatal errors.

`#include <gauge3d/error.h>`

### Public Methods

- GException (GString cause,GString file,int line)
- virtual ~**GException** ()
- GString Cause ()const

  *A human-readable string describing the cause of the error.*

- GString File ()const

  *The source file from which the exception was thrown.*

- int Line ()const

  *The line of the source file from which the exception was thrown.*

### 6.13.1 Detailed Description

Thrown on fatal errors.

**See also:**
GAUGE3D_FATAL_ERROR

An object of type GException may be thrown when a fatal (i.e. unrecoverable) error occurs. The main function of the program should catch this and output an error.

### 6.13.2 Constructor & Destructor Documentation

#### 6.13.2.1 GAUGE3D::GException::GException (GString *cause*, GString *file*, int *line*)

Rather than call this, use the macro GAUGE3D_FATAL_ERROR to create and throw an exception.

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

- gauge3d/common/error.h

## 6.14 GAUGE3D::GFile Class Reference

Represents a block of data.

`#include <gauge3d/files/file.h>`

Inherits GAUGE3D::GObject.

Inherited by GAUGE3D::GDiskFile, and GAUGE3D::GStringFile.

### Public Types

- typedef GSmartPointer<FileMap> **pFileMap**
- enum tMapFlags { READ = 1 << 0, WRITE = 1 << 1 }

  *Flags for mapping a file to memory.*

### Public Methods

- virtual ~**GFile** ()
- virtual int Size ()=0

  *Gets the size of the file in bytes.*

- virtual bool Truncate ()=0

  *Truncates the file to zero length. Returns false if file was not writable.*

- pGIStream GetInputStream ()

  *Get a GIStream for the file. May be null if the file is not readable.*

- pGOStream GetOutputStream ()

  *Get a GOStream for the file. May be null if the file is not readable.*

- virtual pGRawIStream GetRawInputStream ()=0

  *Get a GRawIStream for the file. May be null if the file is not readable.*

- virtual pGRawOStream GetRawOutputStream ()=0

  *Get a GRawOStream for the file. May be null if the file is not writeable.*

- virtual pFileMap Map (int offset,int size,int flags)=0

  *Maps a portion of the file to memory.*

- pFileMap Map (int flags=READ)

  *Maps the entire file to memory.*

- virtual GString MapToDisk ()=0

  *Copies the file to a temporary file on disk, if it is not already on the disk.*

- virtual GString Name ()=0

  *Get the name of the file.*

- virtual operator GString ()

  *Create a string containing the complete file.*

### 6.14.1 Detailed Description

Represents a block of data.

This could be a file on the disk, a file in a compressed archive, a file on an ftp server, etc.

### 6.14.2 Member Enumeration Documentation

#### 6.14.2.1 enum GAUGE3D::GFile::tMapFlags

Flags for mapping a file to memory.

**Enumeration values:**

***READ*** Make the map readable.

***WRITE*** Make it writable.

### 6.14.3 Member Function Documentation

#### 6.14.3.1 pGIStream GAUGE3D::GFile::GetInputStream () `[inline]`

Get a GIStream for the file. May be null if the file is not readable.

**Remarks:**

This just calls GetRawInputStream() and wraps a GIStream around it.

#### 6.14.3.2 pGOStream GAUGE3D::GFile::GetOutputStream () `[inline]`

Get a GOStream for the file. May be null if the file is not readable.

**Remarks:**

This just calls GetRawOutputStream() and wraps a GOStream around it.

#### 6.14.3.3 pFileMap GAUGE3D::GFile::Map (int *offset*, int *size*, int *flags*) `[pure virtual]`

Maps a portion of the file to memory.

**Parameters:**

***offset*** Where to start the map relative to the beginning of the file.

***size*** How many bytes to map to memory.

***flags*** One or more of tMapFlags, bitwise or'd.

**Returns:**

A pointer to a FileMap class describing the map, or NULL if the file could not be mapped.

Reimplemented in GAUGE3D::GDiskFile, and GAUGE3D::GStringFile.

#### 6.14.3.4 pFileMap GAUGE3D::GFile::Map (int *flags* = READ) `[inline]`

Maps the entire file to memory.

**Parameters:**

***flags*** One or more of tMapFlags, bitwise or'd.

**Returns:**

A pointer to a FileMap class describing the map, or NULL if the file could not be mapped.

#### 6.14.3.5 GString GAUGE3D::GFile::MapToDisk () `[pure virtual]`

Copies the file to a temporary file on disk, if it is not already on the disk.

**Returns:**
The path and name of the file on disk.

You should not write to the file on disk as other parts of the engine may be reading from it as well. The file on disk will be deleted when the GFile that created it is deleted (unless the file was on the disk to begin with).

**Remarks:**
Do not use this if you can avoid it. It is mainly meant for use with GLibrary, which must load shared libraries from disk due to the limitations of the library loading functions of every operating system I know of.

Reimplemented in GAUGE3D::GDiskFile, and GAUGE3D::GStringFile.

#### 6.14.3.6 GString GAUGE3D::GFile::Name () `[pure virtual]`

Get the name of the file.

**Note:**
This is not always the same as the name of the directory entry that points to the file, especially if the file is in a virtual filesystem.

Reimplemented in GAUGE3D::GDiskFile, and GAUGE3D::GStringFile.

#### 6.14.3.7 GAUGE3D::GFile::operator GString () `[virtual]`

Create a string containing the complete file.

**Remarks:**
This function pretty much just calls Map and makes a string out of the results. Do not use this on large files. Note that derived classes may define their own implementation of this, but they are not required to.

Reimplemented in GAUGE3D::GStringFile.

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

- gauge3d/files/file.h

## 6.15 GAUGE3D::GFileLoader Class Reference

Base class for plugins that load and save particular file formats.

`#include <gauge3d/plugins/fileloader.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- virtual ∼**GFileLoader** ()
- virtual GString **FileType** ()=0
- virtual pGObject **Load** (pGFile file)=0
- virtual bool **Save** (pGFile file,pGObject object)=0
- virtual bool **CanSave** (pGObject object)=0
- virtual pGObject **Read** (pGIStream input)=0
- virtual int **Write** (pGOStream output,pGObject object)=0

### 6.15.1 Detailed Description

Base class for plugins that load and save particular file formats.

This class is used when a program needs better control over loading and saving particular formats. If you simply want to load a file, you probably don't need this class. Just use GObjectLoader instead.

This class will probably be removed in a future version of GAUGE. It will be replaced by a better system.

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

- gauge3d/plugins/fileloader.h

## 6.16 GAUGE3D::GFileSystem::FileMap Class Reference

Contains information related to a memory mapped file.

`#include <gauge3d/osabstraction/filesystem.h>`

### Public Attributes

- void∗ mMap

  *The location in memory where the file has been mapped.*

- int mOffset

  *The offset from the beginning of the file at which the map starts.*

- int mSize

  *The size of the map.*

- FileMapInternal∗ mInternal

  *A pointer to internal info needed by the filesystem.*

### 6.16.1 Detailed Description

Contains information related to a memory mapped file.

The values of mOffset and mSize will always be exactly what was requested when GFileSystem::Map() was called.

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

- gauge3d/osabstraction/filesystem.h

## 6.17 GAUGE3D::GFile::FileMap Class Reference

Represents a GFile mapped to memory.

`#include <gauge3d/files/file.h>`

### Public Methods

- FileMap (void ∗map)

  *Initialize the file map.*

- virtual ∼**FileMap** ()
- void∗ Map ()

  *Get the location of the map in merory.*

### 6.17.1 Detailed Description

Represents a GFile mapped to memory.

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

- gauge3d/files/file.h

## 6.18 GAUGE3D::GFileSystem Class Reference

Low-level filesystem access.

`#include <gauge3d/osabstraction/filesystem.h>`

### Public Types

- enum tOpenFlags { READ = 1 << 0, WRITE = 1 << 1, CREATE = 1 << 2, NOREPLACE = 1 << 3, TRUNCATE = 1 << 4, APPEND = 1 << 5 }

  *Flags used in the Open() function.*

- enum tSeekType { BEGINNING, CURRENT, END }

  *Flags for the Seek() function.*

### Static Public Methods

**Initialization**

- void Init (GString progName)

  *Initialize the filesystem.*

- void ShutDown ()

  *Shuts down and cleans up the filesystem.*

**Directory Structure**

- GString WorkingDir ()

  *Get the currend working directory.*

- GString ConfigDir ()

  *Get the directory where config files should be placed.*

- GString ResourceDir ()

  *Get the location of the program's resources.*

- GString PluginDir ()

  *Get the location of GAUGE plugins.*

### Stream-based File Access

- File∗ Open (GString filename,int flags)
  *Open a file.*
- int Read (File ∗file,void ∗target,int size)
  *Read from a file.*
- int Write (File ∗file,const void ∗source,int size)
  *Write to a file.*
- void Seek (File ∗file,int offset,tSeekType relativeTo)
  *Seek the position of the stream.*
- int TellPos (File ∗file)
  *Get the position of the stream.*
- void Close (File ∗file)
  *Close the file.*

### Memmory Mapped File Access

- FileMap∗ Map (GString filename,int offset,int size,int flags)
  *Map a file to memory.*
- void UnMap (FileMap ∗map)
  *Close a memory mapped file.*

### Other File Manipulation Functions

- int FileSize (GString filename)
  *Get the size of a file.*
- int FileSize (File ∗file)
  *Get the size of an already open file.*
- bool Exists (GString filename)
  *Check if a file or directory exists.*
- void DeleteFile (GString filename)
  *Delete a file.*

### Directory Manipulation Functions

- Directory∗ OpenDirectory (GString dirName)
  *Open a directory for listing.*
- bool ReadDirectory (GString ∗filename,Directory ∗directory)
  *Read the name of one object in the directory.*
- void RewindDirectory (Directory ∗directory)
  *Restart reading from the beginning of a directory.*
- void CloseDirectory (Directory ∗directory)
  *Close the directory.*
- void CreateDirectory (GString dirname)
  *Create a directory.*
- void DeleteDirectory (GString dirname)
  *Remove an empty directory.*
- void DeleteTree (GString dirname)
  *Recursively delete a directory and all of its contents.*
- bool IsDirectory (GString dirname)
  *Check if an object is a file or a directory.*

## 6.18.1 Detailed Description

Low-level filesystem access.

Do not use this class directly. Use the classes in the File Manipulation module instead.

This is a static class. All members are static. You do not need to instantiate it. So, if you want to open a file, you would say something like GFileSystem::Open(”myfile.txt”, GFileSystem::READ).

## 6.18.2 Member Enumeration Documentation

### 6.18.2.1 enum GAUGE3D::GFileSystem::tOpenFlags

Flags used in the Open() function.

**Enumeration values:**

***READ*** Open for reading.

***WRITE*** Open for writing.

***CREATE*** Create the file if it doesn't exist.

***NOREPLACE*** Don't open if the file already exists.

***TRUNCATE*** Truncate the file to zero length if it exists.

***APPEND*** Place the stream at the end of the file.

#### 6.18.2.2 enum GAUGE3D::GFileSystem::tSeekType

Flags for the Seek() function.

**Enumeration values:**

***BEGINNING*** Seek relative to the beginning of the stream.

***CURRENT*** Seek relative to the current stream position.

***END*** Seek relative to the end of the stream.

### 6.18.3 Member Function Documentation

#### 6.18.3.1 void GAUGE3D::GFileSystem::Init (GString *progName*) `[static]`

Initialize the filesystem.

**Parameters:**

***progName*** A short name for the program, containing only numbers and lowercase letters with no spaces. For example, "gauge3d". This name will be used for directories and such that belong to this program. (i.e. on Unix, the config files are stored in "~/." + progName.)

Specifically, this function does the following things:

- Creates the config directory if it isn't there yet.
- Deletes the temporary directory if it is there, and creates a new one.
- Sticks a text file in the temporary directory telling the user that they can delete it if it is there.

#### 6.18.3.2 void GAUGE3D::GFileSystem::ShutDown () `[static]`

Shuts down and cleans up the filesystem.

Specifically, this function deletes the temporary directory and all of its contents.

#### 6.18.3.3 File ∗ GAUGE3D::GFileSystem::Open (GString *filename*, int *flags*) `[static]`

Open a file.

**Parameters:**

***filename*** The path and name of the file.

***flags*** One or more of the flags from tOpenFlags, bitwise or’d.

**Returns:**

A pointer to the open file, or NULL if an error occured.

#### 6.18.3.4 int GAUGE3D::GFileSystem::Read (File ∗ *file*, void ∗ *target*, int *size*) `[static]`

Read from a file.

**Parameters:**

***file*** A pointer to a file, obtained by calling Open().

***target*** The location to fill with the data read from the file.

***size*** The number of bytes to read.

**Returns:**

The number of bytes read.

#### 6.18.3.5 int GAUGE3D::GFileSystem::Write (File ∗ *file*, const void ∗ *source*, int *size*) `[static]`

Write to a file.

**Parameters:**

***file*** A pointer to a file, obtained by calling Open().

***source*** The location of the data to write to the file.

***size*** The number of bytes to write.

**Returns:**

The number of bytes written.

#### 6.18.3.6 void GAUGE3D::GFileSystem::Seek (File ∗ *file*, int *offset*, tSeekType *relativeTo*) `[static]`

Seek the position of the stream.

**Parameters:**

***file*** A pointer to a file, obtained by calling Open().

***offset*** The position to seek to.

***relativeTo*** The position that offset is relative to. One of the values of tSeekType.

#### 6.18.3.7 int GAUGE3D::GFileSystem::TellPos (File ∗ *file*) `[static]`

Get the position of the stream.

**Parameters:**

***file*** A pointer to a file, obtained by calling Open().

**Returns:**

The position of the stream relative to the beginning of the file.

#### 6.18.3.8 void GAUGE3D::GFileSystem::Close (File ∗ *file*) `[static]`

Close the file.

**Parameters:**

***file*** A pointer to a file, obtained by calling Open().

Call this when you are done with the file.

#### 6.18.3.9 FileMap ∗ GAUGE3D::GFileSystem::Map (GString *filename*, int *offset*, int *size*, int *flags*) `[static]`

Map a file to memory.

**Parameters:**

- ***filename*** The path and name of the file to map.
- ***offset*** The offset in bytes from the beginning of the file at which the map should start.
- ***size*** The number of bytes of the file to map.
- ***flags*** One or more of the flags READ, WRITE, CREATE, and NOREPLACE, from tOpenFlags, bitwise or'd. These have the same effect here as they do for Open().

**Returns:**

A FileMap structure describing the map in memory, or NULL if an error occured.

This function allows you to access a file as if the entire file (or a segment of it) were in RAM. Where available, features of the operating system are used to accellerate this functionality.

#### 6.18.3.10 void GAUGE3D::GFileSystem::UnMap (FileMap ∗ *map*) `[static]`

Close a memory mapped file.

**Parameters:**

- ***map*** A pointer to a FileMap, obtained by calling Map()

Call this when you are done with the file map.

#### 6.18.3.11 int GAUGE3D::GFileSystem::FileSize (GString *filename*) `[static]`

Get the size of a file.

**Parameters:**

- ***filename*** The path and name of the file to get the size of.

**Returns:**

The size of the file in bytes.

#### 6.18.3.12 int GAUGE3D::GFileSystem::FileSize (File ∗ *file*) `[static]`

Get the size of an already open file.

**Parameters:**

***file*** A pointer to a file, obtained by calling Open().

**Returns:**

The size of the file in bytes.

#### 6.18.3.13 bool GAUGE3D::GFileSystem::Exists (GString *filename*) `[static]`

Check if a file or directory exists.

**Parameters:**

***filename*** The path and name of the file to check.

**Returns:**

True if the file exists, or false if it does not.

**Remarks:**

This function will also return true if the file is actually a directory. Use IsDirectory() to find out if it is a file or a directory.

#### 6.18.3.14 void GAUGE3D::GFileSystem::DeleteFile (GString *filename*) `[static]`

Delete a file.

**Parameters:**

***filename*** The path and name of the file to delete.

#### 6.18.3.15 Directory ∗ GAUGE3D::GFileSystem::OpenDirectory (GString *dirName*) `[static]`

Open a directory for listing.

**Parameters:**

***dirName*** The path and name of the directory.

**Returns:**

A pointer to a directory, or NULL if an error occured.

### 6.18.3.16 bool GAUGE3D::GFileSystem::ReadDirectory (GString ∗ *filename*, Directory ∗ *directory*) `[static]`

Read the name of one object in the directory.

**Parameters:**

***filename*** A pointer to a string which should be replaced with the name of the file or sub directory.

***directory*** A pointer to a directory, obtained by calling OpenDirectory().

**Returns:**

True if a name was read, or false if there were no more objects to read

### 6.18.3.17 void GAUGE3D::GFileSystem::RewindDirectory (Directory ∗ *directory*) `[static]`

Restart reading from the beginning of a directory.

**Parameters:**

***directory*** A pointer to a directory, obtained by calling OpenDirectory().

### 6.18.3.18 void GAUGE3D::GFileSystem::CloseDirectory (Directory ∗ *directory*) `[static]`

Close the directory.

**Parameters:**

***directory*** A pointer to a directory, obtained by calling OpenDirectory().

Call this when you are done reading the directory.

#### 6.18.3.19 void GAUGE3D::GFileSystem::CreateDirectory (GString *dirname*) `[static]`

Create a directory.

**Parameters:**

***dirname*** The path and name of the directory to create.

#### 6.18.3.20 void GAUGE3D::GFileSystem::DeleteDirectory (GString *dirname*) `[static]`

Remove an empty directory.

**Parameters:**

***dirname*** The path and name of the directory to remove. This directory must be empty.

#### 6.18.3.21 void GAUGE3D::GFileSystem::DeleteTree (GString *dirname*) `[static]`

Recursively delete a directory and all of its contents.

**Parameters:**

***dirname*** The path and name of the directory to remove. This directory does *not* have to be empty.

#### 6.18.3.22 bool GAUGE3D::GFileSystem::IsDirectory (GString *dirname*) `[static]`

Check if an object is a file or a directory.

**Parameters:**

***dirname*** The path and name of the filesystem object to check.

**Returns:**

True if the object was a directory, or false if it was not.

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

- gauge3d/osabstraction/filesystem.h

## 6.19 GAUGE3D::GImage::FormatDesc Struct Reference

GImage::FormatDesc describes the format of raw image data obtained from a GImage.

