# Assimp v3.1.1 (June 2014)

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Namespaces</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
</table>

Installation
Using the pre-built libraries with Visual C++ 8/9

If you develop at Visual Studio 2005 or 2008, you can simply use the pre-built linker libraries provided in the distribution. Extract all files to a place of your choice. A directory called "assimp" will be created there. Add the assimp/include path to your include paths (Menu->Extras->Options->Projects and Solutions->VC++ Directories->Include files) and the assimp/lib/<Compiler> path to your linker paths (Menu->Extras->Options->Projects and Solutions->VC++ Directories->Library files). This is neccessary only once to setup all paths inside you IDE.

To use the library in your C++ project you have to include either <assimp/Importer.hpp> or <assimp/cimport.h> plus some others starting with <types.h>. If you set up your IDE correctly the compiler should be able to find the files. Then you have to add the linker library to your project dependencies. Link to <assimp_root>/lib/<config-name>/assimp.lib. config-name is one of the predefined project configs. For static linking, use release/debug. See the sections below on this page for more information on the other build configs. If done correctly you should now be able to compile, link, run and use the application. If the linker complains about some integral functions being defined twice you probably have mixed the runtimes. Recheck the project configuration (project properties -> C++ -> Code generation -> Runtime) if you use static runtimes (Multithreaded / Multithreaded Debug) or dynamic runtimes (Multithreaded DLL / Multithreaded Debug DLL). Choose the assimp linker lib accordingly.

Please don't forget to also read the Microsoft Compilers and the C++ Standard Library section on MSVC and the STL.
Microsoft Compilers and the C++ Standard Library

In VC8 and VC9 Microsoft introduced some Standard Library debugging features. A good example are improved iterator checks and various useful debug checks. The problem is the performance penalty that incurs with those extra checks.

Most of these security enhancements are active in release builds by default, rendering assimp several times slower. However, it is possible to disable them by setting

```plaintext
_HAS_ITERATOR_DEBUGGING=0
 SECURE_SCL=0
```

in the preprocessor options (or alternatively in the source code, just before the STL is included for the first time). assimp's vc8 and vc9 configs enable these flags by default.

If you're linking statically against assimp: Make sure your applications uses the same STL settings! If you do not, there are two binary incompatible STL versions mangled together and you'll crash. Alternatively you can disable the fast STL settings for assimp by removing the 'FastSTL' property sheet from the vc project file.

If you're using assimp in a DLL/SO: It's ok. There's no STL used in the binary DLL/SO interface, so it doesn't care whether your application uses the same STL settings or not.

Another option is to build against a different STL implementation, for example STLport. There's a special Building against STLport section that has a description how to achieve this.
Building the library from scratch

To build the library on your own you first have to get hold of the dependencies. Fortunately, special attention was paid to keep the list of dependencies short. Unfortunately, the only dependency is boost which can be a bit painful to set up for certain development environments. Boost is a widely used collection of classes and functions for various purposes. Chances are that it was already installed along with your compiler. If not, you have to install it for yourself. Read the "Getting Started" section of the Boost documentation for how to setup boost. VisualStudio users can use a comfortable installer from http://www.boost-consulting.com/products/free. Choose the appropriate version of boost for your runtime of choice.

If you don't want to use boost, you can build against our "Boost-Workaround". It consists of very small implementations of the various boost utility classes used. However, you'll lose functionality (e.g. threading) by doing this. So, if you can use boost, you should use boost. Otherwise, See the NoBoost-Section later on this page for the details of the workaround.

Once boost is working, you have to set up a project for the assimp library in your favorite IDE. If you use VC2005 or VC2008, you can simply load the solution or project files in the workspaces/ folder, otherwise you have to create a new package and add all the headers and source files from the include/ and code/ directories. Set the temporary output folder to obj/, for example, and redirect the output folder to bin/. Then build the library - it should compile and link fine.

The last step is to integrate the library into your project. This is basically the same task as described in the "Using the pre-built libraries" section above: add the include/ and bin/ directories to your IDE's paths so that the compiler can find the library files. Alternatively you can simply add the assimp project to your project's overall solution and build it inside your solution.
Building without boost.

The Boost-Workaround consists of dummy replacements for some boost utility templates. Currently there are replacements for

- `boost.scoped_ptr`
- `boost.scoped_array`
- `boost.format`
- `boost.random`
- `boost.common_factor`
- `boost.foreach`
- `boost.tuple`
- `boost.make_shared`

These implementations are very limited and are not intended for use outside assimp. A compiler with full support for partial template specializations is required. To enable the workaround, put the following in your compiler's list of predefined macros:

```
#define ASSIMP_BUILD_BOOST_WORKAROUND
```

If you're working with the provided solutions for Visual Studio use the `-noboost` build configs.

`assimp_BUILD_BOOST_WORKAROUND` implies `assimp_BUILD_SINGLETHREADED`.

See the assimp_st section for more details.
assimp can be built as DLL. You just need to select a -dll config from the list of project configs and you're fine.

**NOTE:** Theoretically, assimp-dll can be used with multithreaded (non-dll) runtime libraries, as long as you don't utilize any non-public stuff from the code folder. However, if you happen to encounter very strange problems, try changing the runtime to *Multithreaded (Debug) DLL.*
Building against STLport

STLport is a free, fast and secure STL replacement that works with all major compilers and platforms. To get it, download the latest release from <stlport.org>. Usually you'll just need to run 'configure' + a makefile (see their README for more details). Don't miss to add <stlport_root>/stlport to your compiler's default include paths - prior to the directory where your compiler vendor's headers lie. Do the same for <stlport_root>/lib and recompile assimp. To ensure you're really building against STLport see aiGetCompileFlags(). In our testing, STLport builds tend to be a bit faster than builds against Microsoft's C++ Standard Library.
Usage
Access by C++ class interface

The assimp library can be accessed by both a class or flat function interface. The C++ class interface is the preferred way of interaction: you create an instance of class `Assimp::Importer`, maybe adjust some settings of it and then call `Assimp::Importer::ReadFile()`. The class will read the files and process its data, handing back the imported data as a pointer to an `aiScene` to you. You can now extract the data you need from the file. The importer manages all the resources for itself. If the importer is destroyed, all the data that was created/read by it will be destroyed, too. So the easiest way to use the Importer is to create an instance locally, use its results and then simply let it go out of scope.

C++ example:

```cpp
#include <assimp/Importer.hpp> // C++ importer interface
#include <assimp/scene.h> // Output data structure
#include <assimp/postprocess.h> // Post processing flags

bool DoTheImportThing( const std::string& pFile)
{
    // Create an instance of the Importer class
    Assimp::Importer importer;
    // And have it read the given file with some example postprocessing
    // Usually - if speed is not the most important aspect for you - you'll
    // probably to request more postprocessing than we do in this example.
    const aiScene* scene = importer.ReadFile( pFile,
        aiProcess_CalcTangentSpace | aiProcess_Triangulate | aiProcess_JoinIdenticalVertices | aiProcess_SortByPType);
    // If the import failed, report it
    if( !scene)
    {
        DoTheErrorLogging( importer.GetErrorString());
        return false;
    }
    // Now we can access the file's contents.
```
DoTheSceneProcessing( scene);
// We're done. Everything will be cleaned up by the importer destructor
return true;
}

What exactly is read from the files and how you interpret it is described at the
Data Structures page. The post processing steps that the assimp library can apply
to the imported data are listed at aiPostProcessSteps. See the pp Post processing
page for more details.

Note that the aiScene data structure returned is declared 'const'. Yes, you can get
rid of these 5 letters with a simple cast. Yes, you may do that. No, it's not
recommended (and it's suicide in DLL builds if you try to use new or delete on
any of the arrays in the scene).
Access by plain-c function interface

The plain function interface is just as simple, but requires you to manually call the clean-up after you're done with the imported data. To start the import process, call \texttt{aiImportFile()} with the filename in question and the desired postprocessing flags like above. If the call is successful, an \texttt{aiScene} pointer with the imported data is handed back to you. When you're done with the extraction of the data you're interested in, call \texttt{aiReleaseImport()} on the imported scene to clean up all resources associated with the import.

C example:

```c
#include <assimp/cimport.h> // Plain-C interface
#include <assimp/scene.h> // Output data structure
#include <assimp/postprocess.h> // Post processing flags

bool DoTheImportThing( const char* pFile)
{
    // Start the import on the given file with some example postprocessing
    // Usually - if speed is not the most important aspect for you - you'll t
    // probably to request more postprocessing than we do in this example.
    const aiScene* scene = aiImportFile( pFile,
        aiProcess_CalcTangentSpace | aiProcess_Triangulate | aiProcess_JoinIdenticalVertices | aiProcess_SortByPType);
    // If the import failed, report it
    if( !scene)
    {
        DoTheErrorLogging( aiGetErrorString());
        return false;
    }
    // Now we can access the file's contents
    DoTheSceneProcessing( scene);
    // We're done. Release all resources associated with this import
    aiReleaseImport( scene);
    return true;
} 
```
Using custom IO logic with the C++ class interface

The assimp library needs to access files internally. This of course applies to the file you want to read, but also to additional files in the same folder for certain file formats. By default, standard C/C++ IO logic is used to access these files. If your application works in a special environment where custom logic is needed to access the specified files, you have to supply custom implementations of IOSstream and IOSystem. A shortened example might look like this:

#include <assimp/IOSstream.hpp>
#include <assimp/IOSystem.hpp>
// My own implementation of IOSstream
class MyIOSstream : public Assimp::IOSstream
{
  friend class MyIOSystem;
  protected:
    // Constructor protected for private usage by MyIOSystem
    MyIOSstream(void);
  public:
    ~MyIOSstream(void);
    size_t Read( void* pvBuffer, size_t pSize, size_t pCount) { ... }
    size_t Write( const void* pvBuffer, size_t pSize, size_t pCount) { ... }
    aiReturn Seek( size_t pOffset, aiOrigin pOrigin) { ... }
    size_t Tell() const { ... }
    size_t FileSize() const { ... }
    void Flush () { ... }
};
// Fisher Price - My First Filesystem
class MyIOSystem : public Assimp::IOSystem
{
  MyIOSystem() { ... }
  ~MyIOSystem() { ... }
  // Check whether a specific file exists
  bool Exists( const std::string& pFile) const {
Get the path delimiter character we'd like to see:

```cpp
char GetOsSeparator() const {
    return '/';
}
```

... and finally a method to open a custom stream:

```cpp
IOStream* Open( const std::string& pFile, const std::string& pMode) {
    return new MyIOStream( ... );
}
```

Now that your IO system is implemented, supply an instance of it to the Importer object by calling `Assimp::Importer::SetIOHandler()`.

```cpp
void DoTheImportThing( const std::string& pFile)
{
    Assimp::Importer importer;
    // put my custom IO handling in place
    importer.SetIOHandler( new MyIOSystem());
    // the import process will now use this implementation to access any file
    importer.ReadFile( pFile, SomeFlag | SomeOtherFlag);
}
```
Using custom IO logic with the plain-c function interface

The C interface also provides a way to override the file system. Control is not as fine-grained as for C++ although surely enough for almost any purpose. The process is simple:

- Include `cfileio.h`
- Fill an `aiFileIO` structure with custom file system callbacks (they're self-explanatory as they work similar to the CRT's fXXX functions)
- .. and pass it as parameter to `aiImportFileEx`
Logging

The assimp library provides an easy mechanism to log messages. For instance if you want to check the state of your import and you just want to see, after which preprocessing step the import-process was aborted you can take a look into the log. Per default the assimp-library provides a default log implementation, where you can log your user specific message by calling it as a singleton with the requested logging-type. To see how this works take a look to this:

```cpp
using namespace Assimp;

// Create a logger instance
DefaultLogger::create("", Logger::VERBOSE);
// Now I am ready for logging my stuff
DefaultLogger::get()->info("this is my info-call");
// Kill it after the work is done
DefaultLogger::kill();
```

At first you have to create the default-logger-instance (create). Now you are ready to rock and can log a little bit around. After that you should kill it to release the singleton instance.

If you want to integrate the assimp-log into your own GUI it my be helpful to have a mechanism writing the logs into your own log windows. The logger interface provides this by implementing an interface called LogStream. You can attach and detach this log stream to the default-logger instance or any implementation derived from Logger. Just derivate your own logger from the abstract base class LogStream and overwrite the write-method:

```cpp
// Example stream
class myStream : LogStream
{
public:
// Constructor
myStream()
{
// empty
```
}  // Destructor
~myStream()
{
    // empty
}
 // Write your own functionality
void write(const char* message)
{
    ::printf("%s\n", message);
}

// Select the kinds of messages you want to receive on this log stream
const unsigned int severity =
Logger::DEBUGGING|Logger::INFO|Logger::ERR|Logger::WARN;
// Attaching it to the default logger
Assimp::DefaultLogger::get()->attachStream( new myStream(), severity );

The severity level controls the kind of message which will be written into the
attached stream. If you just want to log errors and warnings set the warn and
error severity flag for those severities. It is also possible to remove a self defined
logstream from an error severity by detaching it with the severity flag set:

unsigned int severity = 0;
severity |= Logger::DEBUGGING;
// Detach debug messages from you self defined stream
Assimp::DefaultLogger::get()->attachStream( new myStream(), severity );

If you want to implement your own logger just derive from the abstract base
class #Logger and overwrite the methods debug, info, warn and error.

If you want to see the debug-messages in a debug-configured build, the Logger-
interface provides a logging-severity. You can set it calling the following
method:

Assimp::DefaultLogger::get()->setLogSeverity( LogSeverity log_severity );

The normal logging severity supports just the basic stuff like, info, warnings and
errors. In the verbose level very fine-grained debug messages will be logged,
too. Note that this kind kind of logging might decrease import performance.
Data Structures

The assimp library returns the imported data in a collection of structures.

aiScene forms the root of the data, from here you gain access to all the nodes, meshes, materials, animations or textures that were read from the imported file. The aiScene is returned from a successful call to assimp::Importer::ReadFile(), aiImportFile() or aiImportFileEx() - see the Usage page for further information on how to use the library.

By default, all 3D data is provided in a right-handed coordinate system such as OpenGL uses. In this coordinate system, +X points to the right, -Z points away from the viewer into the screen and +Y points upwards. Several modeling packages such as 3D Studio Max use this coordinate system as well (or a rotated variant of it). By contrast, some other environments use left-handed coordinate systems, a prominent example being DirectX. If you need the imported data to be in a left-handed coordinate system, supply the aiProcess_MakeLeftHanded flag to the ReadFile() function call.

The output face winding is counter clockwise. Use aiProcess_FlipWindingOrder to get CW data.

x2
x1
x0

Outputted polygons can be literally everything: they're probably concave, self-intersecting or non-planar, although our built-in triangulation (aiProcess_Triangulate postprocessing step) doesn't handle the two latter.

The output UV coordinate system has its origin in the lower-left corner:

0y|1y ---------- 1x|1y
||
||
||
Use the aiProcess_FlipUVs flag to get UV coordinates with the upper-left corner als origin.

All matrices in the library are row-major. That means that the matrices are stored row by row in memory, which is similar to the OpenGL matrix layout. A typical 4x4 matrix including a translational part looks like this:

\[
\begin{bmatrix}
X_1 & Y_1 & Z_1 & T_1 \\
X_2 & Y_2 & Z_2 & T_2 \\
X_3 & Y_3 & Z_3 & T_3 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

... with \((X_1, X_2, X_3)\) being the \(X\) base vector, \((Y_1, Y_2, Y_3)\) being the \(Y\) base vector, \((Z_1, Z_2, Z_3)\) being the \(Z\) base vector and \((T_1, T_2, T_3)\) being the translation part. If you want to use these matrices in DirectX functions, you have to transpose them.

11.24.09: We changed the orientation of our quaternions to the most common convention to avoid confusion. However, if you're a previous user of Assimp and you update the library to revisions beyond SVNREV 502, you have to adapt your animation loading code to match the new quaternion orientation.
The Node Hierarchy

Nodes are little named entities in the scene that have a place and orientation relative to their parents. Starting from the scene's root node all nodes can have 0 to x child nodes, thus forming a hierarchy. They form the base on which the scene is built on: a node can refer to 0..x meshes, can be referred to by a bone of a mesh or can be animated by a key sequence of an animation. DirectX calls them "frames", others call them "objects", we call them aiNode.

A node can potentially refer to single or multiple meshes. The meshes are not stored inside the node, but instead in an array of aiMesh inside the aiScene. A node only refers to them by their array index. This also means that multiple nodes can refer to the same mesh, which provides a simple form of instancing. A mesh referred to by this way lives in the node's local coordinate system. If you want the mesh's orientation in global space, you'd have to concatenate the transformations from the referring node and all of its parents.

Most of the file formats don't really support complex scenes, though, but a single model only. But there are more complex formats such as .3ds, .x or .collada scenes which may contain an arbitrary complex hierarchy of nodes and meshes. I for myself would suggest a recursive filter function such as the following pseudocode:

```c
void CopyNodesWithMeshes( aiNode node, SceneObject targetParent, Matrix4x4 accTransform)
{
    SceneObject parent;
    Matrix4x4 transform;
    // if node has meshes, create a new scene object for it
    if( node.mNumMeshes > 0)
    {
        SceneObjekt newObject = new SceneObject;
        targetParent.addChild( newObject);
        // copy the meshes
        CopyMeshes( node, newObject);
        // the new object is the parent for all child nodes
        parent = newObject;
    }
}
```
transform.SetUnity();
} else
{
  // if no meshes, skip the node, but keep its transformation
  parent = targetParent;
  transform = node.mTransformation * accTransform;
}
// continue for all child nodes
for( all node.mChildren)
  CopyNodesWithMeshes( node.mChildren[a], parent, transform);

This function copies a node into the scene graph if it has children. If yes, a new scene object is created for the import node and the node's meshes are copied over. If not, no object is created. Potential child objects will be added to the old targetParent, but there transformation will be correct in respect to the global space. This function also works great in filtering the bone nodes - nodes that form the bone hierarchy for another mesh/node, but don't have any mesh themselves.
Meshes

All meshes of an imported scene are stored in an array of aiMesh* inside the aiScene. Nodes refer to them by their index in the array and providing the coordinate system for them, too. One mesh uses only a single material everywhere - if parts of the model use a different material, this part is moved to a separate mesh at the same node. The mesh refers to its material in the same way as the node refers to its meshes: materials are stored in an array inside aiScene, the mesh stores only an index into this array.

An aiMesh is defined by a series of data channels. The presence of these data channels is defined by the contents of the imported file: by default there are only those data channels present in the mesh that were also found in the file. The only channels guaranteed to be always present are aiMesh::mVertices and aiMesh::mFaces. You can test for the presence of other data by testing the pointers against NULL or use the helper functions provided by aiMesh. You may also specify several post processing flags at Importer::ReadFile() to let assimp calculate or recalculate additional data channels for you.

At the moment, a single aiMesh may contain a set of triangles and polygons. A single vertex does always have a position. In addition it may have one normal, one tangent and bitangent, zero to AI_MAX_NUMBER_OF_TEXTURECOORDS (4 at the moment) texture coords and zero to AI_MAX_NUMBER_OF_COLOR_SETS (4) vertex colors. In addition a mesh may or may not have a set of bones described by an array of aiBone structures. How to interpret the bone information is described later on.
Materials

See the Material System Page.
Bones

A mesh may have a set of bones in the form of aiBone structures. Bones are a means to deform a mesh according to the movement of a skeleton. Each bone has a name and a set of vertices on which it has influence. Its offset matrix declares the transformation needed to transform from mesh space to the local space of this bone.

Using the bones name you can find the corresponding node in the node hierarchy. This node in relation to the other bones' nodes defines the skeleton of the mesh. Unfortunately there might also be nodes which are not used by a bone in the mesh, but still affect the pose of the skeleton because they have child nodes which are bones. So when creating the skeleton hierarchy for a mesh I suggest the following method:

a) Create a map or a similar container to store which nodes are necessary for the skeleton. Pre-initialise it for all nodes with a "no".
b) For each bone in the mesh:
b1) Find the corresponding node in the scene's hierarchy by comparing their names.
b2) Mark this node as "yes" in the necessityMap.
b3) Mark all of its parents the same way until you 1) find the mesh's node or 2) the parent of the mesh's node.
c) Recursively iterate over the node hierarchy
c1) If the node is marked as necessary, copy it into the skeleton and check its children
c2) If the node is marked as not necessary, skip it and do not iterate over its children.

Reasons: you need all the parent nodes to keep the transformation chain intact. For most file formats and modelling packages the node hierarchy of the skeleton is either a child of the mesh node or a sibling of the mesh node but this is by no means a requirement so you shouldn't rely on it. The node closest to the root node is your skeleton root, from there you start copying the hierarchy. You can skip every branch without a node being a bone in the mesh - that's why the algorithm skips the whole branch if the node is marked as "not necessary".
You should now have a mesh in your engine with a skeleton that is a subset of the imported hierarchy.
Animations

An imported scene may contain zero to x aiAnimation entries. An animation in this context is a set of keyframe sequences where each sequence describes the orientation of a single node in the hierarchy over a limited time span. Animations of this kind are usually used to animate the skeleton of a skinned mesh, but there are other uses as well.

An aiAnimation has a duration. The duration as well as all time stamps are given in ticks. To get the correct timing, all time stamp thus have to be divided by aiAnimation::mTicksPerSecond. Beware, though, that certain combinations of file format and exporter don't always store this information in the exported file. In this case, mTicksPerSecond is set to 0 to indicate the lack of knowledge.

The aiAnimation consists of a series of aiNodeAnim's. Each bone animation affects a single node in the node hierarchy only, the name specifying which node is affected. For this node the structure stores three separate key sequences: a vector key sequence for the position, a quaternion key sequence for the rotation and another vector key sequence for the scaling. All 3d data is local to the coordinate space of the node's parent, that means in the same space as the node's transformation matrix. There might be cases where animation tracks refer to a non-existent node by their name, but this should not be the case in your every-day data.

To apply such an animation you need to identify the animation tracks that refer to actual bones in your mesh. Then for every track:
  a) Find the keys that lay right before the current anim time.
  b) Optional: interpolate between these and the following keys.
  c) Combine the calculated position, rotation and scaling to a tranformation matrix
  d) Set the affected node's transformation to the calculated matrix.

If you need hints on how to convert to or from quaternions, have a look at the Matrix&Quaternion FAQ. I suggest using logarithmic interpolation for the scaling keys if you happen to need them - usually you don't need them at all.
Textures

Normally textures used by assets are stored in separate files, however, there are file formats embedding their textures directly into the model file. Such textures are loaded into an aiTexture structure.

There are two cases:

1) The texture is NOT compressed. Its color data is directly stored in the aiTexture structure as an array of aiTexture::mWidth * aiTexture::mHeight aiTexel structures. Each aiTexel represents a pixel (or "texel") of the texture image. The color data is stored in an unsigned RGBA8888 format, which can be easily used for both Direct3D and OpenGL (swizzling the order of the color components might be necessary). RGBA8888 has been chosen because it is well-known, easy to use and natively supported by nearly all graphics APIs.

2) This applies if aiTexture::mHeight == 0 is fullfilled. Then, texture is stored in a "compressed" format such as DDS or PNG. The term "compressed" does not mean that the texture data must actually be compressed, however the texture was found in the model file as if it was stored in a separate file on the harddisk. Appropriate decoders (such as libjpeg, libpng, D3DX, DevIL) are required to load thesese textures. aiTexture::mWidth specifies the size of the texture data in bytes, aiTexture::pcData is a pointer to the raw image data and aiTexture::achFormatHint is either zeroed or contains the most common file extension of the embedded texture's format. This value is only set if assimp is able to determine the file format.
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Namespaces</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
</table>

Material System
Overview

All materials are stored in an array of `aiMaterial` inside the `aiScene`.

Each `aiMesh` refers to one material by its index in the array. Due to the vastly diverging definitions and usages of material parameters there is no hard definition of a material structure. Instead a material is defined by a set of properties accessible by their names. Have a look at `assimp/material.h` to see what types of properties are defined. In this file there are also various functions defined to test for the presence of certain properties in a material and retrieve their values.
Textures

Textures are organized in stacks, each stack being evaluated independently. The final color value from a particular texture stack is used in the shading equation. For example, the computed color value of the diffuse texture stack (aiTextureType_DIFFUSE) is multiplied with the amount of incoming diffuse light to obtain the final diffuse color of a pixel.

Stack Resulting equation
-------------------------------
| Constant base color | color
-------------------------------
| Blend operation 0 | +
-------------------------------
| Strength factor 0 | 0.25*
-------------------------------
| Texture 0 | texture_0
-------------------------------
| Blend operation 1 | *
-------------------------------
| Strength factor 1 | 1.0*
-------------------------------
| Texture 1 | texture_1
-------------------------------
... ...
## Constants

All material key constants start with 'AI_MATKEY' (it's an ugly macro for historical reasons, don't ask).

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Default Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>aiString</td>
<td>n/a</td>
<td>The name of the material, if available.</td>
</tr>
<tr>
<td>COLOR_DIFFUSE</td>
<td>aiColor3D</td>
<td>black (0,0,0)</td>
<td>Diffuse color of the material. This is typically scaled by the amount of incoming diffuse light (e.g. using gouraud shading)</td>
</tr>
<tr>
<td>COLOR_SPECULAR</td>
<td>aiColor3D</td>
<td>black (0,0,0)</td>
<td>Specular color of the material. This is typically scaled by the amount of incoming specular light (e.g. using phong shading)</td>
</tr>
<tr>
<td>COLOR_AMBIENT</td>
<td>aiColor3D</td>
<td>black (0,0,0)</td>
<td>Ambient color of the material. This is typically scaled by the amount of ambient light</td>
</tr>
<tr>
<td>COLOR_EMISSIVE</td>
<td>aiColor3D</td>
<td>black (0,0,0)</td>
<td>Emissive color of the material. This is the amount of light emitted by the object. In real time applications it will usually not affect surrounding objects, but raytracing applications may wish to treat emissive objects as light sources.</td>
</tr>
</tbody>
</table>

Defines the transparent color.
<table>
<thead>
<tr>
<th>PROPERTY</th>
<th>TYPE</th>
<th>VALUE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR_TRANSPARENT</td>
<td>aiColor3D</td>
<td>black (0,0,0)</td>
<td>of the material, this is the color to be multiplied with the color of translucent light to construct the final 'destination color' for a particular position in the screen buffer. T</td>
</tr>
<tr>
<td>WIREFRAME</td>
<td>int</td>
<td>false</td>
<td>Specifies whether wireframe rendering must be turned on for the material. 0 for false, 1 for true.</td>
</tr>
<tr>
<td>TWOSIDED</td>
<td>int</td>
<td>false</td>
<td>Specifies whether meshes using this material must be rendered without backface culling. 0 for false, 1 for true.</td>
</tr>
<tr>
<td>SHADING_MODEL</td>
<td>int</td>
<td>gouraud</td>
<td>One of the aiShadingMode enumerated values. Defines the library shading model to use for (real time) rendering to approximate the original look of the material as closely as possible.</td>
</tr>
<tr>
<td>BLEND_FUNC</td>
<td>int</td>
<td>false</td>
<td>One of the aiBlendMode enumerated values. Defines how the final color value in the screen buffer is computed from the given color at that position and the newly computed color from the material. Simply said, alpha blending settings.</td>
</tr>
<tr>
<td>OPACITY</td>
<td>float</td>
<td>1.0</td>
<td>Defines the opacity of the material in a range between 0..1. Use this property to activate TWOSIDED to avoid cull artifacts.</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SHININESS</td>
<td>float</td>
<td>0.0f</td>
<td>Defines the shininess of a phong-shaded material. This is actually the exponent of the phong specular equation</td>
</tr>
<tr>
<td>SHININESS_STRENGTH</td>
<td>float</td>
<td>1.0f</td>
<td>Scales the specular color of the material.</td>
</tr>
<tr>
<td>REFRACTIVITY</td>
<td>float</td>
<td>1.0f</td>
<td>Defines the Index Of Refraction for the material.</td>
</tr>
<tr>
<td>TEXTURE(t, n)</td>
<td>aiString</td>
<td>n/a</td>
<td>Defines the path to the n'th texture on the stack 't', where 'n' is any value &gt;= 0 and 't' is one of the aiTextureType enumerated values.</td>
</tr>
<tr>
<td>TEXBLEND(t, n)</td>
<td>float</td>
<td>n/a</td>
<td>Defines the strength the n'th texture on the stack 't'. All color components (rgb) are multiplied with this factor before any further processing is done.</td>
</tr>
<tr>
<td>TEXOP(t, n)</td>
<td>int</td>
<td>n/a</td>
<td>One of the aiTextureOp enumerated values. Defines the arithmetic operation to be used to combine the n'th texture on the stack 't' with the n-1'th. TEXOP(t, 0) refers to the blend operation between the base color for this stack (e.g. COLOR_DIFFUSE for the diffuse stack) and the first texture.</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>MAPPING(t,n)</td>
<td>int n/a</td>
<td>Defines how the input mapping coordinates for sampling the n'th texture on the stack 't' are computed. Usually explicit UV coordinates are provided, but some model file formats might also be using basic shapes, such as spheres or cylinders, to project textures onto meshes.</td>
<td></td>
</tr>
<tr>
<td>UVWSRC(t,n)</td>
<td>int n/a</td>
<td>Defines the UV channel to be used as input mapping coordinates for sampling the n'th texture on the stack 't'. All meshes assigned to this material share the same UV channel setup.</td>
<td></td>
</tr>
<tr>
<td>MAPPINGMODE_U(t,n)</td>
<td>int n/a</td>
<td>Any of the aiTextureMapMode enumerated values. Defines the texture wrapping mode on the x axis for sampling the n'th texture on the stack 't'. 'Wrapping' occurs whenever UVs lie outside the 0..1 range.</td>
<td></td>
</tr>
<tr>
<td>MAPPINGMODE_V(t,n)</td>
<td>int n/a</td>
<td>Wrap mode on the v axis. See MAPPINGMODE_U.</td>
<td></td>
</tr>
<tr>
<td>TEXMAP_AXIS(t,n)</td>
<td>aiVector3D n/a</td>
<td>Defines the base axis to to compute the mapping coordinates for the n'th texture on the stack 't' from. This is not required for UV-mapped textures. For instance, if MAPPING(t,n) is aiTextureMapping_SPHERE, the base axis is used to compute UV coordinates.</td>
<td></td>
</tr>
<tr>
<td>TEXFLAGS(t, n)</td>
<td>int</td>
<td>n/a</td>
<td>U and V would map to longitude and latitude of a sphere around the given axis. The axis is given in local mesh space.</td>
</tr>
<tr>
<td>---------------</td>
<td>-----</td>
<td>-----</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>Defines miscellaneous flag for the n'th texture on the stack 't'. This is a bitwise combination of the aiTextureFlags enumerated values.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
C++-API

Retrieving a property from a material is done using various utility functions. For C++ it's simply calling `aiMaterial::Get()`

```c++
aiMaterial* mat = ..... // The generic way
if(AI_SUCCESS != mat->Get(<material-key>,<where-to-store>)) {
// handle epic failure here
}
```

Simple, isn't it? To get the name of a material you would use

```c++
aiString name;
mat->Get(AI_MATKEY_NAME,name);
```

Or for the diffuse color ('color' won't be modified if the property is not set)

```c++
aiColor3D color (0.f,0.f,0.f);
mat->Get(AI_MATKEY_COLOR_DIFFUSE,color);
```

**Note:** Get() is actually a template with explicit specializations for `aiColor3D`, aiColor4D, `aiString`, float, int and some others. Make sure that the type of the second parameter matches the expected data type of the material property (no compile-time check yet!). Don't follow this advice if you wish to encounter very strange results.
C-API

For good old C it's slightly different. Take a look at the aiGetMaterialGet<datatype> functions.

```c
aiMaterial* mat = ..... 
if(AI_SUCCESS != aiGetMaterialFloat(mat,<material-key>,<where-to-store>))
{
    // handle epic failure here
}
```

To get the name of a material you would use

```c
aiString name;
aiGetMaterialString(mat,AI_MATKEY_NAME,&name);
```

Or for the diffuse color ('color' won't be modified if the property is not set)

```c
aiColor3D color (0.f,0.f,0.f);
aiGetMaterialColor(mat,AI_MATKEY_COLOR_DIFFUSE,&color);
```
How to map UV channels to textures (MATKEY_UVWSRC)

The MATKEY_UVWSRC property is only present if the source format doesn't specify an explicit mapping from textures to UV channels. Many formats don't do this and assimp is not aware of a perfect rule either.

Your handling of UV channels needs to be flexible therefore. Our recommendation is to use logic like this to handle most cases properly:

1. have only one uv channel?
   - assign channel 0 to all textures and break

2. for all textures
   - have uvwsrc for this texture?
     - assign channel specified in uvwsrc
   - else
     - assign channels in ascending order for all texture stacks,
       - i.e. diffuse1 gets channel 1, opacity0 gets channel 0.
Pseudo Code Listing

For completeness, the following is a very rough pseudo-code sample showing how to evaluate Assimp materials in your shading pipeline. You'll probably want to limit your handling of all those material keys to a reasonable subset suitable for your purposes (for example most 3d engines won't support highly complex multi-layer materials, but many 3d modellers do).

Also note that this sample is targeted at a (shader-based) rendering pipeline for real time graphics.