`#include <gauge3d/graphics/image.h>`

### Public Attributes

- int mWidth

  *Width of the image, in pixels.*

- int mHeight

  *Height of the image, in pixels.*

- int mNumChannels

  *Number of color channels in the image.*

- int mBitsPerChannel

  *Number of bits per element per channel.*

- bool mHasMipmaps

  *True if the image has built-in mipmaps.*

- GString mEncoding

  *The encoding/compression used to store the image. (JPEG, FXT1, etc.).*

### 6.19.1 Detailed Description

GImage::FormatDesc describes the format of raw image data obtained from a GImage.

mNumChannels stores the number of channels in the image. This should be a number between 1 and 4, inclusive. A typical RGB image has three channels (red, green, and blue) while RGBA has four and grayscale has one.

mBitsPerChannel can 4 (for 16-bit images), 8 (for 32-bit images), or 5. In the case of 5, the image is 16-bits-per-pixel, and must contain three or four channels. If it contains three channels, the last bit of each 16 is ignored. If it contains four channels, the last bit of each 16 is interpreted as the fourth channel (a one-bit channel).

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

- gauge3d/graphics/image.h

## 6.20 GAUGE3D::GMessageQueue::Handler Class Reference

Base class for message handlers.

`#include <gauge3d/input/messagequeue.h>`

Inherits GAUGE3D::GObject.

Inherited by GAUGE3D::GInputMap.

### Public Methods

- virtual bool HandleMessage (pGMessage message)=0

  *Attempts to handle the message, and returns true if the message was dealt with.*

### 6.20.1 Detailed Description

Base class for message handlers.

### 6.20.2 Member Function Documentation

#### 6.20.2.1 bool GAUGE3D::GMessageQueue::Handler::HandleMessage (pGMessage *message*) `[pure virtual]`

Attempts to handle the message, and returns true if the message was dealt with.

**Returns:**

True if the handler did something with the message, and thus the message has served its purpose. False if the message was not dealt with and should be sent to other handlers.

Reimplemented in GAUGE3D::GInputMap.

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

- gauge3d/input/messagequeue.h

## 6.21 GAUGE3D::GImage Class Reference

Base class for image plugins.

`#include <gauge3d/graphics/image.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- virtual ∼**GImage** ()
- virtual FormatDesc Format ()=0

  *Get the image format.*

- virtual GArray<uint8> Data (int mipmapLevel=0)=0

  *Gets the raw image data in the format given by GetFormat().*

- virtual GArray<uint8> EncodedData (int mipmapLevel=0)=0

  *Gets the encoded image data.*

### 6.21.1 Detailed Description

Base class for image plugins.

GImage describes simple images in an abstract way. Plugins that support specific image formats (like GIF or JPEG) are derived from this.

### 6.21.2 Member Function Documentation

#### 6.21.2.1 GArray< uint8 > GAUGE3D::GImage::Data (int *mipmapLevel* = 0) `[pure virtual]`

Gets the raw image data in the format given by GetFormat().

**Parameters:**

***mipmapLevel*** The mip map to return. For example, for a 64x64 image, level 0 is 64x64, 1 is 32x32, 2 is 16x16, etc. Images that don't provide mip maps should ignore this.

**Returns:**

The raw image data in the given format.

#### 6.21.2.2 GArray< uint8 > GAUGE3D::GImage::EncodedData (int *mipmapLevel* = 0) `[pure virtual]`

Gets the encoded image data.

**Parameters:**

***mipmapLevel*** The mip map to return. For example, for a 64x64 image, level 0 is 64x64, 1 is 32x32, 2 is 16x16, etc. Images that don't provide mip maps and images encodings whing encode multiple mipmap levels together should ignore this.

**Returns:**

The image, encoded in whatever format the image file was in. So, it may be jpeg compressed, or FXT1, or whatever.

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

- gauge3d/graphics/image.h

## 6.22 GAUGE3D::GInputMap Class Reference

Maps user-configured input to game-specific controls.

`#include <gauge3d/input/inputmap.h>`

Inherits GAUGE3D::GMessageQueue::Handler.

### Public Methods

- GInputMap (pGMessageQueue outputQueue)

  *Constructs an input map.*

- virtual ∼GInputMap ()

  *Destructor.*

- virtual bool HandleMessage (pGMessage message)

  *Translates and dispatches a message (called by the input message queue).*

- void NewControl (GString name,int type)

  *Add a new control to the list of game-specific controls.*

- void MapInput (GString input,GString control,float sensitivity=1.0)

  *Maps an input source to a control with a given sensitivity.*

- void ControlList (pGDirectory config)

  *Loads a list of controls from a config file.*

- void ControlMap (pGDirectory config)

  *Loads the input to control map from a config file.*

- void ClearControlList ()

  *Empties the list of controls.*

- void ClearControlMap ()

  *Erases all input-to-control mappings.*

- virtual void DoFrame (tTime time,tTime frameTime)

  *This will be called every frame by the scheduler.*

## 6.22.1 Detailed Description

Maps user-configured input to game-specific controls.

GInputMap makes it easy to allow users to configure their controls rather than use forced key bindings. It also translates the input type to the control type (analog vs. digital, absolute vs. relative) and applies sensitivity settings to the controls.

To make the input map translate messages, you must register it as a message handler in the input message queue. Also, make sure to call GInputMap::DoFrame() every frame.

## 6.22.2 Constructor & Destructor Documentation

### 6.22.2.1 GAUGE3D::GInputMap::GInputMap (pGMessageQueue *outputQueue*)

Constructs an input map.

**Parameters:**

***outputQueue*** A message queue to which the translated messages should be sent. This must not be the same as the input queue for obvious reasons.

## 6.22.3 Member Function Documentation

### 6.22.3.1 void GAUGE3D::GInputMap::NewControl (GString *name*, int *type*)

Add a new control to the list of game-specific controls.

**Parameters:**

***name*** The name that the translated messages will have.

***type*** The type of the control. This is a bitwise OR of either GInputMessage::DIGITAL or GInputMessage::ANALOG with either GInputMessage::ABSOLUTE or GInputMessage::RELATIVE.

This function adds a new game-specific control to the list of controls which the user can configure, but it does not actually bind an input source to it. Use MapInput to bind one or more input sources to the control.

When an input source is mapped to a control, the message is translated from the input's type to the control's type, so you don't have to worry about handling any type of input other than what you expect, yet the user is free to bind any sort of input to any control (almost).

Choosing the right type for the control is tricky, especially if it is an analog control. Here are what the four types represent:

- Absolute Digital: Any sort of simple button-pressing control, like fire, use, jump, etc. You will recieve messages when the user presses or releases the button.
- Relative Digital: Something that cycles in two directions by discreet amounts. For example, a control that cycles though your inventory would be relative digital. You will receive messages when the user cycles forward or back. Think mouse wheel.
- Absolute Analog: Something which has defined limits. The only really way I can describe the difference between this and relative analog is by explaining how the translation is done. When mapping any sort of absolute input to an absolute analog control, the output value is simply the same as the input (after sensitivity is applied). When mapping a relative input to an absolute analog output, a running total of the input is kept and clamped to the range [-1.0, 1.0], and that total is output. An example of an absolute analog control would be a glance left/right function, where you can only turn your head so far to the left or right and the view should snap back to center afterwards.
- Relative Analog: A control with no limits. Again, I can't think of a good way to explain this without explaining exactly how the mapping is done. When mapping any sort of relative input to a relative analog control, the output value is simply the same as the input (after sensitivity is applied). When mapping absolute input, messages are sent every frame as long as the input value is not zero. So, the input becomes sort of a rate of change, as if the input were a mouse and were moving at exactly that speed. Unfortunately, the explanation I just gave you sucked, so you will have to experiment a little or I will have to get a better documentation writer. Examples of relative analog controls include movement, turning, looking, etc.

#### 6.22.3.2 void GAUGE3D::GInputMap::MapInput (GString *input*, GString *control*, float *sensitivity* = 1.0)

Maps an input source to a control with a given sensitivity.

**Parameters:**

***input*** The name of the input source to map.

***control*** The name of the control to which the input should be mapped.

***sensitivity*** The sensitivity of the input. How exactly this is used depends on the types of both the input and the control. Usually, the input value is multiplied by the sensitivity, but in some cases (usually when converting from analog input to a digital control) the sensitivity is actually a threshold value.

**Note:**

You can not map an input to a control before you create the control by calling NewControl (or ControlList()).

#### 6.22.3.3 void GAUGE3D::GInputMap::ControlList (pGDirectory *config*)

Loads a list of controls from a config file.

**Parameters:**

***config*** The directory representing the config file. See GConfigFile for more info on how this works.

The format of the config file is as follows:

*config* is one section of the config file. Each line in that section should have the form ”control name = [absolute|relative] [digital|analog]”. Here’s an example:

[controls]

cycle weapon = relative digital

fire = absolute digital

look Y = absolute analog

move X = analog relative

move Y = analog relative

[/controls]

#### 6.22.3.4 void GAUGE3D::GInputMap::ControlMap (pGDirectory *config*)

Loads the input to control map from a config file.

**Parameters:**

***config*** The directory representing the config file. See GConfigFile for more info on how this works.

**Note:**

You must load the control list before loading the input map.

The format of the config file is as follows:

*config* is one section of the config file. Each line in that section should have the form:

’control name = ”input name”[∗sensitivity] [”input2 name”[∗sensitivity] [...]]

Here’s an example:

[control map]

cycle weapon = ”mouse wheel”

fire = ”space” ”mouse 1”

look Y = ”mouse y”∗2.0

move X = ”left”∗-1.0 ”s”∗-1.0 ”right”∗1.0 ”d”∗1.0 ”joystick x”∗0.83

move Y = ”up”∗1.0 ”w”∗1.0 ”down”∗-1.0 ”a”∗-1.0 ”joystick y”∗0.83

[/control map]

#### 6.22.3.5 void GAUGE3D::GInputMap::DoFrame (tTime *time*, tTime *frameTime*) `[virtual]`

This will be called every frame by the scheduler.

**Remarks:**
> You can safely pretend this function does not exist. Everything will be handled for you. :)

Reimplemented from GAUGE3D::GObject.

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

- gauge3d/input/inputmap.h

## 6.23 GAUGE3D::GInputMessage Class Reference

Represents an input message.

`#include <gauge3d/input/message.h>`

Inherits GAUGE3D::GMessage.

Inherited by GAUGE3D::GKeyboardMessage, GAUGE3D::GMouseButtonMessage, GAUGE3D::GMouseMotionMessage, and GAUGE3D::GMouseWheelMessage.

### Public Types

- enum tFlags { **DIGITAL** = 0, **ANALOG** = 1, **ABSOLUTE** = 0, **RELATIVE** = 2 }

  *Indicates the type of input.*

### Public Methods

- GInputMessage (pGObject origin,GString name,float value,int flags)

  *Create a GInputMessage with the given info.*

- virtual ∼**GInputMessage** ()
- bool IsAnalog ()

  *Returns true if the input is analog, false if digital.*

- bool IsRelative ()

  *Returns true if the input is relate, false if absolute.*

- int Type ()

  *Returns the raw bitpattern type of the message.*

- float Value ()

  *Returns the value of the input.*

- GString Name ()

  *Returns an identifier for the input.*

### 6.23.1 Detailed Description

Represents an input message.

Input messages can be analog or digital, and they can be relative or absolute. Here's what each combination means:

- **Absolute Analog**: The value of the input is always between -1.0 and 1.0, and is measured relative to some central point. For example, joystick positions are normally measured this way. 0.0 is the center of the joystick, and anything else is off to one side or the other.
- **Relative Analog**: The value of the input could be anything. It is measured relative to the last position of the input rather than relative to some reference position. Mice are typically relative analog devices. Each mouse movement message only reports the distance that the mouse moved since the last message.
- **Absolute Digital**: The value of the input is either 0.0 or 1.0. This is mostly used for buttons, including mouse buttons and keys on the keyboard. A value of 1.0 means the button was pressed, whereas 0.0 means it was released.
- **Relative Digital**: The value of the input is either -1.0 or 1.0. The main use for this is mouse wheels, since they normally are measured in some sort of clicking fashion. 1.0 means the wheel was clicked up one notch and -1.0 means it was clicked down one notch.

### 6.23.2 Constructor & Destructor Documentation

#### 6.23.2.1 GAUGE3D::GInputMessage::GInputMessage (pGObject *origin*, GString *name*, float *value*, int *flags*) `[inline]`

Create a GInputMessage with the given info.

**Parameters:**

***origin*** A pointer to the object which is the logical ”originator” of the message. For example, most messages will originate from a GDisplay.

***name*** See Name(), below.

***value*** See Value(), below.

***flags*** A bitwise combination of the enum values of tFlags. The input must be either ANALOG or DIGITAL, and either ABSOLUTE or RELATIVE.

### 6.23.3 Member Function Documentation

#### 6.23.3.1 int GAUGE3D::GInputMessage::Type () `[inline]`

Returns the raw bitpattern type of the message.

**Remarks:**

This does not give you any more info than IsAnalog and IsRelative do.

#### 6.23.3.2 GString GAUGE3D::GInputMessage::Name () `[inline]`

Returns an identifier for the input.

The name of the input source is a human-readable unique identifier for the source of the input. In the case of a key on the keyboard, the name might be ”a” or ”left control”. For a mouse button, it would be ”mouse1” or ”mouse3”. For mouse movement, ”mouse x” or ”mouse y”. Et cetera.

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

- gauge3d/input/message.h

## 6.24 GAUGE3D::GIStream Class Reference

Provides many functions for reading data from a stream.

`#include <gauge3d/files/istream.h>`

Inherits GAUGE3D::GObject.

### Public Types

- enum tSeekType { BEGINNING, CURRENT, END }

  *Position to seek relative to. Used by Seek().*

- enum tReadMode { BINARY, ASCII }

  *Read mode specifiers for ReadMode().*

- enum tEndianness { BIG, LITTLE }

  *Endianness specifiers for Endianness().*

- enum tNumberBase { OCT = 8, DEC = 10, HEX = 16 }

  *Number base specifiers for NumberBase().*

### Public Methods

- GIStream (pGRawIStream streamSource)

  *Create a buffered input stream.*

- GIStream (pGRawIStream streamSource,int bufferSize)

  *Create a possibly buffered input stream with a given buffer size.*

- ∼GIStream ()

  *Destructor.*

- void Seek (int pos,tSeekType seekType)

  *Seek the stream position.*

- int TellPos ()

  *Get the distance in bytes between the beginning of the stream and the current position.*

- int Read (uint64 ∗target,int num=1)

  *Read one or more uint64's into the target buffer. Returns the number successfully read.*

- int Read (uint32 ∗target,int num=1)

  *Read one or more uint32's into the target buffer. Returns the number successfully read.*

- int Read (uint16 ∗target,int num=1)

  *Read one or more uint16's into the target buffer. Returns the number successfully read.*

- int Read (int64 ∗target,int num=1)

  *Read one or more int64's into the target buffer. Returns the number successfully read.*

- int Read (int32 ∗target,int num=1)

  *Read one or more int32's into the target buffer. Returns the number successfully read.*

- int Read (int16 ∗target,int num=1)

  *Read one or more int16's into the target buffer. Returns the number successfully read.*

- int Read (float64 ∗target,int num=1)

  *Read one or more float64's into the target buffer. Returns the number successfully read.*

- int Read (float32 ∗target,int num=1)

  *Read one or more float32's into the target buffer. Returns the number successfully read.*

- int Read (char ∗target,int num=1)

  *Read one or more char's into the target buffer. Returns the number successfully read.*

- int Read (char ∗target[ ],int num=1,char ∗scanCode=NULL,int stringSize=-1)

  *Read one or more strings into the target buffer. Returns the number successfully read.*

- int Read (GString ∗target,int num=1,char ∗scanCode=NULL)

  *Read one or more strings into the target buffer. Returns the number successfully read.*

- int Read (void ∗target,int num)

  *Read* num *bytes of raw data.*

- char Peek ()

  *Check what the next byte in the stream is without extracting it.*

- char Get ()

  *Get the next byte in the stream.*

- bool WaitForInput (tTime timeout=0.0)

  *Wait for input to become available.*

- void ReadMode (tReadMode mode)
- tReadMode ReadMode ()

  *Get the current read mode.*

- void Endianness (tEndianness endianness)

  *Set the endianness of the input data. Only applies to binary streams.*

- tEndianness Endianness ()

  *Get the endianness of the input data.*

- void NumberBase (tNumberBase base)

  *Set the number base of the input data. Only applies to text streams.*

- tNumberBase NumberBase ()

  *Get the number base of the input data.*

- bool InputAvailable ()

  *Returns true if input is currently available.*

- operator void $*$ ()

  *Use to check if the last read operation succeeded.*

## Related Functions

(Note that these are not member functions.)

- GIStream & operator>> (GIStream &is,tType &target)

  *The >> operator works the same as it does for C++ istreams.*

### 6.24.1 Detailed Description

Provides many functions for reading data from a stream.

GIStream does not provide the stream data itself. It wraps around a GRawIStream. GIStream provides many different ways to read the data, however, whereas GRawIStream only lets you read the data raw.

You can use a GIStream in either binary or text mode. Text mode works like regular C++ istreams, where numbers are read as ASCII, human-readable numbers, etc. In binary mode, on the other hand, the numbers are read in actual binary code, where a 32-bit number should be exactly four bytes in size in the file. Also, when in binary mode you can set whether to expect the data in big-endian or little-endian format. The data will automatically be converted to the host's endianness when read.

### 6.24.2 Member Enumeration Documentation

#### 6.24.2.1 enum GAUGE3D::GIStream::tSeekType

Position to seek relative to. Used by Seek().

**Enumeration values:**

- ***BEGINNING*** Seek relative to the start of the stream.
- ***CURRENT*** Seek relative to the current position.
- ***END*** Seek relative to the end of the stream.

#### 6.24.2.2 enum GAUGE3D::GIStream::tReadMode

Read mode specifiers for ReadMode().

**Enumeration values:**

- ***BINARY*** Binary mode reading.
- ***ASCII*** ASCII text mode reading.

#### 6.24.2.3 enum GAUGE3D::GIStream::tEndianness

Endianness specifiers for Endianness().

**Enumeration values:**

- ***BIG*** Big endian. Most significant byte comes first. PPC processors are big endian.
- ***LITTLE*** Little endian. Least significant byte comes first. Intel processors are little endian.

#### 6.24.2.4 enum GAUGE3D::GIStream::tNumberBase

Number base specifiers for NumberBase().

**Enumeration values:**

- ***OCT*** Octal (base 8).
- ***DEC*** Decimal (base 10).
- ***HEX*** Hexidecimal (base 16).

### 6.24.3 Constructor & Destructor Documentation

#### 6.24.3.1 GAUGE3D::GIStream::GIStream (pGRawIStream *streamSource*)

Create a buffered input stream.

**Parameters:**

- ***streamSource*** A pointer to a GRawIStream from which to read data.

#### 6.24.3.2 GAUGE3D::GIStream::GIStream (pGRawIStream *streamSource*, int *bufferSize*)

Create a possibly buffered input stream with a given buffer size.

**Parameters:**

- ***streamSource*** A pointer to a GRawIStream from which to read data.

***bufferSize*** The size of the internal buffer in bytes. You can set this to zero to create an unbuffered stream, but this will usually be slower unless you are reading data in large blocks. Also, only buffered streams can be used in text (non-binary) mode.

## 6.24.4 Member Function Documentation

### 6.24.4.1 void GAUGE3D::GIStream::Seek (int *pos*, tSeekType *seekType*)

Seek the stream position.

**Parameters:**

***pos*** Position to seek to, in bytes.

***seekType*** Point in the stream that pos is relative to.

### 6.24.4.2 int GAUGE3D::GIStream::Read (char ∗ *target*[ ], int *num* = 1, char ∗ *scanCode* = NULL, int *stringSize* = -1)

Read one or more strings into the target buffer. Returns the number successfully read.

**Parameters:**

***target*** A pointer to an array of pointers to arrays of char's to which to write the strings.

***num*** The number of strings to read.

***scanCode*** Specifies what characters to include in the string. It has the same syntax as the scanf %[] conversion specifier, without the brackets. So, "^

" would say to stop when a newline is encountered. By default, the set of all non-whitespace characters is used.

***stringSize*** The number of characters in each character array. If the string being read into one particular character array is longer than this, the remainder will be ignored. Each array will always be zero-terminated. A value of -1 indicates that no limit should be placed on the size of the strings read (not recommended!).

#### 6.24.4.3 int GAUGE3D::GIStream::Read (GString ∗ *target*, int *num* = 1, char ∗ *scanCode* = NULL)

Read one or more strings into the target buffer. Returns the number successfully read.

**Parameters:**

- ***target*** A pointer to an array of strings to which to write the data read.
- ***num*** The number of strings to read.
- ***scanCode*** Specifies what characters to include in the string. It has the same syntax as the scanf %[] conversion specifier, without the brackets. So, ”^
” would say to stop when a newline is encountered. By default, the set of all non-whitespace characters is used.

#### 6.24.4.4 char GAUGE3D::GIStream::Peek ()

Check what the next byte in the stream is without extracting it.

**Note:**

This can only be done with buffered streams.

#### 6.24.4.5 char GAUGE3D::GIStream::Get ()

Get the next byte in the stream.

**Note:**

This can only be done with buffered streams.

#### 6.24.4.6 bool GAUGE3D::GIStream::WaitForInput (tTime *timeout* = 0.0)

Wait for input to become available.

**Parameters:**

- ***timout*** Maximum amount of time to wait for input, or 0 to wait indefinately.

**Returns:**

True if input became available, false otherwise.

**Note:**

This function may return false immediately if it knows that input will not be available in the future. For example, if the stream is a file and it is at the end of the file.

#### 6.24.4.7 void GAUGE3D::GIStream::ReadMode (tReadMode *mode*)

\breif Set how to interpret the data in the stream (binary or text).

**Parameters:**

***mode*** One of tReadMode.

#### 6.24.4.8 bool GAUGE3D::GIStream::InputAvailable ()

Returns true if input is currently available.

**Remarks:**

You can use this to test if you are at the end of a file.

#### 6.24.4.9 GAUGE3D::GIStream::operator void ∗ ()

Use to check if the last read operation succeeded.

**Remarks:**

Using the stream as a truth value returns true if the last read operation succeeded. This way, you can say something like:

- while(myStream >> data) {...}

to keep reading in and processing data until there is no longer data available.

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

- gauge3d/files/istream.h

## 6.25 GAUGE3D::GKeyboardMessage Class Reference

Message sent when a key is pressed or released.

`#include <gauge3d/input/message.h>`

Inherits GAUGE3D::GInputMessage.

### Public Methods

- GKeyboardMessage (pGObject origin,GString key,GString modified,bool down)

  *Create a keyboard message with the given info.*

- virtual ~GKeyboardMessage ()

  *Destructor.*

- GString ModifiedKey ()

  *Get the key name with modifiers applied.*

### 6.25.1 Detailed Description

Message sent when a key is pressed or released.

This class is here for two reasons:

- To make it easier to create common message types.
- To make it possible to determine the source of a message. (Hint: Use QueryInterface)

### 6.25.2 Constructor & Destructor Documentation

#### 6.25.2.1 GAUGE3D::GKeyboardMessage::GKeyboardMessage (pGObject *origin*, GString *key*, GString *modified*, bool *down*) `[inline]`

Create a keyboard message with the given info.

**Parameters:**

***origin*** A pointer to the object which is the logical ”originator” of the message. For example, most messages will originate from a GDisplay.

***key*** The name of the key that was pressed or released. If the key has a direct ASCII translation, this should be it. Otherwise, it should be a short but unique name for the key (like ”left control”).

***modified*** This is the key with modifier keys (like shift or caps lock) applied. So, if shift is held down and the ”a” key is pressed, *modified* should be ”A”, whereas *key* should be ”a”. If the key is not printable, this will be an empty sting.

***down*** True if the key was pressed, false if released.

**Remarks:**

Please, please, if you are writing a plugin which reads input from a keyboard and sends GKeyboardMessages, please make sure to check the system keyboard map! Do NOT assume querty! I have a dvorak keyboard, so I will not use your software if you disregard this.

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

- gauge3d/input/message.h

## 6.26 GAUGE3D::GLibrary Class Reference

Loads shared libraries (dll's and so's).

`#include <gauge3d/osabstraction/library.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- GLibrary (GString filename)

  *Create a GLibrary for a given library file.*
- ∼GLibrary ()

  *Close the library.*
- bool Open ()

  *Open the library.*
- void∗ GetSymbol (GString symbolName)

  *Get a pointer to a function in the library.*
- void Close ()

  *Unload the library from memory.*
- GString Filename ()

  *Get the name of the library, as given to the constructor.*
- bool IsOpen ()

  *Returns true if the library is open.*

### 6.26.1 Detailed Description

Loads shared libraries (dll's and so's).

This class allows you to load executable add-ons to your program. You could use it directly, but we recommend using GObjectLoader instead, as it provides a much nicer system for doing this. GObjectLoader uses GLibrary to do what it does.

## 6.26.2 Constructor & Destructor Documentation

### 6.26.2.1 GAUGE3D::GLibrary::GLibrary (GString *filename*)

Create a GLibrary for a given library file.

**Parameters:**

***filename*** The path and name of the library file on the disk. If no path is given (just a filename), the system's standard library directories will be searched.

**Remarks:**

The library can be closed when not in use and re-openned later, but the filename will never change. This way, the library implementation can opt to keep the libraries open when in debug mode in case the debugger is easily confused (most are! even GDB!).

## 6.26.3 Member Function Documentation

### 6.26.3.1 bool GAUGE3D::GLibrary::Open ()

Open the library.

**Returns:**

True if the library was successfully opened, or false if an error occured.

**Remarks:**

Don't forget that if you close and re-open a library, you have to get all your symbols again. They will not be the same.

### 6.26.3.2 void ∗ GAUGE3D::GLibrary::GetSymbol (GString *symbolName*)

Get a pointer to a function in the library.

**Parameters:**

***symbolName*** The name of the function to get a pointer to.

**Returns:**

A pointer to the function, or NULL if an error occured or the function was not found.

**Remarks:**

Most operating systems allow you to get symbols other than functions from a library. Do not depend on this behavior.

#### 6.26.3.3 void GAUGE3D::GLibrary::Close ()

Unload the library from memory.

This will unload the library from RAM. If you need to use it again, call Open().

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

- gauge3d/osabstraction/library.h

## 6.27 GAUGE3D::GLight Class Reference

Represents a light source in a 3D scene.

`#include <gauge3d/renderer/light.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- virtual ~**GLight** ()
- virtual void Show ()=0

  *Turn on the light.*

- virtual void Hide ()=0

  *Turn off the light.*

- virtual void Remove ()=0

  *Permenantly remove the light from the scene.*

- virtual void Position (const GVector &position)=0

  *Set the position of the light, and make the light local.*

- virtual void Direction (const GVector &direction)=0

  *Set the direction of the light, and make the light infinite.*

- virtual void Color (float red,float green,float blue)=0

  *Set the color of the light.*

- virtual void Color (const float color[ ])=0

  *Set the color of the light. (3-element array: RGB).*

- virtual void LightProps (float ambient,float diffuse,float specular=0.0)=0

  *Set the properties of the light.*

- virtual void LightProps2v (const float lightProps[ ])=0

  *Set the properties of the light. (ambient and diffuse).*

- virtual void LightProps3v (const float lightProps[ ])=0

  *Set the properties of the light. (ambient, diffuse, and specular).*

- virtual void Attenuation (float linear,float quadratic=0.0)=0

  *Sets linear and quadratic attenuation factors. (makes light dimmer with distance).*

- virtual void Spotlight (const GVector &direction,float exponent,float width)=0

  *Make the light act like a spotlight.*

### 6.27.1 Detailed Description

Represents a light source in a 3D scene.

### 6.27.2 Member Function Documentation

#### 6.27.2.1 void GAUGE3D::GLight::Show () `[pure virtual]`

Turn on the light.

All lights start out hidden. After you have cached the light, call Show() to enable it. Show and Hide are a good way to turn lights on and off.

#### 6.27.2.2 void GAUGE3D::GLight::LightProps (float *ambient*, float *diffuse*, float *specular* = 0.0) `[pure virtual]`

Set the properties of the light.

**Parameters:**

***ambient*** The ambient light level. This represets the light that is not coming directly from the source, but has bounced off a few walls first. It is simulated through simple distance attenuation.

***diffuse*** The diffuse light level. This represets the light that is coming directly from the source. The orientation of the surface relative to the light source affects the level of the light, but the viewpoint does not matter.

***specular*** The specular light level. This represents light that has been reflected off the surface of the object. Both the orientation of the object's surface as well as the viewpoint are taken into account.

#### 6.27.2.3 void GAUGE3D::GLight::Spotlight (const GVector & *direction*, float *exponent*, float *width*) `[pure virtual]`

Make the light act like a spotlight.

**Parameters:**

***direction*** The direction of the spot light.

***exponent*** How focused the light is. Setting this higher will make the center of the spot brighter and the edges dimmer.

***width*** The width of the spotlight in degrees. Set to -1.0 to disable the spotlight effect.

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

- gauge3d/renderer/light.h

## 6.28 GAUGE3D::GLine Class Reference

Represents a line in 3D space.

`#include <gauge3d/3dmath/line.h>`

### Public Methods

- **GLine** ()
- **GLine** (const GVector &position,const GVector &direction)
- bool FindIntersection (GVector ∗result,const GLine &other)const

  *Find intersection between two lines.*
- bool Contains (const GVector &point)const

  *test if a point is on the line.*
- GCoordinate FindDistance (const GVector &point)const

  *Finds the shortest distance between the line and a point.*
- GCoordinate FindSquareDistance (const GVector &point)const

  *Finds the square of the shortest distance between the line and a point (faster).*
- GCoordinate FindDistance (const GLine &other)const

  *Finds the shortest distance between two lines.*
- GCoordinate FindSquareDistance (const GLine &point)const

  *Finds the square of the shortest distance between two lines (faster).*
- const GLine& operator+= (const GVector &other)

  *Moves the line.*
- const GLine& operator-= (const GVector &other)

  *Moves the line.*

### Public Attributes

**Coordinates**

*Plücker coordinates for the line.*

- GVector **u**
- GVector **v**

## 6.28.1 Detailed Description

Represents a line in 3D space.

The line is represented using Plücker coordinates. A line is represented by two vectors U and V. U is the direction of the line, and the cross product of U and any point on the line is equal to V.

## 6.28.2 Member Function Documentation

### 6.28.2.1 bool GAUGE3D::GLine::FindIntersection (GVector ∗ *result*, const GLine & *other*) const

Find intersection between two lines.

**Parameters:**

***result*** Pointer to a vector which should be overwritten with the point of intersection. If NULL, it will not be overwritten.

***other*** The line with which to test for intersection.

**Returns:**

True if a point of intersection was found, or false if the lines did not intersect or if they were equal.

### 6.28.2.2 bool GAUGE3D::GLine::Contains (const GVector & *point*) const `[inline]`

test if a point is on the line.

**Returns:**

True if the given point was on the line or false otherwise.

### 6.28.2.3 GCoordinate GAUGE3D::GLine::FindSquareDistance (const GVector & *point*) const

Finds the square of the shortest distance between the line and a point (faster).

This is faster because no square root needs to be performed.

### 6.28.2.4 GCoordinate GAUGE3D::GLine::FindSquareDistance (const GLine & *point*) const

Finds the square of the shortest distance between two lines (faster).

This is faster because no square root needs to be performed.

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

- gauge3d/3dmath/line.h

## 6.29 GAUGE3D::GMessage Class Reference

Represents a message of some sort.

`#include <gauge3d/input/message.h>`

Inherits GAUGE3D::GObject.

Inherited by GAUGE3D::GInputMessage, and GAUGE3D::GQuitMessage.

### Public Methods

- GMessage (pGObject origin)
  *Create a GMessage with the given origin.*
- virtual ∼GMessage ()
  *Destructor.*
- pGObject Origin ()
  *Returns a pointer to the object that this message came from.*

### 6.29.1 Detailed Description

Represents a message of some sort.

GMessage itself does not contain any functions for finding out what sort of message it is. You must use QueryInterface to determine the type of the message.

### 6.29.2 Constructor & Destructor Documentation

#### 6.29.2.1 GAUGE3D::GMessage::GMessage (pGObject *origin*) `[inline]`

Create a GMessage with the given origin.

**Parameters:**

***origin*** A pointer to the object which is the logical ”originator” of the message. For example, most messages will originate from a GDisplay.

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

- gauge3d/input/message.h

## 6.30 GAUGE3D::GMessageBox Class Reference

Opens a simple messagebox using the native GUI.

`#include <gauge3d/osabstraction/messagebox.h>`

Inherits GAUGE3D::GObject.

### Public Types

- enum tEnum { WARNING, ERROR, QUESTION, INFORMATION, ABORT_RETRY_IGNORE, OK_CANCEL, RETRY_CANCEL, YES_NO, YES_NO_CANCEL, OK, ABORT, CANCEL, RETRY, IGNORE, YES, NO }

  *Flags and settings used by GMessageBox.*

### Public Methods

- GMessageBox (GString message,tEnum type,tEnum buttons)

  *Create a message box, but don't display it yet.*

- ~GMessageBox ()

  *Destroy the message box.*

- tEnum Run ()

  *Run the message box (modal).*

### 6.30.1 Detailed Description

Opens a simple messagebox using the native GUI.

This class can be used to display simple message boxes before the graphical sub-system is available. It should not be used once the graphics system is running, however, as the message box may not be visible to the user at that time.

### 6.30.2 Member Enumeration Documentation

#### 6.30.2.1 enum GAUGE3D::GMessageBox::tEnum

Flags and settings used by GMessageBox.

**Remarks:**

Sorry about the limited button sets. That was Windows' fault. :(

**Enumeration values:**

| | |
|---|---|
| ***WARNING*** | Message type. |
| ***ERROR*** | Message type. |
| ***QUESTION*** | Message type. |
| ***INFORMATION*** | Message type. |
| ***ABORT_RETRY_IGNORE*** | Button set. |
| ***OK_CANCEL*** | Button set. |
| ***RETRY_CANCEL*** | Button set. |
| ***YES_NO*** | Button set. |
| ***YES_NO_CANCEL*** | Button set. |
| ***OK*** | Button set / User response. |
| ***ABORT*** | User response. |
| ***CANCEL*** | User response. |
| ***RETRY*** | User response. |
| ***IGNORE*** | User response. |
| ***YES*** | User response. |
| ***NO*** | User response. |

### 6.30.3 Constructor & Destructor Documentation

#### 6.30.3.1 GAUGE3D::GMessageBox::GMessageBox (GString *message*, tEnum *type*, tEnum *buttons*)

Create a message box, but don't display it yet.

**Parameters:**

| | |
|---|---|
| ***message*** | The text of the message to display. |
| ***type*** | One of the message type values in tEnum. |
| ***buttons*** | One of the button sets in tEnum. |

### 6.30.4 Member Function Documentation

#### 6.30.4.1 tEnum GAUGE3D::GMessageBox::Run ()

Run the message box (modal).

**Returns:**

One of the user response values in tEnum. This is the button which the user clicked on.

**Remarks:**

This function does not return until the user has clicked on a button.

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

- gauge3d/osabstraction/messagebox.h

## 6.31 GAUGE3D::GMessageQueue Class Reference

Facilitates the distribution of messages.

`#include <gauge3d/input/messagequeue.h>`

Inherits GAUGE3D::GObject.

### Public Types

- typedef GSmartPointer<Handler> **pHandler**

### Public Methods

- **GMessageQueue** ()
- ~**GMessageQueue** ()
- void EnqueueMessage (pGMessage message)

  *Enqueue a message.*

- void DispatchMessage (pGMessage message)

  *Immediately dispatch a message, bypassing the queue.*

- void DiscardMessages ()

  *Discard all enqueued messages.*

- void AddHandler (pHandler handler,float priority=0.5)

  *Add a handler to the list of handlers.*

- void RemoveHandler (pHandler handler)

  *Remove a message handler from the list.*

- void ClearHandlers ()

  *Remove all handlers from the message handler list.*

- virtual void DoFrame (tTime time,tTime frametime)

  *Dispatch all enqueued messages that have not been sent. Will be called automatically.*

### 6.31.1 Detailed Description

Facilitates the distribution of messages.

This class queues and distributes messages. First, you give it pointers to one or more message handlers. When a message is ready to be dispatched, it will be sent to each handler one by one until one of the handlers reports that it has handled the message.

### 6.31.2 Member Function Documentation

#### 6.31.2.1 void GAUGE3D::GMessageQueue::AddHandler (pHandler *handler*, float *priority* = 0.5)

Add a handler to the list of handlers.

**Parameters:**

***handler*** A pointer to a Handler to add.

***priority*** A priority for the handler, between 0.0 and 1.0. This determines the order in which the handlers will be called. Lower numbers will be called first. Do not set this to exactly 0.0 or 1.0 unless you are doing something special (like creating an input filter).

**Remarks:**

You could have a handler that acts as an input filter by giving it a priority of 0.0. Have the handler return true if the message should be ignored. Since the priority is 0.0, it will always be called first, and if it returns true, no other handlers will be called.