// Evaluate multiple textures stacked on top of each other
float3 EvaluateStack(stack)
{
    // For the 'diffuse' stack stack.base_color would be COLOR_DIFFUSE
    // and TEXTURE(aiTextureType_DIFFUSE,n) the n'th texture.
    float3 base = stack.base_color;
    for (every texture in stack)
    {
        // assuming we have explicit & pretransformed UVs for this texture
        float3 color = SampleTexture(texture,uv);
        // scale by texture blend factor
        color *= texture.blend;
        if (texture.op == add)
            base += color;
        else if (texture.op == multiply)
            base *= color;
        else // other blend ops go here
    }
    return base;
}

// Compute the diffuse contribution for a pixel
float3 ComputeDiffuseContribution()
if (shading == none)
return float3(1,1,1);
float3 intensity (0,0,0);
for (all lights in range)
{
float fac = 1.f;
if (shading == gouraud)
fac = lambert-term ..
else // other shading modes go here
// handling of different types of lights, such as point or spot lights
// ...  
// and finally sum the contribution of this single light ...
intensity += light.diffuse_color * fac;
}
// ... and combine the final incoming light with the diffuse color
return EvaluateStack(diffuse) * intensity;
}
// --------------------------------------------------------------------- 
// Compute the specular contribution for a pixel
float3 ComputeSpecularContribution()
{
if (shading == gouraud || specular_strength == 0 || specular_exponent == 0)
return float3(0,0,0);
float3 intensity (0,0,0);
for (all lights in range)
{
float fac = 1.f;
if (shading == phong)
fac = phong-term ..
else // other specular shading modes go here
// handling of different types of lights, such as point or spot lights
// ...
// and finally sum the specular contribution of this single light ...
intensity += light.specular_color * fac;
}
// ... and combine the final specular light with the specular color
return EvaluateStack(specular) * intensity * specular_strength;
}
// ---------------------------------------------------------------------
// Compute the ambient contribution for a pixel
float3 ComputeAmbientContribution()
{
    if (shading == none)
        return float3(0,0,0);
    float3 intensity (0,0,0);
    for (all lights in range)
    {
        float fac = 1.f;
        // handling of different types of lights, such as point or spot lights
        // ...
        // and finally sum the ambient contribution of this single light ...
        intensity += light.ambient_color * fac;
    }
    // ... and combine the final ambient light with the ambient color
    return EvaluateStack(ambient) * intensity;
}

// Compute the final color value for a pixel
// @param prev Previous color at that position in the framebuffer
float4 PimpMyPixel (float4 prev)
{
    // .. handle displacement mapping per vertex
    // .. handle bump/normal mapping
    // Get all single light contribution terms
    float3 diff = ComputeDiffuseContribution();
    float3 spec = ComputeSpecularContribution();
    float3 ambi = ComputeAmbientContribution();
    // .. and compute the final color value for this pixel
    float3 color = diff + spec + ambi;
    float3 opac = EvaluateStack(opacity);
    // note the *slightly* strange meaning of additive and multiplicative blending here ...
    // those names will most likely be changed in future versions
    if (blend_func == add)
        return prev+color*opac;
    else if (blend_func == multiply)
        return prev*(1.0-opac)+prev*opac;
    return color;
Assimp v3.1.1 (June 2014)

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Namespaces</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
</table>

Performance
Overview

This page discusses general performance issues related to assimp.
Profiling

assimp has built-in support for very basic profiling and time measurement. To turn it on, set the GLOB_MEASURE_TIME configuration switch to true (nonzero). Results are dumped to the log file, so you need to setup an appropriate logger implementation with at least one output stream first (see the Logging Page for the details.).

Note that these measurements are based on a single run of the importer and each of the post processing steps, so a single result set is far away from being significant in a statistic sense. While precision can be improved by running the test multiple times, the low accuracy of the timings may render the results useless for smaller files.

A sample report looks like this (some unrelated log messages omitted, entries grouped for clarity):

Debug, T5488: START `total`
Info, T5488: Found a matching importer for this file format

Debug, T5488: START `import`
Info, T5488: BlendModifier: Applied the `Subdivision` modifier to ` OBMonkey`
Debug, T5488: END `import`, dt= 3.516 s

Debug, T5488: START `preprocess`
Debug, T5488: END `preprocess`, dt= 0.001 s
Info, T5488: Entering post processing pipeline

Debug, T5488: START `postprocess`
Debug, T5488: RemoveRedundantMatsProcess begin
Debug, T5488: RemoveRedundantMatsProcess finished
Debug, T5488: END `postprocess`, dt= 0.001 s

Debug, T5488: START `postprocess`
Debug, T5488: TriangulateProcess begin
Info, T5488: TriangulateProcess finished. All polygons have been tr
Debug, T5488: END `postprocess`, dt= 3.415 s
In this particular example only one fourth of the total import time was spent on the actual importing, while the rest of the time got consumed by the \texttt{aiProcess\_Triangulate}, \texttt{aiProcess\_JoinIdenticalVertices} and \texttt{aiProcess\_ImproveCacheLocality} postprocessing steps. A wise selection of postprocessing steps is therefore essential to getting good performance. Of course this depends on the individual requirements of your application, in many of the typical use cases of assimp performance won't matter (i.e. in an offline content pipeline).
Assimp v3.1.1 (June 2014)

Threading
Overview

This page discusses both assimps scalability in threaded environments and the precautions to be taken in order to use it from multiple threads concurrently.
Thread-safety / using Assimp concurrently from several threads

The library can be accessed by multiple threads simultaneously, as long as the following prerequisites are fulfilled:

- Users of the C++-API should ensure that they use a dedicated 
  Assimp::Importer instance for each thread. Constructing instances of 
  Assimp::Importer is expensive, so it might be a good idea to let every 
  thread maintain its own thread-local instance (which can be used to load as 
  many files as necessary).
- The C-API is thread safe.
- When supplying custom IO logic, one must make sure the underlying 
  implementation is thread-safe.
- Custom log streams or logger replacements have to be thread-safe, too.

Multiple concurrent imports may or may not be beneficial, however. For certain 
file formats in conjunction with little or no post processing IO times tend to be 
the performance bottleneck. Intense post processing together with 'slow' file 
formats like X or Collada might scale well with multiple concurrent imports.
Internal threading

Internal multi-threading is not currently implemented.
Resources

This page lists some useful resources for assimp.

Note that, even though the core team has an eye on them, we cannot guarantee the accuracy of third-party information. If in doubt, it's best to ask either on the mailing list or on our forums on SF.net.

- assimp comes with some sample applications, these can be found in the ./samples folder. Don't forget to read the README file.
- [http://www.drivenbynostalgia.com/files/AssimpOpenGLDemo.rar](http://www.drivenbynostalgia.com/files/AssimpOpenGLDemo.rar) - OpenGl animation sample using the library's animation import facilities.
- [http://ogldev.atspace.co.uk/www/tutorial22/tutorial22.html](http://ogldev.atspace.co.uk/www/tutorial22/tutorial22.html) - Tutorial "Loading models using the Open Asset Import Library", out of a series of OpenGl tutorials.
# Assimp v3.1.1 (June 2014)

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Namespaces</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
</table>

## Importer Notes
Blender

This section contains implementation notes for the Blender3D importer.
Overview

assimp provides a self-contained reimplementation of Blender's so called SDNA system ([http://www.blender.org/development/architecture/notes-on-sdna/](http://www.blender.org/development/architecture/notes-on-sdna/)). SDNA allows Blender to be fully backward and forward compatible and to exchange files across all platforms. The BLEND format is thus a non-trivial binary monster and the loader tries to read the most of it, naturally limited by the scope of the aiScene output data structure. Consequently, if Blender is the only modeling tool in your asset work flow, consider writing a custom exporter from Blender if assimp's format coverage does not meet the requirements.
Current status

The Blender loader does not support animations yet, but is apart from that considered relatively stable.
Notes

When filing bugs on the Blender loader, always give the Blender version (or, even better, post the file caused the error).
IFC

This section contains implementation notes on the IFC-STEP importer.
Overview

The library provides a partial implementation of the IFC2x3 industry standard for automatized exchange of CAE/architectural data sets. See http://en.wikipedia.org/wiki/Industry_Foundation_Classes for more information on the format. We aim at getting as much 3D data out of the files as possible.
Current status

IFC support is new and considered experimental. Please report any bugs you may encounter.
Notes

- Only the STEP-based encoding is supported. IFCZIP and IFCXML are not (but IFCZIP can simply be unzipped to get a STEP file).
- The importer leaves vertex coordinates untouched, but applies a global scaling to the root transform to convert from whichever unit the IFC file uses to **metres**.
- If multiple geometric representations are provided, the choice which one to load is based on how expensive a representation seems to be in terms of import time. The loader also avoids representation types for which it has known deficits.
- Not supported are arbitrary binary operations (binary clipping is implemented, though).
- Of the various relationship types that IFC knows, only aggregation, containment and material assignment are resolved and mapped to the output graph.
- The implementation knows only about IFC2X3 and applies this rule set to all models it encounters, regardless of their actual version. Loading of older or newer files may fail with parsing errors.
Metadata

IFC file properties (IfcPropertySet) are kept as per-node metadata, see aiNode::mMetaData.
ATTENTION*: The Ogre-Loader is currently under development, many things have changed after this documentation was written, but they are not final enough to rewrite the documentation. So things may have changed by now!

This section contains implementations notes for the OgreXML importer.
Overview

Ogre importer is currently optimized for the Blender Ogre exporter, because that's the only one that I use. You can find the Blender Ogre exporter at: http://www.ogre3d.org/forums/viewtopic.php?f=8&t=45922
**What will be loaded?**

Mesh: Faces, Positions, Normals and all TexCoords. The Materialname will be used to load the material.

Material: The right material in the file will be searched, the importer should work with materials who have 1 technique and 1 pass in this technique. From there, the texturename (for 1 color- and 1 normalmap) and the materialcolors (but not in custom materials) will be loaded. Also, the materialname will be set.

Skeleton: Skeleton with Bone hierarchy (Position and Rotation, but no Scaling in the skeleton is supported), names and transformations, animations with rotation, translation and scaling keys.
How to export Files from Blender

You can find informations about how to use the Ogreexporter by your own, so here are just some options that you need, so the assimp importer will load everything correctly:

- Use either "Rendering Material" or "Custom Material" see Materials
- do not use "Flip Up Axies to Y"
- use "Skeleton name follow mesh"
XML Format

There is a binary and a XML mesh Format from Ogre. This loader can only Handle xml files, but don't panic, there is a command line converter, which you can use to create XML files from Binary Files. Just look on the Ogre page for it.

Currently you can only load meshes. So you will need to import the *.mesh.xml file, the loader will try to find the appendant material and skeleton file.

The skeleton file must have the same name as the mesh file, e.g. fish.mesh.xml and fish.skeleton.xml.
Materials

The material file can have the same name as the mesh file (if the file is model.mesh or model.mesh.xml the loader will try to load model.material), or you can use Importer::Importer::SetPropertyString(AI_CONFIG_IMPORT_OGRE_MATERIAL "materialfile.material") to specify the name of the material file. This is especially useful if multiply materials a stored in a single file. The importer will first try to load the material with the same name as the mesh and only if this can't be open try to load the alternate material file. The default material filename is "Scene.material".

We suggest that you use custom materials, because they support multiple textures (like colormap and normalmap). First of all you should read the custom material sektion in the Ogre Blender exporter Help File, and than use the assimp.tlp template, which you can find in scripts/OgreImpoter/Assimp.tlp in the assimp source. If you don't set all values, don't worry, they will be ignored during import.

If you want more properties in custom materials, you can easily expand the ogre material loader, it will be just a few lines for each property. Just look in OgreImporterMaterial.cpp
Properties

- **IMPORT_OGRE_TEXTURETYPE_FROM_FILENAME**: Normally, a texture is loaded as a colormap, if no target is specified in the material file. Is this switch is enabled, texture names ending with _n, _l, _s are used as normalmaps, lightmaps or specularmaps.
  Property type: Bool. Default value: false.

- **IMPORT_OGRE_MATERIAL_FILE**: Ogre Meshes contain only the MaterialName, not the MaterialFile. If there is no material file with the same name as the material, Ogre Importer will try to load this file and search the material in it.
  Property type: String. Default value: guessed.
Todo

- Load colors in custom materials
- extend custom and normal material loading
- fix bone hierarchy bug
- test everything elaboratly
- check for non existent animation keys (what happens if a one time not all bones have a key?)
Extending the Library
Overview

Or - how to write your own loaders. It's easy. You just need to implement the `Assimp::BaseImporter` class, which defines a few abstract methods, register your loader, test it carefully and provide test models for it.

OK, that sounds too easy :-). The whole procedure for a new loader merely looks like this:

- Create a header (`FormatNameImporter.h`) and a unit (`FormatNameImporter.cpp`) in the `<root>/code/` directory
- Add them to the following workspaces: vc8 and vc9 (the files are in the workspaces directory), CMAKE (code/CMakeLists.txt, create a new source group for your importer and put them also to ADD_LIBRARY( assimp SHARED))
- Include `AssimpPCH.h` - this is the PCH file, and it includes already most Assimp-internal stuff.

- Open `Importer.cpp` and include your header just below the `include_new_importers_here` line, guarded by a `#define`:

```cpp
#if (!defined assimp_BUILD_NO_FormatName_IMPORTED)
...
#endif
```

Wrap the same guard around your .cpp!

- Now advance to the `register_new_importers_here` line in the `Importer.cpp` and register your importer there - just like all the others do.
- Setup a suitable test environment (i.e. use AssimpView or your own application), make sure to enable the `aiProcess_ValidateDataStructure` flag and enable verbose logging. That is, simply call before you import anything:

```cpp
DefaultLogger::create("AssimpLog.txt", Logger::VERBOSE)
```

- Implement the `Assimp::BaseImporter::CanRead()`, `Assimp::BaseImporter::InternReadFile()` and `Assimp::BaseImporter::GetExtensionList()`. Just copy'n'paste the template
from Appendix A and adapt it for your needs.

- For error handling, throw a dynamic allocated ImportErrorException (see Appendix A) for critical errors, and log errors, warnings, infos and debuginfos with DefaultLogger::get()->[error, warn, debug, info].
- Make sure that your loader compiles against all build configurations on all supported platforms. This includes -noboot! To avoid problems, see the boost section on this page for a list of all 'allowed' boost classes (again, this grew historically when we had to accept that boost is not THAT widely spread that one could rely on it being available everywhere).
- Provide some free test models in <root>/test/models/<FormatName>/ and credit their authors. Test files for a file format shouldn't be too large (~500 KiB in total), and not too repetitive. Try to cover all format features with test data.
- Done! Please, share your loader that everyone can profit from it!
Properties

You can use properties to change the behavior of your importer. In order to do so, you have to override BaseImporter::SetupProperties, and specify your custom properties in config.h. Just have a look to the other AI_CONFIG_IMPORT_* defines and you will understand, how it works.

The properties can be set with Importer::SetProperty***() and can be accessed in your SetupProperties function with Importer::GetProperty***(). You can store the properties as a member variable of your importer, as they are thread safe.
Notes for text importers

- Try to make your parser as flexible as possible. Don't rely on particular layout, whitespace/tab style, except if the file format has a strict definition, in which case you should always warn about spec violations. But the general rule of thumb is *be strict in what you write and tolerant in what you accept.*
- Call `Assimp::BaseImporter::ConvertToUTF8()` before you parse anything to convert foreign encodings to UTF-8. That's not necessary for XML importers, which must use the provided IrrXML for reading.
Notes for binary importers

- Take care of endianess issues! Assimp importers mostly support big-endian platforms, which define the AI_BUILD_BIG_ENDIAN constant. See the next section for a list of utilities to simplify this task.
- Don't trust the input data! Check all offsets!
Utilities

Mixed stuff for internal use by loaders, mostly documented (most of them are already included by AssimpPCH.h):

- **ByteSwapper** (*ByteSwapper.h*) - manual byte swapping stuff for binary loaders.
- **StreamReader** (*StreamReader.h*) - safe, endianess-correct, binary reading.
- **IrrXML** (*irrXMLWrapper.h*) - for XML-parsing (SAX).
- **CommentRemover** (*RemoveComments.h*) - remove single-line and multi-line comments from a text file.
- fast_atof, strtoul10, strtoul16, SkipSpaceAndLineEnd, SkipToNextToken .. large family of low-level parsing functions, mostly declared in *fast_atof.h*, *StringComparison.h* and *ParsingUtils.h* (a collection that grew historically, so don't expect perfect organization).
- **ComputeNormalsWithSmoothingsGroups()** (*SmoothingGroups.h*) - Computes normal vectors from plain old smoothing groups.
- **SkeletonMeshBuilder** (*SkeletonMeshBuilder.h*) - generate a dummy mesh from a given (animation) skeleton.
- **StandardShapes** (*StandardShapes.h*) - generate meshes for standard solids, such as platonic primitives, cylinders or spheres.
- **BatchLoader** (*BaseImporter.h*) - manage imports from external files. Useful for file formats which spread their data across multiple files.
- **SceneCombiner** (*SceneCombiner.h*) - exhaustive toolset to merge multiple scenes. Useful for file formats which spread their data across multiple files.
Filling materials

The required definitions to set/remove/query keys in aiMaterial structures are declared in MaterialSystem.h, in a aiMaterial derivate called aiMaterial. The header is included by AssimpPCH.h, so you don't need to bother.

```cpp
aiMaterial* mat = new aiMaterial();
const float spec = 16.f;
mat->AddProperty(&spec, 1, AI_MATKEY_SHININESS);
// set the name of the material:
NewMaterial->AddProperty(&aiString(MaterialName.c_str()),
    AI_MATKEY_NAME); // MaterialName is a std::string
// set the first diffuse texture
NewMaterial->AddProperty(&aiString(Texturename.c_str()),
    AI_MATKEY_TEXTURE(aiTextureType_DIFFUSE, 0)); // Texturename is a std::string
```
Boost

The boost whitelist:

- `boost.scoped_ptr`
- `boost.scoped_array`
- `boost.format`
- `boost.random`
- `boost.common_factor`
- `boost.foreach`
- `boost.tuple`

(if you happen to need something else, i.e. boost::thread, make this an optional feature. `assimp_BUILD_BOOST_WORKAROUND` is defined for `-noboot` builds)
Appendix A - Template for BaseImporter's abstract methods

// Returns whether the class can handle the format of the given file.
bool xxxxImporter::CanRead( const std::string& pFile, IOSystem* pIOHandler,
bool checkSig) const
{
const std::string extension = GetExtension(pFile);
if(extension == "xxxx") {
return true;
}
if (!extension.length() || checkSig) {
// no extension given, or we're called a second time because no
// suitable loader was found yet. This means, we're trying to open
// the file and look for and hints to identify the file format.
// #Assimp::BaseImporter provides some utilities:
//
// #Assimp::BaseImporter::SearchFileHeaderForToken - for text files.
// It reads the first lines of the file and does a substring check
// against a given list of 'magic' strings.
//
// #Assimp::BaseImporter::CheckMagicToken - for binary files. It goes
// to a particular offset in the file and and compares the next words
// against a given list of 'magic' tokens.
// These checks MUST be done (even if !checkSig) if the file extension
// is not exclusive to your format. For example, .xml is very common
// and (co)used by many formats.
}
return false;
}
// Get list of file extensions handled by this loader
void xxxxImporter::GetExtensionList(std::set<std::string>& extensions) {

extensions.insert("xxx");
;
//  Adam  Adam  Adam  Adam  Adam
void xxxxImporter::InternReadFile( const std::string& pFile,
aiScene* pScene, IOSystem* pIOHandler)
{
boost::scoped_ptr<IOStream> file( pIOHandler->Open( pFile, "rb"));
// Check whether we can read from the file
if( file.get() == NULL) {
throw DeadlyImportError( "Failed to open xxxx file " + pFile + ".");
}
// Your task: fill pScene
// Throw a ImportErrorException with a meaningful (!) error message if
// something goes wrong.
}
## Assimp v3.1.1 (June 2014)

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Namespaces</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
</table>

**Animation Overview**
Transformations

This diagram shows how you can calculate your transformation matrices for an animated character:

\[
\text{Final Transform}_{\text{Head}} = \text{Offset}_{\text{Head}} \times (\text{Scale} \times \text{Rot} \times \text{Trans})_{\text{Head}} \times (\text{Scale} \times \text{Rot} \times \text{Trans})_{\text{Spine}} \times (\text{Scale} \times \text{Rot} \times \text{Trans})_{\text{Pelvis}}
\]
Deprecated List

Member aiProcess_ConvertToLeftHanded

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Namespace List

Here is a list of all namespaces with brief descriptions:

Namespace List
Assimp's CPP-API and all internal APIs

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp v3.1.1 (June 2014)

Assimp Namespace Reference

Assimp's CPP-API and all internal APIs. More...
Classes

class **BaseImporter**

FOR IMPORTER PLUGINS ONLY: The **BaseImporter** defines a common interface for all importer worker classes. More...

class **DefaultLogger**

CPP-API: Primary logging facility of Assimp. More...

class **Exporter**

class **ExportProperties**

class **Importer**

CPP-API: The **Importer** class forms an C++ interface to the functionality of the Open Asset Import Library. More...

struct **Interpolator**

CPP-API: Utility class to simplify interpolations of various data types. More...

class **IStream**

CPP-API: Class to handle file I/O for C++. More...

class **IOSystem**
CPP-API: Interface to the file system.  

class **Logger**

CPP-API: Abstract interface for logger implementations.  

class **LogStream**

CPP-API: Abstract interface for log stream implementations.  

class **NullLogger**

CPP-API: Empty logging implementation.  

class **ProgressHandler**

CPP-API: Abstract interface for custom progress report receivers.  

struct **ScopeGuard**
Variables

class ExportProperties

CPP-API: The Exporter class forms an C++ interface to the export functionality of the Open Asset Import Library. More...
Detailed Description

Assimp's CPP-API and all internal APIs.
Variable Documentation

class ASSIMP_API Assimp::ExportProperties

CPP-API: The **Exporter** class forms an C++ interface to the export functionality of the Open Asset Import Library.

Note that the export interface is available only if Assimp has been built with ASSIMP_BUILD_NO_EXPORT not defined.

The interface is modelled after the importer interface and mostly symmetric. The same rules for threading etc. apply.

In a nutshell, there are two export interfaces: #Export, which writes the output file(s) either to the regular file system or to a user-supplied #IOSystem, and #ExportToBlob which returns a linked list of memory buffers (blob), each referring to one output file (in most cases there will be only one output file of course, but this extra complexity is needed since Assimp aims at supporting a wide range of file formats).

#ExportToBlob is especially useful if you intend to work with the data in-memory.
Here is a list of all namespace members with links to the namespace documentation for each member:

- ExportProperties : Assimp
• ExportProperties : Assimp
Assimp v3.1.1 (June 2014)

Class List
Here are the classes, structs, unions and interfaces with brief descriptions:
[detail level 123]

▼ NAssimp
Assimp's CPP-API and all internal APIs
FOR IMPORTER PLUGINS ONLY: The BaseImporter

CBaseImporter
defines a common interface for all importer worker classes

CDefaultLogger
CPP-API: Primary logging facility of Assimp

▼ CExporter
Internal description of an Assimp export format option

CExportProperties

CImporter
CPP-API: The Importer class forms an C++ interface to the functionality of the Open Asset Import Library

CInterpolator
CPP-API: Utility class to simplify interpolations of various data types

CIOStream
CPP-API: Class to handle file I/O for C++

CIOSystem
CPP-API: Interface to the file system

CLogger
CPP-API: Abstract interface for logger implementations

CLogStream
CPP-API: Abstract interface for log stream implementations

CNullLogger
CPP-API: Empty logging implementation

CProgressHandler
CPP-API: Abstract interface for custom progress report receivers

CScopeGuard

CaiAnimation
An animation consists of keyframe data for a number of nodes

CaiAnimMesh
NOT CURRENTLY IN USE

CaiBone
A single bone of a mesh
CaiCamera: Helper structure to describe a virtual camera
CaiColor3D: Represents a color in Red-Green-Blue space
CaiColor4t: Represents a color in Red-Green-Blue space including an alpha component
CaiExportDataBlob: Describes a blob of exported scene data
CaiExportFormatDesc: Describes an file format which Assimp can export to
CaiFace: A single face in a mesh, referring to multiple vertices
CaiFile: C-API: File callbacks
CaiFileIO: C-API: File system callbacks
CaiImporterDesc: Meta information about a particular importer
CaiLight: Helper structure to describe a light source
CaiLogStream: C-API: Represents a log stream
CaiMaterial: Data structure for a material
CaiMatrix3x3t: Represents a row-major 3x3 matrix
CaiMatrix4x4t: Represents a row-major 4x4 matrix, use this for homogeneous coordinates
CaiMemoryInfo: Stores the memory requirements for different components (e.g.
CaiMesh: A mesh represents a geometry or model with a single material
CaiMeshAnim: Describes vertex-based animations for a single mesh or a group of meshes
CaiMeshKey: Binds a anim mesh to a specific point in time
CaiMetadata: Container for holding metadata
CaiMetadataEntry: Metadata entry
CaiNode: A node in the imported hierarchy
CaiNodeAnim: Describes the animation of a single node
CaiPlane: Represents a plane in a three-dimensional, euclidean space
CaiPropertyStore: C-API: Represents an opaque set of settings to be used during importing
CaiQuatKey: Represents a quaternion in a 4D vector
CaiQuatKey: A time-value pair specifying a rotation for the given time
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaiRay</td>
<td>Represents a ray</td>
</tr>
<tr>
<td>CaiScene</td>
<td>The root structure of the imported data</td>
</tr>
<tr>
<td>CaiString</td>
<td>Represents an UTF-8 string, zero byte terminated</td>
</tr>
<tr>
<td>CaiTexel</td>
<td>Helper structure to represent a texel in a ARGB8888 format</td>
</tr>
<tr>
<td>CaiTexture</td>
<td>Helper structure to describe an embedded texture</td>
</tr>
<tr>
<td>CaiUVTransform</td>
<td>Defines how an UV channel is transformed</td>
</tr>
<tr>
<td>CaiVector2t</td>
<td>Represents a two-dimensional vector</td>
</tr>
<tr>
<td>CaiVector3t</td>
<td>Represents a three-dimensional vector</td>
</tr>
<tr>
<td>CaiVectorKey</td>
<td>A time-value pair specifying a certain 3D vector for the given time</td>
</tr>
<tr>
<td>CaiVertexWeight</td>
<td>A single influence of a bone on a vertex</td>
</tr>
</tbody>
</table>
Assimp v3.1.1 (June 2014)

- Assimp
- BaseImporter

FOR IMPORTER PLUGINS ONLY: The BaseImporter defines a common interface for all importer worker classes. More...
Public Types
Public Member Functions

**BaseImporter ()**
Constructor to be privately used by **Importer. More...**

**virtual bool CanRead (const std::string &pFile, IOSystem *pIOHandler, bool checkSig) const =0**
Returns whether the class can handle the format of the given file. **More...**

**const std::string & GetErrorText () const**
Returns the error description of the last error that occurred. **More...**

**void GetExtensionList (std::set< std::string > &extensions)**
Called by #Importer::GetExtensionList for each loaded importer. **More...**

**virtual const aiImporterDesc * GetInfo () const =0**
Called by #Importer::GetImporterInfo to get a description of some loader features. **More...**

**aiScene * ReadFile (const Importer *pImp, const std::string &pFile, IOSystem *pIOHandler)**
Imports the given file and returns the imported data. **More...**

**virtual void SetupProperties (const Importer *pImp)**
Called prior to **ReadFile(). More...**

**virtual ~BaseImporter ()**
Destructor, private as well. **More...**
### Static Public Member Functions

**CheckMagicToken** *(IOSystem *pIOHandler, const std::string &pFile, const void *magic, unsigned int num, unsigned int offset=0, unsigned int size=4)*

Check whether a file starts with one or more magic tokens.  [More...](#)

**ConvertToUTF8** *(std::vector<char> &data)*

An utility for all text file loaders.  [More...](#)

**ConvertUTF8toISO8859_1** *(std::string &data)*

An utility for all text file loaders.  [More...](#)

**CopyVector** *(std::vector<T> &vec, T *&out, unsigned int &outLength)*

Utility function to move a std::vector into an aiScene array.  [More...](#)

**GetExtension** *(const std::string &pFile)*

Extract file extension from a string.  [More...](#)

**SearchFileHeaderForToken** *(IOSystem *pIOSystem, const std::string &file, const char **tokens, unsigned int numTokens, unsigned int searchBytes=200, bool tokensSol=false)*

A utility for CanRead().  [More...](#)

**SimpleExtensionCheck** *(const std::string &pFile, const char *ext0, const char *ext1=NULL, const char *ext2=NULL)*
Check whether a file has a specific file extension. More...

```cpp
static void TextFileToBuffer (IStream *stream, std::vector< char > &data, TextFileMode mode=FORBID_EMPTY)
```
Utility for text file loaders which copies the contents of the file into a memory buffer and converts it to our UTF8 representation. More...
Protected Member Functions

```cpp
virtual InternReadFile (const std::string &pFile, aiScene *pScene,_IOSystem void *pIOHandler)=0
    Imports the given file into the given scene structure. More...
```
Protected Attributes

std::string m_ErrorText
    Error description in case there was one. More...

ProgressHandler * m_progress
    Currently set progress handler. More...
Friends

class Importer
**Detailed Description**

FOR IMPORTER PLUGINS ONLY: The `BaseImporter` defines a common interface for all importer worker classes.

The interface defines two functions: `CanRead()` is used to check if the importer can handle the format of the given file. If an implementation of this function returns true, the importer then calls `ReadFile()` which imports the given file. `ReadFile` is not overridable, it just calls `InternReadFile()` and catches any `ImportErrorException` that might occur.
**Member Enumeration Documentation**

define enum `Assimp::BaseImporter::TextFileMode`  

**Enumerator**
- ALLOW_EMPTY
- FORBID_EMPTY
Constructor & Destructor Documentation

Assimp::BaseImporter::BaseImporter ( )

Constructor to be privately used by Importer.

virtual Assimp::BaseImporter::~BaseImporter ( ) virtual

Destructor, private as well.
**Member Function Documentation**

```cpp
class Assimp::BaseImporter
{
public:
    virtual bool CanRead(const std::string &pFile, IOSystem *pIOHandler, bool checkSig) const = 0;
}
```

Returns whether the class can handle the format of the given file.

The implementation should be as quick as possible. A check for the file extension is enough. If no suitable loader is found with this strategy, `CanRead()` is called again, the 'checkSig' parameter set to true this time. Now the implementation is expected to perform a full check of the file structure, possibly searching the first bytes of the file for magic identifiers or keywords.

**Parameters**

- `pFile` Path and file name of the file to be examined.
- `pIOHandler` The IO handler to use for accessing any file.
- `checkSig` Set to true if this method is called a second time. This time, the implementation may take more time to examine the contents of the file to be loaded for magic bytes, keywords, etc to be able to load files with unknown/not existent file extensions.

**Returns**

true if the class can read this file, false if not.

```cpp
class Assimp::BaseImporter
{
public:
    static bool CheckMagicToken(IOSystem *pIOHandler, const std::string &pFile, const void *magic, unsigned int num, unsigned int offset = 0);
}
```

static
Check whether a file starts with one or more magic tokens.

Parameters

- `pFile`: Input file
- `pIOHandler`: IO system to be used
- `magic`: `n` magic tokens
- `num`: Size of magic
- `offset`: Offset from file start where tokens are located
- `Size`: Size of one token, in bytes. Maximally 16 bytes.

Returns

- `true` if one of the given tokens was found

Note

For convenience, the check is also performed for the byte-swapped variant of all tokens (big endian). Only for tokens of size 2, 4.

```cpp
static void Assimp::BaseImporter::ConvertToUTF8(std::vector<char>& data) static
```

An utility for all text file loaders.

It converts a file to our UTF8 character set. Errors are reported, but ignored.

Parameters

- `data`: File buffer to be converted to UTF8 data. The buffer is resized as appropriate.

```cpp
static void Assimp::BaseImporter::ConvertUTF8toISO8859_1(std::string& data) static
```

An utility for all text file loaders.

It converts a file from our UTF8 character set back to ISO-8859-1. Errors are reported, but ignored.
Parameters
data File buffer to be converted from UTF8 to ISO-8859-1. The buffer is resized as appropriate.

template<typename T >
static AI_FORCE_INLINE void Assimp::BaseImporter::CopyVector(std::vector<T> & vec, T *& out, unsigned int & outLength)

Utility function to move a std::vector into an aiScene array.

Parameters
vec The vector to be moved
out The output pointer to the allocated array.
umOut The output count of elements copied.

const std::string& Assimp::BaseImporter::GetErrorText() const inline
Returns the error description of the last error that occurred.

Returns
A description of the last error that occurred. An empty string if there was no error.

static std::string Assimp::BaseImporter::GetExtension(const std::string & pFile) static
Extract file extension from a string.

Parameters
pFile Input file

Returns
Extension without trailing dot, all lowercase
void Assimp::BaseImporter::GetExtensionList ( std::set< std::string > & extensions )

Called by #Importer::GetExtensionList for each loaded importer.

Take the extension list contained in the structure returned by getInfo and insert all file extensions into the given set.

Parameters
  extension set to collect file extensions in

virtual const aiImporterDesc* Assimp::BaseImporter::GetInfo ( ) const pure virtual

Called by #Importer::GetImporterInfo to get a description of some loader features.

Importers must provide this information.

virtual void Assimp::BaseImporter::InternReadFile ( std::string pFile, const aiScene * pScene, protected pure virtual

Imports the given file into the given scene structure.

The function is expected to throw an ImportErrorException if there is an error. If it terminates normally, the data in aiScene is expected to be correct. Override this function to implement the actual importing.

The output scene must meet the following requirements:

- At least a root node must be there, even if its only purpose is to reference one mesh.
- aiMesh::mPrimitiveTypes may be 0. The types of primitives in the mesh are
determined automatically in this case.

- the vertex data is stored in a pseudo-indexed "verbose" format. In fact this means that every vertex that is referenced by a face is unique. Or the other way round: a vertex index may not occur twice in a single aiMesh.
- aiAnimation::mDuration may be -1. Assimp determines the length of the animation automatically in this case as the length of the longest animation channel.
- aiMesh::mBitangents may be NULL if tangents and normals are given. In this case bitangents are computed as the cross product between normal and tangent.
- There needn't be a material. If none is there a default material is generated. However, it is recommended practice for loaders to generate a default material for yourself that matches the default material setting for the file format better than Assimp's generic default material. Note that default materials should be named AI_DEFAULT_MATERIAL_NAME if they're just color-shaded or AI_DEFAULT_TEXTURED_MATERIAL_NAME if they define a (dummy) texture.

If the AI_SCENE_FLAGS_INCOMPLETE-Flag is not set:

- at least one mesh must be there
- there may be no meshes with 0 vertices or faces

This won't be checked (except by the validation step): Assimp will crash if one of the conditions is not met!

Parameters

pFile Path of the file to be imported.
pScene The scene object to hold the imported data. NULL is not a valid parameter.
pIOHandler The IO handler to use for any file access. NULL is not a valid parameter.

aiScene* Assimp::BaseImporter::ReadFile ( const Importer * pImp,
const std::string & pFile,
IOSystem * pIOHandler
)
Imports the given file and returns the imported data.