#### 6.31.2.2 void GAUGE3D::GMessageQueue::DoFrame (tTime *time*, tTime *frametime*) `[virtual]`

Dispatch all enqueued messages that have not been sent. Will be called automatically.

**Remarks:**

You can pretend this function does not exist if you want. You should never call it yourself.

Reimplemented from GAUGE3D::GObject.

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

- gauge3d/input/messagequeue.h

## 6.32 GAUGE3D::GModel Class Reference

Base class for 3D model plugins.

`#include <gauge3d/graphics/model.h>`

Inherits GAUGE3D::GObject.

Inherited by GAUGE3D::GBillboardSprite, GAUGE3D::GSphereModel, and GAUGE3D::GSuperModel.

### Public Types

- typedef GSmartPointer<AnimState> pAnimState

  *Smart pointer to a AnimState.*

- typedef GSmartPointer<RenderInfo> pRenderInfo

  *Smart pointer to a RenderInfo.*

### Public Methods

- virtual ~**GModel** ()
- virtual pAnimState CreateAnimState ()=0

  *Generates an AnimState object associated with this model.*

- virtual pRenderInfo CreateRenderInfo ()=0

  *Create a RenderInfo bound to this model.*

### 6.32.1 Detailed Description

Base class for 3D model plugins.

Plugins that allow GAUGE to load various 3D model formats (like MD2 from Quake 2) are derived from this class.

### 6.32.2 Member Function Documentation

#### 6.32.2.1 pRenderInfo GAUGE3D::GModel::CreateRenderInfo () `[pure virtual]`

Create a RenderInfo bound to this model.

**Remarks:**

> Obviously, this function uses dynamic memory and should not be called every frame. A renderer should call this once when the model is assigned to a sprite and then use the same RenderInfo object every time the sprite is drawn.

Reimplemented in GAUGE3D::GBillboardSprite, GAUGE3D::GSphereModel, and GAUGE3D::GSuperModel.

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

- gauge3d/graphics/model.h

## 6.33 GAUGE3D::GMouseButtonMessage Class Reference

Message sent when a mouse button is pressed or released.

`#include <gauge3d/input/message.h>`

Inherits GAUGE3D::GInputMessage.

### Public Methods

- GMouseButtonMessage (pGObject origin,GString button,bool down)

  *Create a mouse button message with the given info.*

- virtual ∼**GMouseButtonMessage** ()

### 6.33.1 Detailed Description

Message sent when a mouse button is pressed or released.

This class is here for two reasons:

- To make it easier to create common message types.
- To make it possible to determine the source of a message. (Hint: Use QueryInterface)

### 6.33.2 Constructor & Destructor Documentation

#### 6.33.2.1 GAUGE3D::GMouseButtonMessage::GMouseButtonMessage (pGObject *origin*, GString *button*, bool *down*) `[inline]`

Create a mouse button message with the given info.

**Parameters:**

***origin*** A pointer to the object which is the logical ”originator” of the message. For example, most messages will originate from a GDisplay.

***button*** The name of the button pressed. Something like ”left mouse”.

***down*** True if the button was pressed, false if released.

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

- gauge3d/input/message.h

## 6.34 GAUGE3D::GMouseMotionMessage Class Reference

Message sent when the mouse is moved.

`#include <gauge3d/input/message.h>`

Inherits GAUGE3D::GInputMessage.

### Public Methods

- GMouseMotionMessage (pGObject origin,GString axis,float change,float absolute)

  *Create a mouse motion message with the given info.*

- virtual ~**GMouseMotionMessage** ()
- float Absolute ()

  *Returns the absolute position of the mouse.*

### 6.34.1 Detailed Description

Message sent when the mouse is moved.

This class is here for two reasons:

- To make it easier to create common message types.
- To make it possible to determine the source of a message. (Hint: Use QueryInterface)

### 6.34.2 Constructor & Destructor Documentation

#### 6.34.2.1 GAUGE3D::GMouseMotionMessage::GMouseMotionMessage (pGObject *origin*, GString *axis*, float *change*, float *absolute*) `[inline]`

Create a mouse motion message with the given info.

**Parameters:**

***origin*** A pointer to the object which is the logical ”originator” of the message. For example, most messages will originate from a GDisplay.

***button*** The name of the axis on which the motion occurred. Something like ”mouse x”.

***change*** The distance that the mouse moved. This should be measured in the same units as *absolute*.

***absolute*** If the mouse is *not* captured by the display, then this is the absolute position of the mouse. The edges of the screen are 0.0 and 1.0.

**Remarks:**

Yes, mouse motion will usually have to be sent as two separate messages (one for x and one for y).

## 6.34.3 Member Function Documentation

### 6.34.3.1 float GAUGE3D::GMouseMotionMessage::Absolute () `[inline]`

Returns the absolute position of the mouse.

**Remarks:**

This will be zero if the mouse is in captured mode.

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

- gauge3d/input/message.h

## 6.35 GAUGE3D::GMouseWheelMessage Class Reference

Message sent when the mouse wheel is moved.

`#include <gauge3d/input/message.h>`

Inherits GAUGE3D::GInputMessage.

### Public Types

- enum **tDirection** { **UP**, **DOWN** }

### Public Methods

- GMouseWheelMessage (pGObject origin,GString name,tDirection direction)

  *Create a mouse wheel message with the given info.*

- virtual ~**GMouseWheelMessage** ()

### 6.35.1 Detailed Description

Message sent when the mouse wheel is moved.

This class is here for two reasons:

- To make it easier to create common message types.
- To make it possible to determine the source of a message. (Hint: Use QueryInterface)

### 6.35.2 Constructor & Destructor Documentation

#### 6.35.2.1 GAUGE3D::GMouseWheelMessage::GMouseWheelMessage (pGObject *origin*, GString *name*, tDirection *direction*) `[inline]`

Create a mouse wheel message with the given info.

**Parameters:**

***origin*** A pointer to the object which is the logical ”originator” of the message. For example, most messages will originate from a GDisplay.

***name*** The name of the input. Just ”mouse wheel” should do in most cases.

***direction*** Either UP or DOWN, indicating the direction the mouse wheel was moved.

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

- gauge3d/input/message.h

## 6.36 GAUGE3D::GMutex Class Reference

Use to prevent data corruption caused by multiple threads accessing the same data.

`#include <gauge3d/osabstraction/thread.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- GMutex ()

  *Create the mutex.*

- ~GMutex ()

  *Destroy the mutex. It must be unlocked when this happens.*

- void Lock ()

  *Wait until mutex is unlocked, and then lock the mutex to this thread.*

- bool TryLock ()

  *Check if mutex is locked, and lock it to this thread if not.*

- void Unlock ()

  *Release lock on a thread.*

### 6.36.1 Detailed Description

Use to prevent data corruption caused by multiple threads accessing the same data.

If two or more threads can access the same data, and at least one is writing to it, you should use a mutex to prevent them from accessing it at the same time.

### 6.36.2 Member Function Documentation

#### 6.36.2.1 bool GAUGE3D::GMutex::TryLock ()

Check if mutex is locked, and lock it to this thread if not.

**Returns:**

True if the mutex was not locked (and is now locked to the calling thread), or false if the mutex was locked by another thread at the time.

#### 6.36.2.2 void GAUGE3D::GMutex::Unlock ()

Release lock on a thread.

Only the thread that the mutex is locked to may call this.

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

- gauge3d/osabstraction/thread.h

## 6.37 GAUGE3D::GObject Class Reference

Base class objects with smart pointer and QueryInterface capabilities.

`#include <gauge3d/stddefs.h>`

Inherited by GAUGE3D::GCamera, GAUGE3D::GDirectory, GAUGE3D::GDisplay, GAUGE3D::GFile, GAUGE3D::GFileLoader, GAUGE3D::GImage, GAUGE3D::GIStream, GAUGE3D::GLibrary, GAUGE3D::GLight, GAUGE3D::GMessage, GAUGE3D::GMessageBox, GAUGE3D::GMessageQueue, GAUGE3D::GMessageQueue::Handler, GAUGE3D::GModel, GAUGE3D::GModel::AnimState, GAUGE3D::GModel::RenderInfo, GAUGE3D::GMutex, GAUGE3D::GObjectLoader, GAUGE3D::GOStream, GAUGE3D::GPlugin, GAUGE3D::GRawIStream, GAUGE3D::GRawOStream, GAUGE3D::GRenderer, GAUGE3D::GScene, GAUGE3D::GScheduler, GAUGE3D::GSprite, GAUGE3D::GSurface, GAUGE3D::GTexture, and GAUGE3D::GThread.

### Public Types

- typedef const char∗ tError

  *An error identifier.*

### Public Methods

- virtual ∼**GObject** ()
- tError Error ()

  *Get the last error.*

- virtual void DoFrame (tTime time,tTime frameTime)

  *Called by GScheduler every frame if the object requests it.*

- virtual void TimerCallback (tTime time)

  *Called by GScheduler at a desired time.*

### Protected Methods

- void Error (tError error)

*Set the last error.*

### 6.37.1 Detailed Description

Base class objects with smart pointer and QueryInterface capabilities.

GObject is the base class for many, but not all, GAUGE classes. Using it adds several capabilities to the derived class:

- GSmartPointer can point to GObject and classes derived from it.
- The QueryInterface system provides COM-like interface building capabilities.
- Per-class error reporting system.

### 6.37.2 Member Typedef Documentation

#### 6.37.2.1 typedef const char ∗ GAUGE3D::GObject::tError

An error identifier.

tError is an error identifier. In actuality, it is a const char∗ which points to a human-readable string. Every class which has error conditions should define these conditions as member variables of type static const tError. Then, to check which error occured, the caller can check the error identifier against each of these member variables. If it doesn't know how to handle the error, it can just output it to cerr or something since the error identifier points to a human-readable string.

### 6.37.3 Member Function Documentation

#### 6.37.3.1 void GAUGE3D::GObject::DoFrame (tTime *time*, tTime *frametime*) `[virtual]`

Called by GScheduler every frame if the object requests it.

**Parameters:**

***time*** The current time, in seconds. This is measured relative to a completely arbitrary point (usually the point when the engine was initialized).

***frameTime*** The amount of time which elapsed between the start of the previous frame and the start of this frame.

See GScheduler for more info on how to make it call this function every frame. Objects that don't need this functionality can ignore it.

Reimplemented in GAUGE3D::GInputMap, GAUGE3D::GMessageQueue, and GAUGE3D::GScheduler.

#### 6.37.3.2 void GAUGE3D::GObject::TimerCallback (tTime *time*) `[virtual]`

Called by GScheduler at a desired time.

**Parameters:**

***time*** The current time. This will usually be a bit later than the time at which the object requested to be called back, but it will never be earlier.

See GScheduler for more info on how to make it call this function at a particular point in time. Objects that don't need this functionality can ignore it.

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

- gauge3d/common/object.h

## 6.38 GAUGE3D::GPlugin::ObjectInfo Struct Reference

Used to describe the objects that a plugin contains.

`#include <gauge3d/plugins/plugin.h>`

### Public Attributes

- GString mClassname

  *The class name of the object.*
- GString mName

  *The human-readable name of the object.*
- GString mDescription

  *A human-readable description of the object.*
- GArray<GString> mBaseClasses

  *A list of all the base classes of the object (excluding GObject).*
- GArray<GString> mKeywords

  *A list of keywords that users can use to request the object.*
- GArray<GString> mFileTypes

  *A list of file types which are supported by the object.*

### 6.38.1 Detailed Description

Used to describe the objects that a plugin contains.

The plugin tells the object loader what objects are available from it by giving it an array of ObjectInfo objects. Each one gives info for one of the available objects.

### 6.38.2 Member Data Documentation

#### 6.38.2.1 GArray< GString > GAUGE3D::GPlugin::ObjectInfo::mKeywords

A list of keywords that users can use to request the object.

For example, the OpenGL renderer has the keyword ”opengl”, and the D3D renderer has ”direct3d”. Keywords should be all lower-case letters.

#### 6.38.2.2 GArray< GString > GAUGE3D::GPlugin::ObjectInfo::mFileTypes

A list of file types which are supported by the object.

Types are represented by the extension of the file. For example: ”mp3”, ”bsp”, ”tgz”. Names should be all lower-case and should not include the ’.’.

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

- gauge3d/plugins/plugin.h

## 6.39 GAUGE3D::GObjectLoader Class Reference

Loads objects from plugins.

`#include <gauge3d/plugins/objectloader.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- **GObjectLoader** ()
- ~**GObjectLoader** ()
- pGObject LoadObject (const GString &className)

  *Find an object with the given class name and create it.*
- pGObject LoadObjectOfType (const GString &baseClass)

  *Find an object derived from the specified class and create it.*
- pGObject LoadObjectWithKeyword (const GString &keyWord,const GString &baseClass)

  *Find an object with the given keyword and create it.*
- pGObject LoadObjectForFile (pGFile file,const GString &baseClass)

  *Find an object that can load the given file and that is derived from the given base class and create it.*
- void AddPlugin (pGFile library)

  *Add a library to the list of plugins to search for objects.*

### 6.39.1 Detailed Description

Loads objects from plugins.

This class acts sort of like a CORBA ORB, except for GAUGE plugin objects. (Disclaimer: I've never used CORBA, so I don't actually know how a CORBA ORB works. Let me know if that last statement was wrong.) The class loads plugins and queries them to see what objects can be loaded from them, and what functions those objects can perform. It then fields requests for objects that perform certain types of functions by matching the request with a plugin object that performs the task and getting an instance of the object from the corresponding plugin.

## 6.39.2 Member Function Documentation

### 6.39.2.1 pGObject GAUGE3D::GObjectLoader::LoadObject (const GString & *className*)

Find an object with the given class name and create it.

**Returns:**

A pointer to the object, or NULL if an obect with the given name could not be found.

### 6.39.2.2 pGObject GAUGE3D::GObjectLoader::LoadObjectOfType (const GString & *baseClass*)

Find an object derived from the specified class and create it.

**Returns:**

A pointer to the object, or NULL if an obect derived from the given class could not be found.

### 6.39.2.3 pGObject GAUGE3D::GObjectLoader::LoadObjectWithKeyword (const GString & *keyWord*, const GString & *baseClass*)

Find an object with the given keyword and create it.

**Returns:**

A pointer to the object, or NULL if an obect with the given keyword could not be found.

Each object can have several keywords. See GPlugin::ObjectInfo::mKeywords for more info on them.

### 6.39.2.4 pGObject GAUGE3D::GObjectLoader::LoadObjectForFile (pGFile *file*, const GString & *baseClass*)

Find an object that can load the given file and that is derived from the given base class and create it.

**Parameters:**

***file*** The file that the object should load.

***baseClass*** The class that the object should be derived from.

So, if you have a jpeg image file ”jpegFile”, you might say:

- objectLoader.LoadObject(jpegFile, ”GImage”);

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

- gauge3d/plugins/objectloader.h

## 6.40 GAUGE3D::GOStream Class Reference

Provides many functions for writing data to a stream.

`#include <gauge3d/files/ostream.h>`

Inherits GAUGE3D::GObject.

### Public Types

- enum tSeekType { BEGINNING, CURRENT, END }

  *Position to seek relative to. Used by Seek().*

- enum tWriteMode { BINARY, ASCII }

  *Write mode specifiers for WriteMode().*

- enum tEndianness { BIG, LITTLE }

  *Endianness specifiers for Endianness().*

- enum tNumberBase { OCT = 8, DEC = 10, HEX = 16 }

  *Number base specifiers for NumberBase().*

### Public Methods

- GOStream (pGRawOStream streamDest)

  *Create a buffered output stream.*

- GOStream (pGRawOStream streamDest,int bufferSize)

  *Create a buffered output stream.*

- virtual ~GOStream ()

  *Destructor.*

- void Seek (int pos,tSeekType seekType)

  *Seek the stream position.*

- int TellPos ()

  *Get the distance in bytes between the beginning of the stream and the current position.*

- int Write (const uint64 ∗source,int num=1)

  *Write one or more uint64's from the source buffer. Returns the number successfully written.*

- int Write (const uint32 ∗source,int num=1)

  *Write one or more uint32's from the source buffer. Returns the number successfully written.*

- int Write (const uint16 ∗source,int num=1)

  *Write one or more uint16's from the source buffer. Returns the number successfully written.*

- int Write (const int64 ∗source,int num=1)

  *Write one or more int64's from the source buffer. Returns the number successfully written.*

- int Write (const int32 ∗source,int num=1)

  *Write one or more int32's from the source buffer. Returns the number successfully written.*

- int Write (const int16 ∗source,int num=1)

  *Write one or more int16's from the source buffer. Returns the number successfully written.*

- int Write (const float64 ∗source,int num=1)

  *Write one or more float64's from the source buffer. Returns the number successfully written.*

- int Write (const float32 ∗source,int num=1)

  *Write one or more float32's from the source buffer. Returns the number successfully written.*

- int Write (const char ∗source,int num=1)

  *Write one or more char's from the source buffer. Returns the number successfully written.*

- int Write (const char ∗source[ ],int num=1)

  *Write one or more strings from the source buffer. Returns the number successfully written.*

- int Write (const GString ∗source,int num=1)

  *Write one or more strings from the source buffer. Returns the number successfully written.*

- int Write (const void ∗source,int num)

  *Write <u>num</u> bytes of raw data.*

- void Put (char c)

  *Write a character to the stream.*

- int Flush ()

  *Write any data that is in the buffer to the output stream.*

- void WriteMode (tWriteMode mode)
- tWriteMode WriteMode ()

  *Get the current write mode.*

- void Separator (char separator,bool enable)

  *Specify a character to place after each item.*

- char Separator ()

  *Get the current separator character.*

- void Endianness (tEndianness endianness)

  *Set the endianness in which to write the output data. Only applies to binary streams.*

- tEndianness Endianness ()

  *Get the endianness of the output data.*

- void NumberBase (tNumberBase base)

  *Set the number base in which to write the output data. Only applies to text streams.*

- tNumberBase NumberBase ()

  *Get the number base of the output data.*

- operator void ∗ ()

  *Use to check if the last write operation succeeded.*

## Related Functions

(Note that these are not member functions.)

- template<class tType> GOStream & operator<< (GOStream &os,const tType &source)

  *The << operator works the same as it does for C++ ostreams.*

### 6.40.1 Detailed Description

Provides many functions for writing data to a stream.

GOStream does not provide the stream destination itself. It wraps around a GRawOStream. GOStream provides many different ways to write the data, however, whereas GRawOStream only lets you write the data raw.

You can use a GOStream in either binary or text mode. Text mode works like regular C++ ostreams, where numbers are written as ASCII, human-readable numbers, etc. In binary mode, on the other hand, the numbers are written in actual binary code, where a 32-bit number will be exactly four bytes in size in the file. Also, when in binary mode you can set whether to write the data in big-endian or little-endian format. The data will automatically be converted from the host's endianness when written.