If the import succeeds, ownership of the data is transferred to the caller. If the import fails, NULL is returned. The function takes care that any partially constructed data is destroyed beforehand.

Parameters
- `pImp` : Importer object hosting this loader.
- `pFile` : Path of the file to be imported.
- `pIOHandler` : IO-Handler used to open this and possible other files.

Returns
The imported data or NULL if failed. If it failed a human-readable error description can be retrieved by calling `GetErrorText()`.

Note
This function is not intended to be overridden. Implement `InternReadFile()` to do the import. If an exception is thrown somewhere in `InternReadFile()`, this function will catch it and transform it into a suitable response to the caller.

```cpp
static bool Assimp::BaseImporter::SearchFileHeaderForToken
    ( *pIOSystem,
    const std::string & file,
    &
    const char ** tokens,
    unsigned int numTokens,
    unsigned int searchBytes = 200,
    bool tokensSol = false )
```

A utility for `CanRead()`.
The function searches the header of a file for a specific token and returns true if this token is found. This works for text files only. There is a rudimentary handling of UNICODE files. The comparison is case independent.

Parameters

- pIOSystem: IO System to work with
- file: File name of the file
- tokens: List of tokens to search for
- numTokens: Size of the token array
- searchBytes: Number of bytes to be searched for the tokens.

virtual void
Assimp::BaseImporter::SetupProperties
( const Importer* pImp ) virtual

Called prior to ReadFile().

The function is a request to the importer to update its configuration basing on the Importer's configuration property list.

Parameters

- pImp: Importer instance

static bool
Assimp::BaseImporter::SimpleExtensionCheck
( const std::string& pFile,
  const char* ext0,
  const char* ext1 = NULL,
  const char* ext2 = NULL ) static

Check whether a file has a specific file extension.

Parameters

- pFile: Input file
  - ext0: Extension to check for. Lowercase characters only, no dot!
ext1  Optional second extension
ext2  Optional third extension

Note
Case-insensitive

static void Assimp::BaseImporter::TextFileToBuffer ( IOSTream * stream,
std::vector<char> & data,
TextFileMode mode = FORBID_EMPTY )

Utility for text file loaders which copies the contents of the file into a memory
buffer and converts it to our UTF8 representation.

Parameters

stream Stream to read from.
data   Output buffer to be resized and filled with the converted text file
data. The buffer is terminated with a binary 0.
mode   Whether it is OK to load empty text files.
Friends And Related Function Documentation

friend class Importer friend
Member Data Documentation

std::string Assimp::BaseImporter::m_ErrorText protected

Error description in case there was one.

ProgressHandler* Assimp::BaseImporter::m_progress protected

Currently set progress handler.

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

- BaseImporter.h

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp v3.1.1 (June 2014)

- Assimp
- DefaultLogger

Public Member Functions | Static Public Member Functions | List of all members
Assimp::DefaultLogger Class Reference

CPP-API: Primary logging facility of Assimp. More...

Inherits Assimp::Logger.
Public Member Functions

bool attachStream (LogStream *pStream, unsigned int severity)
Attach a new log-stream. More...

bool detachStream (LogStream *pStream, unsigned int severity)
Detach a still attached stream from the logger (or modify the filter flags bits) More...

- Public Member Functions inherited from Assimp::Logger
  void debug (const char *message)
  Writes a debug message. More...

  void debug (const std::string &message)

  void error (const char *message)
  Writes an error message. More...

  void error (const std::string &message)

  LogSeverity getLogSeverity () const
  Get the current log severity. More...

  void info (const char *message)
  Writes a info message. More...

  void info (const std::string &message)

  void setLogSeverity (LogSeverity log_severity)
  Set a new log severity. More...

  void warn (const char *message)
Writes a warning message. More...

void warn (const std::string &message)

virtual ~Logger ()

Virtual destructor. More...
Static Public Member Functions

```c
static Logger
* create (const char *name=ASSIMP_DEFAULT_LOG_NAME,
LogSeverity severity=NORMAL,
unsigned int defStreams=aiDefaultLogStream_DEBUGGER|aiDefaultLogStream_FILE,
IOSystem *io=NULL)

Creates a logging instance. More...
```

```c
static Logger
* get ()

* Getter for singleton instance. More...
```

```c
static bool
isNullLogger ()

* Return whether a #NullLogger is currently active. More...
```

```c
static void
kill ()

* Kills the current singleton logger and replaces it with a #NullLogger instance. More...
```

```c
static void
set (Logger *logger)

* Setup a custom Logger implementation. More...
```
Additional Inherited Members

- Public Types inherited from `Assimp::Logger`
- Protected Member Functions inherited from `Assimp::Logger`
  
  `Logger ()`
  Default constructor. [More...](#)

  `Logger (LogSeverity severity)`
  Construction with a given log severity. [More...](#)

- Protected Attributes inherited from `Assimp::Logger`
  `LogSeverity m_Severity`
  `Logger severity. [More...](#)`
Detailed Description

CPP-API: Primary logging facility of Assimp.

The library stores its primary Logger as a static member of this class. get() returns this primary logger. By default the underlying implementation is just a #NullLogger which rejects all log messages. By calling create(), logging is turned on. To capture the log output multiple log streams (#LogStream) can be attach to the logger. Some default streams for common streaming locations (such as a file, std::cout, OutputDebugString()) are also provided.

If you wish to customize the logging at an even deeper level supply your own implementation of Logger to set().

Note
The whole logging stuff causes a small extra overhead for all imports.
**Member Function Documentation**

```cpp
bool Assimp::DefaultLogger::attachStream ( LogStream * pStream,
                                          unsigned int severity virtual
)
```

Attach a new log-stream.

The logger takes ownership of the stream and is responsible for its destruction (which is done using ::delete when the logger itself is destroyed). Call detachStream to detach a stream and to gain ownership of it again.

Parameters
- `pStream` Log-stream to attach
  - Message filter, specified which types of log messages are dispatched to the stream. Provide a bitwise combination of the ErrorSeverity flags.

Returns
- true if the stream has been attached, false otherwise.

Implements Assimp::Logger.

```cpp
static Logger* Assimp::DefaultLogger::create ( const char * name = ASSIMP_DEFAULT_LOG_NAME,
                                               LogSeverity severity = NORMAL,
                                               unsigned int defStreams = aiDefaultLogStream_DEBUGGER |
                                               IOSystem * io = NULL
)
```

Creates a logging instance.

Parameters
- `name` Name for log file. Only valid in combination with the aiDefaultLogStream_FILE flag.
severity Log severity, VERBOSE turns on debug messages
Default log streams to be attached. Any bitwise combination of
the aiDefaultLogStream enumerated values. If
aiDefaultLogStream_FILE is specified but an empty string is
passed for 'name', no log file is created at all.

IOSystem to be used to open external files (such as the log
file). Pass NULL to rely on the default implementation. This
replaces the default #NullLogger with a #DefaultLogger
instance.

bool Assimp::DefaultLogger::detachStream ( LogStream * pStream,
unsigned int severity  virtual
)

Detach a still attached stream from the logger (or modify the filter flags bits)

Parameters
pStream Log-stream instance for detaching
severity Provide a bitwise combination of the ErrorSeverity flags. This
value is &~ed with the current flags of the stream, if the result is 0
the stream is detached from the Logger and the caller retakes the
possession of the stream.

Returns
true if the stream has been detached, false otherwise.

Implements Assimp::Logger.

static Logger* Assimp::DefaultLogger::get ( ) static

Getter for singleton instance.

Returns
Only instance. This is never null, but it could be a NullLogger. Use
isNullLogger to check this.

static bool Assimp::DefaultLogger::isNullLogger ( ) static
Return whether a #NullLogger is currently active.

Returns
true if the current logger is a #NullLogger. Use create() or set() to setup a logger that does actually do something else than just rejecting all log messages.

static void Assimp::DefaultLogger::kill ( ) static

Kills the current singleton logger and replaces it with a #NullLogger instance.

static void Assimp::DefaultLogger::set ( Logger * logger ) static

Setup a custom Logger implementation.

Use this if the provided #DefaultLogger class doesn't fit into your needs. If the provided message formatting is OK for you, it's much easier to use create() and to attach your own custom output streams to it.

Parameters
logger Pass NULL to setup a default NullLogger

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

- DefaultLogger.hpp

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp  v3.1.1 (June 2014)

- Assimp
- Exporter

Classes | Public Types | Public Member Functions | Protected Attributes | List of all members
Assimp::Exporter Class Reference
Classes

struct ExportFormatEntry

Internal description of an Assimp export format option. More...
Public Types

typedef fpExportFunc (const char *, IOSystem *, const aiScene *, const ExportProperties *)

Function pointer type of a Export worker function. More...
Public Member Functions

aiReturn Export (const aiScene *pScene, const char *pFormatId, const char *pPath, unsigned int pPreprocessing=0u, const ExportProperties *pProperties=NULL)
Convenience function to export directly to a file. More...

aiReturn Export (const aiScene *pScene, const std::string &pFormatId, const std::string &pPath, unsigned int pPreprocessing=0u, const ExportProperties *pProperties=NULL)

Exporter ()

const aiExportDataBlob * ExportToBlob (const aiScene *pScene, const char *pFormatId, unsigned int pPreprocessing=0u, const ExportProperties *pProperties=NULL)
Exports the given scene to a chosen file format. More...

const aiExportDataBlob * ExportToBlob (const aiScene *pScene, const std::string &pFormatId, unsigned int pPreprocessing=0u, const ExportProperties *pProperties=NULL)

void FreeBlob ()
Frees the current blob. More...

const aiExportDataBlob * GetBlob () const
Return the blob obtained from the last call to ExportToBlob. More...

const char * GetErrorString () const
Returns an error description of an error that occurred in Export or ExportToBlob. More...
size_t GetExportFormatCount() const
    Returns the number of export file formats available in the current Assimp build. More...

const aiExportFormatDesc* GetExportFormatDescription(size_t pIndex) const
    Returns a description of the nth export file format. More...

IOSystem* GetIOHandler() const
    Retrieves the IO handler that is currently set. More...

const aiExportDataBlob* GetOrphanedBlob() const
    Orphan the blob from the last call to ExportToBlob. More...

bool IsDefaultIOHandler() const
    Checks whether a default IO handler is active A default handler is active as long the application doesn't supply its own custom IO handler via SetIOHandler(). More...

aiReturn RegisterExporter(const ExportFormatEntry &desc)
    Register a custom exporter. More...

void SetIOHandler(IOSystem*pIOHandler)
    Supplies a custom IO handler to the exporter to use to open and access files. More...

void UnregisterExporter(const char *id)
    Remove an export format previously registered with RegisterExporter from the Exporter instance (this can also be used to drop builtin exporters because those are implicitly registered using RegisterExporter). More...
~Exporter ()
Protected Attributes

ExporterPimpl *  pimpl
Member Typedef Documentation

typedef void(* Assimp::Exporter::fpExportFunc) (const char *, IOSystem *, const aiScene *, const ExportProperties *)

Function pointer type of a Export worker function.
Constructor & Destructor Documentation

Assimp::Exporter::Exporter ( )
Assimp::Exporter::~Exporter ( )
Member Function Documentation

aiReturn
Assimp::Exporter::Export
( const aiScene * pScene,
 const char * pFormatId,
 const char * pPath,
 unsigned int pPreprocessing = 0u,
 const ExportProperties * pProperties = NULL )

Convenience function to export directly to a file.

Use #SetIOSystem to supply a custom IOSystem to gain fine-grained control about the output data flow of the export process.

Parameters

pBlob A data blob obtained from a previous call to aiExportScene. Must not be NULL.
pPath Full target file name. Target must be accessible.
Accepts any choice of the aiPostProcessSteps enumerated flags, but in reality only a subset of them makes sense here. Specifying 'preprocessing' flags is useful if the input scene does not conform to Assimp's default conventions as specified in the Data Structures Page. In short, this means the geometry data should use a right-handed coordinate systems, face winding should be counter-clockwise and the UV coordinate origin is assumed to be in the upper left. The aiProcess_MakeLeftHanded, aiProcess_FlipUVs and aiProcess_FlipWindingOrder flags are used in the import side to allow users to have those defaults automatically adapted to their conventions. Specifying those flags for exporting has the opposite effect, respectively. Some other of the aiPostProcessSteps enumerated values may be useful as well, but you'll need to try out what their effect on the
exported file is. Many formats impose their own restrictions on the structure of the geometry stored therein, so some preprocessing may have little or no effect at all, or may be redundant as exporters would apply them anyhow. A good example is triangulation - whilst you can enforce it by specifying the \texttt{aiProcess_Triangulate} flag, most export formats support only triangulate data so they would run the step even if it wasn't requested.

If assimp detects that the input scene was directly taken from the importer side of the library (i.e. not copied using \texttt{aiCopyScene} and potentially modified afterwards), any postprocessing steps already applied to the scene will not be applied again, unless they show non-idempotent behaviour (\texttt{aiProcess_MakeLeftHanded}, \texttt{aiProcess_FlipUVs} and \texttt{aiProcess_FlipWindingOrder}).

Returns \texttt{AI_SUCCESS} if everything was fine.

Note

Use \texttt{aiCopyScene()} to get a modifiable copy of a previously imported scene.

\texttt{aiReturn Assimp::Exporter::Export}\ ( const \texttt{aiScene} * \texttt{pScene},
const std::string & \texttt{pFormatId},
const std::string & \texttt{pPath},
unsigned int \texttt{pPreprocessing} = \texttt{0u},
const \texttt{ExportProperties} * \texttt{pProperties} = \texttt{NULL}\ )

\texttt{const aiExportDataBlob* Assimp::Exporter::ExportToBlob}\ ( const \texttt{aiScene} * \texttt{pScene},
const char * \texttt{pFormatId},
unsigned int \texttt{pPreprocessing} = \texttt{0u},
...
Exports the given scene to a chosen file format.

Returns the exported data as a binary blob which you can write into a file or something. When you're done with the data, simply let the Exporter instance go out of scope to have it released automatically.

**Parameters**

- **pScene**
  The scene to export. Stays in possession of the caller, is not changed by the function.
- **pFormatId**
  ID string to specify to which format you want to export to. Use GetExportFormatCount / GetExportFormatDescription to learn which export formats are available.
- **pPreprocessing**
  See the documentation for Export

**Returns**

the exported data or NULL in case of error.

**Note**

If the Exporter instance did already hold a blob from a previous call to ExportToBlob, it will be disposed. Any IO handlers set via SetIOHandler are ignored here.

Use aiCopyScene() to get a modifiable copy of a previously imported scene.

```cpp
const aiExportDataBlob * Assimp::Exporter::ExportToBlob(
    const aiScene * pScene,
    const std::string & pFormatId,
    unsigned int pPreprocessing = 0u,
    const ExportProperties * pProperties = NULL
) const
```
void Assimp::Exporter::FreeBlob ( )

Frees the current blob.

The function does nothing if no blob has previously been previously produced via ExportToBlob. FreeBlob is called automatically by the destructor. The only reason to call it manually would be to reclaim as much storage as possible without giving up the Exporter instance yet.

const aiExportDataBlob* Assimp::Exporter::GetBlob ( ) const

Return the blob obtained from the last call to ExportToBlob.

const char* Assimp::Exporter::GetErrorString ( ) const

Returns an error description of an error that occurred in Export or ExportToBlob.

Returns an empty string if no error occurred.

Returns
A description of the last error, an empty string if no error occurred. The string is never NULL.

Note
The returned function remains valid until one of the following methods is called: Export, ExportToBlob, FreeBlob

size_t Assimp::Exporter::GetExportFormatCount ( ) const

Returns the number of export file formats available in the current Assimp build.

Use #Exporter::GetExportFormatDescription to retrieve infos of a specific export format.

This includes built-in exporters as well as exporters registered using RegisterExporter.

const aiExportFormatDesc*
Assimp::Exporter::GetExportFormatDescription ( size_t pIndex ) const

Returns a description of the nth export file format.

Use # #Exporter::GetExportFormatCount to learn how many export formats are supported.

The returned pointer is of static storage duration if the pIndex pertains to a built-in exporter (i.e. one not registered via #RegisterExporter). It is restricted to the life-time of the Exporter instance otherwise.

Parameters

pIndex Index of the export format to retrieve information for. Valid range is 0 to #Exporter::GetExportFormatCount

Returns

A description of that specific export format. NULL if pIndex is out of range.

IOSystem* Assimp::Exporter::GetIOHandler ( ) const

Retrieves the IO handler that is currently set.

You can use IsDefaultIOHandler() to check whether the returned interface is the default IO handler provided by ASSIMP. The default handler is active as long the application doesn't supply its own custom IO handler via SetIOHandler().

Returns

A valid IOSystem interface, never NULL.

const aiExportDataBlob* Assimp::Exporter::GetOrphanedBlob ( ) const

Orphan the blob from the last call to ExportToBlob.

This means the caller takes ownership and is thus responsible for calling the C API function aiReleaseExportBlob to release it.

bool Assimp::Exporter::IsDefaultIOHandler ( ) const
Checks whether a default IO handler is active A default handler is active as long the application doesn't supply its own custom IO handler via `SetIOHandler()`.

Returns
true by default

```c
aiReturn Assimp::Exporter::RegisterExporter ( const ExportFormatEntry & desc )
```

Register a custom exporter.

Custom export formats are limited to the current `Exporter` instance and do not affect the library globally. The indexes under which the format's export format description can be queried are assigned monotonously.

Parameters
desc `Exporter` description.

Returns
aiReturn_SUCCESS if the export format was successfully registered. A common cause that would prevent an exporter from being registered is that its format id is already occupied by another format.

```c
void Assimp::Exporter::SetIOHandler ( IOSystem * pIOHandler )
```

Supplies a custom IO handler to the exporter to use to open and access files.

If you need `Export` to use custom IO logic to access the files, you need to supply a custom implementation of `IOSystem` and IOFile to the exporter.

`Exporter` takes ownership of the object and will destroy it afterwards. The previously assigned handler will be deleted. Pass NULL to take again ownership of your `IOSystem` and reset Assimp to use its default implementation, which uses plain file IO.

Parameters
pIOHandler The IO handler to be used in all file accesses of the `Importer`.

```c
void Assimp::Exporter::UnregisterExporter ( const char * id )
```
Remove an export format previously registered with `RegisterExporter` from the `Exporter` instance (this can also be used to drop builtin exporters because those are implicitly registered using `RegisterExporter`).

Parameters

- `id` Format id to be unregistered, this refers to the 'id' field of `aiExportFormatDesc`.

Note

- Calling this method on a format description not yet registered has no effect.
Member Data Documentation

ExporterPimpl* Assimp::Exporter::pimpl protected

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

- **Exporter.hpp**
Internal description of an Assimp export format option. More...
Public Member Functions

ExportFormatEntry (const char *pId, const char *pDesc, const char *pExtension, fpExportFunc pFunction, unsigned int pEnforcePP=0u)

ExportFormatEntry ()
Public Attributes

\texttt{aiExportFormatDesc} \hspace{1em} \texttt{mDescription}

Public description structure to be returned by \texttt{aiGetExportFormatDescription()}

\texttt{unsigned int} \hspace{1em} \texttt{mEnforcePP}

\texttt{fpExportFunc} \hspace{1em} \texttt{mExportFunction}
Detailed Description

Internal description of an Assimp export format option.
Constructor & Destructor Documentation

Assimp::Exporter::ExportFormatEntry::ExportFormatEntry ( const char * pId,
const char * pDesc
const char * pEx
fpExportFunc pFu
unsigned int pEn
= 0u )

Assimp::Exporter::ExportFormatEntry::ExportFormatEntry ( ) inline
Member Data Documentation

aiExportFormatDesc Assimp::Exporter::ExportFormatEntry::mDescription

Public description structure to be returned by aiGetExportFormatDescription()

unsigned int Assimp::Exporter::ExportFormatEntry::mEnforcePP

fpExportFunc Assimp::Exporter::ExportFormatEntry::mExportFunction

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

- Exporter.hpp

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Public Types

typedef std::map< KeyType, float > FloatPropertyMap

typedef std::map< KeyType, int > IntPropertyMap

typedef unsigned int KeyType

typedef std::map< KeyType, aiMatrix4x4 > MatrixPropertyMap

typedef std::map< KeyType, std::string > StringPropertyMap
Public Member Functions

**ExportProperties()**
Standard constructor. [More...](#)

**ExportProperties(const ExportProperties &other)**
Copy constructor. [More...](#)

**bool GetPropertyBool(const char *szName, bool bErrorReturn=false) const**
Get a boolean configuration property. [More...](#)

**float GetPropertyFloat(const char *szName, float fErrorReturn=10e10f) const**
Get a floating-point configuration property. [More...](#)

**int GetPropertyInteger(const char *szName, int iErrorReturn=0xffffffff) const**
Get a configuration property. [More...](#)

**const aiMatrix4x4 GetPropertyMatrix(const char *szName, const aiMatrix4x4 &sErrorReturn=aiMatrix4x4()) const**
Get a matrix configuration property. [More...](#)

**const std::string GetPropertyString(const char *szName, const std::string &sErrorReturn="") const**
Get a string configuration property. [More...](#)

**bool HasPropertyBool(const char *szName) const**
Determine a boolean configuration property has been set. [More...](#)

**bool HasPropertyFloat(const char *szName) const**
Determine a boolean configuration property has been set. [More...](#)
bool HasPropertyInteger (const char *szName) const  
Determine a integer configuration property has been set.  More...

bool HasPropertyMatrix (const char *szName) const  
Determine a Matrix configuration property has been set.  More...

bool HasPropertyString (const char *szName) const  
Determine a String configuration property has been set.  More...

bool SetPropertyBool (const char *szName, bool value)  
Set a boolean configuration property.  More...

bool SetPropertyFloat (const char *szName, float fValue)  
Set a floating-point configuration property.  More...

bool SetPropertyInteger (const char *szName, int iValue)  
Set an integer configuration property.  More...

bool SetPropertyMatrix (const char *szName, const aiMatrix4x4 &sValue)  
Set a matrix configuration property.  More...

bool SetPropertyString (const char *szName, const std::string &sValue)  
Set a string configuration property.  More...
Protected Attributes

**FloatPropertyMap**  **mFloatProperties**
List of floating-point properties.  **More...**

**IntPropertyMap**  **mIntProperties**
List of integer properties.  **More...**

**MatrixPropertyMap**  **mMatrixProperties**
List of Matrix properties.  **More...**

**StringPropertyMap**  **mStringProperties**
List of string properties.  **More...**
typedef std::map<KeyType, float> Assimp::ExportProperties::FloatPropertyMap
typedef std::map<KeyType, int> Assimp::ExportProperties::IntPropertyMap
typedef unsigned int Assimp::ExportProperties::KeyType
typedef std::map<KeyType, aiMatrix4x4> Assimp::ExportProperties::MatrixPropertyMap
typedef std::map<KeyType, std::string> Assimp::ExportProperties::StringPropertyMap
Constructor & Destructor Documentation

Assimp::ExportProperties::ExportProperties ( )

Standard constructor.

See also
  ExportProperties()

Assimp::ExportProperties::ExportProperties ( const ExportProperties & other )

Copy constructor.

This copies the configuration properties of another ExportProperties.

See also
  ExportProperties(const ExportProperties& other)
Member Function Documentation

bool Assimp::ExportProperties::GetPropertyBool(const char * szName, bool bErrorReturn = false)

Get a boolean configuration property.

Boolean properties are stored on the integer stack internally so it's possible to set them via `SetPropertyBool` and query them with `GetPropertyBool` and vice versa.

See also `GetPropertyInteger()`

float Assimp::ExportProperties::GetPropertyFloat(const char * szName, float fErrorReturn = 10e10f)

Get a floating-point configuration property.

See also `GetPropertyInteger()`

int Assimp::ExportProperties::GetPropertyInteger(const char * szName, int iErrorReturn = 0xffffffff)

Get a configuration property.

Parameters
szName Name of the property. All supported properties are defined in 
the aiConfig.g header (all constants share the prefix 
AI_CONFIG_XXX).

iErrorReturn Value that is returned if the property is not found.

Returns 
Current value of the property

Note 
Property of different types (float, int, string ..) are kept on different lists, so 
calling SetPropertyInteger() for a floating-point property has no effect - the 
loader will call GetPropertyFloat() to read the property, but it won't be 
there.

const aiMatrix4x4 
Assimp::ExportProperties::GetPropertyMatrix 
( const char * szName, 
const aiMatrix4x4 & sErrorReturn = aiMatrix4x4() 
) const

Get a matrix configuration property.

The return value remains valid until the property is modified.

See also 
GetPropertyInteger()

const std::string 
Assimp::ExportProperties::GetPropertyString 
( const char * szName, 
const std::string & sErrorReturn = "" 
) const

Get a string configuration property.

The return value remains valid until the property is modified.
See also

\texttt{GetPropertyInteger()}

bool \texttt{Assimp::ExportProperties::HasPropertyBool ( const char * \textit{szName} ) const}

Determine a boolean configuration property has been set.

See also

\texttt{HasPropertyBool()}

bool \texttt{Assimp::ExportProperties::HasPropertyFloat ( const char * \textit{szName} ) const}

Determine a boolean configuration property has been set.

See also

\texttt{HasPropertyFloat()}

bool \texttt{Assimp::ExportProperties::HasPropertyInteger ( const char \* \textit{szName} ) const}

Determine a integer configuration property has been set.

See also

\texttt{HasPropertyInteger()}

bool \texttt{Assimp::ExportProperties::HasPropertyMatrix ( const char \* \textit{szName} ) const}

Determine a Matrix configuration property has been set.

See also

\texttt{HasPropertyMatrix()}

bool \texttt{Assimp::ExportProperties::HasPropertyString ( const char \* \textit{szName} ) const}

Determine a String configuration property has been set.

See also
bool Assimp::ExportProperties::SetPropertyBool ( const char * szName, bool value inline )

Set a boolean configuration property.

Boolean properties are stored on the integer stack internally so it's possible to set them via `SetPropertyBool` and query them with `GetPropertyBool` and vice versa.

See also

bool Assimp::ExportProperties::SetPropertyFloat ( const char * szName, float fValue )

Set a floating-point configuration property.

See also

bool Assimp::ExportProperties::SetPropertyInteger ( const char * szName, int iValue )

Set an integer configuration property.

Parameters
Name of the property. All supported properties are defined in the `szName aiConfig.g` header (all constants share the prefix `AI_CONFIG_XXX` and are simple strings).

iValue New value of the property

Returns
true if the property was set before. The new value replaces the previous value in this case.
Note

Property of different types (float, int, string ..) are kept on different stacks, so calling `SetPropertyInteger()` for a floating-point property has no effect - the loader will call `GetPropertyFloat()` to read the property, but it won't be there.

```cpp
bool Assimp::ExportProperties::SetPropertyMatrix ( const char * szName,
                                                  const aiMatrix4x4 & sValue
                                                )
```

Set a matrix configuration property.

See also

`SetPropertyInteger()`

```cpp
bool Assimp::ExportProperties::SetPropertyString ( const char * szName,
                                                  const std::string & sValue
                                                )
```

Set a string configuration property.

See also

`SetPropertyInteger()`
**Member Data Documentation**

**FloatPropertyMap** Assimp::ExportProperties::mFloatProperties protected

List of floating-point properties.

**IntPropertyMap** Assimp::ExportProperties::mIntProperties protected

List of integer properties.

**MatrixPropertyMap** Assimp::ExportProperties::mMatrixProperties protected

List of Matrix properties.

**StringPropertyMap** Assimp::ExportProperties::mStringProperties protected

List of string properties.

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

- **Exporter.hpp**
Assimp v3.1.1 (June 2014)

- Assimp
- Importer

Public Member Functions | Static Public Attributes | Protected Attributes | List of all members
Assimp::Importer Class Reference

CPP-API: The Importer class forms an C++ interface to the functionality of the Open Asset Import Library. More...
Public Member Functions

const aiScene * ApplyCustomizedPostProcessing (BaseProcess *rootProcess, bool requestValidation)

const aiScene * ApplyPostProcessing (unsigned int pFlags)
Apply post-processing to an already-imported scene. More...

void FreeScene ()
Frees the current scene. More...

const char * GetErrorString () const
Returns an error description of an error that occurred in ReadFile(). More...

void GetExtensionList (aiString &szOut) const
Get a full list of all file extensions supported by ASSIMP. More...

void GetExtensionList (std::string &szOut) const
Get a full list of all file extensions supported by ASSIMP. More...

BaseImporter * GetImporter (size_t index) const
Find the importer corresponding to a specific index. More...

BaseImporter * GetImporter (const char *szExtension) const
Find the importer corresponding to a specific file extension. More...

size_t GetImporterCount () const
Get the number of importers currently registered with Assimp. More...
size_t GetImporterIndex (const char *szExtension) const

Find the importer index corresponding to a specific file extension. More...

const aiImporterDesc* GetImporterInfo (size_t index) const

* Get meta data for the importer corresponding to a specific index. More...

IOSystem * GetIOHandler () const

Retrieves the IO handler that is currently set. More...

void GetMemoryRequirements (aiMemoryInfo &in) const

Returns the storage allocated by ASSIMP to hold the scene data in memory. More...

aiScene * GetOrphanedScene ()

Returns the scene loaded by the last successful call to ReadFile() and releases the scene from the ownership of the Importer instance. More...

ProgressHandler * GetProgressHandler () const

Retrieves the progress handler that is currently set. More...

bool GetPropertyBool (const char *szName, bool bErrorReturn=false) const

Get a boolean configuration property. More...

float GetPropertyFloat (const char *szName, float fErrorReturn=10e10f) const

Get a floating-point configuration property. More...
int GetPropertyInteger (const char *szName, int iErrorReturn=0xffffffff) const
Get a configuration property. More...

const aiMatrix4x4 GetPropertyMatrix (const char *szName, const aiMatrix4x4 &sErrorReturn=aiMatrix4x4()) const
Get a matrix configuration property. More...

const std::string GetPropertyString (const char *szName, const std::string &sErrorReturn="") const
Get a string configuration property. More...

const aiScene * GetScene () const
Returns the scene loaded by the last successful call to ReadFile() More...

Importer ()
Constructor. More...

Importer (const Importer &other)
Copy constructor. More...

bool IsDefaultIOHandler () const
Checks whether a default IO handler is active A default handler is active as long the application doesn't supply its own custom IO handler via SetIOHandler(). More...

bool IsDefaultProgressHandler () const
Checks whether a default progress handler is active A default handler is active as long the application doesn't supply its own custom progress handler via SetProgressHandler(). More...

bool IsExtensionSupported (const char *szExtension) const
Returns whether a given file extension is supported by ASSIMP. More...

bool IsExtensionSupported (const std::string &szExtension) const
Returns whether a given file extension is supported by ASSIMP. More...

ImporterPimpl * Pimpl ()
Private, do not use. More...

const ImporterPimpl * Pimpl () const

const aiScene * ReadFile (const char *pFile, unsigned int pFlags)
Reads the given file and returns its contents if successful. More...

const aiScene * ReadFile (const std::string &pFile, unsigned int pFlags)
Reads the given file and returns its contents if successful. More...

const aiScene * ReadFileFromMemory (const void *pBuffer, size_t pLength, unsigned int pFlags, const char *pHint="")
Reads the given file from a memory buffer and returns its contents if successful. More...

aiReturn RegisterLoader (BaseImporter *pImp)
 Registers a new loader. More...

aiReturn RegisterPPStep (BaseProcess *pImp)
 Registers a new post-process step. More...
void SetExtraVerbose (bool bDo)
    Enables "extra verbose" mode. More...

void SetIOHandler (IOSystem *pIOHandler)
    Supplies a custom IO handler to the importer to use to open and access files. More...

void SetProgressHandler (ProgressHandler *pHandler)
    Supplies a custom progress handler to the importer. More...

bool SetPropertyBool (const char *szName, bool value)
    Set a boolean configuration property. More...

bool SetPropertyFloat (const char *szName, float fValue)
    Set a floating-point configuration property. More...

bool SetPropertyInteger (const char *szName, int iValue)
    Set an integer configuration property. More...

bool SetPropertyMatrix (const char *szName, const aiMatrix4x4 &sValue)
    Set a matrix configuration property. More...

bool SetPropertyString (const char *szName, const std::string &sValue)
    Set a string configuration property. More...

aiReturn UnregisterLoader (BaseImporter *pImp)
    Unregisters a loader. More...

aiReturn UnregisterPPStep (BaseProcess *pImp)
    Unregisters a post-process step. More...
bool ValidateFlags (unsigned int pFlags) const
Check whether a given set of postprocessing flags is supported. More...

~Importer ()
Destructor. More...
Static Public Attributes

static const unsigned int MaxLenHint = 200
The upper limit for hints. More...
Protected Attributes

ImporterPimpl *  pimpl
Detailed Description

CPP-API: The `Importer` class forms an C++ interface to the functionality of the Open Asset Import Library.

Create an object of this class and call `ReadFile()` to import a file. If the import succeeds, the function returns a pointer to the imported data. The data remains property of the object, it is intended to be accessed read-only. The imported data will be destroyed along with the `Importer` object. If the import fails, `ReadFile()` returns a NULL pointer. In this case you can retrieve a human-readable error description by calling `GetErrorString()`. You can call `ReadFile()` multiple times with a single `Importer` instance. Actually, constructing `Importer` objects involves quite many allocations and may take some time, so it's better to reuse them as often as possible.

If you need the `Importer` to do custom file handling to access the files, implement `IOSystem` and `IStream` and supply an instance of your custom `IOSystem` implementation by calling `SetIOHandler()` before calling `ReadFile()`. If you do not assign a custom IO handler, a default handler using the standard C++ IO logic will be used.

Note

One `Importer` instance is not thread-safe. If you use multiple threads for loading, each thread should maintain its own `Importer` instance.


Constructor & Destructor Documentation

Assimp::Importer::Importer ( )

Constructor.

Creates an empty importer object.

Call ReadFile() to start the import process. The configuration property table is initially empty.

Assimp::Importer::Importer ( const Importer & other )

Copy constructor.

This copies the configuration properties of another Importer. If this Importer owns a scene it won't be copied. Call ReadFile() to start the import process.

Assimp::Importer::~Importer ( )

Destructor.

The object kept ownership of the imported data, which now will be destroyed along with the object.
Member Function Documentation

const aiScene* Assimp::Importer::ApplyCustomizedPostProcessing ( * rootProcess, bool requestValidation )

const aiScene* Assimp::Importer::ApplyPostProcessing ( unsigned int pFlags )

Apply post-processing to an already-imported scene.

This is strictly equivalent to calling ReadFile() with the same flags. However, you can use this separate function to inspect the imported scene first to fine-tune your post-processing setup.

Parameters

pFlags Provide a bitwise combination of the aiPostProcessSteps flags.

Returns

A pointer to the post-processed data. This is still the same as the pointer returned by ReadFile(). However, if post-processing fails, the scene could now be NULL. That's quite a rare case, post processing steps are not really designed to 'fail'. To be exact, the #aiProcess_ValidateDS flag is currently the only post processing step which can actually cause the scene to be reset to NULL.

Note

The method does nothing if no scene is currently bound to the Importer instance.

void Assimp::Importer::FreeScene ( )

Frees the current scene.

The function does nothing if no scene has previously been read via ReadFile(). FreeScene() is called automatically by the destructor and ReadFile() itself.

const char* Assimp::Importer::GetErrorString ( ) const
Returns an error description of an error that occurred in \texttt{ReadFile()}. Returns an empty string if no error occurred.

Returns

A description of the last error, an empty string if no error occurred. The string is never NULL.

Note

The returned function remains valid until one of the following methods is called: \texttt{ReadFile()}, \texttt{FreeScene()}. 

\begin{verbatim}
void Assimp::Importer::GetExtensionList ( aiString & szOut ) const

Get a full list of all file extensions supported by ASSIMP.

If a file extension is contained in the list this does of course not mean that ASSIMP is able to load all files with this extension — it simply means there is an importer loaded which claims to handle files with this file extension.

Parameters

- String to receive the extension list. Format of the list: \texttt{szOut "*.3ds;*.obj;*.dae"}. This is useful for use with the WinAPI call GetOpenFileName(Ex).

\end{verbatim} 

\begin{verbatim}
AI_FORCE_INLINE void
Assimp::Importer::GetExtensionList ( std::string & szOut ) const inline

Get a full list of all file extensions supported by ASSIMP.

This function is provided for backward compatibility. See the \texttt{aiString} version for detailed and up-to-date docs.

See also

GetExtensionList(aiString&)

\end{verbatim} 

\begin{verbatim}
BaseImporter* Assimp::Importer::GetImporter ( size_t index ) const

Find the importer corresponding to a specific index.

\end{verbatim}
Parameters

index Index to query, must be within [0,\texttt{GetImporterCount}()]

Returns

\texttt{Importer} instance. NULL if the index does not exist.

\texttt{BaseImporter}\* 
\texttt{Assimp::Importer::GetImporter(const char\* szExtension) const}

Find the importer corresponding to a specific file extension.

This is quite similar to \texttt{IsExtensionSupported} except a \texttt{BaseImporter} instance is returned.

Parameters

\texttt{szExtension} Extension to check for. The following formats are recognized (BAH being the file extension): "BAH" (comparison is case-insensitive), ".bah", ":*.bah" (wild card and dot characters at the beginning of the extension are skipped).

Returns

NULL if no importer is found

\texttt{size_t Assimp::Importer::GetImporterCount( ) const}

Get the number of importers currently registered with \texttt{Assimp}.

\texttt{size_t Assimp::Importer::GetImporterIndex(const char\* szExtension) const}

Find the importer index corresponding to a specific file extension.

Parameters

\texttt{szExtension} Extension to check for. The following formats are recognized (BAH being the file extension): "BAH" (comparison is case-insensitive), ".bah", ":*.bah" (wild card and dot characters at the beginning of the extension are skipped).

Returns

(size_t)-1 if no importer is found
const aiImporterDesc* Assimp::Importer::GetImporterInfo (size_t index) const

Get meta data for the importer corresponding to a specific index.

For the declaration of aiImporterDesc, include <assimp/importerdesc.h>.

Parameters
  index Index to query, must be within [0, GetImporterCount()]

Returns
  Importer meta data structure, NULL if the index does not exist or if the importer doesn't offer meta information (importers may do this at the cost of being hated by their peers).

IOSystem* Assimp::Importer::GetIOHandler () const

Retrieves the IO handler that is currently set.

You can use IsDefaultIOHandler() to check whether the returned interface is the default IO handler provided by ASSIMP. The default handler is active as long the application doesn't supply its own custom IO handler via SetIOHandler().

Returns
  A valid IOSystem interface, never NULL.

void Assimp::Importer::GetMemoryRequirements (aiMemoryInfo & in) const

Returns the storage allocated by ASSIMP to hold the scene data in memory.

This refers to the currently loaded file, see ReadFile().

Parameters
  in Data structure to be filled.

Note
  The returned memory statistics refer to the actual size of the use data of the aiScene. Heap-related overhead is (naturally) not included.
aiScene* Assimp::Importer::GetOrphanedScene ( )

Returns the scene loaded by the last successful call to ReadFile() and releases the scene from the ownership of the Importer instance.

The application is now responsible for deleting the scene. Any further calls to GetScene() or GetOrphanedScene() will return NULL - until a new scene has been loaded via ReadFile().

Returns
  Current scene or NULL if there is currently no scene loaded

Note
  Use this method with maximal caution, and only if you have to. By design, aiScene's are exclusively maintained, allocated and deallocated by Assimp and no one else. The reasoning behind this is the golden rule that deallocations should always be done by the module that did the original allocation because heaps are not necessarily shared. GetOrphanedScene() enforces you to delete the returned scene by yourself, but this will only be fine if and only if you're using the same heap as assimp. On Windows, it's typically fine provided everything is linked against the multithreaded-dll version of the runtime library. It will work as well for static linkage with Assimp.

ProgressDialogHandler* Assimp::Importer::GetProgressDialogHandler ( ) const

Retrieves the progress handler that is currently set.

You can use IsDefaultProgressDialogHandler() to check whether the returned interface is the default handler provided by ASSIMP. The default handler is active as long the application doesn't supply its own custom handler via SetProgressDialogHandler().

Returns
  A valid ProgressDialogHandler interface, never NULL.

bool Assimp::Importer::GetPropertyBool ( const char* szName, bool bErrorReturn = false )

const char* Assimp::Importer::GetOrphanedScene ( )

Returns the scene loaded by the last successful call to ReadFile() and releases the scene from the ownership of the Importer instance.

The application is now responsible for deleting the scene. Any further calls to GetScene() or GetOrphanedScene() will return NULL - until a new scene has been loaded via ReadFile().

Returns
  Current scene or NULL if there is currently no scene loaded

Note
  Use this method with maximal caution, and only if you have to. By design, aiScene's are exclusively maintained, allocated and deallocated by Assimp and no one else. The reasoning behind this is the golden rule that deallocations should always be done by the module that did the original allocation because heaps are not necessarily shared. GetOrphanedScene() enforces you to delete the returned scene by yourself, but this will only be fine if and only if you're using the same heap as assimp. On Windows, it's typically fine provided everything is linked against the multithreaded-dll version of the runtime library. It will work as well for static linkage with Assimp.

ProgressDialogHandler* Assimp::Importer::GetProgressDialogHandler ( ) const

Retrieves the progress handler that is currently set.

You can use IsDefaultProgressDialogHandler() to check whether the returned interface is the default handler provided by ASSIMP. The default handler is active as long the application doesn't supply its own custom handler via SetProgressDialogHandler().

Returns
  A valid ProgressDialogHandler interface, never NULL.
Get a boolean configuration property.

Boolean properties are stored on the integer stack internally so it's possible to set them via `SetPropertyBool` and query them with `GetPropertyBool` and vice versa.

See also

`GetPropertyInteger()`

```cpp
float Assimp::Importer::GetPropertyFloat ( const char * szName,
                                           float fErrorReturn = 10e10f
                                           )
const
```

Get a floating-point configuration property.

See also

`GetPropertyInteger()`

```cpp
int Assimp::Importer::GetPropertyInteger ( const char * szName,
                                          int iErrorReturn = 0xffffffff
                                          )
const
```

Get a configuration property.

Parameters

Name of the property. All supported properties are defined in the `aiConfig.g` header (all constants share the prefix `AI_CONFIG_XXX`).

`szName` Value that is returned if the property is not found.

`iErrorReturn` Current value of the property

Note

Property of different types (float, int, string ..) are kept on different lists, so
calling `setPropertyInteger()` for a floating-point property has no effect - the loader will call `getPropertyFloat()` to read the property, but it won't be there.

```cpp
const aiMatrix4x4 Assimp::Importer::getPropertyMatrix ( const char * szName, const aiMatrix4x4 & sErrorReturn = aiMatrix4x4() ) const
```

Get a matrix configuration property.

The return value remains valid until the property is modified.

See also `getPropertyInteger()`

```cpp
const std::string Assimp::Importer::getPropertyString ( const char * szName, const std::string & sErrorReturn = "" ) const
```

Get a string configuration property.

The return value remains valid until the property is modified.

See also `getPropertyInteger()`

```cpp
const aiScene* Assimp::Importer::getScene ( ) const
```

Returns the scene loaded by the last successful call to `readFile()`

Returns
Current scene or NULL if there is currently no scene loaded

```cpp
bool Assimp::Importer::isDefaultIOHandler ( ) const
```
Checks whether a default IO handler is active. A default handler is active as long the application doesn't supply its own custom IO handler via `SetIOHandler()`.

Returns
true by default

```cpp
bool Assimp::Importer::IsDefaultProgressHandler() const
```

Checks whether a default progress handler is active. A default handler is active as long the application doesn't supply its own custom progress handler via `SetProgressHandler()`.

Returns
true by default

```cpp
bool Assimp::Importer::IsProgressHandlerActive() const
```

Returns whether a given file extension is supported by ASSIMP.

Parameters
szExtension
Extension to be checked. Must include a trailing dot '.'. Example: ".3ds", ".md3". Cases-insensitive.

Returns
true if the extension is supported, false otherwise

```cpp
AI_FORCE_INLINE bool Assimp::Importer::IsExtensionSupported(const std::string & szExtension) const inline
```

Returns whether a given file extension is supported by ASSIMP.

This function is provided for backward compatibility. See the const char* version for detailed and up-to-date docs.

See also
IsExtensionSupported(const char*)
Private, do not use.

const ImporterPimpl* Assimp::Importer::Pimpl ( ) const inline

const aiScene* Assimp::Importer::ReadFile ( const char * pFile,
                        unsigned int pFlags
                      )

Reads the given file and returns its contents if successful.

If the call succeeds, the contents of the file are returned as a pointer to an aiScene object. The returned data is intended to be read-only, the importer object keeps ownership of the data and will destroy it upon destruction. If the import fails, NULL is returned. A human-readable error description can be retrieved by calling GetLastErrorString(). The previous scene will be deleted during this call.

Parameters

pFile  Path and filename to the file to be imported.
       Optional post processing steps to be executed after a successful import. Provide a bitwise combination of the aiPostProcessSteps pFlags flags. If you wish to inspect the imported scene first in order to fine-tune your post-processing setup, consider to use ApplyPostProcessing().

Returns

A pointer to the imported data, NULL if the import failed. The pointer to the scene remains in possession of the Importer instance. Use GetOrphanedScene() to take ownership of it.

Note

Assimp is able to determine the file format of a file automatically.

AI_FORCE_INLINE const aiScene *
Assimp::Importer::ReadFile
                        ( const std::string & pFile,
                          unsigned int pFlags
                        )
Reads the given file and returns its contents if successful.

class **Importer**

This function is provided for backward compatibility. See the const char* version for detailed docs.

See also

**ReadFile(const char*, pFlags)**

```c++
const aiScene* Assimp::Importer::ReadFileFromMemory(
    const void* pBuffer,  
    size_t pLength,  
    unsigned int pFlags,  
    const char* pHint = ""
)
```

Reads the given file from a memory buffer and returns its contents if successful.

If the call succeeds, the contents of the file are returned as a pointer to an aiScene object. The returned data is intended to be read-only, the importer object keeps ownership of the data and will destroy it upon destruction. If the import fails, NULL is returned. A human-readable error description can be retrieved by calling **GetErrorString()**. The previous scene will be deleted during this call. Calling this method doesn't affect the active IOSystem.

Parameters

- **pBuffer** Pointer to the file data
- **pLength** Length of pBuffer, in bytes
- **pFlags** Optional post processing steps to be executed after a successful import. Provide a bitwise combination of the aiPostProcessSteps flags. If you wish to inspect the imported scene first in order to fine-tune your post-processing setup, consider to use **ApplyPostProcessing()**.
- **pHint** An additional hint to the library. If this is a non empty string, the library looks for a loader to support the file extension specified by
pHint and passes the file to the first matching loader. If this loader is unable to completely the request, the library continues and tries to determine the file format on its own, a task that may or may not be successful. Check the return value, and you'll know ...

Returns
A pointer to the imported data, NULL if the import failed. The pointer to the scene remains in possession of the Importer instance. Use GetOrphanedScene() to take ownership of it.

Note
This is a straightforward way to decode models from memory buffers, but it doesn't handle model formats that spread their data across multiple files or even directories. Examples include OBJ or MD3, which outsource parts of their material info into external scripts. If you need full functionality, provide a custom IOSystem to make Assimp find these files and use the regular ReadFile() API.

aiReturn Assimp::Importer::RegisterLoader ( BaseImporter * pImp )

Registers a new loader.

Parameters
pImp Importer to be added. The Importer instance takes ownership of the pointer, so it will be automatically deleted with the Importer instance.

Returns
AI_SUCCESS if the loader has been added. The registration fails if there is already a loader for a specific file extension.

aiReturn Assimp::Importer::RegisterPPStep ( BaseProcess * pImp )

Registers a new post-process step.

At the moment, there's a small limitation: new post processing steps are added to end of the list, or in other words, executed last, after all built-in steps.

Parameters
Post-process step to be added. The Importer instance takes ownership
pImp of the pointer, so it will be automatically deleted with the Importer instance.

Returns

AI_SUCCESS if the step has been added correctly.

void Assimp::Importer::SetExtraVerbose ( bool bDo )

Enables "extra verbose" mode.

'Extra verbose' means the data structure is validated after every single post processing step to make sure everyone modifies the data structure in a well-defined manner. This is a debug feature and not intended for use in production environments.

void Assimp::Importer::SetIOHandler ( IOSystem * pIOHandler )

Supplies a custom IO handler to the importer to use to open and access files.

If you need the importer to use custion IO logic to access the files, you need to provide a custom implementation of IOSystem and IOFile to the importer. Then create an instance of your custion IOSystem implementation and supply it by this function.

The Importer takes ownership of the object and will destroy it afterwards. The previously assigned handler will be deleted. Pass NULL to take again ownership of your IOSystem and reset Assimp to use its default implementation.

Parameters

pIOHandler The IO handler to be used in all file accesses of the Importer.

void Assimp::Importer::SetProgressHandler ( ProgressHandler * pHandler )

Supplies a custom progress handler to the importer.

This interface exposes a #Update() callback, which is called more or less periodically (please don't sue us if it isn't as periodically as you'd like it to have ...). This can be used to implement progress bars and loading timeouts.
Parameters
  pHandler  Progress callback interface. Pass NULL to disable progress reporting.

Note
  Progress handlers can be used to abort the loading at almost any time.

bool Assimp::Importer::SetPropertyBool ( const char * szName,
                                           bool       value       inline
                                    )

Set a boolean configuration property.

Boolean properties are stored on the integer stack internally so it's possible to set them via SetPropertyBool and query them with GetPropertyBool and vice versa.

See also
  SetPropertyInteger()

bool Assimp::Importer::SetPropertyFloat ( const char * szName,
                                          float       fValue
                                   )

Set a floating-point configuration property.

See also
  SetPropertyInteger()

bool Assimp::Importer::SetPropertyInteger ( const char * szName,
                                            int       iValue
                                     )

Set an integer configuration property.

Parameters
  szName  Name of the property. All supported properties are defined in the aiConfig.g header (all constants share the prefix AI_CONFIG_XXX and are simple strings).
iValue   New value of the property

Returns
true if the property was set before. The new value replaces the previous
value in this case.

Note
Property of different types (float, int, string ..) are kept on different stacks,
so calling SetPropertyInteger() for a floating-point property has no effect -
the loader will call GetPropertyFloat() to read the property, but it won't be
there.

bool Assimp::Importer::SetPropertyMatrix ( const char *         szName,
                                           const aiMatrix4x4 & sValue
                                           )

Set a matrix configuration property.

See also
SetPropertyInteger()

bool Assimp::Importer::SetPropertyString ( const char *         szName,
                                           const std::string & sValue
                                           )

Set a string configuration property.

See also
SetPropertyInteger()

aiReturn Assimp::Importer::UnregisterLoader ( BaseImporter * pImp )

Unregisters a loader.

Parameters
pImp Importer to be unregistered.

Returns
AI_SUCCESS if the loader has been removed. The function fails if the loader is currently in use (this could happen if the `Importer` instance is used by more than one thread) or if it has not yet been registered.

```c
aiReturn Assimp::Importer::UnregisterPPStep ( BaseProcess * pImp )
```

Unregisters a post-process step.

Parameters
   pImp Step to be unregistered.

Returns
   AI_SUCCESS if the step has been removed. The function fails if the step is currently in use (this could happen if the `Importer` instance is used by more than one thread) or if it has not yet been registered.

```c
bool Assimp::Importer::ValidateFlags ( unsigned int pFlags ) const
```

Check whether a given set of postprocessing flags is supported.

Some flags are mutually exclusive, others are probably not available because your excluded them from your `Assimp` builds. Calling this function is recommended if you're unsure.

Parameters
   pFlags Bitwise combination of the aiPostProcess flags.

Returns
   true if this flag combination is fine.
Member Data Documentation

const unsigned int Assimp::Importer::MaxLenHint = 200 static

The upper limit for hints.

ImporterPimpl* Assimp::Importer::pimpl protected

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

- Importer.hpp

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp v3.1.1 (June 2014)

- Assimp
- Interpolator

Public Member Functions | List of all members
Assimp::Interpolator< T > Struct Template Reference

CPP-API: Utility class to simplify interpolations of various data types. More...
Public Member Functions

void operator() (T &out, const T &a, const T &b, float d) const
Get the result of the interpolation between a,b. More...
Detailed Description

```cpp
template<typename T>
struct Assimp::Interpolator<T>
```

CPP-API: Utility class to simplify interpolations of various data types.

The type of interpolation is chosen automatically depending on the types of the arguments.
template<typename T>
void Assimp::Interpolator<T>::operator() ( T & out, const T & a, const T & b, inline float d ) const

Get the result of the interpolation between a,b.

The interpolation algorithm depends on the type of the operands. aiQuaternion's and aiQuatKey's SLERP, the rest does a simple linear interpolation.

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

- anim.h
Assimp v3.1.1 (June 2014)

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Namespaces</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class List</td>
<td>Class Index</td>
<td>Class Hierarchy</td>
<td>Class Members</td>
<td></td>
</tr>
</tbody>
</table>

- Assimp
- IOSStream

**Public Member Functions** | **Protected Member Functions** | **List of all members**
Assimp::IOStream Class Reference
abstract

CPP-API: Class to handle file I/O for C++. [More...]

Inherits AllocateFromAssimpHeap.
Public Member Functions

virtual size_t FileSize () const =0
Returns filesize Returns the filesize. More...

virtual void Flush ()=0
Flush the contents of the file buffer (for writers) See fflush() for more details. More...

virtual size_t Read (void *pvBuffer, size_t pSize, size_t pCount)=0
Read from the file. More...

virtual aiReturn Seek (size_t pOffset, aiOrigin pOrigin)=0
Set the read/write cursor of the file. More...

virtual size_t Tell () const =0
Get the current position of the read/write cursor. More...

virtual size_t Write (const void *pvBuffer, size_t pSize, size_t pCount)=0
Write to the file. More...

virtual ~IOStream ()
Destructor. More...
Protected Member Functions

**Iostream (void)**
Constructor protected, use **IOSystem::Open()** to create an instance. [More...]
Detailed Description

CPP-API: Class to handle file I/O for C++.

Derive an own implementation from this interface to provide custom IO handling to the Importer. If you implement this interface, be sure to also provide an implementation for IOSystem that creates instances of your custom IO class.
**Constructor & Destructor Documentation**

Assimp::IOStream::IOStream ( void ) inlineprotected

Constructor protected, use **IOSystem::Open()** to create an instance.

class **IOStream**

Assimp::IOStream::~IOStream ( ) inlinevirtual

Destructor.

Deleting the object closes the underlying file, alternatively you may use **IOSystem::Close()** to release the file.


**Member Function Documentation**

virtual size_t Assimp::IOStream::FileSize ( ) const pure virtual

Returns filesize Returns the filesize.

virtual void Assimp::IOStream::Flush ( ) pure virtual

Flush the contents of the file buffer (for writers) See fflush() for more details.

virtual size_t Assimp::IOStream::Read ( void * pvBuffer,
    size_t pSize,
    size_t pCount   ) pure virtual

Read from the file.

See fread() for more details This fails for write-only files

virtual aiReturn Assimp::IOStream::Seek ( size_t pOffset,
    aiOrigin pOrigin   ) pure virtual

Set the read/write cursor of the file.

Note that the offset is *negative* for aiOrigin_END. See fseek() for more details

virtual size_t Assimp::IOStream::Tell ( ) const pure virtual

Get the current position of the read/write cursor.

See ftell() for more details

virtual size_t Assimp::IOStream::Write ( const void * pvBuffer,
    size_t pSize,
size_t pCount pure virtual

Write to the file.

See fwrite() for more details This fails for read-only files

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

- IOSstream.hpp

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp v3.1.1 (June 2014)

- Assimp
- IOSystem

Public Member Functions | List of all members
Assimp::IOSystem Class Referenceabstract

CPP-API: Interface to the file system. More...

Inherits AllocateFromAssimpHeap.
**Public Member Functions**

virtual void **Close** (IOStream *pFile)=0
Closes the given file and releases all resources associated with it. **More...**

virtual bool **ComparePaths** (const char *one, const char *second) const
Compares two paths and check whether the point to identical files. **More...**

bool **ComparePaths** (const std::string &one, const std::string &second) const
For backward compatibility. **More...**

virtual const std::string & **CurrentDirectory** () const
Returns the top directory from the stack. **More...**

**AI_FORCE_INLINE**
bool **Exists** (const std::string &pFile) const
For backward compatibility. **More...**

virtual bool **Exists** (const char *pFile) const =0
Tests for the existence of a file at the given path. **More...**

virtual char **getOsSeparator** () const =0
Returns the system specific directory separator. **More...**

**IOSystem** ()
Default constructor. **More...**

virtual IOStream * **Open** (const char *pFile, const char *pMode="rb")=0
Open a new file with a given path. More...

**IOStream** *Open* (const std::string &pFile, const std::string &pMode=std::string("rb"))
For backward compatibility. More...

virtual bool **PopDirectory** ()
Pops the top directory from the stack. More...

virtual bool **PushDirectory** (const std::string &path)
Pushes a new directory onto the directory stack. More...

virtual size_t **StackSize** () const
Returns the number of directories stored on the stack. More...

virtual **~IOSystem** ()
Virtual destructor. More...
Detailed Description

CPP-API: Interface to the file system.

Derive an own implementation from this interface to supply custom file handling to the importer library. If you implement this interface, you also want to supply a custom implementation for \texttt{IStream}.

See also

\texttt{Importer::SetIOHandler()}
Constructor & Destructor Documentation

**AI_FORCE_INLINE** Assimp::IOSystem::IOSystem ( )

Default constructor.

Create an instance of your derived class and assign it to an **Assimp::Importer** instance by calling **Importer::SetIOHandler()**.

**AI_FORCE_INLINE** Assimp::IOSystem::~IOSystem ( ) virtual

Virtual destructor.

It is safe to be called from within DLL **Assimp**, we're constructed on **Assimp**'s heap.
Member Function Documentation

virtual void Assimp::IOSystem::Close ( std::iostream * pFile ) pure virtual

Closes the given file and releases all resources associated with it.

Parameters
   pFile The file instance previously created by Open().

virtual bool Assimp::IOSystem::ComparePaths ( const char * one,
                                            const char * second  virtual
                                           ) const

Compares two paths and check whether the point to identical files.

The dummy implementation of this virtual member performs a case-insensitive comparison of the given strings. The default IO system implementation uses OS mechanisms to convert relative into absolute paths, so the result can be trusted.

Parameters
   one First file
   second Second file

Returns
   true if the paths point to the same file. The file needn't be existing, however.

bool Assimp::IOSystem::ComparePaths ( const std::string & one,
                                        const std::string & second  inline
                                       ) const

For backward compatibility.

See also
   ComparePaths(const char*, const char*)
const std::string & Assimp::IOSystem::CurrentDirectory ( ) const inline virtual

Returns the top directory from the stack.

Returns
The directory on the top of the stack. Returns empty when no directory was pushed to the stack.

AI_FORCE_INLINE bool Assimp::IOSystem::Exists ( const std::string & pFile ) const

For backward compatibility.

See also
Exists(const char*)

virtual bool Assimp::IOSystem::Exists ( const char * pFile ) const pure virtual

Tests for the existence of a file at the given path.

Parameters
pFile Path to the file

Returns
true if there is a file with this path, else false.

virtual char Assimp::IOSystem::getOsSeparator ( ) const pure virtual

Returns the system specific directory separator.

Returns
System specific directory separator

virtual IOStream* Assimp::IOSystem::Open ( * pFile,
const char pMode = "rb"
) pure virtual
Open a new file with a given path.

When the access to the file is finished, call Close() to release all associated resources (or the virtual dtor of the IOSstream).

Parameters

- **pFile** Path to the file
- **pMode** Desired file I/O mode. Required are: "wb", "w", "wt", "rb", "r", "rt".

Returns

- New IOSstream interface allowing the lib to access the underlying file.

Note

- When implementing this class to provide custom IO handling, you probably have to supply an own implementation of IOSstream as well.

```cpp
AI_FORCE_INLINE IOSstream * Assimp::IOSystem::Open
cost const pFile,
&
cost const pMode =
std::string("rb")

For backward compatibility.

See also

Open(const char*, const char*)

bool Assimp::IOSystem::PopDirectory ( ) inlinevirtual

Pops the top directory from the stack.

Returns

- True, when a directory was on the stack. False if no directory was on the stack.
bool Assimp::IOSystem::PushDirectory(const std::string &path) inline virtual

Pushes a new directory onto the directory stack.

Parameters
  path Path to push onto the stack.

Returns
  True, when push was successful, false if path is empty.

size_t Assimp::IOSystem::StackSize() const inline virtual

Returns the number of directories stored on the stack.

Returns
  The number of directories of the stack.

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

- IOSystem.hpp

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp v3.1.1 (June 2014)

- Assimp
- Logger

Public Types | Public Member Functions | Protected Member Functions | Protected Attributes | List of all members
Assimp::Logger Class Reference
abstract

CPP-API: Abstract interface for logger implementations. More...

Inherits AllocateFromAssimpHeap.

Inherited by Assimp::DefaultLogger, and Assimp::NullLogger.
Public Types
Public Member Functions

virtual bool attachStream (LogStream *pStream, unsigned int severity=Debugging|Err|Warn|Info)=0
Attach a new log-stream. More...

void debug (const char *message)
Writes a debug message. More...

void debug (const std::string &message)

virtual bool detachStream (LogStream *pStream, unsigned int severity=Debugging|Err|Warn|Info)=0
Detach a still attached stream from the logger (or modify the filter flags bits) More...

void error (const char *message)
Writes an error message. More...

void error (const std::string &message)

LogSeverity getLogSeverity () const
Get the current log severity. More...

void info (const char *message)
Writes a info message. More...

void info (const std::string &message)

void setLogSeverity (LogSeverity log_severity)
Set a new log severity. More...
void **warn** (const char *message)
  Writes a warning message. **More...**

void **warn** (const std::string &message)

virtual **~Logger** ()
  Virtual destructor. **More...**
Protected Member Functions

Logger ()
Default constructor. More...

Logger (LogSeverity severity)
Construction with a given log severity. More...

virtual void OnDebug (const char *message)=0
Called as a request to write a specific debug message. More...

virtual void OnError (const char *message)=0
Called as a request to write a specific error message. More...

virtual void OnInfo (const char *message)=0
Called as a request to write a specific info message. More...

virtual void OnWarn (const char *message)=0
Called as a request to write a specific warn message. More...
Protected Attributes

LogSeverity  m_Severity
Logger severity. More...
Detailed Description

CPP-API: Abstract interface for logger implementations.

Assimp provides a default implementation and uses it for almost all logging stuff ('DefaultLogger'). This class defines just basic logging behaviour and is not of interest for you. Instead, take a look at #DefaultLogger.
Member Enumeration Documentation

enum `Assimp::Logger::ErrorSeverity`

Description for severity of a log message.

Every `LogStream` has a bitwise combination of these flags. A `LogStream` doesn't receive any messages of a specific type if it doesn't specify the corresponding `ErrorSeverity` flag.

**Enumerator**

Debugging  Debug log message.

Info  Info log message.

Warn  Warn log message.

Err  Error log message.

enum `Assimp::Logger::LogSeverity`

Log severity to describe the granularity of logging.

**Enumerator**

NORMAL  Normal granularity of logging.

VERBOSE  Debug infos will be logged, too.
Constructor & Destructor Documentation

Assimp::Logger::~Logger ( ) inlinevirtual

Virtual destructor.

Assimp::Logger::Logger ( ) inlineprotected

Default constructor.

Assimp::Logger::Logger ( LogSeverity severity ) inlineexplicitprotected

Construction with a given log severity.
Member Function Documentation

virtual bool Assimp::Logger::attachStream ( LogStream * pStream, unsigned int severity = Debugging | Err | Warn | Info )

Attach a new log-stream.

The logger takes ownership of the stream and is responsible for its destruction (which is done using ::delete when the logger itself is destroyed). Call detachStream to detach a stream and to gain ownership of it again.

Parameters
  pStream Log-stream to attach
  severity Message filter, specified which types of log messages are dispatched to the stream. Provide a bitwise combination of the ErrorSeverity flags.

Returns
  true if the stream has been attached, false otherwise.

Implemented in Assimp::DefaultLogger, and Assimp::NullLogger.

void Assimp::Logger::debug ( const char * message )

Writes a debug message.

Parameters
  message Debug message

void Assimp::Logger::debug ( const std::string & message ) inline

virtual bool Assimp::Logger::detatchStream ( LogStream * pStream, unsigned int severity = Debugging | Err | Warn | Info )

pure virtual
Debugging | Err | Warn | Info

Detach a still attached stream from the logger (or modify the filter flags bits)

Parameters

pStream Log-stream instance for detaching

severity Provide a bitwise combination of the ErrorSeverity flags. This value is &~ed with the current flags of the stream, if the result is 0 the stream is detached from the Logger and the caller retakes the possession of the stream.

Returns

true if the stream has been detached, false otherwise.

Implemented in Assimp::DefaultLogger, and Assimp::NullLogger.

void Assimp::Logger::error ( const char * message )

Writes an error message.

Parameters

message Error message

void Assimp::Logger::error ( const std::string & message ) inline

Logger::LogSeverity Assimp::Logger::getLogSeverity ( ) const inline

Get the current log severity.

void Assimp::Logger::info ( const char * message )

Writes a info message.

Parameters

message Info message

void Assimp::Logger::info ( const std::string & message ) inline
virtual void Assimp::Logger::OnDebug (const char * message) protected 

Called as a request to write a specific debug message.

Parameters
message Debug message. Never longer than MAX_LOG_MESSAGE_LENGTH characters (excluding the '0').

Note
The message string is only valid until the scope of the function is left.

Implemented in Assimp::NullLogger.

virtual void Assimp::Logger::OnError (const char * message) protected

Called as a request to write a specific error message.

Parameters
message Error message. Never longer than MAX_LOG_MESSAGE_LENGTH characters (exclunding the '0').

Note
The message string is only valid until the scope of the function is left.

Implemented in Assimp::NullLogger.

virtual void Assimp::Logger::OnInfo (const char * message) protected

Called as a request to write a specific info message.

Parameters
message Info message. Never longer than MAX_LOG_MESSAGE_LENGTH characters (excluding the '0').

Note
The message string is only valid until the scope of the function is left.

Implemented in `Assimp::NullLogger`.

```cpp
template <typename T>
void Assimp::Logger::warn(T *message);
```

Called as a request to write a specific warn message.

Parameters
- `message` Warn message. Never longer than MAX_LOG_MESSAGE_LENGTH characters (excluding the '0').

Note
The message string is only valid until the scope of the function is left.

Implemented in `Assimp::NullLogger`.

```cpp
void Assimp::Logger::setLogSeverity(LogSeverity log_severity);
```

Set a new log severity.

Parameters
- `log_severity` New severity for logging

```cpp
void Assimp::Logger::warn(const char *message);
```

Writes a warning message.

Parameters
- `message` Warn message

```cpp
void Assimp::Logger::warn(const std::string &message);
```

Parameters
- `message` Warn message
Member Data Documentation

LogSeverity Assimp::Logger::m_Severity protected

Logger severity.

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

- Logger.hpp

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp v3.1.1 (June 2014)

- Assimp
- LogStream

Public Member Functions | Static Public Member Functions | Protected Member Functions | List of all members
Assimp::LogStream Class Reference

CPP-API: Abstract interface for log stream implementations. More...

Inherits AllocateFromAssimpHeap.
Public Member Functions

virtual void write (const char *message)=0
      Overwrite this for your own output methods. More...

virtual ~LogStream ()
      Virtual destructor. More...
Static Public Member Functions

```cpp
static LogStream createDefaultStream (aiDefaultLogStream stream, const char *name="AssimpLog.txt", IOSystem *io=NULL)

Creates a default log stream. More...
```
Protected Member Functions

LogStream ()
Default constructor. More...
Detailed Description

CPP-API: Abstract interface for log stream implementations.

Several default implementations are provided, see aiDefaultLogStream for more details. Writing your own implementation of LogStream is just necessary if these are not enough for your purpose.
Constructor & Destructor Documentation

Assimp::LogStream::LogStream() inline protected

Default constructor.

virtual Assimp::LogStream::~LogStream() inline virtual

Virtual destructor.


Member Function Documentation

static LogStream* Assimp::LogStream::createDefaultStream ( aiDefaultLogStream stream,
const char * name = "AssimpLog.txt",
IOSystem * io = NULL )

Creates a default log stream.

Parameters

  streams Type of the default stream
  name For aiDefaultLogStream_FILE: name of the output file
  io For aiDefaultLogStream_FILE: IOSystem to be used to open the output file. Pass NULL for the default implementation.

Returns

  New LogStream instance.

virtual void Assimp::LogStream::write ( const char * message ) pure virtual

Overwrite this for your own output methods.

Log messages may consist of multiple lines and you shouldn't expect a consistent formatting. If you want custom formatting (e.g. generate HTML), supply a custom instance of Logger to #DefaultLogger:set(). Usually you can expect that a log message is exactly one line and terminated with a single character.

Parameters

  message Message to be written

The documentation for this class was generated from the following file:
Main Page | Related Pages | Namespaces | Classes | Files
---|---|---|---|---
Class List | Class Index | Class Hierarchy | Class Members

- Assimp
- NullLogger

Public Member Functions | List of all members
Assimp::NullLogger Class Reference

CPP-API: Empty logging implementation. More...

Inherits Assimp::Logger.
Public Member Functions

bool attachStream (LogStream *pStream, unsigned int severity)
   Detach a still attached stream from logger. More...

bool detachStream (LogStream *pStream, unsigned int severity)
   Detach a still attached stream from logger. More...

void OnDebug (const char *message)
   Logs a debug message. More...

void OnError (const char *message)
   Logs an error message. More...

void OnInfo (const char *message)
   Logs an info message. More...

void OnWarn (const char *message)
   Logs a warning message. More...

Public Member Functions inherited from Assimp::Logger
void debug (const char *message)
   Writes a debug message. More...

void debug (const std::string &message)

void error (const char *message)
   Writes an error message. More...

void error (const std::string &message)

LogSeverity getLogSeverity () const
Get the current log severity. More...