### 6.40.2 Member Enumeration Documentation

#### 6.40.2.1 enum GAUGE3D::GOStream::tSeekType

Position to seek relative to. Used by Seek().

**Enumeration values:**

- ***BEGINNING*** Seek relative to the start of the stream.
- ***CURRENT*** Seek relative to the current position.
- ***END*** Seek relative to the end of the stream.

#### 6.40.2.2 enum GAUGE3D::GOStream::tWriteMode

Write mode specifiers for WriteMode().

**Enumeration values:**

- ***BINARY*** Binary mode writing.
- ***ASCII*** ASCII text mode writing.

#### 6.40.2.3 enum GAUGE3D::GOStream::tEndianness

Endianness specifiers for Endianness().

**Enumeration values:**

***BIG*** Big endian. Most significant byte comes first. PPC processors are big endian.

***LITTLE*** Little endian. Least significant byte comes first. Intel processors are little endian.

#### 6.40.2.4 enum GAUGE3D::GOStream::tNumberBase

Number base specifiers for NumberBase().

**Enumeration values:**

***OCT*** Octal (base 8).

***DEC*** Decimal (base 10).

***HEX*** Hexidecimal (base 16).

### 6.40.3 Constructor & Destructor Documentation

#### 6.40.3.1 GAUGE3D::GOStream::GOStream (pGRawOStream *streamDest*)

Create a buffered output stream.

**Parameters:**

***streamDest*** A pointer to a GRawOStream to which to write data.

#### 6.40.3.2 GAUGE3D::GOStream::GOStream (pGRawOStream *streamDest*, int *bufferSize*)

Create a buffered output stream.

**Parameters:**

***streamDest*** A pointer to a GRawOStream to which to write data.

***bufferSize*** The size of the internal buffer in bytes. You can set this to zero to create an unbuffered stream, but this will usually be slower unless you are writing data in large blocks. Also, only buffered streams can be used in text (non-binary) mode.

## 6.40.4 Member Function Documentation

### 6.40.4.1 void GAUGE3D::GOStream::Seek (int *pos*, tSeekType *seekType*)

Seek the stream position.

**Parameters:**

***pos*** Position to seek to, in bytes.

***seekType*** Point in the stream that pos is relative to.

### 6.40.4.2 int GAUGE3D::GOStream::Write (const char ∗ *source*[ ], int *num* = 1)

Write one or more strings from the source buffer. Returns the number successfully written.

**Parameters:**

***source*** An array of pointers to null-terminated character strings.

***num*** The number of strings to write.

**Returns:**

The number of strings successfully written.

### 6.40.4.3 int GAUGE3D::GOStream::Write (const GString ∗ *source*, int *num* = 1)

Write one or more strings from the source buffer. Returns the number successfully written.

**Parameters:**

***source*** An array of strings.

***num*** The number of strings to write.

**Returns:**
The number of strings successfully written.

#### 6.40.4.4 void GAUGE3D::GOStream::Put (char *c*)

Write a character to the stream.

**Note:**
This can only be done with buffered streams.

#### 6.40.4.5 int GAUGE3D::GOStream::Flush ()

Write any data that is in the buffer to the output stream.

**Returns:**
The number of bytes still in the buffer (should always be zero).

#### 6.40.4.6 void GAUGE3D::GOStream::WriteMode (tWriteMode *mode*)

\breif Set how to output the data in the stream (binary or text).

**Parameters:**
***mode*** One of tWriteMode.

#### 6.40.4.7 void GAUGE3D::GOStream::Separator (char *separator*, bool *enable*)

Specify a character to place after each item.

**Parameters:**
***separator*** A character that should be placed after every item that is written.
***enable*** Whether or not to enable this functionality.

**Remarks:**

This allows you to output an array of numbers without getting them all mushed together.

**Note:**

This only applies to test i/o, not binary. The separator will *not* be used after Put() calls (as the other write calls use Put() for text output).

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

- gauge3d/files/ostream.h

## 6.41 GAUGE3D::GPlane Class Reference

Represents a plane in 3D space.

`#include <gauge3d/3dmath/plane.h>`

### Public Methods

- GPlane ()

  *Creates an uninitialized plane.*
- GPlane (const GVector &point,const GVector &normal)

  *Creates a plane with the given normal that contains the given point.*
- GPlane (const GVector &normal,GCoordinate _d)

  *Initialized n and d to the given values.*
- bool IsFacing (const GVector &point)const

  *Returns true if the plane is facing the point.*
- bool FindIntersection (GVector ∗result,const GLine &line)const

  *Finds the intersection between the plane and a line.*
- bool FindIntersection (GLine ∗result,const GPlane &other)const

  *Finds the intersection between two planes.*
- bool FindIntersection (GVector ∗result,const GPlane &other1,GPlane &other2)const

  *Finds the intersection between three planes.*
- bool Contains (const GVector &point)const

  *Returns true if the point is on the plane.*
- GCoordinate FindDistance (const GVector &point)const

  *Finds the shortest distance between the plane and the point.*
- GCoordinate FindSquareDistance (const GVector &point)const

  *Finds the square of the shortest distance between the plane and the point (faster).*
- const GPlane& operator+= (const GVector &vec)

*Moves the plane a certain distance.*

- const GPlane& operator-= (const GVector &vec)

  *Moves the plane a certain distance.*

## Public Attributes

- GVector n

  *Normal vector to plane.*

- GCoordinate d

  *Umm... d... it's d... yeah...*

### 6.41.1 Detailed Description

Represents a plane in 3D space.

The plane is represented as a normal vector and a scalar so that n $*$ p + d = 0 for all points p on the plane.

### 6.41.2 Member Function Documentation

#### 6.41.2.1 bool GAUGE3D::GPlane::IsFacing (const GVector & *point*) const `[inline]`

Returns true if the plane is facing the point.

The plane is considered to be "facing" the point if the normal vector of the plane points in the direction of the point from any point on the plane. In other (non-)words: (point $*$ n + d >= 0)

#### 6.41.2.2 bool GAUGE3D::GPlane::FindIntersection (GVector $*$ *result*, const GLine & *line*) const

Finds the intersection between the plane and a line.

**Parameters:**

***result*** Pointer to a vector which should be overwritten with the point of intersection. If NULL, it will not be overwritten.

***line*** The line with which to test for intersection.

**Returns:**

True if a point of intersection was found, or false if the line and plane did not intersect or if the line was on the plane.

#### 6.41.2.3 bool GAUGE3D::GPlane::FindIntersection (GLine ∗ *result*, const GPlane & *other*) const

Finds the intersection between two planes.

**Parameters:**

***result*** Pointer to a line which should be overwritten with the line of intersection. If NULL, it will not be overwritten.

***other*** The plane with which to test for intersection.

**Returns:**

True if a line of intersection was found, or false if the planes did not intersect or if they were equal.

#### 6.41.2.4 bool GAUGE3D::GPlane::FindIntersection (GVector ∗ *result*, const GPlane & *other1*, GPlane & *other2*) const

Finds the intersection between three planes.

**Parameters:**

***result*** Pointer to a point which should be overwritten with the point of intersection. If NULL, it will not be overwritten.

***other1*** One of the two planes with which to test for intersection.

***other2*** The other of the two planes with which to test for intersection.

**Returns:**

True if a point of intersection was found, or false if the planes did not intersect or if two or more of them were equal.

#### 6.41.2.5 GCoordinate GAUGE3D::GPlane::FindSquareDistance (const GVector & *point*) const

Finds the square of the shortest distance between the plane and the point (faster).

This is faster because the square root of the result does not need to be found.

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

- gauge3d/3dmath/plane.h

## 6.42 GAUGE3D::GPlugin Class Reference

Base class for objects that tell GAUGE what a plugin can do.

`#include <gauge3d/plugins/plugin.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- virtual ~**GPlugin** ()
- virtual GString Name ()=0

  *Returns the human-readable name of the plugin.*

- virtual GString Description ()=0

  *Returns a human-readable description of the plugin.*

- virtual bool InUse ()=0

  *Returns true if the plugin is in use.*

- virtual pGObject CreateObject (const GString &className)=0

  *Creates an instance of the class with the given name.*

- virtual pGObject CreateObject (const GString &className,pGFile file)=0

  *Creates an object of the class with the given name using the given file.*

- virtual GArray<ObjectInfo> ObjectList ()=0

  *Gets ObjectInfos for every object available from this plugin.*

### 6.42.1 Detailed Description

Base class for objects that tell GAUGE what a plugin can do.

You only need to know about this class if you are writing a plugin, not if you are only using them.

Derivatives of GPlugin are used to describe what sort of objects a plugin contains, and create those objects for the object loader.

Here is what happens when a plugin is opened:

1. GObjectLoader uses GLibrary to load a shared library on disk.
2. The shared library exports an object derived from GPlugin. GObjectLoader imports this object.
3. GObjectLoader calls GPlugin::ObjectList() on the imported plugin object to find out what other types of objects the plugin contains.
4. Later on, when an object is requested, GObjectLoader finds a plugin that contains a suitable object. It calls one of this plugin's GPlugin::CreateObject() functions to create the object.
5. The plugin creates the object and returns a pointer to it.

## 6.42.2 Member Function Documentation

### 6.42.2.1 bool GAUGE3D::GPlugin::InUse () `[pure virtual]`

Returns true if the plugin is in use.

**Remarks:**
GObjectLoader uses this to determine when it is safe to unload the library from RAM. It should return true as long as an instance of any class in the plugin exists.

### 6.42.2.2 pGObject GAUGE3D::GPlugin::CreateObject (const GString & *className*, pGFile *file*) `[pure virtual]`

Creates an object of the class with the given name using the given file.

For example, if the class loads jpeg images, the file would be a jpeg image to load.

**Returns:**
A pointer to the created object, or NULL if the file could not be loaded for some reason.

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

- gauge3d/plugins/plugin.h

## 6.43 GAUGE3D::GPolygon Class Reference

Represents a polygon with an arbitrary number of sides.

`#include <gauge3d/3dmath/polygon.h>`

### Public Methods

- GPolygon ()

  *Creates a polygon with no sides.*

- GPolygon (const GTriangle &triangle)

  *Creates a polygon equivalent to the given triangle.*

- GPolygon (const GPolygon &other)

  *Copy constructor.*

- GPolygon (GVector vertices[ ],int num)

  *Creates a polygon with the given vertices.*

- ∼GPolygon ()

  *Destructor.*

- void GetPlane (GPlane ∗plane)const

  *Gets the plane which the polygon is on.*

- void Resize (int num,bool copy)

  *Change the number of vertices.*

- void Split (GTriangle triangles[ ])const

  *Splits the polygon into triangles.*

- void Reverse ()

  *Reverses the direction of the vertices.*

- bool Contains (const GVector &point)

  *Returns true if the point is on the polygon.*

- int8 Intersects (const GLine &line)

*Finds whether or not the line intersects the polygon, and from which side if so.*

- int NumVertices ()const

  *The number of vertices in the polygon.*

- GVector& operator[ ] (int index)

  *Array-like access to vertices.*

- const GVector& operator[ ] (int index)const

  *Array-like access to vertices.*

- const GPolygon& operator= (const GPolygon other)

  *Assignment operator.*

## 6.43.1 Detailed Description

Represents a polygon with an arbitrary number of sides.

I recommend against using this class except when other methods are extremely inconvenient. This class uses dynamic memory, which makes it unsuitable for fast math computations.

## 6.43.2 Member Function Documentation

### 6.43.2.1 void GAUGE3D::GPolygon::GetPlane (GPlane ∗ *plane*) const

Gets the plane which the polygon is on.

**Parameters:**

***plane*** Points to the location to which to write the plane.

**Note:**

The plane will face in the direction from which the vertices of the polygon appear clockwise.

#### 6.43.2.2 void GAUGE3D::GPolygon::Resize (int *num*, bool *copy*)

Change the number of vertices.

**Parameters:**

***num*** New number of vertices in the polygon.

***copy*** True if the old vertices should be copied into the new list.

#### 6.43.2.3 void GAUGE3D::GPolygon::Split (GTriangle *triangles*[ ]) const

Splits the polygon into triangles.

**Parameters:**

***triangles*** An array containing at least the number of vertices minus two triangles. These will be overwritten with the triangles that make up the polygon.

#### 6.43.2.4 void GAUGE3D::GPolygon::Reverse ()

Reverses the direction of the vertices.

**Remarks:**

This does not reverse the order of the vertices. It just swaps two of the vertices so that clockwise becomes counter-clockwise. This also means that the direction that the polygon faces will be reversed, since that direction is the direction from which the vertices appear in clockwise order.

#### 6.43.2.5 int8 GAUGE3D::GPolygon::Intersects (const GLine & *line*)

Finds whether or not the line intersects the polygon, and from which side if so.

**Parameters:**

***line*** The line with which to test for intersection.

**Returns:**

0 if there is no intersection, 1 if the line hits the front of the polygon, or -1 if the line hits the back of the polygon. (the front side is the side from which the vertices of the polygon appear clockwise.)

**Remarks:**

> This function is very fast. It is actually *faster* than Contains(), so if you want to find the point of intersection, first call this and then call GPlane::FindIntersection if necessary.

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

- gauge3d/3dmath/polygon.h

## 6.44 GAUGE3D::GQuaternion Class Reference

Represents a quaternion, or an axis-angle rotation.

`#include <gauge3d/3dmath/quaternion.h>`

### Public Methods

- GQuaternion ()

  *Creates an uninitialized quaternion.*

- GQuaternion (const GVector &axis,GCoordinate angle)

  *Creates a quaternion that represents the given axis-angle rotation. The axis must be normalized!*

- void Rotate (const GVector in[ ],GVector out[ ],int num)const

  *Applies the quaternion rotation to a set of vectors.*

- void Rotate (GVector ∗vec)const

  *Rotate a single vector in place.*

- void Interpolate (const GQuaternion &q1,const GQuaternion &q2,GCoordinate t)

  *Spherical linear interpolation between quaternions.*

- GCoordinate Angle ()const

  *Extracts the angle of the quaternion rotation.*

- GVector Axis ()const

  *Extracts the axis of the quaternion rotation.*

- GVector RawAxis ()const

  *Returns the axis without normalizing it. Faster.*

- void Set (const GVector &axis,GCoordinate angle)

  *Sets the quaternion to represent the given axis-angle rotation. The axis must be a unit vector!*

- GQuaternion operator ∗ (const GQuaternion &other)const

  *Combines two quaternion rotations into one.*

- const GQuaternion& operator *= (const GQuaternion &other)

  *Combines two quaternion rotations into one.*

## Public Attributes

### Coordinates

- GCoordinate x

  *imaginary component 1.*

- GCoordinate y

  *imaginary component 2.*

- GCoordinate z

  *imaginary component 3.*

- GCoordinate w

  *real component.*

## 6.44.1 Detailed Description

Represents a quaternion, or an axis-angle rotation.

A quaternion is a complex number with one real component and three imaginary components. But you don't need to worry about that. All you care about is the fact that quaternions are an efficient way to represent axis-angle rotations.

## 6.44.2 Member Function Documentation

### 6.44.2.1 void GAUGE3D::GQuaternion::Rotate (const GVector *in*[ ], GVector *out*[ ], int *num*) const

Applies the quaternion rotation to a set of vectors.

**Parameters:**

***in*** An array of vectors to rotate.

***out*** Location to which to write the rotated vectors. May be the same as in.

***num*** Number of vectors to rotate.

**Remarks:**

Rotating many vectors at once is faster than rotating vectors individually.

#### 6.44.2.2 void GAUGE3D::GQuaternion::Interpolate (const GQuaternion & *q1*, const GQuaternion & *q2*, GCoordinate *t*)

Spherical linear interpolation between quaternions.

Interpolate() interpolates along a sphere between q1 and q2 using time value t, which is between 0.0 (q1), and 1.0 (q2). The results are stored in this. This is useful if you want to find the orientation between two quaternions, or want to animate an object moving between these two orientations.

#### 6.44.2.3 GQuaternion GAUGE3D::GQuaternion::operator * (const GQuaternion & *other*) const

Combines two quaternion rotations into one.

Applying the resulting quaternion to a vector will have the same effect as if you had first applied the second operand of the multiplication to the vector, and then the first operand. This is similar to matrix multiplication.

#### 6.44.2.4 const GQuaternion & GAUGE3D::GQuaternion::operator *= (const GQuaternion & *other*)

Combines two quaternion rotations into one.

Applying the resulting quaternion to a vector will have the same effect as if you had first applied the second operand of the multiplication to the vector, and then the first operand. This is similar to matrix multiplication.

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

- gauge3d/3dmath/quaternion.h

## 6.45 GAUGE3D::GQuitMessage Class Reference

Represents a message asking the program to quit.

`#include <gauge3d/input/message.h>`

Inherits GAUGE3D::GMessage.

### Public Methods

- GQuitMessage (pGObject origin)

  *Create a GQuitMessage with the given origin.*

- virtual ~GQuitMessage ()

  *Destructor.*

### 6.45.1 Detailed Description

Represents a message asking the program to quit.

This message is sent when the user requests that the program exit. It is up to the receiver of the message to actually shut down the program when it receives this message. So, for example, if the user tries to close the program window, the renderer will send a quit message, but the window will not actually be closed until some other part of the engine asks the renderer to close it. So, if the quit message is ignored, the program will stay open.

### 6.45.2 Constructor & Destructor Documentation

#### 6.45.2.1 GAUGE3D::GQuitMessage::GQuitMessage (pGObject *origin*) `[inline]`

Create a GQuitMessage with the given origin.

**Parameters:**

***origin*** A pointer to the object which is the logical ”originator” of the message. For example, most messages will originate from a GDisplay.

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

- gauge3d/input/message.h

## 6.46 GAUGE3D::GRawIStream Class Reference

A source of data.

`#include <gauge3d/files/istream.h>`

Inherits GAUGE3D::GObject.

### Public Types

- enum tSeekType { BEGINNING, CURRENT, END }

  *Position to seek relative to.*

### Public Methods

- virtual ∼**GRawIStream** ()
- virtual void Seek (int pos,int seekType)

  *Seek the stream position.*

- virtual int TellPos ()

  *Get the distance in bytes between the beginning of the stream and the current position.*

- virtual int Read (void ∗buffer,int size)=0

  *Attempts to fill the buffer and returns the amount actually read.*

- virtual bool WaitForInput (tTime timeout=0.0)

  *Wait for input to become available.*

- virtual bool InputAvailable ()

  *Returns true if input is currently available.*

### 6.46.1 Detailed Description

A source of data.