```cpp
void info (const char *message)
  Writes a info message. More...
```

```cpp
void info (const std::string &message)
```

```cpp
void setLogSeverity (LogSeverity log_severity)
  Set a new log severity. More...
```

```cpp
void warn (const char *message)
  Writes a warning message. More...
```

```cpp
void warn (const std::string &message)
```

```cpp
virtual ~Logger ()
  Virtual destructor. More...
```
Additional Inherited Members

- Public Types inherited from **Assimp::Logger**
- Protected Member Functions inherited from **Assimp::Logger**
  
  **Logger** ()
  Default constructor. **More...**

  **Logger** (**LogSeverity** severity)
  Construction with a given log severity. **More...**

- Protected Attributes inherited from **Assimp::Logger**
  **LogSeverity**  **m_Severity**
  **Logger** severity. **More...**
Detailed Description

CPP-API: Empty logging implementation.

Does nothing! Used by default if the application hasn't requested a custom logger via #DefaultLogger::set() or #DefaultLogger::create();
Member Function Documentation

bool Assimp::NullLogger::attachStream (LogStream * pStream, unsigned int severity) inline virtual

Detach a still attached stream from logger.

Implements Assimp::Logger.

bool Assimp::NullLogger::detachStream (LogStream * pStream, unsigned int severity) inline virtual

Detach a still attached stream from logger.

Implements Assimp::Logger.

void Assimp::NullLogger::OnDebug (const char * message) inline virtual

Logs a debug message.

Implements Assimp::Logger.

void Assimp::NullLogger::OnError (const char * message) inline virtual

Logs an error message.

Implements Assimp::Logger.

void Assimp::NullLogger::OnInfo (const char * message) inline virtual

Logs an info message.

Implements Assimp::Logger.
void Assimp::NullLogger::OnWarn ( const char * message ) inline virtual

Logs a warning message.

Implements Assimp::Logger.

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

- NullLogger.hpp
Assimp v3.1.1 (June 2014)

- Assimp
- ProgressHandler

Public Member Functions | Protected Member Functions | List of all members
Assimp::ProgressHandler Class Reference abstract

CPP-API: Abstract interface for custom progress report receivers. More...

Inherits AllocateFromAssimpHeap.
Public Member Functions

virtual bool Update (float percentage=-1.f)=0
    Progress callback. More...

virtual void UpdateFileRead (int currentStep, int numberOfSteps)
    Progress callback for file loading steps. More...

virtual void UpdatePostProcess (int currentStep, int numberOfSteps)
    Progress callback for post-processing steps. More...

virtual ~ProgressHandler ()
    Virtual destructor. More...
Protected Member Functions

ProgressHandler ()
Default constructor. More...
Detailed Description

CPP-API: Abstract interface for custom progress report receivers.

Each #Importer instance maintains its own ProgressHandler. The default implementation provided by Assimp doesn't do anything at all.
Constructor & Destructor Documentation

Assimp::ProgressHandler::ProgressHandler() inline protected

Default constructor.

virtual Assimp::ProgressHandler::~ProgressHandler() inline virtual

Virtual destructor.
Member Function Documentation

virtual bool Assimp::ProgressHandler::Update ( float percentage = -1.f )

Progress callback.

Parameters
percentage An estimate of the current loading progress, in percent. Or -1.f if such an estimate is not available.

There are restriction on what you may do from within your implementation of this method: no exceptions may be thrown and no non-const #Importer methods may be called. It is not generally possible to predict the number of callbacks fired during a single import.

Returns
Return false to abort loading at the next possible occasion (loaders and Assimp are generally allowed to perform all needed cleanup tasks prior to returning control to the caller). If the loading is aborted, #Importer::ReadFile() returns always NULL.

virtual void Assimp::ProgressHandler::UpdateFileRead ( int currentStep, int numberOfSteps )

Progress callback for file loading steps.

Parameters
numberOfSteps The number of total post-processing steps
currentStep The index of the current post-processing step that will run, or equal to numberOfSteps if all of them has finished. This number is always strictly monotone increasing, although not necessarily linearly.
virtual void
Assimp::ProgressHandler::UpdatePostProcess ( int currentStep,
        int numberOfSteps )

Progress callback for post-processing steps.

Parameters

- numberOfSteps: The number of total post-processing steps
- currentStep: The index of the current post-processing step that will run, or equal to numberOfSteps if all of them has finished. This number is always strictly monotone increasing, although not necessarily linearly.

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

- ProgressHandler.hpp
• Assimp
• ScopeGuard

Public Member Functions | List of all members
Assimp::ScopeGuard< T > Struct Template Reference
Public Member Functions

T * dismiss ()

operator T * ()

T * operator-> ()

ScopeGuard (T *obj)

~ScopeGuard () throw ()
Constructor & Destructor Documentation

template<typename T>
Assimp::ScopeGuard<T>::ScopeGuard(T * obj) inline explicit

template<typename T>
Assimp::ScopeGuard<T>::~ScopeGuard() throw ( )
throw ( inline )
Member Function Documentation

template<typename T >
  T* Assimp::ScopeGuard<T>::dismiss( ) inline

template<typename T >
  Assimp::ScopeGuard<T>::operator T *( ) inline

template<typename T >
  T* Assimp::ScopeGuard<T>::operator->( ) inline

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

- BaseImporter.h

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
An animation consists of keyframe data for a number of nodes. More...
Public Member Functions

aiAnimation() 

~aiAnimation()
Public Attributes

C_STRUCT aiNodeAnim ** mChannels
The node animation channels. More...

double mDuration
Duration of the animation in ticks. More...

C_STRUCT aiMeshAnim ** mMeshChannels
The mesh animation channels. More...

C_STRUCT aiString mName
The name of the animation. More...

unsigned int mNumChannels
The number of bone animation channels. More...

unsigned int mMNumMeshChannels
The number of mesh animation channels. More...

double mTicksPerSecond
Ticks per second. More...
Detailed Description

An animation consists of keyframe data for a number of nodes.

For each node affected by the animation a separate series of data is given.
Constructor & Destructor Documentation

aiAnimation::aiAnimation( ) inline
aiAnimation::~aiAnimation( ) inline
Member Data Documentation

C_STRUCT aiNodeAnim** aiAnimation::mChannels

The node animation channels.

Each channel affects a single node. The array is mNumChannels in size.

double aiAnimation::mDuration

Duration of the animation in ticks.

C_STRUCT aiMeshAnim** aiAnimation::mMeshChannels

The mesh animation channels.

Each channel affects a single mesh. The array is mNumMeshChannels in size.

C_STRUCT aiString aiAnimation::mName

The name of the animation.

If the modeling package this data was exported from does support only a single animation channel, this name is usually empty (length is zero).

unsigned int aiAnimation::mNumChannels

The number of bone animation channels.

Each channel affects a single node.

unsigned int aiAnimation::mNumMeshChannels

The number of mesh animation channels.

Each channel affects a single mesh and defines vertex-based animation.

double aiAnimation::mTicksPerSecond
Ticks per second.

0 if not specified in the imported file

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

- anim.h

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp v3.1.1 (June 2014)

Public Member Functions | Public Attributes | List of all members
aiAnimMesh Struct Reference

NOT CURRENTLY IN USE. More...
Public Member Functions

aiAnimMesh ()

bool HasNormals () const
Check whether the anim mesh overrides the vertex normals of its host mesh. More...

bool HasPositions () const
Check whether the anim mesh overrides the vertex positions of its host mesh. More...

bool HasTangentsAndBitangents () const
Check whether the anim mesh overrides the vertex tangents and bitangents of its host mesh. More...

bool HasTextureCoords (unsigned int pIndex) const
Check whether the anim mesh overrides a particular set of texture coordinates on his host mesh. More...

bool HasVertexColors (unsigned int pIndex) const
Check whether the anim mesh overrides a particular set of vertex colors on his host mesh. More...

~aiAnimMesh ()
Public Attributes

**aiVector3D** mBitangents

Replacement for aiMesh::mBitangents. More...

**aiColor4D** mColors [AI_MAX_NUMBER_OF_COLOR_SETS]

Replacement for aiMesh::mColors. More...

**aiVector3D** mNormals

Replacement for aiMesh::mNormals. More...

unsigned int mNumVertices

The number of vertices in the aiAnimMesh, and thus the length of all the member arrays. More...

**aiVector3D** mTangents

Replacement for aiMesh::mTangents. More...

**aiVector3D** mTextureCoords [AI_MAX_NUMBER_OF_TEXTURECOORDS]

Replacement for aiMesh::mTextureCoords. More...

**aiVector3D** mVertices

Replacement for aiMesh::mVertices. More...
Detailed Description

NOT CURRENTLY IN USE.

An AnimMesh is an attachment to an aiMesh stores per-vertex animations for a particular frame.

You may think of an aiAnimMesh as a patch for the host mesh, which replaces only certain vertex data streams at a particular time. Each mesh stores n attached attached meshes (aiMesh::mAnimMeshes). The actual relationship between the time line and anim meshes is established by aiMeshAnim, which references singular mesh attachments by their ID and binds them to a time offset.
Constructor & Destructor Documentation

aiAnimMesh::aiAnimMesh ( ) inline
aiAnimMesh::~aiAnimMesh ( ) inline
**Member Function Documentation**

bool aiAnimMesh::HasNormals ( ) const inline

Check whether the anim mesh overrides the vertex normals of its host mesh.

bool aiAnimMesh::HasPositions ( ) const inline

Check whether the anim mesh overrides the vertex positions of its host mesh.

bool aiAnimMesh::HasTangentsAndBitangents ( ) const inline

Check whether the anim mesh overrides the vertex tangents and bitangents of its host mesh.

As for aiMesh, tangents and bitangents always go together.

bool aiAnimMesh::HasTextureCoords ( unsigned int pIndex ) const inline

Check whether the anim mesh overrides a particular set of texture coordinates on his host mesh.

Parameters
    pIndex 0<index<AI_MAX_NUMBER_OF_TEXTURECOORDS

bool aiAnimMesh::HasVertexColors ( unsigned int pIndex ) const inline

Check whether the anim mesh overrides a particular set of vertex colors on his host mesh.

Parameters
    pIndex 0<index<AI_MAX_NUMBER_OF_COLOR_SETS
Member Data Documentation

\texttt{aiVector3D}* \texttt{aiAnimMesh::mBitangents}

Replacement for \texttt{aiMesh::mBitangents}.

\texttt{aiColor4D}* \texttt{aiAnimMesh::mColors[AI\_MAX\_NUMBER\_OF\_COLOR\_SETS]}

Replacement for \texttt{aiMesh::mColors}.

\texttt{aiVector3D}* \texttt{aiAnimMesh::mNormals}

Replacement for \texttt{aiMesh::mNormals}.

unsigned int \texttt{aiAnimMesh::mNumVertices}

The number of vertices in the \texttt{aiAnimMesh}, and thus the length of all the member arrays.

This has always the same value as the \texttt{mNumVertices} property in the corresponding \texttt{aiMesh}. It is duplicated here merely to make the length of the member arrays accessible even if the \texttt{aiMesh} is not known, e.g. from language bindings.

\texttt{aiVector3D}* \texttt{aiAnimMesh::mTangents}

Replacement for \texttt{aiMesh::mTangents}.

\texttt{aiVector3D}* \texttt{aiAnimMesh::mTextureCoords[AI\_MAX\_NUMBER\_OF\_TEXTURECOORDS]}

Replacement for \texttt{aiMesh::mTextureCoords}.

\texttt{aiVector3D}* \texttt{aiAnimMesh::mVertices}

Replacement for \texttt{aiMesh::mVertices}. 
If this array is non-NULL, it must contain mNumVertices entries. The corresponding array in the host mesh must be non-NULL as well - animation meshes may neither add or nor remove vertex components (if a replacement array is NULL and the corresponding source array is not, the source data is taken instead)

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

- mesh.h
Assimp v3.1.1 (June 2014)

Public Member Functions | Public Attributes | List of all members
aiBone Struct Reference

A single bone of a mesh. More...
Public Member Functions

aiBone ()
Default constructor. More...

aiBone (const aiBone &other)
Copy constructor. More...

~aiBone ()
Destructor - deletes the array of vertex weights. More...
Public Attributes

**aiString**  *mName*

The name of the bone. [More...](#)

unsigned int  *mNumWeights*

The number of vertices affected by this bone. The maximum value for this member is `AI_MAX_BONE_WEIGHTS`. [More...](#)

**aiMatrix4x4**  *mOffsetMatrix*

Matrix that transforms from mesh space to bone space in bind pose. [More...](#)

**aiVertexWeight**  *mWeights*

The vertices affected by this bone. [More...](#)
Detailed Description

A single bone of a mesh.

A bone has a name by which it can be found in the frame hierarchy and by which it can be addressed by animations. In addition it has a number of influences on vertices.
Constructor & Destructor Documentation

aiBone::aiBone ( ) inline

Default constructor.

aiBone::aiBone ( const aiBone & other ) inline

Copy constructor.

aiBone::~aiBone ( ) inline

Destructor - deletes the array of vertex weights.
**Member Data Documentation**

**aiString** aiBone::mName

The name of the bone.

unsigned int aiBone::mNumWeights

The number of vertices affected by this bone. The maximum value for this member is **AI_MAX_BONE_WEIGHTS**.

**aiMatrix4x4** aiBone::mOffsetMatrix

Matrix that transforms from mesh space to bone space in bind pose.

**aiVertexWeight*** aiBone::mWeights

The vertices affected by this bone.

---

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

- **mesh.h**

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by [doxygen](http://www.stackoverflow.com) 1.8.11
Public Member Functions | Public Attributes | List of all members
aiCamera Struct Reference

Helper structure to describe a virtual camera. More...
Public Member Functions

aiCamera ()

void GetCameraMatrix (aiMatrix4x4 &out) const
Get a right-handed camera matrix from me. More...
Public Attributes

float **mAspect**
Screen aspect ratio. [More...]

float **mClipPlaneFar**
Distance of the far clipping plane from the camera. [More...]

float **mClipPlaneNear**
Distance of the near clipping plane from the camera. [More...]

float **mHorizontalFOV**
Half horizontal field of view angle, in radians. [More...]

**aiVector3D** **mLookAt**
'LookAt' - vector of the camera coordinate system relative to the coordinate space defined by the corresponding node. [More...]

**aiString** **mName**
The name of the camera. [More...]

**aiVector3D** **mPosition**
Position of the camera relative to the coordinate space defined by the corresponding node. [More...]

**aiVector3D** **mUp**
'Up' - vector of the camera coordinate system relative to the coordinate space defined by the corresponding node. [More...]
Detailed Description

Helper structure to describe a virtual camera.

Cameras have a representation in the node graph and can be animated. An important aspect is that the camera itself is also part of the scenegraph. This means, any values such as the look-at vector are not absolute, they're relative to the coordinate system defined by the node which corresponds to the camera. This allows for camera animations. For static cameras parameters like the 'look-at' or 'up' vectors are usually specified directly in aiCamera, but beware, they could also be encoded in the node transformation. The following (pseudo)code sample shows how to do it:

```cpp
// Get the camera matrix for a camera at a specific time
// if the node hierarchy for the camera does not contain
// at least one animated node this is a static computation
get-camera-matrix (node sceneRoot, camera cam) : matrix
{
  node cnd = find-node-for-camera(cam)
  matrix cmt = identity()
  // as usual - get the absolute camera transformation for this frame
  for each node nd in hierarchy from sceneRoot to cnd
    matrix cur
    if (is-animated(nd))
      cur = eval-animation(nd)
    else cur = nd->mTransformation;
    cmt = mult-matrices( cmt, cur )
  end for
  // now multiply with the camera's own local transform
  cam = mult-matrices (cam, get-camera-matrix(cmt) )
}
```

Note some file formats (such as 3DS, ASE) export a "target point" - the point the camera is looking at (it can even be animated). Assimp writes the target point as a subnode of the camera's main node, called "<camName>.Target". However this is just additional information then the transformation tracks of
the camera main node make the camera already look in the right direction.
Constructor & Destructor Documentation

aiCamera::aiCamera() inline
**Member Function Documentation**

void aiCamera::GetCameraMatrix ( aiMatrix4x4 & out ) const inline

Get a *right-handed* camera matrix from me.

Parameters

- out Camera matrix to be filled

todo: test ... should work, but i'm not absolutely sure

We don't know whether these vectors are already normalized ...
Member Data Documentation

float aiCamera::mAspect

Screen aspect ratio.

This is the ration between the width and the height of the screen. Typical values are 4/3, 1/2 or 1/1. This value is 0 if the aspect ratio is not defined in the source file. 0 is also the default value.

float aiCamera::mClipPlaneFar

Distance of the far clipping plane from the camera.

The far clipping plane must, of course, be further away than the near clipping plane. The default value is 1000.f. The ratio between the near and the far plane should not be too large (between 1000-10000 should be ok) to avoid floating-point inaccuracies which could lead to z-fighting.

float aiCamera::mClipPlaneNear

Distance of the near clipping plane from the camera.

The value may not be 0.f (for arithmetic reasons to prevent a division through zero). The default value is 0.1f.

float aiCamera::mHorizontalFOV

Half horizontal field of view angle, in radians.

The field of view angle is the angle between the center line of the screen and the left or right border. The default value is 1/4PI.

aiVector3D aiCamera::mLookAt

'LookAt' - vector of the camera coordinate system relative to the coordinate space defined by the corresponding node.
This is the viewing direction of the user. The default value is 0|0|1. The vector may be normalized, but it needn't.

**aiString** aiCamera::mName

The name of the camera.

There must be a node in the scenegraph with the same name. This node specifies the position of the camera in the scene hierarchy and can be animated.

**aiVector3D** aiCamera::mPosition

Position of the camera relative to the coordinate space defined by the corresponding node.

The default value is 0|0|0.

**aiVector3D** aiCamera::mUp

'Up' - vector of the camera coordinate system relative to the coordinate space defined by the corresponding node.

The 'right' vector of the camera coordinate system is the cross product of the up and lookAt vectors. The default value is 0|1|0. The vector may be normalized, but it needn't.

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

- camera.h
Represents a color in Red-Green-Blue space. More...
Public Member Functions

aiColor3D ()

aiColor3D (float _r, float _g, float _b)

aiColor3D (float _r)

aiColor3D (const aiColor3D &o)

bool IsBlack () const
Check whether a color is black. More...

bool operator!= (const aiColor3D &other) const
Component-wise inverse comparison. More...

aiColor3D operator* (const aiColor3D &c) const
Component-wise multiplication. More...

aiColor3D operator* (float f) const
Multiply with a scalar. More...

aiColor3D operator+ (const aiColor3D &c) const
Component-wise addition. More...

aiColor3D operator- (const aiColor3D &c) const
Component-wise subtraction. More...

bool operator< (const aiColor3D &other) const
Component-wise comparison. More...

bool operator== (const aiColor3D &other) const
Component-wise comparison. More...

float operator[] (unsigned int i) const
Access a specific color component. More...

float & operator[] (unsigned int i)
Access a specific color component. More...
Public Attributes

float $b$

float $g$

float $r$

Red, green and blue color values. More...
Detailed Description

Represents a color in Red-Green-Blue space.
Constructor & Destructor Documentation

aiColor3D::aiColor3D ( ) inline
aiColor3D::aiColor3D ( float _r,
                              float _g, inline
                              float _b
                      )
aiColor3D::aiColor3D ( float _r ) inline explicit
aiColor3D::aiColor3D ( const aiColor3D & o ) inline
**Member Function Documentation**

bool aiColor3D::IsBlack ( ) const inline

Check whether a color is black.

bool aiColor3D::operator!= ( const aiColor3D & other ) const inline

Component-wise inverse comparison.

aiColor3D aiColor3D::operator* ( const aiColor3D & c ) const inline

Component-wise multiplication.

aiColor3D aiColor3D::operator* ( float f ) const inline

Multiply with a scalar.

aiColor3D aiColor3D::operator+ ( const aiColor3D & c ) const inline

Component-wise addition.

aiColor3D aiColor3D::operator- ( const aiColor3D & c ) const inline

Component-wise subtraction.

bool aiColor3D::operator< ( const aiColor3D & other ) const inline

Component-wise comparison.

bool aiColor3D::operator== ( const aiColor3D & other ) const inline

Component-wise comparison.

float aiColor3D::operator[] ( unsigned int i ) const inline
Access a specific color component.

float& aiColor3D::operator[](unsigned int i) inline

Access a specific color component.
Member Data Documentation

float aiColor3D::b
float aiColor3D::g
float aiColor3D::r

Red, green and blue color values.

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

- types.h

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
aiColor4t< TReal > Class Template Reference

Represents a color in Red-Green-Blue space including an alpha component.

More...
Public Member Functions

aiColor4t ()

aiColor4t (TReal _r, TReal _g, TReal _b, TReal _a)

aiColor4t (TReal _r)

aiColor4t (const aiColor4t &o)

bool IsBlack () const
check whether a color is (close to) black More...

bool operator!= (const aiColor4t &other) const

const aiColor4t & operator*= (TReal f)

const aiColor4t & operator+= (const aiColor4t &o)

const aiColor4t & operator-= (const aiColor4t &o)

const aiColor4t & operator/= (TReal f)

bool operator< (const aiColor4t &other) const

bool operator== (const aiColor4t &other) const

TReal operator[] (unsigned int i) const

TReal & operator[] (unsigned int i)
Public Attributes

union {
  struct {
    TReal a
    TReal b
    TReal g
    TReal r
  }
  TReal c [4]
};
Detailed Description

template<typename TReal>
class aiColor4t<< TReal >

Represents a color in Red-Green-Blue space including an alpha component.

Color values range from 0 to 1.
Constructor & Destructor Documentation

template<typename TReal >
aiColor4t< TReal >::aiColor4t ( ) inline

template<typename TReal >
aiColor4t< TReal >::aiColor4t ( TReal _r,
       TReal _g,
       TReal _b, inline
       TReal _a
       )

template<typename TReal >
aiColor4t< TReal >::aiColor4t ( TReal _r ) inline explicit

template<typename TReal >
aiColor4t< TReal >::aiColor4t ( const aiColor4t< TReal > & o ) inline
Member Function Documentation

template<typename TReal >
bool aiColor4t< TReal >::IsBlack ( ) const inline

check whether a color is (close to) black

template<typename TReal >
AI_FORCE_INLINE bool aiColor4t< TReal >::operator!=( const aiColor4t< TReal > & other ) const

template<typename TReal >
AI_FORCE_INLINE const aiColor4t< TReal > & aiColor4t< TReal >::operator*=( TReal f )

template<typename TReal >
AI_FORCE_INLINE const aiColor4t< TReal > & aiColor4t< TReal >::operator+=( const aiColor4t< TReal > & o )

template<typename TReal >
AI_FORCE_INLINE const aiColor4t< TReal > & aiColor4t< TReal >::operator-=( const aiColor4t< TReal > & o )

template<typename TReal >
AI_FORCE_INLINE const aiColor4t< TReal > & aiColor4t< TReal >::operator/=( TReal f )

template<typename TReal >
AI_FORCE_INLINE bool aiColor4t< TReal >::operator< ( const aiColor4t< TReal > & other ) const

template<typename TReal >
AI_FORCE_INLINE bool aiColor4t< TReal >::operator<= ( const aiColor4t< TReal > & other ) const

template<typename TReal >
AI_FORCE_INLINE bool aiColor4t< TReal >::operator> ( const aiColor4t< TReal > & other ) const

template<typename TReal >
AI_FORCE_INLINE bool aiColor4t< TReal >::operator>= ( const aiColor4t< TReal > & other ) const

template<typename TReal >
TReal aiColor4t< TReal >::operator[]( unsigned int i ) const inline

template<typename TReal >
TReal & aiColor4t< TReal >::operator[]( unsigned int i ) inline
Member Data Documentation

union { ... }
template<typename TReal >
TReal aiColor4t< TReal >::a
template<typename TReal >
TReal aiColor4t< TReal >::b
template<typename TReal >
TReal aiColor4t< TReal >::c[4]
template<typename TReal >
TReal aiColor4t< TReal >::g
template<typename TReal >
TReal aiColor4t< TReal >::r

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

- color4.h
- color4.inl
Describes a blob of exported scene data. More...
Public Member Functions

`aiExportDataBlob()`
Default constructor. More...

`~aiExportDataBlob()`
Releases the data. More...
Public Attributes

void * data
The data. More...

aiString name
Name of the blob. More...

aiExportDataBlob * next
Pointer to the next blob in the chain or NULL if there is none. More...

size_t size
Size of the data in bytes. More...
Detailed Description

Describes a blob of exported scene data.

Use `aiExportSceneToBlob()` to create a blob containing an exported scene. The memory referred by this structure is owned by Assimp. to free its resources. Don't try to free the memory on your side - it will crash for most build configurations due to conflicting heaps.

Blobs can be nested - each blob may reference another blob, which may in turn reference another blob and so on. This is used when exporters write more than one output file for a given `aiScene`. See the remarks for `aiExportDataBlob::name` for more information.
Constructor & Destructor Documentation

aiExportDataBlob::aiExportDataBlob() inline

Default constructor.

aiExportDataBlob::~aiExportDataBlob() inline

Releases the data.
**Member Data Documentation**

void* aiExportDataBlob::data

The data.

*aiString* aiExportDataBlob::name

Name of the blob.

An empty string always indicates the first (and primary) blob, which contains the actual file data. Any other blobs are auxiliary files produced by exporters (i.e. material files). Existence of such files depends on the file format. Most formats don't split assets across multiple files.

If used, blob names usually contain the file extension that should be used when writing the data to disc.

*aiExportDataBlob* aiExportDataBlob::next

Pointer to the next blob in the chain or NULL if there is none.

size_t aiExportDataBlob::size

Size of the data in bytes.

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

- cexport.h

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11*
Assimp v3.1.1 (June 2014)

Public Attributes | List of all members
aiExportFormatDesc Struct Reference

Describes an file format which Assimp can export to. More...
Public Attributes

const char * description

A short description of the file format to present to users. More...

const char * fileExtension

Recommended file extension for the exported file in lower case. More...

const char * id

a short string ID to uniquely identify the export format. More...
Detailed Description

Describes an file format which Assimp can export to.

Use aiGetExportFormatCount() to learn how many export formats the current Assimp build supports and aiGetExportFormatDescription() to retrieve a description of an export format option.
Member Data Documentation

const char* aiExportFormatDesc::description

A short description of the file format to present to users.
Useful if you want to allow the user to select an export format.

const char* aiExportFormatDesc::fileExtension

Recommended file extension for the exported file in lower case.

const char* aiExportFormatDesc::id

a short string ID to uniquely identify the export format.

Use this ID string to specify which file format you want to export to when calling aiExportScene(). Example: "dae" or "obj"

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

- cexport.h
aiFace Struct Reference

A single face in a mesh, referring to multiple vertices. More...
Public Member Functions

aiFace ()
Default constructor. More...

aiFace (const aiFace &o)
Copy constructor. Copy the index array. More...

bool operator!= (const aiFace &o) const
Inverse comparison operator. More...

aiFace & operator= (const aiFace &o)
Assignment operator. Copy the index array. More...

bool operator==(const aiFace &o) const
Comparison operator. More...

~aiFace ()
Default destructor. Delete the index array. More...
Public Attributes

unsigned int * mIndices
   Pointer to the indices array. Size of the array is given in numIndices. More...

unsigned int mNumIndices
   Number of indices defining this face. More...
Detailed Description

A single face in a mesh, referring to multiple vertices.

If mNumIndices is 3, we call the face 'triangle', for mNumIndices > 3 it's called 'polygon' (hey, that's just a definition!).

aiMesh::mPrimitiveTypes can be queried to quickly examine which types of primitive are actually present in a mesh. The aiProcess_SortByPType flag executes a special post-processing algorithm which splits meshes with different primitive types mixed up (e.g. lines and triangles) in several 'clean' submeshes. Furthermore there is a configuration option (AI_CONFIG_PP_SBP_REMOVE) to force aiProcess_SortByPType to remove specific kinds of primitives from the imported scene, completely and forever. In many cases you'll probably want to set this setting to

aiPrimitiveType_LINE|aiPrimitiveType_POINT

Together with the aiProcess_Triangulate flag you can then be sure that aiFace::mNumIndices is always 3.

Note
Take a look at the Data Structures page for more information on the layout and winding order of a face.
Constructor & Destructor Documentation

aiFace::aiFace ( ) inline

Default constructor.

aiFace::~aiFace ( ) inline

Default destructor. Delete the index array.

aiFace::aiFace ( const aiFace & o ) inline

Copy constructor. Copy the index array.
Member Function Documentation

bool aiFace::operator!= ( const aiFace & o ) const inline

Inverse comparison operator.
Checks whether the index array of two faces is NOT identical

aiFace & aiFace::operator= ( const aiFace & o ) inline

Assignment operator. Copy the index array.

bool aiFace::operator== ( const aiFace & o ) const inline

Comparison operator.
Checks whether the index array of two faces is identical
Member Data Documentation

unsigned int* aiFace::mIndices

Pointer to the indices array. Size of the array is given in numIndices.

unsigned int aiFace::mNumIndices

Number of indices defining this face.

The maximum value for this member is A_MAX_FACE_INDICES.

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

- mesh.h
Public Attributes | List of all members
aiFile Struct Reference

C-API: File callbacks. More...
Public Attributes

aiFileTellProc  FileSizeProc
Callback to retrieve the size of the file, in bytes. More...

aiFileFlushProc  FlushProc
Callback to flush the file contents. More...

aiFileReadProc  ReadProc
Callback to read from a file. More...

aiFileSeek  SeekProc
Callback to set the current position of the file cursor (fseek()) More...

aiFileTellProc  TellProc
Callback to retrieve the current position of the file cursor (ftell()) More...

aiUserData  UserData
User-defined, opaque data. More...

aiFileWriteProc  WriteProc
Callback to write to a file. More...
Detailed Description

C-API: File callbacks.

Actually, it's a data structure to wrap a set of fXXX (e.g. fopen) replacement functions.

The default implementation of the functions utilizes the fXXX functions from the CRT. However, you can supply a custom implementation to Assimp by delivering a custom aiFileIO. Use this to enable reading from other sources, such as ZIP archives or memory locations.
Member Data Documentation

aiFileTellProc aiFile::FileSizeProc

Callback to retrieve the size of the file, in bytes.

aiFileFlushProc aiFile::FlushProc

Callback to flush the file contents.

aiFileReadProc aiFile::ReadProc

Callback to read from a file.

aiFileSeek aiFile::SeekProc

Callback to set the current position of the file cursor (fseek())

aiFileTellProc aiFile::TellProc

Callback to retrieve the current position of the file cursor (ftell())

aiUserData aiFile::UserData

User-defined, opaque data.

aiFileWriteProc aiFile::WriteProc

Callback to write to a file.

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

- cfileio.h
Public Attributes | List of all members

aiFileIO Struct Reference

C-API: File system callbacks. More...
Public Attributes

aiFileCloseProc  CloseProc
Function used to close an existing file. More...

aiFileOpenProc  OpenProc
Function used to open a new file. More...

aiUserData  UserData
User-defined, opaque data. More...
Detailed Description

C-API: File system callbacks.

Provided are functions to open and close files. Supply a custom structure to the import function. If you don't, a default implementation is used. Use custom file systems to enable reading from other sources, such as ZIPs or memory locations.
Member Data Documentation

aiFileCloseProc aiFileIO::CloseProc

Function used to close an existing file.

aiFileOpenProc aiFileIO::OpenProc

Function used to open a new file.

aiUserData aiFileIO::UserData

User-defined, opaque data.

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

- cfileio.h
Public Attributes | List of all members

aiImporterDesc Struct Reference

Meta information about a particular importer. More...
Public Attributes

const char * mAuthor

Original author (left blank if unknown or whole assimp team) More...

const char * mComments

Implementation comments, i.e. More...

const char * mFileExtensions

List of file extensions this importer can handle. More...

unsigned int mFlags

These flags indicate some characteristics common to many importers. More...

const char * mMaintainer

Current maintainer, left blank if the author maintains. More...

unsigned int mMaxMajor

Maximum format version that can be loaded im major.minor format, both are set to 0 if there is either no version scheme or if the loader doesn't care. More...

unsigned int mMaxMinor

unsigned mMinMajor
Minimum format version that can be loaded in major.minor format, both are set to 0 if there is either no version scheme or if the loader doesn't care. More...

unsigned int mMinMinor

const char * mName

Full name of the importer (i.e. More...)
**Detailed Description**

Meta information about a particular importer.

Importers need to fill this structure, but they can freely decide how talkative they are. A common use case for loader meta info is a user interface in which the user can choose between various import/export file formats. Building such an UI by hand means a lot of maintenance as importers/exporters are added to Assimp, so it might be useful to have a common mechanism to query some rough importer characteristics.
**Member Data Documentation**

*const char* aiImporterDesc::mAuthor

Original author (left blank if unknown or whole assimp team)

*const char* aiImporterDesc::mComments

Implementation comments, i.e.

unimplemented features

*const char* aiImporterDesc::mFileExtensions

List of file extensions this importer can handle.

List entries are separated by space characters. All entries are lower case without a leading dot (i.e. "xml dae" would be a valid value. Note that multiple importers may respond to the same file extension - assimp calls all importers in the order in which they are registered and each importer gets the opportunity to load the file until one importer "claims" the file. Apart from file extension checks, importers typically use other methods to quickly reject files (i.e. magic words) so this does not mean that common or generic file extensions such as XML would be tediously slow.

*unsigned int* aiImporterDesc::mFlags

These flags indicate some characteristics common to many importers.

*const char* aiImporterDesc::mMaintainer

Current maintainer, left blank if the author maintains.

*unsigned int* aiImporterDesc::mMaxMajor

Maximum format version that can be loaded im major.minor format, both are set to 0 if there is either no version scheme or if the loader doesn't care.
Loaders that expect to be forward-compatible to potential future format versions should indicate zero, otherwise they should specify the current maximum version.

```c
unsigned int aiImporterDesc::mMaxMinor
unsigned int aiImporterDesc::mMinMajor
```

Minimum format version that can be loaded im major.minor format, both are set to 0 if there is either no version scheme or if the loader doesn't care.

```c
unsigned int aiImporterDesc::mMinMinor
const char* aiImporterDesc::mName
```

Full name of the importer (i.e. Blender3D importer)

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

- importerdesc.h

---

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11*
aiLight Struct Reference

Helper structure to describe a light source. More...
Public Member Functions

aiLight ()
Public Attributes

float mAngleInnerCone
Inner angle of a spot light's light cone. More...

float mAngleOuterCone
Outer angle of a spot light's light cone. More...

float mAttenuationConstant
Constant light attenuation factor. More...

float mAttenuationLinear
Linear light attenuation factor. More...

float mAttenuationQuadratic
Quadratic light attenuation factor. More...

aiColor3D mColorAmbient
Ambient color of the light source. More...

aiColor3D mColorDiffuse
Diffuse color of the light source. More...

aiColor3D mColorSpecular
Specular color of the light source. More...

aiVector3D mDirection
Direction of the light source in space. More...

aiString mName
The name of the light source. More...
aiVector3D  mPosition
Position of the light source in space. More...

aiLightSourceType  mType
The type of the light source. More...
**Detailed Description**

Helper structure to describe a light source.

Assimp supports multiple sorts of light sources, including directional, point and spot lights. All of them are defined with just a single structure and distinguished by their parameters. Note - some file formats (such as 3DS, ASE) export a "target point" - the point a spot light is looking at (it can even be animated). Assimp writes the target point as a subnode of a spotlights's main node, called "<spotName>.Target". However, this is just additional information then, the transformation tracks of the main node make the spot light already point in the right direction.
Constructor & Destructor Documentation

aiLight::aiLight() inline
Member Data Documentation

float aiLight::mAngleInnerCone

Inner angle of a spot light's light cone.

The spot light has maximum influence on objects inside this angle. The angle is given in radians. It is 2PI for point lights and undefined for directional lights.

float aiLight::mAngleOuterCone

Outer angle of a spot light's light cone.

The spot light does not affect objects outside this angle. The angle is given in radians. It is 2PI for point lights and undefined for directional lights. The outer angle must be greater than or equal to the inner angle. It is assumed that the application uses a smooth interpolation between the inner and the outer cone of the spot light.

float aiLight::mAttenuationConstant

Constant light attenuation factor.

The intensity of the light source at a given distance 'd' from the light's position is

\[
\text{Atten} = \frac{1}{(att0 + att1 * d + att2 * d^2)}
\]

This member corresponds to the att0 variable in the equation. Naturally undefined for directional lights.

float aiLight::mAttenuationLinear

Linear light attenuation factor.

The intensity of the light source at a given distance 'd' from the light's position is

\[
\text{Atten} = \frac{1}{(att0 + att1 * d + att2 * d^2)}
\]
This member corresponds to the att1 variable in the equation. Naturally undefined for directional lights.

float aiLight::mAttenuationQuadratic

Quadratic light attenuation factor.

The intensity of the light source at a given distance 'd' from the light's position is

\[ \text{Atten} = \frac{1}{\text{att0} + \text{att1} \times d + \text{att2} \times d^2} \]

This member corresponds to the att2 variable in the equation. Naturally undefined for directional lights.

\[ \text{aiColor3D aiLight::mColorAmbient} \]

Ambient color of the light source.

The ambient light color is multiplied with the ambient material color to obtain the final color that contributes to the ambient shading term. Most renderers will ignore this value it, is just a remaining of the fixed-function pipeline that is still supported by quite many file formats.

\[ \text{aiColor3D aiLight::mColorDiffuse} \]

Diffuse color of the light source.

The diffuse light color is multiplied with the diffuse material color to obtain the final color that contributes to the diffuse shading term.

\[ \text{aiColor3D aiLight::mColorSpecular} \]

Specular color of the light source.

The specular light color is multiplied with the specular material color to obtain the final color that contributes to the specular shading term.

\[ \text{aiVector3D aiLight::mDirection} \]
Direction of the light source in space.

Relative to the transformation of the node corresponding to the light.

The direction is undefined for point lights. The vector may be normalized, but it needn't.

`aiString` aiLight::mName

The name of the light source.

There must be a node in the scenegraph with the same name. This node specifies the position of the light in the scene hierarchy and can be animated.

`aiVector3D` aiLight::mPosition

Position of the light source in space.

Relative to the transformation of the node corresponding to the light.

The position is undefined for directional lights.

`aiLightSourceType` aiLight::mType

The type of the light source.

aiLightSource_UNDEFINED is not a valid value for this member.

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

- light.h
Public Attributes | List of all members

aiLogStream Struct Reference

C-API: Represents a log stream. More...
Public Attributes

`aiLogStreamCallback`  callback

callback to be called [More...](#)

char *  `user`

user data to be passed to the callback [More...](#)
Detailed Description

C-API: Represents a log stream.

A log stream receives all log messages and streams them somewhere.

See also
- aiGetPredefinedLogStream
- aiAttachLogStream
- aiDetachLogStream
Member Data Documentation

`aiLogStreamCallback` aiLogStream::callback

callback to be called

`char* aiLogStream::user`

user data to be passed to the callback

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

- `cimport.h`

Generated on Sun Feb 21 2016 19:42:29 for Assimp by Doxygen 1.8.11
Data structure for a material. More...
Public Member Functions

- **AddBinaryProperty** (const void *pInput, unsigned int pSizeInBytes, const char *pKey, unsigned int type, unsigned int index, aiPropertyTypeInfo pType)
  Add a property with a given key and type info to the material structure. More...

- **AddProperty** (const aiString *pInput, const char *pKey, unsigned int type=0, unsigned int index=0)
  Add a string property with a given key and type info to the material structure. More...

- **template<class TYPE >**
  **AddProperty** (const TYPE *pInput, unsigned int pNumValues, const char *pKey, unsigned int type=0, unsigned int index=0)
  Add a property with a given key to the material structure. More...

- **AddProperty** (const aiVector3D *pInput, unsigned int pNumValues, const char *pKey, unsigned int type=0, unsigned int index=0)

- **AddProperty** (const aiColor3D *pInput, unsigned int pNumValues, const char *pKey, unsigned int type=0, unsigned int index=0)

- **AddProperty** (const aiColor4D *pInput, unsigned int pNumValues, const char *pKey, unsigned int type=0, unsigned int index=0)

- **AddProperty** (const int *pInput, unsigned int pNumValues, const char *pKey, unsigned int type=0, unsigned int index=0)

- **AddProperty** (const float *pInput, unsigned int pNumValues, const char *pKey, unsigned int type=0, unsigned int index=0)
aiReturn **AddProperty** (const aiUVTransform *pInput, unsigned int pNumValues, const char *pKey, unsigned int type=0, unsigned int index=0)

**aiMaterial** ()

void **Clear** ()
Removes all properties from the material. [More...]

template<typename Type >
aiReturn **Get** (const char *pKey, unsigned int type, unsigned int idx, Type *pOut, unsigned int *pMax) const
Retrieve an array of Type values with a specific key from the material. [More...]

aiReturn **Get** (const char *pKey, unsigned int type, unsigned int idx, int *pOut, unsigned int *pMax) const

aiReturn **Get** (const char *pKey, unsigned int type, unsigned int idx, float *pOut, unsigned int *pMax) const

template<typename Type >
aiReturn **Get** (const char *pKey, unsigned int type, unsigned int idx, Type &pOut) const
Retrieve a Type value with a specific key from the material. [More...]

aiReturn **Get** (const char *pKey, unsigned int type, unsigned int idx, int &pOut) const

aiReturn **Get** (const char *pKey, unsigned int type, unsigned int idx, float &pOut) const

aiReturn **Get** (const char *pKey, unsigned int type, unsigned int idx, aiString &pOut) const
Get (const char *pKey, unsigned int type, unsigned int idx, aiColor3D &pOut) const

Get (const char *pKey, unsigned int type, unsigned int idx, aiColor4D &pOut) const

Get (const char *pKey, unsigned int type, unsigned int idx, aiUVTransform &pOut) const

GetTexture (aiTextureType type, unsigned int index, aiString *path, aiTextureMapping *mapping=NULL, unsigned int *uvindex=NULL, float *blend=NULL, aiTextureOp *op=NULL, aiTextureMapMode *mapmode=NULL) const

Helper function to get all parameters pertaining to a particular texture slot from a material. More...

GetTextureCount (aiTextureType type) const

Get the number of textures for a particular texture type. More...

RemoveProperty (const char *pKey, unsigned int type=0, unsigned int index=0)
Remove a given key from the list. More...

~aiMaterial ()
Static Public Member Functions

static void CopyPropertyList (aiMaterial *pcDest, const aiMaterial *pcSrc)
Copy the property list of a material. More...
Public Attributes

unsigned int mNumAllocated
Storage allocated. More...

unsigned int mNumProperties
Number of properties in the data base. More...

aiMaterialProperty ** mProperties
List of all material properties loaded. More...
Detailed Description

Data structure for a material.

Material data is stored using a key-value structure. A single key-value pair is called a 'material property'. C++ users should use the provided member functions of aiMaterial to process material properties, C users have to stick with the aiMaterialGetXXX family of unbound functions. The library defines a set of standard keys (AI_MATKEY_XXX).
Constructor & Destructor Documentation

aiMaterial::aiMaterial ( )
aiMaterial::~aiMaterial ( )
Member Function Documentation

```cpp
aiReturn aiMaterial::AddBinaryProperty ( const void * pInput,
                                           unsigned int pSizeInBytes,
                                           const char * pKey,
                                           unsigned int type,
                                           unsigned int index,
                                           aiPropertyTypeInfo pType
                                        )
```

Add a property with a given key and type info to the material structure.

Parameters
- **pInput**: Pointer to input data
- **pSizeInBytes**: Size of input data
- **pKey**: Key/Usage of the property (AI_MATKEY_XXX)
- **type**: Set by the AI_MATKEY_XXX macro
- **index**: Set by the AI_MATKEY_XXX macro
- **pType**: Type information hint

```cpp
aiReturn aiMaterial::AddProperty ( const aiString * pInput,
                                      const char * pKey,
                                      unsigned int type = 0,
                                      unsigned int index = 0
                                    )
```

Add a string property with a given key and type info to the material structure.

Parameters
- **pInput**: Input string
- **pKey**: Key/Usage of the property (AI_MATKEY_XXX)
- **type**: Set by the AI_MATKEY_XXX macro
- **index**: Set by the AI_MATKEY_XXX macro
template<class TYPE >
aiReturn aiMaterial::AddProperty ( const TYPE * pInput,
   unsigned int pNumValues,
   const char * pKey,
   unsigned int type = 0,
   unsigned int index = 0 )

Add a property with a given key to the material structure.

Parameters
pInput Pointer to the input data
pNumValues Number of values in the array
pKey Key/Usage of the property (AI_MATKEY_XXX)
type Set by the AI_MATKEY_XXX macro
index Set by the AI_MATKEY_XXX macro

aiReturn aiMaterial::AddProperty ( const aiVector3D * pInput,
   unsigned int pNumValues,
   const char * pKey,
   unsigned int type = 0,
   unsigned int index = 0 )

aiReturn aiMaterial::AddProperty ( const aiColor3D * pInput,
   unsigned int pNumValues,
   const char * pKey,
   unsigned int type = 0,
   unsigned int index = 0 )

aiReturn aiMaterial::AddProperty ( const aiColor4D * pInput,
   unsigned int pNumValues,
   const char * pKey,
   unsigned int type = 0,
   unsigned int index = 0 )
aiReturn aiMaterial::AddProperty ( const int * pInput,
    unsigned int pNumValues,
    const char * pKey,
    unsigned int type = 0,
    unsigned int index = 0 )
)
aiReturn aiMaterial::AddProperty ( const float * pInput,
    unsigned int pNumValues,
    const char * pKey,
    unsigned int type = 0,
    unsigned int index = 0 )
)
aiReturn aiMaterial::AddProperty ( const aiUVTransform * pInput,
    unsigned int pNumValues,
    const char * pKey,
    unsigned int type = 0,
    unsigned int index = 0 )
)

void aiMaterial::Clear ( )

Removes all properties from the material.

The data array remains allocated so adding new properties is quite fast.

static void aiMaterial::CopyPropertyList ( aiMaterial * pcDest,
    const aiMaterial * pcSrc  static )
)

Copy the property list of a material.

Parameters
   pcDest Destination material
   pcSrc  Source material
template<typename Type >
aiReturn aiMaterial::Get ( const char * pKey,
    unsigned int type,
    unsigned int idx,
    Type * pOut,
    unsigned int * pMax
) const

Retrieve an array of Type values with a specific key from the material.

Parameters
pKey Key to search for. One of the AI_MATKEY_XXX constants.
type .. set by AI_MATKEY_XXX
idx .. set by AI_MATKEY_XXX
pOut Pointer to a buffer to receive the result.
pMax Specifies the size of the given buffer, in Type's. Receives the number of values (not bytes!) read. NULL is a valid value for this parameter.
Retrieve a Type value with a specific key from the material.

Parameters

- **pKey** Key to search for. One of the AI_MATKEY_XXX constants.
- **type** Specifies the type of the texture to be retrieved (e.g. diffuse, specular, height map ...)
- **idx** Index of the texture to be retrieved.
- **pOut** Reference to receive the output value

```cpp
aiReturn aiMaterial::Get(const char *pKey,
                         unsigned int type,
                         unsigned int idx,
                         int & pOut)

aiReturn aiMaterial::Get(const char *pKey,
                         unsigned int type,
                         unsigned int idx,
                         float & pOut)

aiReturn aiMaterial::Get(const char *pKey,
                         unsigned int type,
                         unsigned int idx,
                         aiString & pOut)

aiReturn aiMaterial::Get(const char *pKey,
                         unsigned int type,
                         unsigned int idx,
                         aiColor3D & pOut)
```
unsigned int  idx,
aiColor4D &  pOut
) const

aiReturn aiMaterial::Get ( const char *  pKey,
unsigned int  type,
unsigned int  idx,
aiUVTransform &  pOut
) const

aiReturn aiMaterial::GetTexture ( aiTextureType  type,
unsigned int  index,
aiString *  path,
aiTextureMapping *  mapping = NULL,
unsigned int *  uvindex = NULL,
float *  blend = NULL,
aiTextureOp *  op = NULL,
aiTextureMapMode *  mapmode = NULL
) const

Helper function to get all parameters pertaining to a particular texture slot from a material.

This function is provided just for convenience, you could also read the single material properties manually.

Parameters

- **type**: Specifies the type of the texture to be retrieved (e.g. diffuse, specular, height map ...)
- **index**: Index of the texture to be retrieved. The function fails if there is no texture of that type with this index. GetTextureCount() can be used to determine the number of textures per texture type.
- **path**: Receives the path to the texture. NULL is a valid value.
- **mapping**: The texture mapping. NULL is allowed as value.
- **uvindex**: Receives the UV index of the texture. NULL is a valid value.
- **blend**: Receives the blend factor for the texture NULL is a valid value.
op Receives the texture operation to be performed between this texture and the previous texture. NULL is allowed as value.
mapmode Receives the mapping modes to be used for the texture. The parameter may be NULL but if it is a valid pointer it MUST point to an array of 3 aiTextureMapMode's (one for each axis: UVW order (=XYZ)).

unsigned int aiMaterial::GetTextureCount (aiTextureType type) const

Get the number of textures for a particular texture type.

Parameters
    type Texture type to check for

Returns
    Number of textures for this type.

Note
    A texture can be easily queried using GetTexture()

aiReturn aiMaterial::RemoveProperty ( const char * pKey,
                                          unsigned int type = 0,
                                          unsigned int index = 0 )

Remove a given key from the list.

The function fails if the key isn't found

Parameters
    pKey Key to be deleted
    type Set by the AI_MATKEY_XXX macro
    index Set by the AI_MATKEY_XXX macro
**Member Data Documentation**

unsigned int aiMaterial::mNumAllocated

Storage allocated.

unsigned int aiMaterial::mNumProperties

Number of properties in the data base.

aiMaterialProperty** aiMaterial::mProperties

List of all material properties loaded.

---

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

- [material.h](#)

---

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by [doxygen](#) 1.8.11*
aiMatrix3x3t< TReal > Class Template Reference

Represents a row-major 3x3 matrix. More...
Public Member Functions

aiMatrix3x3t ()

aiMatrix3x3t (TReal _a1, TReal _a2, TReal _a3, TReal _b1, TReal _b2, TReal _b3, TReal _c1, TReal _c2, TReal _c3)

aiMatrix3x3t (const aiMatrix4x4t< TReal > &pMatrix)
Construction from a 4x4 matrix. More...

TReal Determinant () const

bool Equal (const aiMatrix4x4t< TReal > &m, TReal epsilon=1e-6) const

aiMatrix3x3t & Inverse ()
Invert the matrix. More...

template<typename TOther >
operator aiMatrix3x3t< TOther > () const

bool operator!= (const aiMatrix4x4t< TReal > &m) const

aiMatrix3x3t operator* (const aiMatrix3x3t &m) const

aiMatrix3x3t & operator*= (const aiMatrix3x3t &m)

bool operator== (const aiMatrix4x4t< TReal > &m) const

TReal * operator[] (unsigned int p_iIndex)
const TReal * operator[](unsigned int p_iIndex) const

aiMatrix3x3t & Transpose()

Transpose the matrix. More...
Static Public Member Functions

static aiMatrix3x3t FromToMatrix (const aiVector3t< TReal >& from, const aiVector3t< TReal >& to, aiMatrix3x3t &out)

A function for creating a rotation matrix that rotates a vector called "from" into another vector called "to". More...

static aiMatrix3x3t Rotation (TReal a, const aiVector3t< TReal >& axis, aiMatrix3x3t &out)

Returns a rotation matrix for a rotation around an arbitrary axis. More...

static aiMatrix3x3t RotationZ (TReal a, aiMatrix3x3t &out)

Returns a rotation matrix for a rotation around z. More...

static aiMatrix3x3t Translation (const aiVector2t< TReal >& v, aiMatrix3x3t &out)

Returns a translation matrix. More...
Public Attributes

union {
    struct {
        TReal   a1
        TReal   a2
        TReal   a3
        TReal   b1
        TReal   b2
        TReal   b3
        TReal   c1
        TReal   c2
        TReal   c3
    }
    TReal   m [3][3]
    TReal   mData [9]
};
Detailed Description

template<	typename TReal>
class aiMatrix3x3t< TReal >

Represents a row-major 3x3 matrix.

There's much confusion about matrix layouts (column vs. row order). This is always a row-major matrix. Not even with the aiProcess_ConvertToLeftHanded flag, which absolutely does not affect matrix order - it just affects the handedness of the coordinate system defined thereby.
Constructor & Destructor Documentation

template<typename TReal >
aiMatrix3x3t< TReal >::aiMatrix3x3t ( ) inline
template<typename TReal >
aiMatrix3x3t< TReal >::aiMatrix3x3t ( TReal _a1,
                      TReal _a2,
                      TReal _a3,
                      TReal _b1,
                      TReal _b2,
                      TReal _b3, inline
                      TReal _c1,
                      TReal _c2,
                      TReal _c3 )

template<typename TReal >
aiMatrix3x3t< TReal >::aiMatrix3x3t ( const aiMatrix4x4t< TReal > & pMatrix ) inline explicit

Construction from a 4x4 matrix.

The remaining parts of the matrix are ignored.
Member Function Documentation

template<typename TReal >
TReal aiMatrix3x3t<TReal>::Determinant ( ) const inline

template<typename TReal >
bool aiMatrix3x3t<TReal>::Equal ( const aiMatrix4x4t<TReal> & m,
TReal epsilon = inline 1e-6 const
)

template<typename TReal >
aiMatrix3x3t<TReal> & aiMatrix3x3t<TReal>::FromToMatrix ( const aiVector3t<TReal> & from,
const aiVector3t<TReal> & to, inlinestatic
aiMatrix3x3t<TReal> & out
)

A function for creating a rotation matrix that rotates a vector called "from" into another vector called "to".


template<typename TReal >
aiMatrix3x3t<TReal> & aiMatrix3x3t<TReal>::Inverse ( ) inline

Invert the matrix.

If the matrix is not invertible all elements are set to qnan. Beware, use (f != f) to check whether a TReal f is qnan.
template<typename TReal>
template<typename TOther>
aiMatrix3x3t<TReal>::operator aiMatrix3x3t<TOther> ( ) const

bool aiMatrix3x3t<TReal>::operator!= ( const aiMatrix4x4t<TReal> & m ) const inline

aiMatrix3x3t<TReal> & aiMatrix3x3t<TReal>::operator* ( const aiMatrix3x3t<TReal> & m ) const inline

aiMatrix3x3t<TReal> & aiMatrix3x3t<TReal>::operator*=( const aiMatrix3x3t<TReal> & m ) inline

bool aiMatrix3x3t<TReal>::operator== ( const aiMatrix4x4t<TReal> & m ) const inline

TReal * aiMatrix3x3t<TReal>::operator[]( unsigned int p_iIndex ) inline

const TReal * aiMatrix3x3t<TReal>::operator[]( unsigned int p_iIndex ) const inline

aiMatrix3x3t<TReal> &
aiMatrix3x3t<TReal>::Rotation ( TReal a,
const aiVector3t<TReal> & axis, inlinestatic
aiMatrix3x3t<TReal> & out )

Returns a rotation matrix for a rotation around an arbitrary axis.

Parameters
  a  Rotation angle, in radians
  axis Axis to rotate around
template<typename TReal >
aiMatrix3x3t< TReal > & aiMatrix3x3t< TReal >::RotationZ(TReal a, aiMatrix3x3t< TReal > & out)
inline static

Returns a rotation matrix for a rotation around z.

Parameters
  a  Rotation angle, in radians
  out Receives the output matrix

Returns
  Reference to the output matrix

template<typename TReal >
aiMatrix3x3t< TReal > & aiMatrix3x3t< TReal >::Translation(const aiVector2t< TReal > & v, aiMatrix3x3t< TReal > & out)
inline static

Returns a translation matrix.

Parameters
  v  Translation vector
  out Receives the output matrix

Returns
  Reference to the output matrix

template<typename TReal >
aiMatrix3x3t< TReal > & aiMatrix3x3t< TReal >::Transpose()
inline
Transpose the matrix.
Member Data Documentation

union { ... }

template<typename TReal >
TReal aiMatrix3x3t< TReal >::a1

template<typename TReal >
TReal aiMatrix3x3t< TReal >::a2

template<typename TReal >
TReal aiMatrix3x3t< TReal >::a3

template<typename TReal >
TReal aiMatrix3x3t< TReal >::b1

template<typename TReal >
TReal aiMatrix3x3t< TReal >::b2

template<typename TReal >
TReal aiMatrix3x3t< TReal >::b3

template<typename TReal >
TReal aiMatrix3x3t< TReal >::c1

template<typename TReal >
TReal aiMatrix3x3t< TReal >::c2

template<typename TReal >
TReal aiMatrix3x3t< TReal >::c3

template<typename TReal >
TReal aiMatrix3x3t< TReal >::m[3][3]

template<typename TReal >
TReal aiMatrix3x3t< TReal >::mData[9]

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

- matrix3x3.h
- matrix3x3.inl

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp v3.1.1 (June 2014)

Public Member Functions | Static Public Member Functions | Public Attributes | List of all members
aiMatrix4x4t< TReal > Class Template Reference

 Represents a row-major 4x4 matrix, use this for homogeneous coordinates.
More...
Public Member Functions

aiMatrix4x4t ()
set to identity More...

aiMatrix4x4t (TReal _a1, TReal _a2, TReal _a3, TReal _a4,
TReal _b1, TReal _b2, TReal _b3, TReal _b4, TReal _c1, TReal
_c2, TReal _c3, TReal _c4, TReal _d1, TReal _d2, TReal _d3,
TReal _d4)
construction from single values More...

aiMatrix4x4t (const aiMatrix3x3t< TReal >& m)
construction from 3x3 matrix, remaining elements are set to
identity More...

aiMatrix4x4t (const aiVector3t< TReal >& scaling, const
aiQuaterniont< TReal >& rotation, const aiVector3t< TReal >&
position)
construction from position, rotation and scaling components
More...

void Decompose (aiVector3t< TReal >& scaling, aiQuaterniont<
TReal >& rotation, aiVector3t< TReal >& position) const
Decompose a trafo matrix into its original components. More...

void DecomposeNoScaling (aiQuaterniont< TReal >& rotation,
aiVector3t< TReal >& position) const
Decompose a trafo matrix with no scaling into its original
components. More...

TReal Determinant () const

bool Equal (const aiMatrix4x4t & m, TReal epsilon=1e-6) const
aiMatrix4x4t & FromEulerAnglesXYZ (TReal x, TReal y, TReal z)
    Creates a trafo matrix from a set of euler angles. More...

aiMatrix4x4t & FromEulerAnglesXYZ (const aiVector3t<TReal> &blubb)

aiMatrix4x4t & Inverse ()
    Invert the matrix. More...

bool IsIdentity () const
    Returns true of the matrix is the identity matrix. More...

template<typename TOther >
    operator aiMatrix4x4t<TOther > () const

    bool operator!= (const aiMatrix4x4t &m) const

aiMatrix4x4t & operator* (const aiMatrix4x4t &m) const

aiMatrix4x4t & operator*=(const aiMatrix4x4t &m)

    bool operator== (const aiMatrix4x4t &m) const

    TReal * operator[] (unsigned int p_iIndex)

const TReal * operator[] (unsigned int p_iIndex) const

aiMatrix4x4t & Transpose ()
Transpose the matrix. More...
**Static Public Member Functions**

```cpp
class aiMatrix4x4t
{
    public:
        static aiMatrix4x4t & FromToMatrix (const aiVector3t<TReal> &from, const aiVector3t<TReal> &to, aiMatrix4x4t &out)
        {
            // A function for creating a rotation matrix that rotates a vector called "from" into another vector called "to". More...
        }

        static aiMatrix4x4t & Rotation (TReal a, const aiVector3t<TReal> &axis, aiMatrix4x4t &out)
        {
            // Returns a rotation matrix for a rotation around an arbitrary axis. More...
        }

        static aiMatrix4x4t & RotationX (TReal a, aiMatrix4x4t &out)
        {
            // Returns a rotation matrix for a rotation around the x axis. More...
        }

        static aiMatrix4x4t & RotationY (TReal a, aiMatrix4x4t &out)
        {
            // Returns a rotation matrix for a rotation around the y axis. More...
        }

        static aiMatrix4x4t & RotationZ (TReal a, aiMatrix4x4t &out)
        {
            // Returns a rotation matrix for a rotation around the z axis. More...
        }

        static aiMatrix4x4t & Scaling (const aiVector3t<TReal> &v, aiMatrix4x4t &out)
        {
            // Returns a scaling matrix. More...
        }
};
```
static aiMatrix4x4t Translation (const aiVector3t<TReal>& v, aiMatrix4x4t& out)

  Returns a translation matrix. More...
Public Attributes

union {
   struct {
      TReal  a1
      TReal  a2
      TReal  a3
      TReal  a4
      TReal  b1
      TReal  b2
      TReal  b3
      TReal  b4
      TReal  c1
      TReal  c2
      TReal  c3
      TReal  c4
      TReal  d1
      TReal  d2
   }
}

TReal  \texttt{d3}

TReal  \texttt{d4}

}

TReal  \texttt{m} [4][4]

TReal  \texttt{mData} [16]

};
Detailed Description

template<typename TReal>
class aiMatrix4x4t<TReal>

Represents a row-major 4x4 matrix, use this for homogeneous coordinates.

There's much confusion about matrix layouts (column vs. row order). This is always a row-major matrix. Not even with the aiProcess_ConvertToLeftHanded flag, which absolutely does not affect matrix order - it just affects the handedness of the coordinate system defined thereby.
Constructor & Destructor Documentation

template<typename TReal>
aiMatrix4x4t<TReal>::aiMatrix4x4t()  
set to identity

template<typename TReal>
aiMatrix4x4t<TReal>::aiMatrix4x4t(TReal _a1,  
TReal _a2,  
TReal _a3,  
TReal _a4,  
TReal _b1,  
TReal _b2,  
TReal _b3,  
TReal _b4,  
TReal _c1,  
TReal _c2,  
TReal _c3,  
TReal _c4,  
TReal _d1,  
TReal _d2,  
TReal _d3,  
TReal _d4  
)  
construction from single values

template<typename TReal>
aiMatrix4x4t<TReal>::aiMatrix4x4t(const aiMatrix3x3t<TReal>& m) inline explicit  
construction from 3x3 matrix, remaining elements are set to identity
template<typename TReal>
aiMatrix4x4t<TReal>
> ::= aiMatrix4x4t

( const aiVector3t<TReal> & scaling,
const aiQuaternion<TReal> & rotation,
const aiVector3t<TReal> & position
)

construction from position, rotation and scaling components

Parameters

scaling   The scaling for the x,y,z axes
rotation  The rotation as a hamilton quaternion
position  The position for the x,y,z axes
Member Function Documentation

```cpp
template<typename TReal>
void aiMatrix4x4t<TReal>::Decompose
    ( aiVector3t<TReal> & scaling,
      aiQuaterniont<TReal> & rotation, inline
      aiVector3t<TReal> & position
    ) const

Decompose a trafo matrix into its original components.

Parameters
  scaling  Receives the output scaling for the x,y,z axes
  rotation Receives the output rotation as a hamilton quaternion
  position Receives the output position for the x,y,z axes

```

```cpp
template<typename TReal>
void aiMatrix4x4t<TReal>::DecomposeNoScaling
    ( aiQuaterniont<TReal> & rotation,
      aiVector3t<TReal> & position inline
    ) const

Decompose a trafo matrix with no scaling into its original components.