\ingoup files

GRawIStream is the actual source of data read using a GIStream. GIStream itself provides buffering and other services for reading the data more easily. Often, however, it is desireable to read the raw data directory, especially when layering streams (you don't want double buffering).

GRawIStream is the class that should be inherited when creating new sources for stream data. When inheriting from GRawIStream, you do not have to write implementations of all of the functions. For example, if your stream does not support seeking, you do not have to write Seek() or TellPos(). If input is always available for your class, you need not define WaitForInput() or InputAvailable().

## 6.46.2 Member Enumeration Documentation

### 6.46.2.1 enum GAUGE3D::GRawIStream::tSeekType

Position to seek relative to.

**Enumeration values:**

***BEGINNING*** Seek relative to the start of the stream.

***CURRENT*** Seek relative to the current position.

***END*** Seek relative to the end of the stream.

## 6.46.3 Member Function Documentation

### 6.46.3.1 void GAUGE3D::GRawIStream::Seek (int *pos*, int *seekType*) `[virtual]`

Seek the stream position.

**Parameters:**

***pos*** Position to seek to, in bytes.

***seekType*** Point in the stream that pos is relative to.

### 6.46.3.2 bool GAUGE3D::GRawIStream::WaitForInput (tTime *timeout* = 0.0) `[virtual]`

Wait for input to become available.

**Parameters:**

***timout*** Maximum amount of time to wait for input, or 0 to wait indefinately.

**Returns:**

True if input became available, false otherwise.

**Note:**

This function may return false immediately if it knows that input will not be available in the future. For example, if the stream is a file and it is at the end of the file.

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

- gauge3d/files/istream.h

## 6.47 GAUGE3D::GRawOStream Class Reference

A destination for data.

`#include <gauge3d/files/ostream.h>`

Inherits GAUGE3D::GObject.

### Public Types

- enum tSeekType { BEGINNING, CURRENT, END }

  *Position to seek relative to.*

### Public Methods

- virtual ∼**GRawOStream** ()
- virtual void Seek (int pos,int seekType)

  *Seek the stream position.*

- virtual int TellPos ()

  *Get the distance in bytes between the beginning of the stream and the current position.*

- virtual int Write (const void ∗buffer,int size)=0

  *Attempts to write the contents of buffer and returns the amount actually written.*

### 6.47.1 Detailed Description

A destination for data.

\ingoup files

GRawOStream is the actual destination of data written using a GOStream. GOStream itself provides buffering and other services for writing the data more easily. Often, however, it is desireable to write the raw data directory, especially when layering streams (you don't want double buffering).

GRawOStream is the class that should be inherited when creating new destinations for stream data. When inheriting from GRawOStream, you do not have to write implementations of all of the functions. If your stream does not support seeking, you do not have to write Seek() or TellPos().

## 6.47.2 Member Enumeration Documentation

### 6.47.2.1 enum GAUGE3D::GRawOStream::tSeekType

Position to seek relative to.

**Enumeration values:**

***BEGINNING*** Seek relative to the start of the stream.

***CURRENT*** Seek relative to the current position.

***END*** Seek relative to the end of the stream.

## 6.47.3 Member Function Documentation

### 6.47.3.1 void GAUGE3D::GRawOStream::Seek (int *pos*, int *seekType*) `[virtual]`

Seek the stream position.

**Parameters:**

***pos*** Position to seek to, in bytes.

***seekType*** Point in the stream that pos is relative to.

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

- gauge3d/files/ostream.h

## 6.48 GAUGE3D::GRenderer Class Reference

Represents a hardware 3D rendering device.

`#include <gauge3d/renderer/renderer.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- virtual ~**GRenderer** ()
- virtual pGDisplay NewDisplay ()=0

  *Create a new GDisplay associated with this renderer.*
- virtual pGTexture NewTexture ()=0

  *Create a new GTexture associated with this renderer.*
- virtual pGScene NewScene ()=0

  *Create a new GScene associated with this renderer.*

### 6.48.1 Detailed Description

Represents a hardware 3D rendering device.

This class is the one that is exported by rendering plugins. Given a GRenderer, you can do anything graphics-wise. GRenderer does not do anything itself, but it acts as a vendor for other rendering classes which you can use to get the real work done.

### 6.48.2 Member Function Documentation

#### 6.48.2.1 pGDisplay GAUGE3D::GRenderer::NewDisplay () `[pure virtual]`

Create a new GDisplay associated with this renderer.

**Remarks:**

> Not all rendering API's (or hardware) support creating more than one display. Be prepared for NewDisplay() to return NULL if you have already created a display with this renderer.

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

- gauge3d/renderer/renderer.h

## 6.49 GAUGE3D::GModel::RenderInfo Class Reference

Provides information required to render a model.

`#include <gauge3d/graphics/model.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- virtual ~**RenderInfo** ()
- virtual int StartFrame (pAnimState animState,const GVector &viewPoint,const GQuaternion &viewAngle,const GFrustum &viewFrustum,float geoLod)=0

  *Sets everything up to render a frame with the given parameters.*
- virtual void GetStaticInfo (RenderSet ∗target,int setNum)=0

  *Fills in the static parts of the render set.*
- virtual void GetDynamicInfo (RenderSet ∗target,int setNum)=0

  *Fills in the dynamic parts of the render set.*

### 6.49.1 Detailed Description

Provides information required to render a model.

A RenderInfo object is bound to a particular GModel object and is used to request the information necessary to render the model. The renderer will need one RenderInfo for each sprite in the scene which uses a particular model. The idea here is to allow safe multithreaded access to this information. (So, the renderer could be drawing two different sprites that have the same model at the same time.)

### 6.49.2 Member Function Documentation

#### 6.49.2.1 int GAUGE3D::GModel::RenderInfo::StartFrame (pAnimState *animState*, const GVector & *viewPoint*, const GQuaternion & *viewAngle*, const GFrustum & *viewFrustum*, float *geoLod*) `[pure virtual]`

Sets everything up to render a frame with the given parameters.

**Parameters:**

***animState*** The animation state of the model.

***viewPoint*** The position of the camera.

***geoLod*** The geometric level of detail in which to render the model. This is simply the area that an average triangle in the model should have, if it were one meter (one unit) from the camera. This value is ∗not∗ adjusted for distance from the viewpoint, so the model will have to do that internally. (hint: multiply by viewPoint.SquareMagnitude().) Note that a lower geoLOD value means a higher detail image (smaller triangles).

**Returns:**

The number of RenderSet structures needed to complete the model. The renderer will then call GetStaticInfo and GetDynamicInfo once for each RenderSet.

#### 6.49.2.2 void GAUGE3D::GModel::RenderInfo::GetStaticInfo (RenderSet ∗ *target*, int *setNum*) `[pure virtual]`

Fills in the static parts of the render set.

**Parameters:**

***target*** The RenderSet to which the info will be written.

***setNum*** The number of the RenderSet which is to be filled in.

The renderer will use the values of the mDynamicFlags, mNumVertices, and mNumIndices fields of *target* to allocate buffers to store the dynamically generated parts of the model. Then, GetDynamicInfo will be called to fill in those buffers with the model data.

#### 6.49.2.3 void GAUGE3D::GModel::RenderInfo::GetDynamicInfo (RenderSet ∗ *target*, int *setNum*) `[pure virtual]`

Fills in the dynamic parts of the render set.

**Parameters:**

***target*** The RenderSet to which the info will be written.

***setNum*** The number of the RenderSet which is to be filled in.

Before calling this, the renderer will fill in *target* with pointers to buffers where the dynamic data for the model should be written. For example, if target->mDynamicFlags was set to VERTEX_POSITION by GetStaticInfo, then when GetDynamicInfo is called, target->mVertices and target->mNormals will point to arrays of vertices allocated by the renderer, ready to be filled in with vertex data. This way, the renderer can efficiently handle allocating memory for dynamic model data.

**Remarks:**

> When writing to the buffers which the renderer has allocated, a model plugin should write linearly and should not attempt to read from the buffers. This is because the buffers may be allocated in AGP RAM or even VRAM, and are therefore uncached. This means that the CPU will be able to write much more efficiently if it can combine several writes into one.

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

- gauge3d/graphics/model.h

## 6.50 GAUGE3D::GModel::RenderInfo::RenderSet Struct Reference

Contains all of the data needed to render a model.

`#include <gauge3d/graphics/model.h>`

### Public Types

- enum tDynamicFlags { STATIC = 0, VERTEX_POSITION = 1 << 0, VERTEX_TEXCOORD = 1 << 1, INDEX_LIST = 1 << 16 }

  *Specifies which properties of the model are dynamic.*

- enum tIndexType { POINTS, LINES, TRIANGLES }

  *Specifies how to make geometry out of the indices.*

### Public Methods

- void SetDefaults ()

  *Called by the renderer to reset the RenderSet.*

### Public Attributes

- int mDynamicFlags

  *One or more of tDynamicFlags.*

- int mNumVertices

  *Number of elements in each of the arrays.*

- GVector∗ mVertices

  *Array of vertex positions.*

- GVector∗ mNormals

  *Array of normal vectors. Must be unit vectors!*

- GVector∗ mTexCoords

*Array of texture coordinates. Can be NULL if texturing is not supported.*

- int mIndexType

  *One of the values in tIndexType.*

- int mNumIndices

  *The number of elements in the index array.*

- uint16∗ mIndices

  *Pointer to an array of indices.*

## 6.50.1 Detailed Description

Contains all of the data needed to render a model.

A RenderSet contains a list of all of the vertices in a GModel. For each vertex, it gives the position, surface normal, and texture coordinates. These lists are all strided arrays and can be interleaved. Interleaving your arrays can increase performance.

The RenderSet also contains a list of indices forming triangles, lines, or points.

## 6.50.2 Member Enumeration Documentation

### 6.50.2.1 enum GAUGE3D::GModel::RenderInfo::RenderSet::tDynamicFlags

Specifies which properties of the model are dynamic.

**Remarks:**

Typically, vertex positions and normals are dynamic (if anything).

**Enumeration values:**

***STATIC*** Model is not dynamic.

***VERTEX_POSITION*** Vertex positions (and normals) are dynamic.

***VERTEX_TEXCOORD*** Vertex texture coordinates are dynamic.

***INDEX_LIST*** Index list is dynamic.

#### 6.50.2.2 enum GAUGE3D::GModel::RenderInfo::RenderSet::tIndexType

Specifies how to make geometry out of the indices.

**Enumeration values:**

***POINTS*** Every indice is a point / particle.

***LINES*** Every two indices make one line segment.

***TRIANGLES*** Every three indices make one triangle.

### 6.50.3 Member Function Documentation

#### 6.50.3.1 void GAUGE3D::GModel::RenderInfo::RenderSet::SetDefaults () `[inline]`

Called by the renderer to reset the RenderSet.

**Remarks:**

> The idea is that older model plugins may not support newer RenderSet data fields. In order to make sure the renderer does not read uninitialized data from those fields, they are set to some default value. This is all for binary and source code backwards compatibility, but also makes writing model plugins easier.

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

- gauge3d/graphics/model.h

## 6.51 GAUGE3D::GScene Class Reference

Represents a 3D scene.

`#include <gauge3d/renderer/scene.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- virtual ~**GScene** ()
- virtual pGSprite NewSprite ()=0

  *Create a new GSprite inside the scene.*

- virtual pGLight NewLight ()=0

  *Create a new GLight inside the scene.*

- virtual pGCamera NewCamera ()=0

  *Create a new GCamera looking into the scene.*

### 6.51.1 Detailed Description

Represents a 3D scene.

A scene is basically a collection of sprites and lights within a world map. You can create multiple cameras looking in on a scene.

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

- gauge3d/renderer/scene.h

## 6.52 GAUGE3D::GScheduler Class Reference

`#include <gauge3d/scheduler.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- GScheduler ()

  *Constructor.*

- ~GScheduler ()

  *Destructor.*

- void CallEveryFrame (GObject ∗object,int priority=0)

  *Tell the scheduler to call object->DoFrame() every frame.*

- void StopCallingEveryFrame (GObject ∗object)

  *Remove the object from the list of objects to call every frame.*

- int GetObjectPriority (GObject ∗object)

  *Get the priority of an object on the call every frame list.*

- void SetObjectPriority (GObject ∗object,int priority)

  *Change the priority of an object on the call every frame list.*

- virtual void DoFrame (tTime time,tTime frametime)

  *Call the DoFrame() functions for all objects which requested it.*

- tTime Time ()

  *Get the current time.*

- tTime FrameTime ()

  *Get the amount of time spent processing the previous frame.*

## 6.52.1 Detailed Description

\breif Schedules events.

GScheduler is used to synchronize events throughout the engine. Each subsystem has a scheduler for events which relate to that subsystem. Many classes which are part of that subsystem will register themselves with the scheduler. Then, the scheduler will call those objects' DoFrame() functions every frame.

For example, when you create a GDisplay and give it a GCamera to render from, you do not have to tell it when to render each frame. This is because the GDisplay asks gVideoScheduler to call it every frame. When it gets called back by gVideoScheduler, the GDisplay renders the scene.

## 6.52.2 Member Function Documentation

### 6.52.2.1 void GAUGE3D::GScheduler::CallEveryFrame (GObject ∗ *object*, int *priority* = 0)

Tell the scheduler to call object->DoFrame() every frame.

**Parameters:**

***priority*** This value can optionally be used to tell the scheduler in what order to call objects. Objects with lower priorities will be called earlier in the frame.

**Remarks:**

Typically, this function will be called by the constructor of an object, and StopCallingEveryFrame() will be called by the destructor.

### 6.52.2.2 void GAUGE3D::GScheduler::DoFrame (tTime *time*, tTime *frametime*) `[virtual]`

Call the DoFrame() functions for all objects which requested it.

**Parameters:**

***time*** The current time.

***frametime*** The amount of time spent processing the previous frame.

This function is normally called by GEngine::MainLoop().

Reimplemented from GAUGE3D::GObject.

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

- gauge3d/scheduler.h

## 6.53 GAUGE3D::GSegment Class Reference

Represents a line segment.

`#include <gauge3d/3dmath/segment.h>`

### Public Methods

- GSegment ()

  *Creates an uninitialized segment.*

- GSegment (const GVector &endPoint1,const GVector &endPoint2)

  *Creates a segment with the specified endpoints.*

- void GetLine (GLine ∗line)const

  *Gets the line which the segment is on.*

- bool Contains (const GVector &point)

  *Returns true if the point is on the segment.*

- GVector& operator[ ] (int index)

  *Array-like access to endpoints.*

- const GVector& operator[ ] (int index)const

  *Array-like access to endpoints.*

### 6.53.1 Detailed Description

Represents a line segment.

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

- gauge3d/3dmath/segment.h

## 6.54 GAUGE3D::GSmartPointer Class Template Reference

A reference-counting smart pointer class.

`#include <gauge3d/object.h>`

Inherits GAUGE3D::GSmartPointerBase.

### Public Methods

- tType& operator[ ] (int index)const
  *Array access for smart pointers that point to arrays.*
- operator tType ∗ ()const
  *Cast to standard pointer.*

#### Constructors and Destructors

- GSmartPointer ()
  *Sets pointer to NULL.*
- GSmartPointer (tType ∗object)
  *Cast from regular pointer.*
- GSmartPointer (const GSmartPointer &other)
  *Copy constructor.*
- GSmartPointer (const GSmartPointerBase &other)
  *Cast constructor from smart pointers for other types.*
- ∼GSmartPointer ()
  *Decrement reference count and delete if zero.*

#### Assignment Operators

*These work like you'd expect them to.*

- const GSmartPointer& **operator=** (const GSmartPointer &other)
- const GSmartPointer& **operator=** (const GSmartPointerBase &other)
- const GSmartPointer& **operator=** (tType ∗other)

**Equality Tests**

*These work like you'd expect them to.*

- bool **operator==** (const GSmartPointer &other)const
- bool **operator==** (const GSmartPointerBase &other)const
- bool **operator==** (const tType ∗other)const

**Dereferencing**

*These work like you'd expect them to.*

- tType& **operator** ∗ ()
- const tType& **operator** ∗ ()const
- tType∗ **operator** → ()
- const tType∗ **operator** → ()const

### 6.54.1 Detailed Description

**template<class tType> class GAUGE3D::GSmartPointer**

A reference-counting smart pointer class.

GSmartPointer is a reference counting smart pointer class. It can point to any class derived from GObject. For each smart pointer pointing to an object, the object's reference count will be incremented. When the smart pointer is destroyed, the reference count will be decremented. When the count hits zero, the object will be deleted.

By using smart pointers, you can forget about who has ownership of an object and should therefore delete it when done. The object will be deleted automatically when it is done with.

In GAUGE, every class that is derived from GObject has an associated typedef'd smartpointer type. The smartpointer type is named after the object with the prefix 'p'. So, pGObject is a smart pointer to a GObject, and pGFile is a smart pointer to a GFile, etc.

Some important things to remember when using smart pointers:

- Avoid circular links. This occurs when two objects contain smart pointers to each other. In this case, the objects will not be removed automatically as they will always have reference counts of one or more.
- Smart pointers are meant to be passed by *value*, not by address or reference. Furthermore, smart pointers should always be created either on the stack or as members of larger structures rather than be created wich new. In general, you should never find yourself dealing with either a pointer or a reference to a smart pointer unless you know what you are doing.

### 6.54.2 Constructor & Destructor Documentation

#### 6.54.2.1 template<class tType> GAUGE3D::GSmartPointer<tType>::GSmartPointer<tType> (tType * *object*) `[inline]`

Cast from regular pointer.

WARNING: This can be dangerous. If you have both regular pointers and smart pointers pointing to the same object, and all the smart pointers are destroyed, the regular pointers will be left dangling, and could cause a program crash. Try to avoid using both regular and smart pointers to an object.

#### 6.54.2.2 template<class tType> GAUGE3D::GSmartPointer<tType>::GSmartPointer<tType> (const GSmartPointerBase & *other*) `[inline]`

Cast constructor from smart pointers for other types.

Note that type checking is not performed. If you cast to a type which the object isn't, you will not be warned. If you need to do dynamic casting, you must do it manually.

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

- gauge3d/common/memory.h

## 6.55 GAUGE3D::GSphereModel Class Reference

A sphere model.

`#include <gauge3d/graphics/spheremodel.h>`

Inherits GAUGE3D::GModel.

### Public Types

- typedef GSmartPointer<SphereModelAnimState> **pSphereModelAnimState**

### Public Methods

- **GSphereModel** ()
- virtual ~**GSphereModel** ()
- virtual pAnimState CreateAnimState ()

  *Generates an AnimState object associated with this model.*

- pRenderInfo CreateRenderInfo ()

  *Create a RenderInfo bound to this model.*

### Friends

- class **SphereModelAnimState**

### 6.55.1 Detailed Description

A sphere model.

GSphereModel is a derivative of GModel. You can use it to display spheres. The sphere is of the geodesic type, which makes it look pretty.

Note that you cannot set the radius of the sphere; it is always 1.0. You can, however, give GSprite a scale factor, which will cause the sphere to be drawn with a radius equal to that factor.

## 6.55.2 Member Function Documentation

### 6.55.2.1 pRenderInfo GAUGE3D::GSphereModel::CreateRenderInfo () `[virtual]`

Create a RenderInfo bound to this model.

**Remarks:**

> Obviously, this function uses dynamic memory and should not be called every frame. A renderer should call this once when the model is assigned to a sprite and then use the same RenderInfo object every time the sprite is drawn.

Reimplemented from GAUGE3D::GModel.

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

- gauge3d/graphics/spheremodel.h

## 6.56 GAUGE3D::GSphereModel::SphereModelAnimState Class Reference

AnimState for a GSphereModel.

`#include <gauge3d/graphics/spheremodel.h>`

Inherits GAUGE3D::GModel::AnimState.

### Public Methods

- virtual ~**SphereModelAnimState** ()

### Friends

- class **GSphereModel**

### 6.56.1 Detailed Description

AnimState for a GSphereModel.

There is absolutely nothing special about this class. It doesn't actually do anything at all. Spheres are not animated. They are static models.

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

- gauge3d/graphics/spheremodel.h

## 6.57 GAUGE3D::GSprite Class Reference

Represents an object in the 3D scene.

`#include <gauge3d/renderer/sprite.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- virtual ~**GSprite** ()
- virtual void Show ()=0

  *Tells the renderer to start drawing this object whenever it is visible.*
- virtual void Hide ()=0

  *Tells the renderer not to draw this object at all.*
- virtual void Remove ()=0

  *Permenantly removes the sprite from the scene.*
- virtual void Model (pGModel model)=0

  *Sets the sprite's GModel.*
- virtual void AnimState (GModel::pAnimState anim)=0

  *Sets the animation state for the sprite's model.*
- virtual void Texture (pGTexture texture)=0

  *Sets the texture to apply to the model.*
- virtual void Position (const GVector &position)=0

  *Sets the position of the sprite.*
- virtual void Angle (const GQuaternion &angle)=0

  *Sets the direction that the sprite is facing.*
- virtual void Color (float red,float green,float blue)=0

  *Set the color of the sprite. (alpha unchanged).*
- virtual void Color (float red,float green,float blue,float alpha)=0

  *Set the color and alpha of the sprite.*

- virtual void Color3v (const float color[ ])=0

  *Set the color of the sprite from a three-element array. (alpha unchanged).*

- virtual void Color4v (const float color[ ])=0

  *Set the color and alpha of the sprite from a four-element array.*

- virtual void Alpha (float alpha)=0

  *Set the alpha of the sprite. (color unchanged).*

- virtual void LightProps (float specular,float shininess,float emission)=0

  *Set the reflectivity properties of the sprite.*

- virtual void FullBright (bool fullBright)=0

  *Set true to ignore lighting and render the sprite at maximum brightness.*

- virtual void Scale (GCoordinate magnification)=0

  *Sets a uniform scale factor for the sprite.*

### 6.57.1 Detailed Description

Represents an object in the 3D scene.

A sprite is any independently moving object in the scene. In GAUGE, a ”sprite” is not necessarily a ”billboard object”, although it could be. A sprite can have any geometry which can be described by a GModel.

### 6.57.2 Member Function Documentation

#### 6.57.2.1 void GAUGE3D::GSprite::Angle (const GQuaternion & *angle*) `[pure virtual]`

Sets the direction that the sprite is facing.

**Remarks:**

Don’t forget that you can convert a GAngles into a GQuaternion.

#### 6.57.2.2 void GAUGE3D::GSprite::Color (float *red*, float *green*, float *blue*) `[pure virtual]`

Set the color of the sprite. (alpha unchanged).

**Remarks:**

The color is multiplied component-wise with the texture color. If you don't want any color other than the texture color to be applied, just set the color to 1.0, 1.0, 1.0, 1.0 (full white).

#### 6.57.2.3 void GAUGE3D::GSprite::LightProps (float *specular*, float *shininess*, float *emission*) `[pure virtual]`

Set the reflectivity properties of the sprite.

**Parameters:**

***specular*** Amount of specular light to reflect.

***shininess*** How focused teh specular highlight is.

***emission*** The amount of light emitted from the sprite.

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

- gauge3d/renderer/sprite.h

## 6.58 GAUGE3D::GStridedArray Class Template Reference

Represents a strided or interleaved array.

`#include <gauge3d/array.h>`

### Public Methods

- GStridedArray ()

  *Create NULL array.*

- GStridedArray (tType ∗ptr,int stride)

  *Create array that starts at ptr with given stride.*

- GStridedArray (tType ∗ptr)

  *Create array using non-strided data.*

- const tType& operator[ ] (int index)const

  *Array access.*

- tType& operator[ ] (int index)

  *Array access.*

- int Stride ()

  *Get the stride.*

- operator tType ∗ ()const

  *Cast to type.*

- bool operator== (const void ∗other)

  *Test for equality with pointer.*

### 6.58.1 Detailed Description

**template<class tType> class GAUGE3D::GStridedArray**

Represents a strided or interleaved array.

This class allows you to create an array with space between the elements. This is useful for interleaved data, where you have two types of interleaved into one big array.

The stride of the array is the distance in bytes between the beginning of one array element and the beginning of the next. For example, say you wanted to interleave a set of vertex positions and normals. So, you have an array: GVector∗ data = new GVector[numVertices ∗ 2]; In that array, the even-numbered elements are vertex positions while the odd-numbered elements are vertex normals (starting at zero). Then, you could create two strided arrays like so: GStridedArray<GVector> vertices(data, sizeof(GVector)∗2); GStridedArray<GVector> normals(data + 1, sizeof(GVector)∗2); Then, you could access ”vertices” and ”normals” as if they were regular arrays.

Note that GStridedArray is NOT reference-counted, unlike GArray.

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

- gauge3d/common/array.h

## 6.59 GAUGE3D::GString Class Reference

A unicode-capable string class.

`#include <gauge3d/gstring.h>`

### Public Methods

- GString ()

  *Create an empty (zero-length) string.*

- GString (const GString &other)

  *Copy constructor.*

- GString (const char cstr[ ])

  *Construct from ASCII character array.*

- GString (const wchar_t cstr[ ])

  *Construct from Unicode character array.*

- GString (const char cstr[ ],int length)

  *Construct from ASCII character array of given length (in characters).*

- GString (const wchar_t cstr[ ],int length)

  *Construct from Unicode character array of given length (in wide characters).*

- GString (int i)

  *Create a string representation of an int (base-10).*

- GString (double f)

  *Create a string representation of a float (base-10).*

- **~GString** ()
- int Compare (const GString &other)const

  *Compare one string to another.*

- int Compare (const char other[ ])const

  *Compare to a null-terminated char string.*

- int Compare (const wchar_t other[ ])const

  *Compare to a null-terminated wide char string.*

- int FindFirstOf (wchar_t c)const

  *Returns the index of the first instance of the given character in the string.*

- int FindLastOf (wchar_t c)const

  *Returns the index of the last instance of the given character in the string.*

- int FindNext (wchar_t c,int start)const

  *Returns the index of the first instance of the character occurring after the start position.*

- GString SubString (int start)const

  *Returns a sub string of the string.*

- GString SubString (int start,int length)const

  *Returns a sub string of the string.*

- int Length ()const

  *Gets the length of the string in characters (or wide characters).*

- bool IsUnicode ()const

  *Returns true if the string is represented as Unicode internally.*

- GString ToUnicode ()const

  *Returns the same string, except with a Unicode internal representation.*

- GString ToAscii ()const

  *Returns the same string, except with an ASCII internal representation.*

- const char* AsciiData ()const

  *Returns a pointer to the internal buffer of the string.*

- const wchar_t* UnicodeData ()const

  *Returns a pointer to the internal buffer of the string.*

- void SetChar (int index,wchar_t value)

  *Set the given character in the string to the given value.*

- operator int ()

  *Interpret string as an integer and convert.*

- operator float ()

  *Interpret string as a floating point value and convert.*

- wchar_t operator[ ] (int index)const

  *Array-like access to the characters in the string.*

- const GString& operator= (const GString &other)

  *Assignment operator.*

**String concatination operators.**

- GString **operator+** (const GString &other)const
- const GString& **operator+=** (const GString &other)
- GString **operator+** (char other)const
- const GString& **operator+=** (char other)
- GString **operator+** (char other[ ])const
- const GString& **operator+=** (char other[ ])
- GString **operator+** (wchar_t other[ ])const
- const GString& **operator+=** (wchar_t other[ ])

**String comparison operators.**

- bool **operator==** (const GString &other)const
- bool **operator!=** (const GString &other)const
- bool **operator<** (const GString &other)const
- bool **operator>** (const GString &other)const
- bool **operator<=** (const GString &other)const
- bool **operator>=** (const GString &other)const
- bool **operator==** (const char other[ ])const
- bool **operator!=** (const char other[ ])const
- bool **operator<** (const char other[ ])const
- bool **operator>** (const char other[ ])const
- bool **operator<=** (const char other[ ])const
- bool **operator>=** (const char other[ ])const
- bool **operator==** (const wchar_t other[ ])const
- bool **operator!=** (const wchar_t other[ ])const
- bool **operator<** (const wchar_t other[ ])const
- bool **operator>** (const wchar_t other[ ])const
- bool **operator<=** (const wchar_t other[ ])const
- bool **operator>=** (const wchar_t other[ ])const

## Related Functions

(Note that these are not member functions.)

- GString **operator+** (const char ∗str1,const GString &str2)
- GString **operator+** (const wchar_t str1[ ],const GString &str2)

## 6.59.1 Detailed Description

A unicode-capable string class.

GString can represent both ASCII and Unicode strings and it can convert between them. The internal buffer is reference-counted, so it is perfectly performance-safe to pass strings by value. When a write operation on the string occurs, the internal buffer will be copied if and only if it does not have a reference count of one.

## 6.59.2 Member Function Documentation

### 6.59.2.1 int GAUGE3D::GString::Compare (const GString & *other*) const

Compare one string to another.

**Returns:**

Negative if the string is less than *other*, positive if it is greater than *other*, or zero if the strings are equal.

**Remarks:**

This function basically just calls strcmp (or wcscmp for Unicode) and returns the result.

### 6.59.2.2 GString GAUGE3D::GString::SubString (int *start*) const

Returns a sub string of the string.

**Parameters:**

***start*** The index of the character at which to start the substring.

**Returns:**

The substring, or an empty string if *start* was out-of-range.

### 6.59.2.3 GString GAUGE3D::GString::SubString (int *start*, int *length*) const

Returns a sub string of the string.

**Parameters:**

***start*** The index of the character at which to start the substring.

***length*** The number of characters to place in the substring.

**Returns:**

The substring, or an empty string if *start* was out-of-range.

### 6.59.2.4 bool GAUGE3D::GString::IsUnicode () const

Returns true if the string is represented as Unicode internally.

**Remarks:**

Internally, the string could be represented in either ASCII or Unicode. Do not assume to know what a string is without testing it.

### 6.59.2.5 GString GAUGE3D::GString::ToUnicode () const

Returns the same string, except with a Unicode internal representation.

**Remarks:**

You only need to use this if you are planning to call UnicodeData() and the string is not already in Unicode format. Note that if the string *is* already in Unicode format, then this function will return the same string, so calling IsUnicode() first is not faster.

### 6.59.2.6 GString GAUGE3D::GString::ToAscii () const

Returns the same string, except with an ASCII internal representation.

**Remarks:**

You only need to use this if you are planning to call AsciiData() and the string is not already in ASCII format. Note that if the string *is* already in ASCII format, then this function will return the same string, so calling IsUnicode() first is not faster.

**Note:**

If any non-ASCII characters are part of the string, they will be corrupted by this function.

### 6.59.2.7 const char ∗ GAUGE3D::GString::AsciiData () const

Returns a pointer to the internal buffer of the string.

**Returns:**

A pointer to a null-terminated ASCII character array, or NULL if the string is not represented in ASCII internally.

**Remarks:**

Avoid doing operations on raw character arrays. You should try to use GString's member functions to manipulate the string. If you must operate on char arrays directly, remember to write versions of the operation for both ASCII **and** Unicode!

### 6.59.2.8 const wchar_t ∗ GAUGE3D::GString::UnicodeData () const

Returns a pointer to the internal buffer of the string.

**Returns:**

A pointer to a null-terminated Unicode character array, or NULL if the string is not represented in Unicode internally.

**Remarks:**

Avoid doing operations on raw character arrays. You should try to use GString's member functions to manipulate the string. If you must operate on char arrays directly, remember to write versions of the operation for both ASCII **and** Unicode!

### 6.59.2.9 void GAUGE3D::GString::SetChar (int *index*, wchar_t *value*)

Set the given character in the string to the given value.

**Note:**

This function is relatively slow (compared to real array access).

### 6.59.2.10 wchar_t GAUGE3D::GString::operator[ ] (int *index*) const

Array-like access to the characters in the string.

**Remarks:**

Remember, this function does not return a reference to the character, so you cannot assign to it. Use SetChar() to change a character if you must.

**Note:**

This function is relatively slow (compared to real array access).

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

- gauge3d/common/gstring.h

## 6.60 GAUGE3D::GStringFile Class Reference

File represented by a string.

`#include <gauge3d/files/stringfile.h>`

Inherits GAUGE3D::GFile.

### Public Methods

- GStringFile ()

  *Construct an empty file.*

- GStringFile (GString data)

  *Construct a GStringFile with the given contents.*

- virtual ~GStringFile ()

  *Destructor.*

- virtual int Size ()

  *Gets the size of the file in bytes.*

- virtual bool Truncate ()

  *Truncates the file to zero length. Returns false if file was not writable.*

- virtual pGRawIStream GetRawInputStream ()

  *Get a GRawIStream for the file. May be null if the file is not readable.*

- virtual pGRawOStream GetRawOutputStream ()

  *Get a GRawOStream for the file. May be null if the file is not writeable.*

- virtual pFileMap Map (int offset,int size,int flags)

  *Maps a portion of the file to memory.*

- virtual GString MapToDisk ()

  *Copies the file to a temporary file on disk, if it is not already on the disk.*

- virtual GString Name ()

  *Get the name of the file.*

- void Name (GString name)

  *Set the internal name of the file.*

- void Data (GString data)

  *Set the data contained by the file.*

- virtual operator GString ()

  *Create a string containing the complete file.*

## 6.60.1 Detailed Description

File represented by a string.

The data in a GStringFile is stored as a GString. The idea here is that we want a file which can easily be cast to a string. This is used heavily by GConfigFile, which loads config files into a virtual filesystem where directories are sections of the config file and files are individual settings.

## 6.60.2 Member Function Documentation

### 6.60.2.1 virtual pFileMap GAUGE3D::GStringFile::Map (int *offset*, int *size*, int *flags*) `[virtual]`

Maps a portion of the file to memory.

**Parameters:**

***offset*** Where to start the map relative to the beginning of the file.

***size*** How many bytes to map to memory.

***flags*** One or more of tMapFlags, bitwise or'd.

**Returns:**

A pointer to a FileMap class describing the map, or NULL if the file could not be mapped.

Reimplemented from GAUGE3D::GFile.

#### 6.60.2.2 virtual GString GAUGE3D::GStringFile::MapToDisk () `[virtual]`

Copies the file to a temporary file on disk, if it is not already on the disk.

**Returns:**
The path and name of the file on disk.

You should not write to the file on disk as other parts of the engine may be reading from it as well. The file on disk will be deleted when the GFile that created it is deleted (unless the file was on the disk to begin with).

**Remarks:**
Do not use this if you can avoid it. It is mainly meant for use with GLibrary, which must load shared libraries from disk due to the limitations of the library loading functions of every operating system I know of.

Reimplemented from GAUGE3D::GFile.

#### 6.60.2.3 GString GAUGE3D::GStringFile::Name () `[inline, virtual]`

Get the name of the file.

**Note:**
This is not always the same as the name of the directory entry that points to the file, especially if the file is in a virtual filesystem.

Reimplemented from GAUGE3D::GFile.

#### 6.60.2.4 GAUGE3D::GStringFile::operator GString () `[inline, virtual]`

Create a string containing the complete file.

**Remarks:**
This function pretty much just calls Map and makes a string out of the results. Do not use this on large files. Note that derived classes may define their own implementation of this, but they are not required to.

Reimplemented from GAUGE3D::GFile.

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

- gauge3d/files/stringfile.h

# 6.61 GAUGE3D::GSuperModel Class Reference

Base class for most normal model formats.

`#include <gauge3d/graphcs/supermodel.h>`

Inherits GAUGE3D::GModel.

## Public Types

- typedef GSmartPointer<SuperAnimState> **pSuperAnimState**

## Public Methods

- **GSuperModel** ()
- virtual ~**GSuperModel** ()
- Model∗ **ModelData** ()
- pAnimState CreateAnimState ()

  *Generates an AnimState object associated with this model.*
- pRenderInfo CreateRenderInfo ()

  *Create a RenderInfo bound to this model.*

## Protected Methods

- void **ModelData** (Model ∗model)

### 6.61.1 Detailed Description

Base class for most normal model formats.

This class is slated to be removed and replaced by a better system in the near future. Thus, I don't really feel like documenting this beast for nothing.

### 6.61.2 Member Function Documentation

#### 6.61.2.1 pRenderInfo GAUGE3D::GSuperModel::CreateRenderInfo () `[virtual]`

Create a RenderInfo bound to this model.

**Remarks:**

> Obviously, this function uses dynamic memory and should not be called every frame. A renderer should call this once when the model is assigned to a sprite and then use the same RenderInfo object every time the sprite is drawn.

Reimplemented from GAUGE3D::GModel.

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

- gauge3d/graphics/supermodel.h

## 6.62 GAUGE3D::GSurface Class Reference

Represents a surface map.

`#include <gauge3d/graphics/surface.h>`

Inherits GAUGE3D::GObject.

### Public Types

- enum tMapType { COLOR, TRANSPARENCY, NORMAL, OPACITY, BINARY_OPACITY, GLOSS, ELEVATION }

  *Describes how to interpret a particular channel of a map.*

### Public Methods

- virtual ~**GSurface** ()
- virtual GArray<SurfaceMap> **SurfaceMaps** ()=0

### 6.62.1 Detailed Description

Represents a surface map.