Parameters
  rotation Receives the output rotation as a hamilton quaternion
  position Receives the output position for the x,y,z axes
```

```cpp
template<typename TReal>
TReal aiMatrix4x4t<TReal>::Determinant ( ) const inline
```

```cpp
bool aiMatrix4x4t<TReal> const aiMatrix4x4t<TReal>
```

template<typename TReal>
aiMatrix4x4t<TReal> & aiMatrix4x4t<TReal>::FromEulerAnglesXYZ(TReal x, TReal y, TReal z)

Creates a trafo matrix from a set of euler angles.

Parameters
x Rotation angle for the x-axis, in radians
y Rotation angle for the y-axis, in radians
z Rotation angle for the z-axis, in radians

A function for creating a rotation matrix that rotates a vector called "from" into another vector called "to".

Input : from[3], to[3] which both must be normalized non-zero vectors Output: mtx[3][3] – a 3x3 matrix in column-major form Authors: Tomas Mueller, John

Input: from[3], to[3] which both must be normalized non-zero vectors
Output: mtx[3][3] – a 3x3 matrix in column-major form


template<typename TReal >


aiMatrix4x4t<TReal> & aiMatrix4x4t<TReal>::Inverse ( ) inline

Invert the matrix.

If the matrix is not invertible all elements are set to qnan. Beware, use (f != f) to check whether a TReal f is qnan.

template<typename TReal >


bool aiMatrix4x4t<TReal>::IsIdentity ( ) const inline

Returns true if the matrix is the identity matrix.

The check is performed against a not so small epsilon.

template<typename TReal >
template<typename TOther >

aiMatrix4x4t<TReal> ::operator aiMatrix4x4t<TOther> ( ) const

template<typename TReal >


bool aiMatrix4x4t<TReal> ::operator!= ( const aiMatrix4x4t<TReal> & m ) const inline

template<typename TReal >


aiMatrix4x4t<TReal> & aiMatrix4x4t<TReal> ::operator* ( const aiMatrix4x4t<TReal> & m ) const inline

template<typename TReal >


aiMatrix4x4t<TReal> & aiMatrix4x4t<TReal> ::operator*= ( const aiMatrix4x4t<TReal> & m ) inline

bool aiMatrix4x4t<TReal> & aiMatrix4x4t<TReal> ::operator* ( const aiMatrix4x4t<TReal> & m ) inline

bool aiMatrix4x4t<TReal> & aiMatrix4x4t<TReal> ::operator* ( const aiMatrix4x4t<TReal> & m ) inline

const aiMatrix4x4t<TReal> & aiMatrix4x4t<TReal> ::operator* ( const aiMatrix4x4t<TReal> & m ) inline

const aiMatrix4x4t<TReal> & aiMatrix4x4t<TReal> ::operator* ( const aiMatrix4x4t<TReal> & m ) inline
template<typename TReal>
TReal * aiMatrix4x4t<TReal>::operator[]( unsigned int p_iIndex ) const inline

template<typename TReal>
const TReal * aiMatrix4x4t<TReal>::operator[]( unsigned int p_iIndex ) const inline

aiMatrix4x4t<TReal> & aiMatrix4x4t<TReal>::RotationX( TReal a, aiMatrix4x4t<TReal> & out )

Returns a rotation matrix for a rotation around the x axis.

Parameters

Returns

Reference to the output matrix
a Rotation angle, in radians
out Receives the output matrix

Returns
Reference to the output matrix

template<typename TReal>
aiMatrix4x4t< TReal > & aiMatrix4x4t< TReal >::RotationY
( TReal a,
aiMatrix4x4t< TReal > & out )
inlinestatic

Returns a rotation matrix for a rotation around the y axis.

Parameters
  a Rotation angle, in radians
  out Receives the output matrix

Returns
Reference to the output matrix

template<typename TReal>
aiMatrix4x4t< TReal > & aiMatrix4x4t< TReal >::RotationZ
( TReal a,
aiMatrix4x4t< TReal > & out )
inlinestatic

Returns a rotation matrix for a rotation around the z axis.

Parameters
  a Rotation angle, in radians
  out Receives the output matrix

Returns
Reference to the output matrix
template<typename TReal>
aiMatrix4x4t<TReal> & aiMatrix4x4t<TReal>::Scaling(const aiVector3t<TReal> & v, aiMatrix4x4t<TReal> & out) inline static

Returns a scaling matrix.

Parameters

v Scaling vector
out Receives the output matrix

Returns
Reference to the output matrix

template<typename TReal>
aiMatrix4x4t<TReal> & aiMatrix4x4t<TReal>::Translation(const aiVector3t<TReal> & v, aiMatrix4x4t<TReal> & out) inline static

Returns a translation matrix.

Parameters

v Translation vector
out Receives the output matrix

Returns
Reference to the output matrix

template<typename TReal>
aiMatrix4x4t<TReal> & aiMatrix4x4t<TReal>::Transpose() inline

Transpose the matrix.
Member Data Documentation

union { ... }
template<
    typename TReal>
    TReal aiMatrix4x4t<TReal>::a1
template<
    typename TReal>
    TReal aiMatrix4x4t<TReal>::a2
template<
    typename TReal>
    TReal aiMatrix4x4t<TReal>::a3
template<
    typename TReal>
    TReal aiMatrix4x4t<TReal>::a4
template<
    typename TReal>
    TReal aiMatrix4x4t<TReal>::b1
template<
    typename TReal>
    TReal aiMatrix4x4t<TReal>::b2
template<
    typename TReal>
    TReal aiMatrix4x4t<TReal>::b3
template<
    typename TReal>
    TReal aiMatrix4x4t<TReal>::b4
template<
    typename TReal>
    TReal aiMatrix4x4t<TReal>::c1
template<
    typename TReal>
    TReal aiMatrix4x4t<TReal>::c2
template<
    typename TReal>
    TReal aiMatrix4x4t<TReal>::c3
template<
    typename TReal>
    TReal aiMatrix4x4t<TReal>::c4
template<
    typename TReal>
    TReal aiMatrix4x4t<TReal>::d1
template<
    typename TReal>
    TReal aiMatrix4x4t<TReal>::d2
template<
    typename TReal>
    TReal aiMatrix4x4t<TReal>::d3
template<typename TReal>
TReal aiMatrix4x4t<TReal>::d4

template<typename TReal>
TReal aiMatrix4x4t<TReal>::m[4][4]

template<typename TReal>
TReal aiMatrix4x4t<TReal>::mData[16]

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

- matrix3x3.h
- matrix4x4.h
- matrix4x4.inl

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Stores the memory requirements for different components (e.g. More...)
Public Member Functions

aiMemoryInfo ()
Default constructor. More...
Public Attributes

unsigned int **animations**
Storage allocated for animation data. [More...]

unsigned int **cameras**
Storage allocated for camera data. [More...]

unsigned int **lights**
Storage allocated for light data. [More...]

unsigned int **materials**
Storage allocated for material data. [More...]

unsigned int **meshes**
Storage allocated for mesh data. [More...]

unsigned int **nodes**
Storage allocated for node data. [More...]

unsigned int **textures**
Storage allocated for texture data. [More...]

unsigned int **total**
Total storage allocated for the full import. [More...]
Detailed Description

Stores the memory requirements for different components (e.g. meshes, materials, animations) of an import. All sizes are in bytes.

See also
   Importer::GetMemoryRequirements()
Constructor & Destructor Documentation

aiMemoryInfo::aiMemoryInfo ( ) inline

Default constructor.
Member Data Documentation

unsigned int aiMemoryInfo::animations
Storage allocated for animation data.

unsigned int aiMemoryInfo::cameras
Storage allocated for camera data.

unsigned int aiMemoryInfo::lights
Storage allocated for light data.

unsigned int aiMemoryInfo::materials
Storage allocated for material data.

unsigned int aiMemoryInfo::meshes
Storage allocated for mesh data.

unsigned int aiMemoryInfo::nodes
Storage allocated for node data.

unsigned int aiMemoryInfo::textures
Storage allocated for texture data.

unsigned int aiMemoryInfo::total
Total storage allocated for the full import.

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

• types.h
A mesh represents a geometry or model with a single material. More...
Public Member Functions

aiMesh ()
Default constructor. Initializes all members to 0. More...

unsigned int GetNumColorChannels () const
Get the number of vertex color channels the mesh contains. More...

unsigned int GetNumUVChannels () const
Get the number of UV channels the mesh contains. More...

bool HasBones () const
Check whether the mesh contains bones. More...

bool HasFaces () const
Check whether the mesh contains faces. More...

bool HasNormals () const
Check whether the mesh contains normal vectors. More...

bool HasPositions () const
Check whether the mesh contains positions. More...

bool HasTangentsAndBitangents () const
Check whether the mesh contains tangent and bitangent vectors. 
It is not possible that it contains tangents and no bitangents (or the other way round). More...

bool HasTextureCoords (unsigned int pIndex) const
Check whether the mesh contains a texture coordinate set. More...
bool  **HasVertexColors** (unsigned int pIndex) const
    Check whether the mesh contains a vertex color set.  **More...**

**~aiMesh** ()
Deletes all storage allocated for the mesh.  **More...**
Public Attributes

aiAnimMesh ** mAnimMeshes

NOT CURRENTLY IN USE. More...

aiVector3D * mBitangents

Vertex bitangents. More...

aiBone ** mBones

The bones of this mesh. More...

aiColor4D * mColors [AI_MAX_NUMBER_OF_COLOR_SETS]

Vertex color sets. More...

aiFace * mFaces

The faces the mesh is constructed from. More...

unsigned int mMaterialIndex

The material used by this mesh. More...

aiString mName

Name of the mesh. More...

aiVector3D * mNormals

Vertex normals. More...

unsigned int mNumAnimMeshes

NOT CURRENTLY IN USE. More...

unsigned int mNumBones

The number of bones this mesh contains. More...
unsigned int mNumFaces
   The number of primitives (triangles, polygons, lines) in this mesh. More...

unsigned int mNumUVComponents
   [AI_MAX_NUMBER_OF_TEXTURECOORDS]
   Specifies the number of components for a given UV channel. More...

unsigned int mNumVertices
   The number of vertices in this mesh. More...

unsigned int mPrimitiveTypes
   Bitwise combination of the members of the aiPrimitiveType enum. More...

aiVector3D * mTangents
   Vertex tangents. More...

aiVector3D * mTextureCoords
   [AI_MAX_NUMBER_OF_TEXTURECOORDS]
   Vertex texture coords, also known as UV channels. More...

aiVector3D * mVertices
   Vertex positions. More...
**Detailed Description**

A mesh represents a geometry or model with a single material. It usually consists of a number of vertices and a series of primitives/faces referencing the vertices. In addition there might be a series of bones, each of them addressing a number of vertices with a certain weight. Vertex data is presented in channels with each channel containing a single per-vertex information such as a set of texture coords or a normal vector. If a data pointer is non-null, the corresponding data stream is present. From C++-programs you can also use the comfort functions Has*() to test for the presence of various data streams.

A Mesh uses only a single material which is referenced by a material ID.

**Note**

The mPositions member is usually not optional. However, vertex positions *could* be missing if the `AI_SCENE_FLAGS_INCOMPLETE` flag is set in `aiScene::mFlags`.

```
Constructor & Destructor Documentation

aiMesh::aiMesh ( ) inline

Default constructor. Initializes all members to 0.

aiMesh::~aiMesh ( ) inline

Deletes all storage allocated for the mesh.
Member Function Documentation

unsigned int aiMesh::GetNumColorChannels ( ) const inline
Get the number of vertex color channels the mesh contains.

unsigned int aiMesh::GetNumUVChannels ( ) const inline
Get the number of UV channels the mesh contains.

bool aiMesh::HasBones ( ) const inline
Check whether the mesh contains bones.

bool aiMesh::HasFaces ( ) const inline
Check whether the mesh contains faces.
If no special scene flags are set this should always return true

bool aiMesh::HasNormals ( ) const inline
Check whether the mesh contains normal vectors.

bool aiMesh::HasPositions ( ) const inline
Check whether the mesh contains positions.
Provided no special scene flags are set, this will always be true

bool aiMesh::HasTangentsAndBitangents ( ) const inline
Check whether the mesh contains tangent and bitangent vectors It is not possible that it contains tangents and no bitangents (or the other way round).
The existence of one of them implies that the second is there, too.
bool aiMesh::HasTextureCoords ( unsigned int pIndex ) const inline

Check whether the mesh contains a texture coordinate set.

Parameters
  pIndex Index of the texture coordinates set

bool aiMesh::HasVertexColors ( unsigned int pIndex ) const inline

Check whether the mesh contains a vertex color set.

Parameters
  pIndex Index of the vertex color set
Member Data Documentation

aiAnimMesh** aiMesh::mAnimMeshes

NOT CURRENTLY IN USE.

Attachment meshes for this mesh, for vertex-based animation. Attachment meshes carry replacement data for some of the mesh'es vertex components (usually positions, normals).

aiVector3D* aiMesh::mBitangents

Vertex bitangents.

The bitangent of a vertex points in the direction of the positive Y texture axis. The array contains normalized vectors, NULL if not present. The array is mNumVertices in size.

Note

If the mesh contains tangents, it automatically also contains bitangents.

aiBone** aiMesh::mBones

The bones of this mesh.

A bone consists of a name by which it can be found in the frame hierarchy and a set of vertex weights.

aiColor4D* aiMesh::mColors[AI_MAX_NUMBER_OF_COLOR_SETS]

Vertex color sets.

A mesh may contain 0 to AI_MAX_NUMBER_OF_COLOR_SETS vertex colors per vertex. NULL if not present. Each array is mNumVertices in size if present.

aiFace* aiMesh::mFaces
The faces the mesh is constructed from.

Each face refers to a number of vertices by their indices. This array is always present in a mesh, its size is given in mNumFaces. If the AI_SCENE_FLAGS_NON_VERBOSE_FORMAT is NOT set each face references an unique set of vertices.

unsigned int aiMesh::mMaterialIndex

The material used by this mesh.

A mesh uses only a single material. If an imported model uses multiple materials, the import splits up the mesh. Use this value as index into the scene's material list.

aiString aiMesh::mName

Name of the mesh.

Meshes can be named, but this is not a requirement and leaving this field empty is totally fine. There are mainly three uses for mesh names:

- some formats name nodes and meshes independently.
- importers tend to split meshes up to meet the one-material-per-mesh requirement. Assigning the same (dummy) name to each of the result meshes aids the caller at recovering the original mesh partitioning.
- Vertex animations refer to meshes by their names.

aiVector3D* aiMesh::mNormals

Vertex normals.

The array contains normalized vectors, NULL if not present. The array is mNumVertices in size. Normals are undefined for point and line primitives. A mesh consisting of points and lines only may not have normal vectors. Meshes with mixed primitive types (i.e. lines and triangles) may have normals, but the normals for vertices that are only referenced by point or line primitives are undefined and set to QNaN (WARN: qNaN compares to inequal to everything, even to qNaN itself. Using code like this to check whether a field is qnan is:

#define IS_QNAN(f) (f != f)

still dangerous because even 1.f == 1.f could evaluate to false! (remember the subtleties of IEEE754 arithmetics). Use stuff like fpclassify instead.

Note

Normal vectors computed by Assimp are always unit-length. However, this needn't apply for normals that have been taken directly from the model file.

unsigned int aiMesh::mNumAnimMeshes

NOT CURRENTLY IN USE.

The number of attachment meshes

unsigned int aiMesh::mNumBones

The number of bones this mesh contains.

Can be 0, in which case the mBones array is NULL.

unsigned int aiMesh::mNumFaces

The number of primitives (triangles, polygons, lines) in this mesh.

This is also the size of the mFaces array. The maximum value for this member is AI_MAX_FACES.

unsigned int
aiMesh::mNumUVComponents[AI_MAX_NUMBER_OF_TEXTURECOORDS]

Specifies the number of components for a given UV channel.

Up to three channels are supported (UVW, for accessing volume or cube maps). If the value is 2 for a given channel n, the component p.z of mTextureCoords[n][p] is set to 0.0f. If the value is 1 for a given channel, p.y is set to 0.0f, too.

Note

4D coords are not supported
unsigned int aiMesh::mNumVertices

The number of vertices in this mesh.

This is also the size of all of the per-vertex data arrays. The maximum value for this member is **AI_MAX_VERTICES**.

unsigned int aiMesh::mPrimitiveTypes

Bitwise combination of the members of the **aiPrimitiveType** enum.

This specifies which types of primitives are present in the mesh. The "SortByPrimitiveType"-Step can be used to make sure the output meshes consist of one primitive type each.

**aiVector3D*** aiMesh::mTangents

Vertex tangents.

The tangent of a vertex points in the direction of the positive X texture axis. The array contains normalized vectors, NULL if not present. The array is mNumVertices in size. A mesh consisting of points and lines only may not have normal vectors. Meshes with mixed primitive types (i.e. lines and triangles) may have normals, but the normals for vertices that are only referenced by point or line primitives are undefined and set to qNaN. See the **mNormals** member for a detailed discussion of qNaNs.

**Note**

If the mesh contains tangents, it automatically also contains bitangents.

**aiVector3D***

aiMesh::mTextureCoords[**AI_MAX_NUMBER_OF_TEXTURECOORDS**]

Vertex texture coords, also known as UV channels.

A mesh may contain 0 to **AI_MAX_NUMBER_OF_TEXTURECOORDS** per vertex. NULL if not present. The array is mNumVertices in size.

**aiVector3D*** aiMesh::mVertices
Vertex positions.

This array is always present in a mesh. The array is mNumVertices in size.

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

- mesh.h
aiMeshAnim Struct Reference

Describes vertex-based animations for a single mesh or a group of meshes.
More...
Public Member Functions

aiMeshAnim ()

~aiMeshAnim ()
Public Attributes

C_STRUCT aiMeshKey * mKeys
Key frames of the animation. More...

C_STRUCT aiString mName
Name of the mesh to be animated. More...

unsigned int mNumKeys
Size of the mKeys array. More...
**Detailed Description**

Describes vertex-based animations for a single mesh or a group of meshes.

Meshes carry the animation data for each frame in their `aiMesh::mAnimMeshes` array. The purpose of `aiMeshAnim` is to define keyframes linking each mesh attachment to a particular point in time.
Constructor & Destructor Documentation

aiMeshAnim::aiMeshAnim () inline
aiMeshAnim::~aiMeshAnim () inline
Member Data Documentation

**C_STRUCT aiMeshKey** aiMeshAnim::mKeys

Key frames of the animation.

May not be NULL.

**C_STRUCT aiString** aiMeshAnim::mName

Name of the mesh to be animated.

An empty string is not allowed, animated meshes need to be named (not necessarily uniquely, the name can basically serve as wildcard to select a group of meshes with similar animation setup)

unsigned int aiMeshAnim::mNumKeys

Size of the mKeys array.

Must be 1, at least.

---

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

- **anim.h**

---

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11*
Binds a anim mesh to a specific point in time. More...
Public Types

typedef unsigned int elem_type
Public Member Functions

aiMeshKey ()

aiMeshKey (double time, const unsigned int value)
Construction from a given time and key value. More...

bool operator!= (const aiMeshKey &o) const

bool operator< (const aiMeshKey &o) const

bool operator== (const aiMeshKey &o) const

bool operator> (const aiMeshKey &o) const
**Public Attributes**

double **mTime**

The time of this key. [More...](#)

unsigned int **mValue**

Index into the [aiMesh::mAnimMeshes](#) array of the mesh corresponding to the [aiMeshAnim](#) hosting this key frame. [More...](#)
Detailed Description

Binds a anim mesh to a specific point in time.
Member Typedef Documentation

typedef unsigned int aiMeshKey::elem_type
Constructor & Destructor Documentation

aiMeshKey::aiMeshKey ( ) inline
aiMeshKey::aiMeshKey ( double time,
              const unsigned int value  inline
            )

Construction from a given time and key value.
Member Function Documentation

bool aiMeshKey::operator!= ( const aiMeshKey & o ) const inline
bool aiMeshKey::operator< ( const aiMeshKey & o ) const inline
bool aiMeshKey::operator==( const aiMeshKey & o ) const inline
bool aiMeshKey::operator> ( const aiMeshKey & o ) const inline
Member Data Documentation

double aiMeshKey::mTime

The time of this key.

unsigned int aiMeshKey::mValue

Index into the aiMesh::mAnimMeshes array of the mesh corresponding to the aiMeshAnim hosting this key frame.

The referenced anim mesh is evaluated according to the rules defined in the docs for aiAnimMesh.

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

- anim.h

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Public Member Functions | Public Attributes | List of all members
aiMetadata Struct Reference

Container for holding metadata. More...
Public Member Functions

aiMetadata ()
Constructor. More...

template<typename T >
bool Get (unsigned index, T &value)

template<typename T >
bool Get (const aiString &key, T &value)

template<typename T >
bool Get (const std::string &key, T &value)

template<typename T >
void Set (unsigned index, const std::string &key, const T &value)

~aiMetadata ()
Destructor. More...
Public Attributes

**C_STRUCT aiString** * mKeys
Arrays of keys, may not be NULL. [More...](#)

unsigned int **mNumProperties**
Length of the mKeys and mValues arrays, respectively. [More...](#)

**C_STRUCT aiMetadataEntry** * mValues
Arrays of values, may not be NULL. [More...](#)
**Detailed Description**

Container for holding metadata.

Metadata is a key-value store using string keys and values.
Constructor & Destructor Documentation

aiMetadata::aiMetadata ( ) inline

Constructor.

aiMetadata::~aiMetadata ( ) inline

Destructor.
Member Function Documentation

template<typename T >
bool aiMetadata::Get ( unsigned index,
    T & value  inline
)
template<typename T >
bool aiMetadata::Get ( const aiString & key,
    T & value  inline
)
template<typename T >
bool aiMetadata::Get ( const std::string & key,
    T & value  inline
)
template<typename T >
void aiMetadata::Set ( unsigned index,
    const std::string & key,
    const T & value  inline
)
Member Data Documentation

**C_STRUCT aiString** aiMetadata::mKeys

Arrays of keys, may not be NULL.

Entries in this array may not be NULL as well.

**unsigned int aiMetadata::mNumProperties**

Length of the mKeys and mValues arrays, respectively.

**C_STRUCT aiMetadataEntry** aiMetadata::mValues

Arrays of values, may not be NULL.

Entries in this array may be NULL if the corresponding property key has no assigned value.

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

- [metadata.h](https://example.com/metadata.h)
Public Attributes | List of all members
aiMetadataEntry Struct Reference

Metadata entry. More..
Public Attributes

void * mData

aiMetadataType mType
Detailed Description

Metadata entry.

The type field uniquely identifies the underlying type of the data field.
**Member Data Documentation**

void* aiMetadataEntry::mData  
aiMetadataType aiMetadataEntry::mType

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

- [metadata.h](#)

---

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by [doxygen](#) 1.8.11*
Main Page | Related Pages | Namespaces | Classes | Files
---|---|---|---|---
Class List | Class Index | Class Hierarchy | Class Members | Assimp v3.1.1 (June 2014)

Public Member Functions | Public Attributes | List of all members
aiNode Struct Reference

A node in the imported hierarchy. More...
Public Member Functions

aiNode ()
Constructor. More...

aiNode (const std::string &name)
Construction from a specific name. More...

const aiNode * FindNode (const aiString &name) const
Searches for a node with a specific name, beginning at this nodes. More...

aiNode * FindNode (const aiString &name)

const aiNode * FindNode (const char *name) const

aiNode * FindNode (const char *name)

~aiNode ()
Destructor. More...
Public Attributes

aiNode ** mChildren
The child nodes of this node. More...

unsigned int * mMesses
The meshes of this node. More...

aiMetadata * mMetaData
Metadata associated with this node or NULL if there is no metadata. More...

aiString mName
The name of the node. More...

unsigned int mNumChildren
The number of child nodes of this node. More...

unsigned int mNumMeshes
The number of meshes of this node. More...

aiNode * mParent
Parent node. More...

aiMatrix4x4 mTransformation
The transformation relative to the node's parent. More...
Detailed Description

A node in the imported hierarchy.

Each node has name, a parent node (except for the root node), a transformation relative to its parent and possibly several child nodes. Simple file formats don't support hierarchical structures - for these formats the imported scene does consist of only a single root node without children.
Constructor & Destructor Documentation

aiNode::aiNode ( ) inline

Constructor.

aiNode::aiNode ( const std::string & name ) inline explicit

Construction from a specific name.

aiNode::~aiNode ( ) inline

Destructor.
Member Function Documentation

const aiNode* aiNode::FindNode ( const aiString & name ) const inline

Searches for a node with a specific name, beginning at this node.

Normally you will call this method on the root node of the scene.

Parameters
   name Name to search for

Returns
   NULL or a valid Node if the search was successful.
Member Data Documentation

**aiNode** aiNode::mChildren

The child nodes of this node.

NULL if mNumChildren is 0.

**unsigned int** aiNode::mMeshes

The meshes of this node.

Each entry is an index into the mesh list of the aiScene.

**aiMetadata** aiNode::mMetaData

Metadata associated with this node or NULL if there is no metadata.

Whether any metadata is generated depends on the source file format. See the importer_notes page for more information on every source file format. Importers that don't document any metadata don't write any.

**aiString** aiNode::mName

The name of the node.

The name might be empty (length of zero) but all nodes which need to be referenced by either bones or animations are named. Multiple nodes may have the same name, except for nodes which are referenced by bones (see aiBone and aiMesh::mBones). Their names must be unique.

Cameras and lights reference a specific node by name - if there are multiple nodes with this name, they are assigned to each of them.

There are no limitations with regard to the characters contained in the name string as it is usually taken directly from the source file.

Implementations should be able to handle tokens such as whitespace, tabs, line feeds, quotation marks, ampersands etc.
Sometimes assimp introduces new nodes not present in the source file into the hierarchy (usually out of necessity because sometimes the source hierarchy format is simply not compatible). Their names are surrounded by

<>

e.g.

<DummyRootNode>

.

unsigned int aiNode::mNumChildren

The number of child nodes of this node.

unsigned int aiNode::mNumMeshes

The number of meshes of this node.

aiNode* aiNode::mParent

Parent node.

NULL if this node is the root node.

aiMatrix4x4 aiNode::mTransformation

The transformation relative to the node's parent.

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

- scene.h

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Describes the animation of a single node. More...
Public Member Functions

\texttt{aiNodeAnim} ()

\texttt{~aiNodeAnim} ()
Public Attributes

C_STRUCT
aiString mNodeName
The name of the node affected by this animation. More...

unsigned int mNumPositionKeys
The number of position keys. More...

unsigned int mNumRotationKeys
The number of rotation keys. More...

unsigned int mNumScalingKeys
The number of scaling keys. More...

C_STRUCT
aiVectorKey * mPositionKeys
The position keys of this animation channel. More...

C_ENUM
aiAnimBehaviour mPostState
Defines how the animation behaves after the last key was processed. More...

C_ENUM
aiAnimBehaviour mPreState
Defines how the animation behaves before the first key is encountered. More...

C_STRUCT
aiQuatKey * mRotationKeys
The rotation keys of this animation channel. More...
C_STRUCT aiVectorKey * mScalingKeys

The scaling keys of this animation channel. More...
Detailed Description

Describes the animation of a single node.

The name specifies the bone/node which is affected by this animation channel. The keyframes are given in three separate series of values, one each for position, rotation and scaling. The transformation matrix computed from these values replaces the node's original transformation matrix at a specific time. This means all keys are absolute and not relative to the bone default pose. The order in which the transformations are applied is

- as usual - scaling, rotation, translation.

Note
All keys are returned in their correct, chronological order. Duplicate keys don't pass the validation step. Most likely there will be no negative time values, but they are not forbidden also (so implementations need to cope with them!)
Constructor & Destructor Documentation

aiNodeAnim::aiNodeAnim ( ) inline
aiNodeAnim::~aiNodeAnim ( ) inline
Member Data Documentation

**C_STRUCT aiString** aiNodeAnim::mNodeName

The name of the node affected by this animation.
The node must exist and it must be unique.

unsigned int aiNodeAnim::mNumPositionKeys

The number of position keys.

unsigned int aiNodeAnim::mNumRotationKeys

The number of rotation keys.

unsigned int aiNodeAnim::mNumScalingKeys

The number of scaling keys.

**C_STRUCT aiVectorKey*** aiNodeAnim::mPositionKeys

The position keys of this animation channel.

Positions are specified as 3D vector. The array is mNumPositionKeys in size.

If there are position keys, there will also be at least one scaling and one rotation key.

**C_ENUM aiAnimBehaviour** aiNodeAnim::mPostState

Defines how the animation behaves after the last key was processed.

The default value is aiAnimBehaviour_DEFAULT (the original transformation matrix of the affected node is taken).

**C_ENUM aiAnimBehaviour** aiNodeAnim::mPreState
Defines how the animation behaves before the first key is encountered.

The default value is aiAnimBehaviour_DEFAULT (the original transformation matrix of the affected node is used).

C_STRUCT aiQuatKey* aiNodeAnim::mRotationKeys

The rotation keys of this animation channel.

Rotations are given as quaternions, which are 4D vectors. The array is mNumRotationKeys in size.

If there are rotation keys, there will also be at least one scaling and one position key.

C_STRUCT aiVectorKey* aiNodeAnim::mScalingKeys

The scaling keys of this animation channel.

Scalings are specified as 3D vector. The array is mNumScalingKeys in size.

If there are scaling keys, there will also be at least one position and one rotation key.

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

- anim.h
Assimp v3.1.1 (June 2014)

Public Member Functions | Public Attributes | List of all members
aiPlane Struct Reference

Represents a plane in a three-dimensional, euclidean space. More...
Public Member Functions

aiPlane ()

aiPlane (float _a, float _b, float _c, float _d)

aiPlane (const aiPlane &o)
Public Attributes

float $a$
    Plane equation. More...

float $b$

float $c$

float $d$
Detailed Description

Represents a plane in a three-dimensional, euclidean space.
Constructor & Destructor Documentation

aiPlane::aiPlane( ) inline
aiPlane::aiPlane( float _a,
        float _b,
        float _c, inline
        float _d
    )
aiPlane::aiPlane( const aiPlane & o ) inline
Member Data Documentation

float aiPlane::a

Plane equation.

float aiPlane::b
float aiPlane::c
float aiPlane::d

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

- types.h

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
C-API: Represents an opaque set of settings to be used during importing. More...
Public Attributes

char sentinel
Detailed Description

C-API: Represents an opaque set of settings to be used during importing.

See also
- aiCreatePropertyStore
- aiReleasePropertyStore
- aiImportFileExWithProperties
- aiSetPropertyInteger
- aiSetPropertyFloat
- aiSetPropertyString
- aiSetPropertyMatrix
Member Data Documentation

char aiPropertyStore::sentinel

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

- cimport.h

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
aiQuaternion< TReal > Class Template Reference

Represents a quaternion in a 4D vector. More...
Public Member Functions

aiQuaternion()  

aiQuaternion(TReal pw, TReal px, TReal py, TReal pz)

aiQuaternion(const aiMatrix3x3t<TReal>& pRotMatrix)  
Construct from rotation matrix.  More...

aiQuaternion(TReal rotx, TReal roty, TReal rotz)  
Construct from euler angles.  More...

aiQuaternion(aiVector3t<TReal> axis, TReal angle)  
Construct from an axis-angle pair.  More...

aiQuaternion(aiVector3t<TReal> normalized)  
Construct from a normalized quaternion stored in a vec3.  More...

aiQuaternion & Conjugate()  
Compute quaternion conjugate.  More...

bool Equal(const aiQuaternion &o, TReal epsilon=1e-6) const

aiMatrix3x3t<TReal> GetMatrix() const  
Returns a matrix representation of the quaternion.  More...

aiQuaternion & Normalize()  
Normalize the quaternion.  More...
bool operator!=(const aiQuaternion &o) const

aiQuaternion operator* (const aiQuaternion &two) const
Multiply two quaternions. More...

bool operator==(const aiQuaternion &o) const

aiVector3<TReal> Rotate (const aiVector3<TReal> &in)
Rotate a point by this quaternion. More...
Static Public Member Functions

static Interpolate (aiQuaternion &pOut, const aiQuaternion &pStart, const aiQuaternion &pEnd, TReal pFactor)
  Performs a spherical interpolation between two quaternions and writes the result into the third. More...
Public Attributes

TReal \( w \)
\( w, x, y, z \) components of the quaternion More...

TReal \( x \)

TReal \( y \)

TReal \( z \)
Detailed Description

template<typename TReal>
class aiQuaternion<TReal>

Represents a quaternion in a 4D vector.
Constructor & Destructor Documentation

template<typename TReal >
aiQuaternion< TReal >::aiQuaternion( ) inline

template<typename TReal >
aiQuaternion< TReal >::aiQuaternion( TReal pw, TReal px, TReal py, inline TReal pz )

Construct from rotation matrix.
Result is undefined if the matrix is not orthonormal.

template<typename TReal >
aiQuaternion< TReal >::aiQuaternion( const aiMatrix3x3< TReal > & pRotMatrix ) inline explicit

Construct from euler angles.

template<typename TReal >
aiQuaternion< TReal >::aiQuaternion( TReal rotx, TReal roty, inline TReal rotz )

Construct from an axis-angle pair.
Construct from a normalized quaternion stored in a vec3.
Member Function Documentation

template<typename TReal >
aiQuaternion<TReal> & aiQuaternion<TReal>::Conjugate()
inline
Compute quaternion conjugate.

template<typename TReal >
bool aiQuaternion<TReal>::Equal(const aiQuaternion<TReal> & o,
TReal epsilon = 1e-6
) const
inline
Returns a matrix representation of the quaternion.

template<typename TReal >
void aiQuaternion<TReal>::Interpolate(aiQuaternion<TReal> & pOut,
const aiQuaternion<TReal> & pStart,
const aiQuaternion<TReal> & pEnd,
TReal pFactor
)
inline static
Performs a spherical interpolation between two quaternions and writes the result into the third.

Parameters
pOut Target object to received the interpolated rotation.
pStart Start rotation of the interpolation at factor == 0.
pEnd End rotation, factor == 1.
PFactor Interpolation factor between 0 and 1. Values outside of this range yield undefined results.

template<typename TReal >
aiQuaternion< TReal > & aiQuaternion< TReal >::Normalize ( ) inline

Normalize the quaternion.

template<typename TReal >
bool aiQuaternion< TReal >::operator!=( const aiQuaternion< TReal > & o ) const
template<typename TReal >
aiQuaternion< TReal > aiQuaternion< TReal >::operator* ( const aiQuaternion< TReal > & two ) const inline

Multiply two quaternions.

template<typename TReal >
bool aiQuaternion< TReal >::operator==( const aiQuaternion< TReal > & o ) const
template<typename TReal >
aiVector3< TReal > aiQuaternion< TReal >::Rotate ( const aiVector3< TReal > & in ) inline

Rotate a point by this quaternion.


**Member Data Documentation**

```cpp
template<typename TReal >
TReal aiQuaternion< typename TReal >::w
```

w, x, y, z components of the quaternion

```cpp
template<typename TReal >
TReal aiQuaternion< typename TReal >::x
```

```cpp
template<typename TReal >
TReal aiQuaternion< typename TReal >::y
```

```cpp
template<typename TReal >
TReal aiQuaternion< typename TReal >::z
```

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

- [matrix4x4.h](#)
- [quaternion.h](#)
- [quaternion.inl](#)
aiQuatKey Struct Reference

A time-value pair specifying a rotation for the given time. More...
Public Types

typedef aiQuat ernion elem_type
Public Member Functions

aiQuatKey ()

aiQuatKey (double time, const aiQuaternion &value)
Construction from a given time and key value. More...

bool operator!= (const aiQuatKey &o) const

bool operator< (const aiQuatKey &o) const

bool operator== (const aiQuatKey &o) const

bool operator> (const aiQuatKey &o) const
Public Attributes

double mTime
The time of this key. More...

C_STRUCT aiQuaternion mValue
The value of this key. More...
Detailed Description

A time-value pair specifying a rotation for the given time.

Rotations are expressed with quaternions.
Member Typedef Documentation

typedef aiQuaternion aiQuatKey::elem_type
Constructor & Destructor Documentation

aiQuatKey::aiQuatKey ( ) inline

aiQuatKey::aiQuatKey ( double time,
                      const aiQuaternion & value  inline )

Construction from a given time and key value.
Member Function Documentation

bool aiQuatKey::operator!= ( const aiQuatKey & o ) const inline
bool aiQuatKey::operator< ( const aiQuatKey & o ) const inline
bool aiQuatKey::operator== ( const aiQuatKey & o ) const inline
bool aiQuatKey::operator> ( const aiQuatKey & o ) const inline
Member Data Documentation

double aiQuatKey::mTime

The time of this key.

C_STRUCT aiQuaternion aiQuatKey::mValue

The value of this key.

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

- anim.h

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Represents a ray. More...
Public Member Functions

aiRay ()

aiRay (const aiVector3D &pos, const aiVector3D &dir)

aiRay (const aiRay &o)
Public Attributes

aiVector3D dir

aiVector3D pos

Position and direction of the ray. More...
Detailed Description

Represents a ray.
Constructor & Destructor Documentation

aiRay::aiRay ( ) inline
aiRay::aiRay ( const aiVector3D & _pos,
              const aiVector3D & _dir inline
              )
aiRay::aiRay ( const aiRay & o ) inline
Member Data Documentation

aiVector3D aiRay::dir
aiVector3D aiRay::pos

Position and direction of the ray.