GSurface represents a surface map, which is like a texture map but maps more than just colors and alpha. A point on a surface map is referenced by only one set of texture coordinates, but the layers of the map do not need to line up (see below).

### 6.62.2 Member Enumeration Documentation

#### 6.62.2.1 enum GAUGE3D::GSurface::tMapType

Describes how to interpret a particular channel of a map.

All maps have between one and four channels. The channels are split into the RGB channel(s) and the alpha channel. Note that the RGB channel does not necessarily represent a color, and the alpha channel is not necessarily an opacity value. They are only named as such for historical reasons.

The channels of a map are broken up as follows:

- **1 channel map:** The one channel is the RGB channel (think grayscale).
- **2 channel map:** First channel is RGB, second channel is alpha.
- **3 channel map:** All three channels go into the RGB component.
- **4 channel map:** The first three channels are the RGB, and the fourth is the alpha.

The RGB and the alpha channels are each assigned a map type from the following values.

**Enumeration values:**

***COLOR*** The color (RGB or grayscale) of the image.

Duh, this is just regular texture mapping. Color can be RGB or grayscale. A surface may contain multiple color maps. They will be multiplied together to get the final color. (hint: Detail maps.)

***TRANSPARENCY*** Transparency map. This is different from alpha.

When objects are viewed through a transparent polygon, the colors are modulated (multiplied) by the transparency map. This looks much nicer than opacity maps (alpha transparency) in some situations, and can be used in combination with an opacity map to give good results in all situations. The transparency map can be RGB color or grayscale.

***NORMAL*** Normal map, used for dot product bump mapping.

A normal map must have three channels (duh). The channels are the X, Y, and Z components of the normal vector to the surface. These are then dot-product multiplied with the light direction to get per-pixel lighting.

***OPACITY*** This is the same thing as alpha mapping.

An opacity map can only have one channel.

***BINARY_OPACITY*** An opacity map where each pixel is either fully opaque or fully transparent.

Each pixel in a binary opacity map is either 1.0 or 0.0. When rendered, the edges between opaque regions and transparent regions will be drawn sharp rather than being bi-linearly interpolated like a regular opacity map.

***GLOSS*** Gives the reflectivity of a surface.

One channel. The specular lighting and environmental reflection effects for each pixel will be multiplied by the gloss map.

***ELEVATION*** The height of each pixel.

This is a single-channel map which gives the height of each pixel. This can be used to draw an elevation map, which is done by drawing multiple layers, and is rather expensive. Note that this has nothing to do with bump mapping.

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

- gauge3d/graphics/surface.h

## 6.63 GAUGE3D::GSurface::SurfaceMap Struct Reference

Describes a map of one or two aspects of a surface.

`#include </gauge3d/graphics/surface.h>`

### Public Attributes

- pGImage mImage

  *The image to use for this map.*
- tMapType mRGBType

  *How to interpret the RGB (or grayscale) channel(s) of the image.*
- tMapType mAlphaType

  *How to interpret the alpha channel of the image.*
- GCoordinate mOffset [2]

  *Offset to apply to texture coordinates.*
- GCoordinate mScale [2]

  *Scale to apply to texture coordinates.*
- GCoordinate mAngle

  *Angle by which to rotate texture coordinates. (radians).*
- GCoordinate mThresholdNear [2]

  *The distance at which to start showing this map. (see above).*
- GCoordinate mThresholdFar [2]

  *The distance at which to stop showing this map. (see above).*

### 6.63.1 Detailed Description

Describes a map of one or two aspects of a surface.

Each surface map can map two aspects of a surface. For example, an often-used surface map is one which maps colors and opacity values across the surface (this is your typical RGBA texture). Other ideas for surface maps include:

- Color and gloss (reflectivity)
- Normals (bump map) and elevation
- Transparency and opacity (they are different things!)
- Any other combination where at least one of the two components is a single-channel map.

Each SurfaceMap in a GSurface is referenced by the same texture coordinates, but the texture coordinates are transformed differently for each map. This allows you to apply maps offset from each other to prevent ugly patters from being visible. The incoming texture coordinates are first rotated by mAngle, then scaled by mScale, then offset by mOffset.

The more surface maps you have, the more rendering passes will be used to render the polygons that use this map.

mThresholdNear and mThresholdFar allow you to control level-of-detail. If the object to which this surface is being applied is further away than mThresholdFar[1] or closer than mThresholdNear[0], the map will not be used. If the object is between mThresholdNear[1] and mThresholdFar[0], the map will be applied as normal. In the area between mThresholdNear[0] and mThresholdNear[1], and in the area between mThresholdFar[0] and mThresholdFar[1], the map will be smoothly blended in and out to eliminate popup. (Note that some renderers may not do anything about popup, or may use other methods to avoid it.)

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

- gauge3d/graphics/surface.h

## 6.64 GAUGE3D::GTexture Class Reference

Represents a texture loaded into the rendering device.

`#include <gauge3d/renderer/texture.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- virtual ∼**GTexture** ()
- virtual void Surface (pGSurface surface)=0

  *Sets the surface data for the texture.*
- virtual void Image (pGImage image)=0

  *Sets the color and (possibly) opacity data for the texture.*

### 6.64.1 Detailed Description

Represents a texture loaded into the rendering device.

A GTexture can be more than just a color map. It can be a complete surface descriptor, including gloss map, elevation map, bump map, etc.

### 6.64.2 Member Function Documentation

#### 6.64.2.1 void GAUGE3D::GTexture::Surface (pGSurface *surface*) `[pure virtual]`

Sets the surface data for the texture.

**Note:**

> Most hardware requires textures to have power-of-two dimentions. If you specify a surface which contains images that break this rule, the renderer will scale the image up to the next power-of-two dimentions for you. So, don't worry abuot it unless you have load time problems.

### 6.64.2.2 void GAUGE3D::GTexture::Image (pGImage *image*) `[pure virtual]`

Sets the color and (possibly) opacity data for the texture.

**Remarks:**

> This is basically the same as specifying a surface which only contains this one image interpreted as a color-opacity map (or just color, if the image has no alpha channel).

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

- gauge3d/renderer/texture.h

## 6.65 GAUGE3D::GThread Class Reference

Multithreading support.

`#include <gauge3d/osabstraction/thread.h>`

Inherits GAUGE3D::GObject.

### Public Methods

- GThread (void ∗(∗start)(void ∗))

  *Create the thread, but don't start it yet.*

- ~GThread ()

  *Kill and destroy the thread.*

- void Start (void ∗arg)

  *Start the thread with the given argument.*

- void∗ Join ()

  *Wait for the thread to finish and get the return value.*

- void Detach ()

  *Call if you will not be joining the thread.*

### Static Public Methods

- void Return (void ∗retVal)

  *Quit the current thread.*

### 6.65.1 Detailed Description

Multithreading support.

This class allows for multitasking support.

### 6.65.2 Constructor & Destructor Documentation

#### 6.65.2.1 GAUGE3D::GThread::GThread (void $*$($*$ *start*)(void $*$))

Create the thread, but don't start it yet.

**Parameters:**

***start*** A pointer to the function to run from the new thread. The function takes one parameter of type void$*$ and returns void$*$.

### 6.65.3 Member Function Documentation

#### 6.65.3.1 void GAUGE3D::GThread::Start (void $*$ *arg*)

Start the thread with the given argument.

**Parameters:**

***arg*** The argument to pass to the thread's start function.

#### 6.65.3.2 void $*$ GAUGE3D::GThread::Join ()

Wait for the thread to finish and get the return value.

**Returns:**

The return value from the threads start function.

**Remarks:**

This function will not return until the thread has finished executing. Be careful when you call this.

#### 6.65.3.3 void GAUGE3D::GThread::Detach ()

Call if you will not be joining the thread.

If you do not call this and do not call Join(), some resources associated with the thread may never be freed.

### 6.65.3.4 void GAUGE3D::GThread::Return (void $*$ *retVal*) `[static]`

Quit the current thread.

**Parameters:**

***retVal*** Value to use as the return value from the start function.

Execution of the thread will not continue after calling this.

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

- gauge3d/osabstraction/thread.h

## 6.66 GAUGE3D::GTimer Class Reference

Access to the system clock.

`#include <timer.h>`

### Static Public Methods

- void Init (tTime startTime)

  *Initialized the engine timer and set it to the given time.*

- tTime Time ()

  *Get the current time.*

- void Sleep (tTime length)

  *Pause the current thread for the given amount of time.*

### 6.66.1 Detailed Description

Access to the system clock.

GTimer provides access to the system clock with millisecond accuracy.

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

- gauge3d/osabstraction/timer.h

## 6.67 GAUGE3D::GTriangle Class Reference

Represents a triangle in 3-space.

`#include <gauge3d/3dmath/triangle.h>`

### Public Methods

- GTriangle ()

  *Leaves vertices uninitialized.*

- GTriangle (const GVector &vertex1,const GVector &vertex2,const GVector &vertex3)

  *Initializes vertices to specified values.*

- GTriangle (GVector vertices[ ])

  *Initializes to a three-vertex array.*

- void GetPlane (GPlane ∗plane)const

  *Gets the plane which the triangle is on.*

- void Reverse ()

  *Reverses the direction of the vertices.*

- bool Contains (const GVector &point)

  *Returns true if the point is on the triangle.*

- int8 Intersects (const GLine &line)

  *Finds whether or not the line intersects the triangle, and from which side if so.*

- GVector& operator[ ] (int index)

  *Array access to vertices.*

- const GVector& operator[ ] (int index)const

  *Array access to vertices.*

### 6.67.1 Detailed Description

Represents a triangle in 3-space.

## 6.67.2 Member Function Documentation

### 6.67.2.1 void GAUGE3D::GTriangle::GetPlane (GPlane ∗ *plane*) const

Gets the plane which the triangle is on.

**Parameters:**

***plane*** Points to the location to which to write the plane.

**Note:**

The plane will face in the direction from which the vertices of the triangle appear clockwise.

### 6.67.2.2 void GAUGE3D::GTriangle::Reverse ()

Reverses the direction of the vertices.

**Remarks:**

This does not reverse the order of the vertices. It just swaps two of the vertices so that clockwise becomes counter-clockwise. This also means that the direction that the triangle faces will be reversed, since that direction is the direction from which the vertices appear in clockwise order.

### 6.67.2.3 int8 GAUGE3D::GTriangle::Intersects (const GLine & *line*)

Finds whether or not the line intersects the triangle, and from which side if so.

**Parameters:**

***line*** The line with which to test for intersection.

**Returns:**

0 if there is no intersection, 1 if the line hits the front of the triangle, or -1 if the line hits the back of the triangle. (the front side is the side from which the vertices of the triangle appear clockwise.)

**Remarks:**

This function is very fast. It is actually *faster* than Contains(), so if you want to find the point of intersection, first call this and then call GPlane::FindIntersection if necessary.

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

- gauge3d/3dmath/triangle.h

## 6.68 GAUGE3D::GVector Class Reference

A basic vector class.

`#include <gauge3d/3dmath/vector.h>`

### Public Methods

#### Constructors

- GVector (void)
- GVector (GCoordinate _x,GCoordinate _y,GCoordinate _z)
- GVector (GCoordinate coords[ ])

#### Operators

*These are all standard vector math.*

- const GVector& operator+= (const GVector &other)

  *Standard vector add.*

- const GVector& operator-= (const GVector &other)

  *Standard vector subtract.*

- GCoordinate operator ∗ (const GVector &other)const

  *Dot product.*

- GVector X (const GVector &other)const

  *Cross product.*

- const GVector& operator ∗= (GCoordinate other)

  *Scalar multiplication.*

- GVector operator/ (GCoordinate div)const

  *Scalar division.*

- const GVector& operator/= (GCoordinate other)

  *Scalar division.*

- GCoordinate& operator[ ] (int index)

  *Array-like access to coordinates.*

- const GCoordinate& operator[ ] (int index)const

*Array-like access to coordinates.*

- bool operator== (const GVector &other)const
  *Comparison with error compensation.*
- bool operator!= (const GVector &other)const
  *Comparison with error compensation.*

### Advanced Ops

- GCoordinate Magnitude (void)const
  *Find the magnitude (length) of vector.*
- GCoordinate SquareMagnitude (void)const
  *Find the magnitude squared (faster to calculate).*
- void Normalize (void)
  *Make the vector into a unit vector with the same direction.*
- void Slerp (const GVector &v1,const GVector &v2,GCoordinate t)
  *Spherical Linear Interpolation.*

## Public Attributes

### Coordinates

- GCoordinate **x**
- GCoordinate **y**
- GCoordinate **z**

## Related Functions

(Note that these are not member functions.)

- GVector operator+ (const GVector &vec1,const GVector &vec2)
  *Vector addition.*
- GVector operator- (const GVector &vec1,const GVector &vec2)
  *Vector subtraction.*
- GVector operator ∗ (const GVector &vec,GCoordinate coord)
  *Vector and scalar multiplication.*

- GVector operator ∗ (GCoordinate coord,const GVector &vec)

  *Vector and scalar multiplication.*

### 6.68.1 Detailed Description

A basic vector class.

This class represents a vector in three dimentional space. Consult your highschool math books for more information. :)

### 6.68.2 Constructor & Destructor Documentation

#### 6.68.2.1 GAUGE3D::GVector::GVector (void) `[inline]`

Leaves x, y, and z uninitialized.

#### 6.68.2.2 GAUGE3D::GVector::GVector (GCoordinate *_x*, GCoordinate *_y*, GCoordinate *_z*) `[inline]`

Initialize to given x, y, and z.

#### 6.68.2.3 GAUGE3D::GVector::GVector (GCoordinate *coords*[ ]) `[inline, explicit]`

Initialize from 3-coordinate array containing x, y, and z.

### 6.68.3 Member Function Documentation

#### 6.68.3.1 void GAUGE3D::GVector::Slerp (const GVector & *v1*, const GVector & *v2*, GCoordinate *t*)

Spherical Linear Interpolation.

Slerp() replaces the vector with the result of a spherical linear interpolation between v1 and v2. Imagin interpolating between two points on the surface of a sphere. The resulting vector will be a unit vector.

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

- gauge3d/3dmath/vector.h

## 6.69 GAUGE3D::GVirtualDirectory Class Reference

Allows you to construct a virtual filesystem in RAM.

`#include <gauge3d/files/virtualdirectory.h>`

Inherits GAUGE3D::GDirectory.

### Public Methods

- **GVirtualDirectory** ()
- virtual ~**GVirtualDirectory** ()
- virtual GArray<GString> ListFiles ()

  *Get a list of all the files in the directory.*

- virtual GArray<GString> ListSubdirs ()

  *Get a list of all the subdirectories of the directory.*

- virtual pGFile OpenFile (GString name)

  *Get a pointer to a file in the directory.*

- virtual pGDirectory OpenSubdir (GString name)

  *Get a pointer to a sub directory in the directory.*

- virtual pGFile AddFile (GString name)

  *Create a new file in the directory.*

- virtual pGDirectory AddSubdir (GString name)

  *Create a new sub directory to the directory.*

- virtual void RemoveFile (GString name)

  *Remove a file from the dircetory.*

- virtual void RemoveDirectory (GString name)

  *Remove a sub directory from the directory (recursively if necessary).*

- void AddFile (GString name,pGFile file)

  *Link a file into the virtual directory.*

- void AddSubdir (GString name,pGVirtualDirectory subdir)

  *Link a sub directory into the virtual directory.*

### 6.69.1 Detailed Description

Allows you to construct a virtual filesystem in RAM.

A virtual directory exists only in RAM, and files within the directory are simply pointers to any sort of object derived from GFile. These may be disk files, temporary files, files in archives, etc.

### 6.69.2 Member Function Documentation

#### 6.69.2.1 virtual pGFile GAUGE3D::GVirtualDirectory::OpenFile (GString *name*) `[virtual]`

Get a pointer to a file in the directory.

**Parameters:**

***name*** The name of the file to open.

**Remarks:**

Use this to open the contents of a directory. This way, if you have a pointer to the root directory of any filesystem, you can access everything in the system.

**Note:**

It is perfectly legal to say:

- myDirectory.OpenFile(”this/is/a/deeply/nested/file”);

Likewise with OpenSubdir(). You need not manually traverse the directory tree. Always use forward slashes (’/’) to separate directories. Using back slashes (’\’) will not work, even on Windows.

Reimplemented from GAUGE3D::GDirectory.

#### 6.69.2.2 virtual pGDirectory GAUGE3D::GVirtualDirectory::OpenSubdir (GString *name*) `[virtual]`

Get a pointer to a sub directory in the directory.

**Parameters:**

***name*** The name of the sub directory to open.

**Remarks:**

Use this to open the contents of a directory. This way, if you have a pointer to the root directory of any filesystem, you can access everything in the system.

**Note:**

It is perfectly legal to say:

- myDirectory.OpenFile("this/is/a/deeply/nested/subdir");

Likewise with OpenSubdir(). You need not manually traverse the directory tree. Always use forward slashes ('/') to separate directories. Using back slashes ('\') will not work, even on Windows.

Reimplemented from GAUGE3D::GDirectory.

#### 6.69.2.3 virtual pGFile GAUGE3D::GVirtualDirectory::AddFile (GString *name*) `[virtual]`

Create a new file in the directory.

**Parameters:**

***name*** The name of the file to create.

**Returns:**

A pointer to the new file, or NULL if the file already existed or could not be created.

Reimplemented from GAUGE3D::GDirectory.

#### 6.69.2.4 virtual pGDirectory GAUGE3D::GVirtualDirectory::AddSubdir (GString *name*) `[virtual]`

Create a new sub directory to the directory.

**Parameters:**

***name*** The name of the sub directory to create.

**Returns:**

A pointer to the new directory, or NULL if the directory already existed or could not be created.

Reimplemented from GAUGE3D::GDirectory.

#### 6.69.2.5 void GAUGE3D::GVirtualDirectory::AddFile (GString *name*, pGFile *file*)

Link a file into the virtual directory.

**Parameters:**

***name*** The name to give the file entry in the virtual directory.

***file*** The file that the new entry should refer to. When the entry is requested, this same pointer will be returned.

#### 6.69.2.6 void GAUGE3D::GVirtualDirectory::AddSubdir (GString *name*, pGVirtualDirectory *subdir*)

Link a sub directory into the virtual directory.

**Parameters:**

***name*** The name to give the directory entry in the virtual directory.

***subdir*** The directory that the new entry should refer to. When the entry is requested, this same pointer will be returned. Must be a virtual directory, to avoid confusion.

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

- gauge3d/files/virtualdirectory.h

# Index