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

- types.h

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
aiScene Struct Reference

The root structure of the imported data. More...
Public Member Functions

**ASSIMP_API aiScene ()**
Default constructor - set everything to 0/NULL. [More...](#)

bool **HasAnimations** () const
Check whether the scene contains animations. [More...](#)

bool **HasCameras** () const
Check whether the scene contains cameras. [More...](#)

bool **HasLights** () const
Check whether the scene contains lights. [More...](#)

bool **HasMaterials** () const
Check whether the scene contains materials Unless no special scene flags are set this will always be true. [More...](#)

bool **HasMeshes** () const
Check whether the scene contains meshes Unless no special scene flags are set this will always be true. [More...](#)

bool **HasTextures** () const
Check whether the scene contains textures. [More...](#)

**ASSIMP_API ~aiScene ()**
Destructor. [More...](#)
Public Attributes

aiAnimation ** mAnimations
The array of animations. More...

aiCamera ** mCameras
The array of cameras. More...

unsigned int mFlags
Any combination of the AI_SCENE_FLAGS_XXX flags. More...

aiLight ** mLights
The array of light sources. More...

aiMaterial ** mMaterials
The array of materials. More...

aiMesh ** mMeshes
The array of meshes. More...

unsigned int mNumAnimations
The number of animations in the scene. More...

unsigned int mNumCameras
The number of cameras in the scene. More...

unsigned int mNumLights
The number of light sources in the scene. More...

unsigned int mNumMaterials
The number of materials in the scene. More...

unsigned int \textbf{mNumMeshes}

The number of meshes in the scene. More...

unsigned int \textbf{mNumTextures}

The number of textures embedded into the file. More...

\textbf{void * mPrivate}

Internal data, do not touch. More...

\textbf{aiNode * mRootNode}

The root node of the hierarchy. More...

\textbf{aiTexture ** mTextures}

The array of embedded textures. More...
**Detailed Description**

The root structure of the imported data.

Everything that was imported from the given file can be accessed from here. Objects of this class are generally maintained and owned by Assimp, not by the caller. You shouldn't want to instance it, nor should you ever try to delete a given scene on your own.
Constructor & Destructor Documentation

**ASSIMP_API** aiScene::aiScene ( )

Default constructor - set everything to 0/NULL.

**ASSIMP_API** aiScene::~aiScene ( )

Destructor.
Member Function Documentation

bool aiScene::HasAnimations ( ) const inline
Check whether the scene contains animations.

bool aiScene::HasCameras ( ) const inline
Check whether the scene contains cameras.

bool aiScene::HasLights ( ) const inline
Check whether the scene contains lights.

bool aiScene::HasMaterials ( ) const inline
Check whether the scene contains materials Unless no special scene flags are set this will always be true.

bool aiScene::HasMeshes ( ) const inline
Check whether the scene contains meshes Unless no special scene flags are set this will always be true.

bool aiScene::HasTextures ( ) const inline
Check whether the scene contains textures.
Member Data Documentation

**aiAnimation**  aiScene::mAnimations

The array of animations.

All animations imported from the given file are listed here. The array is mNumAnimations in size.

**aiCamera**  aiScene::mCameras

The array of cameras.

All cameras imported from the given file are listed here. The array is mNumCameras in size. The first camera in the array (if existing) is the default camera view into the scene.

unsigned int aiScene::mFlags

Any combination of the AI_SCENE_FLAGS_XXX flags.

By default this value is 0, no flags are set. Most applications will want to reject all scenes with the AI_SCENE_FLAGS_INCOMPLETE bit set.

**aiLight**  aiScene::mLights

The array of light sources.

All light sources imported from the given file are listed here. The array is mNumLights in size.

**aiMaterial**  aiScene::mMaterials

The array of materials.

Use the index given in each aiMesh structure to access this array. The array is mNumMaterials in size. If the AI_SCENE_FLAGS_INCOMPLETE flag is not set there will always be at least ONE material.
The array of meshes.

Use the indices given in the aiNode structure to access this array. The array is mNumMeshes in size. If the AI_SCENE_FLAGS_INCOMPLETE flag is not set there will always be at least ONE material.

unsigned int aiScene::mNumAnimations

The number of animations in the scene.

unsigned int aiScene::mNumCameras

The number of cameras in the scene.
Cameras are fully optional, in most cases this attribute will be 0

unsigned int aiScene::mNumLights

The number of light sources in the scene.
Light sources are fully optional, in most cases this attribute will be 0

unsigned int aiScene::mNumMaterials

The number of materials in the scene.

unsigned int aiScene::mNumMeshes

The number of meshes in the scene.

unsigned int aiScene::mNumTextures

The number of textures embedded into the file.

void* aiScene::mPrivate

Internal data, do not touch.
aiNode* aiScene::mRootNode

The root node of the hierarchy.

There will always be at least the root node if the import was successful (and no special flags have been set). Presence of further nodes depends on the format and content of the imported file.

aiTexture** aiScene::mTextures

The array of embedded textures.

Not many file formats embed their textures into the file. An example is Quake's MDL format (which is also used by some GameStudio versions)

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

- scene.h

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp  v3.1.1 (June 2014)

Public Member Functions | Public Attributes | List of all members
aiString Struct Reference

Represents an UTF-8 string, zero byte terminated. More...
Public Member Functions

aiString ()
Default constructor, the string is set to have zero length. More...

aiString (const aiString &rOther)
Copy constructor. More...

aiString (const std::string &pString)
Constructor from std::string. More...

void Append (const char *app)
Append a string to the string. More...

const char * C_Str () const
Returns a pointer to the underlying zero-terminated array of characters. More...

void Clear ()
Clear the string - reset its length to zero. More...

bool operator!= (const aiString &other) const
Inverse comparison operator. More...

aiString & operator= (const char *sz)
Assign a const char* to the string. More...

aiString & operator= (const std::string &pString)
Assign a cstd::string to the string. More...
bool operator==(const aiString &other) const
Comparison operator. More...

void Set (const std::string &pString)
Copy a std::string to the aiString. More...

void Set (const char *sz)
Copy a const char* to the aiString. More...
Public Attributes

char data [MAXLEN]
String buffer. More...

size_t length
Binary length of the string excluding the terminal 0. More...
Detailed Description

Represents an UTF-8 string, zero byte terminated.

The character set of an aiString is explicitly defined to be UTF-8. This Unicode transformation was chosen in the belief that most strings in 3d files are limited to ASCII, thus the character set needed to be strictly ASCII compatible.

Most text file loaders provide proper Unicode input file handling, special unicode characters are correctly transcoded to UTF8 and are kept throughout the libraries' import pipeline.

For most applications, it will be absolutely sufficient to interpret the aiString as ASCII data and work with it as one would work with a plain char*. Windows users in need of proper support for i.e asian characters can use the MultiByteToWideChar(), WideCharToMultiByte() WinAPI functionality to convert the UTF-8 strings to their working character set (i.e. MBCS, WideChar).

We use this representation instead of std::string to be C-compatible. The (binary) length of such a string is limited to MAXLEN characters (including the the terminating zero).
Constructor & Destructor Documentation

aiString::aiString ( ) inline

Default constructor, the string is set to have zero length.

aiString::aiString ( const aiString & rOther ) inline

Copy constructor.

aiString::aiString ( const std::string & pString ) inline explicit

Constructor from std::string.
**Member Function Documentation**

void aiString::Append ( const char * app ) inline

Append a string to the string.

const char* aiString::C_Str ( ) const inline

Returns a pointer to the underlying zero-terminated array of characters.

void aiString::Clear ( ) inline

Clear the string - reset its length to zero.

bool aiString::operator!= ( const aiString & other ) const inline

Inverse comparison operator.

aiString& aiString::operator= ( const char * sz ) inline

Assign a const char* to the string.

aiString& aiString::operator= ( const std::string & pString ) inline

Assign a cstd::string to the string.

bool aiString::operator== ( const aiString & other ) const inline

Comparison operator.

void aiString::Set ( const std::string & pString ) inline

Copy a std::string to the aiString.

void aiString::Set ( const char * sz ) inline
Copy a const char* to the `aiString`.
Member Data Documentation

char aiString::data[\texttt{MAXLEN}]

String buffer.

Size limit is \texttt{MAXLEN}

\texttt{size_t aiString::length}

Binary length of the string excluding the terminal 0.

This is NOT the logical length of strings containing UTF-8 multibyte sequences! It's the number of bytes from the beginning of the string to its end.

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

- \texttt{types.h}
aiTexel Struct Reference

Helper structure to represent a texel in a ARGB8888 format. More...
Public Member Functions

operator aiColor4D () const
Conversion to a floating-point 4d color.  More...

bool operator!= (const aiTexel &other) const
Inverse comparison operator.  More...

bool operator== (const aiTexel &other) const
Comparison operator.  More...
Public Attributes

unsigned char a

unsigned char b

unsigned char g

unsigned char r
Detailed Description

Helper structure to represent a texel in a ARGB8888 format.
Used by aiTexture.
Member Function Documentation

aiTexel::operator aiColor4D ( ) const inline

Conversion to a floating-point 4d color.

bool aiTexel::operator!= ( const aiTexel & other ) const inline

Inverse comparison operator.

bool aiTexel::operator== ( const aiTexel & other ) const inline

Comparison operator.
Member Data Documentation

unsigned char aiTexel::a
unsigned char aiTexel::b
unsigned char aiTexel::g
unsigned char aiTexel::r

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

- texture.h
aiTexture Struct Reference

Helper structure to describe an embedded texture. More...
Public Member Functions

aiTexture ()

bool CheckFormat (const char *s) const
For compressed textures (mHeight == 0): compare the format hint against a given string. More...

~aiTexture ()
Public Attributes

**char achFormatHint [4]**
A hint from the loader to make it easier for applications to determine the type of embedded compressed textures. [More...](#)

**unsigned int mHeight**
Height of the texture, in pixels. [More...](#)

**unsigned int mWidth**
Width of the texture, in pixels. [More...](#)

**aiTexel * pcData**
Data of the texture. [More...](#)
Detailed Description

Helper structure to describe an embedded texture.

Normally textures are contained in external files but some file formats embed them directly in the model file. There are two types of embedded textures:

1. Uncompressed textures. The color data is given in an uncompressed format.
2. Compressed textures stored in a file format like png or jpg. The raw file bytes are given so the application must utilize an image decoder (e.g. DevIL) to get access to the actual color data.

Embedded textures are referenced from materials using strings like "*0", "*1", etc. as the texture paths (a single asterisk character followed by the zero-based index of the texture in the aiScene::mTextures array).
Constructor & Destructor Documentation

aiTexture::aiTexture ( ) inline
aiTexture::~aiTexture ( ) inline
bool aiTexture::CheckFormat ( const char * s ) const inline

For compressed textures (mHeight == 0): compare the format hint against a given string.

Parameters

Input string. 3 characters are maximally processed. Example values: s "jpg", "png"

Returns

true if the given string matches the format hint
Member Data Documentation

char aiTexture::achFormatHint[4]

A hint from the loader to make it easier for applications to determine the type of embedded compressed textures.

If mHeight != 0 this member is undefined. Otherwise it is set set to '\0\0\0\0' if the loader has no additional information about the texture file format used OR the file extension of the format without a trailing dot. If there are multiple file extensions for a format, the shortest extension is chosen (JPEG maps to 'jpg', not to 'jpeg'). E.g. 'dds\0', 'pcx\0', 'jpg\0'. All characters are lower-case. The fourth character will always be '\0'.

unsigned int aiTexture::mHeight

Height of the texture, in pixels.

If this value is zero, pcData points to an compressed texture in any format (e.g. JPEG).

unsigned int aiTexture::mWidth

Width of the texture, in pixels.

If mHeight is zero the texture is compressed in a format like JPEG. In this case mWidth specifies the size of the memory area pcData is pointing to, in bytes.

aiTexel* aiTexture::pcData

Data of the texture.

Points to an array of mWidth * mHeight aiTexel's. The format of the texture data is always ARGB8888 to make the implementation for user of the library as easy as possible. If mHeight = 0 this is a pointer to a memory buffer of size mWidth containing the compressed texture data. Good luck, have fun!
The documentation for this struct was generated from the following file:

- texture.h
Assimp v3.1.1 (June 2014)

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Namespaces</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class List</td>
<td>Class Index</td>
<td>Class Hierarchy</td>
<td>Class Members</td>
<td></td>
</tr>
</tbody>
</table>

**aiUVTransform Struct Reference**

Defines how an UV channel is transformed. [More...](#)
Public Member Functions

aiUVTransform()
Public Attributes

float  mRotation
Rotation - in counter-clockwise direction.  More...

aiVector2D  mScaling
Scaling on the u and v axes.  More...

aiVector2D  mTranslation
Translation on the u and v axes.  More...
**Detailed Description**

Defines how an UV channel is transformed.

This is just a helper structure for the `AI_MATKEY_UVTRANSFORM` key. See its documentation for more details.

Typically you'll want to build a matrix of this information. However, we keep separate scaling/translation/rotation values to make it easier to process and optimize UV transformations internally.
Constructor & Destructor Documentation

aiUVTransform::aiUVTransform () inline
Member Data Documentation

float aiUVTransform::mRotation

Rotation - in counter-clockwise direction.

The rotation angle is specified in radians. The rotation center is 0.5f|0.5f. The default value 0.f.

aiVector2D aiUVTransform::mScaling

Scaling on the u and v axes.

The default value is (1|1).

aiVector2D aiUVTransform::mTranslation

Translation on the u and v axes.

The default value is (0|0).

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

- material.h

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
aiVector2t< TReal > Class Template Reference

Represents a two-dimensional vector. More...
Public Member Functions

aiVector2t ()

aiVector2t (TReal _x, TReal _y)

aiVector2t (TReal _xyz)

aiVector2t (const aiVector2t &o)

bool Equal (const aiVector2t &other, TReal epsilon=1e-6) const

TReal Length () const

aiVector2t & Normalize ()

template<typename TOther >

operator aiVector2t< TOther > () const

bool operator!= (const aiVector2t &other) const

const aiVector2t & operator*= (TReal f)

const aiVector2t & operator+= (const aiVector2t &o)

const aiVector2t & operator-= (const aiVector2t &o)

const aiVector2t & operator/= (TReal f)

aiVector2t & operator= (TReal f)

bool operator== (const aiVector2t &other) const
TReal operator[](unsigned int i) const

TReal & operator[](unsigned int i)

void Set(TReal pX, TReal pY)

TReal SquareLength() const

const aiVector2t SymMul(const aiVector2t &o)
Public Attributes

union {
    struct {
        TReal   x
        TReal   y
    }
    TReal   v [2]
};
Detailed Description

template<typename TReal>
class aiVector2t<TReal>

Represents a two-dimensional vector.
Constructor & Destructor Documentation

template<typename TReal>
aiVector2t< TReal >::aiVector2t ( ) inline
template<typename TReal>
aiVector2t< TReal >::aiVector2t ( TReal _x,
            TReal _y inline
            )

template<typename TReal>
aiVector2t< TReal >::aiVector2t ( TReal _xyz ) inlineexplicit
template<typename TReal>
aiVector2t< TReal >::aiVector2t ( const aiVector2t< TReal >& o ) inline
Member Function Documentation

template<typename TReal>
bool aiVector2t<TReal>::Equal ( const aiVector2t<TReal> & other,
TReal epsilon = 1e-6 const
)

template<typename TReal>
TReal aiVector2t<TReal>::Length ( ) const
template<typename TReal>
aiVector2t<TReal> & aiVector2t<TReal>::Normalize ( )
template<typename TReal>
template<typename TOther>
aiVector2t<TReal>::operator aiVector2t<TOther>( ) const

template<typename TReal>
bool aiVector2t<TReal> & aiVector2t<TReal>::operator!=( const aiVector2t<TReal> & other ) const

template<typename TReal>
const aiVector2t<TReal> & aiVector2t<TReal>::operator*= ( TReal f )
template<typename TReal>
const aiVector2t<TReal> & aiVector2t<TReal>::operator+= ( const aiVector2t<TReal> & o )
template<typename TReal>
const aiVector2t<TReal> & aiVector2t<TReal>::operator-= ( const aiVector2t<TReal> & o )
template<typename TReal>
const aiVector2t<TReal> & aiVector2t<TReal>::operator/=( TReal f )
template<typename TReal>
aiVector2t<TReal> & aiVector2t<TReal>::operator=( TReal f )

template<typename TReal>
bool aiVector2t<TReal> & aiVector2t<TReal>::operator[] ( unsigned int i ) const

template<typename TReal>
TReal & aiVector2t<TReal>::operator[](unsigned int i)

template<typename TReal>
void aiVector2t<TReal>::Set(TReal pX,
                           TReal pY)

template<typename TReal>
TReal aiVector2t<TReal>::SquareLength() const

template<typename TReal>
const aiVector2t<TReal>& aiVector2t<TReal>::SymMul(const aiVector2t<TReal>& o)
Member Data Documentation

union { ... }
template<typename TReal>
TReal aiVector2t<TReal>::v[2]
template<typename TReal>
TReal aiVector2t<TReal>::x
template<typename TReal>
TReal aiVector2t<TReal>::y

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

- matrix3x3.h
- vector2.h
- vector2.inl

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Represents a three-dimensional vector. More...
Public Member Functions

aiVector3t ()

aiVector3t (TReal _x, TReal _y, TReal _z)

aiVector3t (TReal _xyz)

aiVector3t (const aiVector3t &o)

bool Equal (const aiVector3t &other, TReal epsilon=1e-6) const

TReal Length () const
Get the length of the vector. More...

aiVector3t & Normalize ()
Normalize the vector. More...

aiVector3t & NormalizeSafe ()
Normalize the vector with extra check for zero vectors. More...

template<typename TOther >
operator aiVector3t< TOther > () const

bool operator!= (const aiVector3t &other) const

const aiVector3t & operator*= (TReal f)

aiVector3t & operator*= (const aiMatrix3x3t< TReal > &mat)
aiVector3t & operator*=(const aiMatrix4x4< TReal > &mat)

const aiVector3t & operator+= (const aiVector3t &o)

const aiVector3t & operator-= (const aiVector3t &o)

const aiVector3t & operator/= (TReal f)

bool operator< (const aiVector3t &other) const

bool operator== (const aiVector3t &other) const

TReal operator[](unsigned int i) const

TReal & operator[](unsigned int i)

void Set (TReal pX, TReal pY, TReal pZ)
Set the components of a vector. More...

TReal SquareLength () const
Get the squared length of the vector. More...

const aiVector3t SymMul (const aiVector3t &o)
Componentwise multiplication of two vectors. More...
Public Attributes

union {
  struct {
    TReal x
    TReal y
    TReal z
  }
  TReal v[3]
};
Detailed Description

template<typename TReal>
class aiVector3t<TReal>

Represents a three-dimensional vector.
Constructor & Destructor Documentation

template<typename TReal>
aiVector3t< TReal >::aiVector3t ( ) inline
template<typename TReal>
aiVector3t< TReal >::aiVector3t ( TReal _x,
    TReal _y, inline
    TReal _z
)

template<typename TReal>
aiVector3t< TReal >::aiVector3t ( TReal _xyz ) inlineexplicit
template<typename TReal>
aiVector3t< TReal >::aiVector3t ( const aiVector3t< TReal >& o ) inline
Member Function Documentation

template<
typename TReal>
    \_AI\_FORCE\_INLINE bool aiVector3t<TReal>::Equal(const aiVector3t<TReal>& other, TReal epsilon = 1e-6)

template<
typename TReal>
    \_AI\_FORCE\_INLINE TReal aiVector3t<TReal>::Length()

Get the length of the vector.

Returns
    length

template<
typename TReal>
    \_AI\_FORCE\_INLINE aiVector3t<TReal> & aiVector3t<TReal>::Normalize()

Normalize the vector.

template<
typename TReal>
    \_AI\_FORCE\_INLINE aiVector3t<TReal> & aiVector3t<TReal>::NormalizeSafe()

Normalize the vector with extra check for zero vectors.

template<
typename TReal>
template<
typename TOther>
    aiVector3t<TReal>::operator aiVector3t<TOther>( ) const

template<
typename TReal>
    \_AI\_FORCE\_INLINE bool aiVector3t<TReal>::operator!=( const aiVector3t<TReal>& other ) const

template<
typename TReal>
    \_AI\_FORCE\_INLINE const aiVector3t<TReal> &
TReal >::operator*= ( TReal f )

template<typename TReal>
AI_FORCE_INLINE aiVector3t< TReal > &
aiVector3t< TReal >::operator*= ( const aiMatrix3x3t< TReal > & mat )
template<typename TReal>
AI_FORCE_INLINE aiVector3t< TReal > &
aiVector3t< TReal >::operator*= ( const aiMatrix4x4t< TReal > & mat )
template<typename TReal>
AI_FORCE_INLINE aiVector3t< TReal > &
aiVector3t< TReal >::operator+= ( const aiVector3t< TReal > & o )
template<typename TReal>
AI_FORCE_INLINE aiVector3t< TReal > &
aiVector3t< TReal >::operator-= ( const aiVector3t< TReal > & o )
template<typename TReal>
AI_FORCE_INLINE const aiVector3t< TReal > &
aiVector3t< TReal >::operator/ ( TReal f )
template<typename TReal>
AI_FORCE_INLINE bool aiVector3t< TReal > &
aiVector3t< TReal >::operator< ( const aiVector3t< TReal > & other ) const
template<typename TReal>
AI_FORCE_INLINE bool aiVector3t< TReal > &
aiVector3t< TReal >::operator== ( const aiVector3t< TReal > & other ) const
template<typename TReal>
AI_FORCE_INLINE TReal aiVector3t< TReal >::operator[] ( unsigned int i ) const
template<typename TReal>
AI_FORCE_INLINE TReal & aiVector3t< TReal >::operator[] ( unsigned int i )
template<typename TReal>
AI_FORCE_INLINE void aiVector3t< TReal >::Set ( TReal pX, TReal pY, TReal pZ )

Set the components of a vector.
Parameters
   pX X component
   pY Y component
   pZ Z component

\[
\text{template<typename TReal >}
\]
\[
\text{AI\_FORCE\_INLINE TReal aiVector3t<TReal>::SquareLength ( ) const}
\]

Get the squared length of the vector.

Returns
   Square length

\[
\text{template<typename TReal >}
\]
\[
\text{AI\_FORCE\_INLINE const aiVector3t<TReal> aiVector3t<TReal>::SymMul ( const aiVector3t<TReal>& o )}
\]

Componentwise multiplication of two vectors.

Note that vec*vec yields the dot product.

Parameters
   o Second factor
Member Data Documentation

union {
    ...
}

template<typename TReal>
TReal aiVector3t< TReal >::v[3]

template<typename TReal>
TReal aiVector3t< TReal >::x

template<typename TReal>
TReal aiVector3t< TReal >::y

template<typename TReal>
TReal aiVector3t< TReal >::z

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

- quaternion.h
- vector3.h
- vector3.inl
A time-value pair specifying a certain 3D vector for the given time. [More...]
Public Types

typedef aiVector3D elem_type
Public Member Functions

`aiVectorKey()`
Default constructor. [More...](#)

`aiVectorKey(double time, const aiVector3D &value)`
Construction from a given time and key value. [More...](#)

bool `operator!=`(const aiVectorKey &o) const

bool `operator<`(const aiVectorKey &o) const

bool `operator==`(const aiVectorKey &o) const

bool `operator>`(const aiVectorKey &o) const
Public Attributes

double mTime
The time of this key. More...

C_STRUCT aiVector3D mValue
The value of this key. More...
Detailed Description

A time-value pair specifying a certain 3D vector for the given time.
Member Typedef Documentation

typedef aiVector3D aiVectorKey::elem_type
Constructor & Destructor Documentation

aiVectorKey::aiVectorKey ( ) inline

Default constructor.

aiVectorKey::aiVectorKey ( double time,
                        const aiVector3D & value  inline )

Construction from a given time and key value.
Member Function Documentation

bool aiVectorKey::operator!= ( const aiVectorKey & o ) const inline
bool aiVectorKey::operator< ( const aiVectorKey & o ) const inline
bool aiVectorKey::operator== ( const aiVectorKey & o ) const inline
bool aiVectorKey::operator> ( const aiVectorKey & o ) const inline
Member Data Documentation

double aiVectorKey::mTime

The time of this key.

C_STRUCT aiVector3D aiVectorKey::mValue

The value of this key.

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

- anim.h

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp v3.1.1 (June 2014)

Public Member Functions | Public Attributes | List of all members
aiVertexWeight Struct Reference

A single influence of a bone on a vertex. More...
Public Member Functions

aiVertexWeight ()
Default constructor. More...

aiVertexWeight (unsigned int pID, float pWeight)
Initialisation from a given index and vertex weight factor. More...
Public Attributes

unsigned int mVertexId
Index of the vertex which is influenced by the bone. More...

float mWeight
The strength of the influence in the range (0...1). More...
Detailed Description

A single influence of a bone on a vertex.
Constructor & Destructor Documentation

aiVertexWeight::aiVertexWeight ( ) inline

Default constructor.

aiVertexWeight::aiVertexWeight ( unsigned int  pID,
                                  float       pWeight  inline
                                   )

Initialisation from a given index and vertex weight factor.

Parameters
    pID       ID
    pWeight   Vertex weight factor
Member Data Documentation

unsigned int aiVertexWeight::mVertexId

Index of the vertex which is influenced by the bone.

float aiVertexWeight::mWeight

The strength of the influence in the range (0...1).

The influence from all bones at one vertex amounts to 1.

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

- mesh.h

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen

1.8.11
## Assimp v3.1.1 (June 2014)

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Namespace</th>
<th>Class</th>
<th>Description</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseImporter</td>
<td>(Assimp)</td>
<td></td>
<td>IOStream</td>
<td>(Assimp)</td>
<td>aiLogStream</td>
</tr>
<tr>
<td>IOSystem</td>
<td>(Assimp)</td>
<td></td>
<td>a</td>
<td></td>
<td>aiMaterial</td>
</tr>
<tr>
<td>DefaultLogger</td>
<td>(Assimp)</td>
<td></td>
<td>LogStream</td>
<td>(Assimp)</td>
<td>aiAnimation</td>
</tr>
<tr>
<td>Logger</td>
<td>(Assimp)</td>
<td></td>
<td>aiAnimMesh</td>
<td></td>
<td>aiMatrix3x3</td>
</tr>
<tr>
<td>Exporter</td>
<td>(Assimp)</td>
<td></td>
<td>NullLogger</td>
<td>(Assimp)</td>
<td>aiMatrix4x4</td>
</tr>
<tr>
<td>ExportProperties</td>
<td>(Assimp)</td>
<td></td>
<td>aiExportDataBlob</td>
<td></td>
<td>aiMemory</td>
</tr>
<tr>
<td>Exporter::ExportFormatEntry</td>
<td>(Assimp)</td>
<td></td>
<td>aiExportFormatDesc</td>
<td></td>
<td>aiMetadata</td>
</tr>
<tr>
<td>ProgressHandler</td>
<td>(Assimp)</td>
<td></td>
<td>aiFace</td>
<td></td>
<td>aiNode</td>
</tr>
<tr>
<td>Interpolator</td>
<td>(Assimp)</td>
<td></td>
<td>aiFile</td>
<td></td>
<td>aiNodeAn</td>
</tr>
</tbody>
</table>

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Class Hierarchy
This inheritance list is sorted roughly, but not completely, alphabetically:
[detail level 123]

CaiAnimation
An animation consists of keyframe data for a number of nodes

CaiAnimMesh
NOT CURRENTLY IN USE

CaiBone
A single bone of a mesh

CaiCamera
Helper structure to describe a virtual camera

CaiColor3D
Represents a color in Red-Green-Blue space

CaiColor4t<TReal>
Represents a color in Red-Green-Blue space including an alpha component

CaiExportDataBlob
Describes a blob of exported scene data

CaiExportFormatDesc
Describes an file format which Assimp can export to

CaiFace
A single face in a mesh, referring to multiple vertices

CaiFile
C-API: File callbacks

CaiFileIO
C-API: File system callbacks

CaiImporterDesc
Meta information about a particular importer

CaiLight
Helper structure to describe a light source

CaiLogStream
C-API: Represents a log stream

CaiMaterial
Data structure for a material

CaiMatrix3x3t<TReal>
Represents a row-major 3x3 matrix

CaiMatrix4x4t<TReal>
Represents a row-major 4x4 matrix, use this for homogeneous coordinates
CaiMatrix4x4t< float >
- Stores the memory requirements for different components (e.g.

CaiMemoryInfo
- A mesh represents a geometry or model with a single material

CaiMesh
- Describes vertex-based animations for a single mesh or a group of meshes

CaiMeshAnim
- Binds a mesh to a specific point in time

CaiMeshKey
- Container for holding metadata

CaiMeshAnim
- Metadata entry

CaiNode
- A node in the imported hierarchy

CaiNodeAnim
- Describes the animation of a node

CaiPlane
- Represents a plane in a three-dimensional, euclidean space

CaiPropertyStore
- C-API: Represents an opaque set of settings to be used during importing

CaiQuatnt< TReal >
- Represents a quaternion in a 4D vector

CaiQuatKey
- A time-value pair specifying a rotation for the given time

CaiRay
- Represents a ray

CaiScene
- The root structure of the imported data

CaiString
- Represents an UTF-8 string, zero byte terminated

CaiString
- Helper structure to represent a texel in a ARGB8888 format

CaiTexture
- Helper structure to describe an embedded texture

CaiUVTransform
- Defines how an UV channel is transformed

CaiVector2t< TReal >
- Represents a two-dimensional vector

CaiVector2t< float >

CaiVector3t< TReal >

CaiVector3t< float >
- Represents a three-dimensional vector
CaiVectorKey
A time-value pair specifying a certain 3D vector for the given time

CaiVertexWeight
A single influence of a bone on a vertex

CAssimp::BaseImporter
FOR IMPORTER PLUGINS ONLY: The BaseImporter defines a common interface for all importer worker classes

CAssimp::Exporter
Internal description of an Assimp export format option

CAssimp::Exporter::ExportFormatEntry

CAssimp::ExportProperties

CAssimp::Importer
CPP-API: The Importer class forms an C++ interface to the functionality of the Open Asset Import Library

CAssimp::Interpolator< T >
CPP-API: Utility class to simplify interpolations of various data types

CAssimp::ScopeGuard< T >

▼ CAllocateFromAssimpHeap

CAssimp::IOStream
CPP-API: Class to handle file I/O for C++

CAssimp::IOSystem
CPP-API: Interface to the file system

▼ CAssimp::Logger

CAssimp::DefaultLogger
CPP-API: Primary logging facility of Assimp

CAssimp::NullLogger
CPP-API: Empty logging implementation

CAssimp::LogStream
CPP-API: Abstract interface for log stream implementations

CAssimp::ProgressHandler
CPP-API: Abstract interface for custom progress report receivers
Here is a list of all class members with links to the classes they belong to:

- a -

- a : aiColor4t< TReal >, aiPlane, aiTexel
- a1 : aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >
- a2 : aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >
- a3 : aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >
- a4 : aiMatrix4x4t< TReal >
- achFormatHint : aiTexture
- AddBinaryProperty() : aiMaterial
- AddProperty() : aiMaterial
- aiAnimation() : aiAnimation
- aiAnimMesh() : aiAnimMesh
- aiBone() : aiBone
- aiCamera() : aiCamera
- aiColor3D() : aiColor3D
- aiColor4t() : aiColor4t< TReal >
- aiExportDataBlob() : aiExportDataBlob
- aiFace() : aiFace
- aiLight() : aiLight
- aiMaterial() : aiMaterial
- aiMatrix3x3t() : aiMatrix3x3t< TReal >
- aiMatrix4x4t() : aiMatrix4x4t< TReal >
- aiMemoryInfo() : aiMemoryInfo
- aiMesh() : aiMesh
- aiMeshAnim() : aiMeshAnim
- aiMeshKey(): aiMeshKey
- aiMetadata(): aiMetadata
- aiNode(): aiNode
- aiNodeAnim(): aiNodeAnim
- aiPlane(): aiPlane
- aiQuaternion(): aiQuaternion< TReal >
- aiQuatKey(): aiQuatKey
- aiRay(): aiRay
- aiScene(): aiScene
- aiString(): aiString
- aiTexture(): aiTexture
- aiUVTransform(): aiUVTransform
- aiVector2t(): aiVector2t< TReal >
- aiVector3t(): aiVector3t< TReal >
- aiVectorKey(): aiVectorKey
- aiVertexWeight(): aiVertexWeight
- ALLOW_EMPTY: Assimp::BaseImporter
- animations: aiMemoryInfo
- Append(): aiString
- ApplyCustomizedPostProcessing(): Assimp::Importer
- ApplyPostProcessing(): Assimp::Importer
- attachStream(): Assimp::DefaultLogger, Assimp::Logger, Assimp::NullLogger
Here is a list of all class members with links to the classes they belong to:

- **b** -

  - b : [aiColor3D](#), [aiColor4t< TReal >](#), [aiPlane](#), [aiTexel](#)
  - b1 : [aiMatrix3x3t< TReal >](#), [aiMatrix4x4t< TReal >](#)
  - b2 : [aiMatrix3x3t< TReal >](#), [aiMatrix4x4t< TReal >](#)
  - b3 : [aiMatrix3x3t< TReal >](#), [aiMatrix4x4t< TReal >](#)
  - b4 : [aiMatrix4x4t< TReal >](#)
  - Baseline": [Assimp::Baseimporter](#)
Here is a list of all class members with links to the classes they belong to:

- **C** -

- `c : aiColor4t< TReal >`, `aiPlane`
- `c1 : aiMatrix3x3t< TReal >`, `aiMatrix4x4t< TReal >`
- `c2 : aiMatrix3x3t< TReal >`, `aiMatrix4x4t< TReal >`
- `c3 : aiMatrix3x3t< TReal >`, `aiMatrix4x4t< TReal >`
- `c4 : aiMatrix4x4t< TReal >`
- `C_Str() : aiString`
- `callback : aiLogStream`
- `cameras : aiMemoryInfo`
- `CanRead() : Assimp::BaseImporter`
- `CheckFormat() : aiTexture`
- `CheckMagicToken() : Assimp::BaseImporter`
- `Clear() : aiMaterial`, `aiString`
- `Close() : Assimp::IOSystem`
- `CloseProc : aiFileIO`
- `ComparePaths() : Assimp::IOSystem`
- `Conjugate() : aiQuaternion< TReal >`
- `ConvertToUTF8() : Assimp::BaseImporter`
- `ConvertUTF8toISO8859_1() : Assimp::BaseImporter`
- `CopyPropertyList() : aiMaterial`
- `CopyVector() : Assimp::BaseImporter`
- `create() : Assimp::DefaultLogger`
- `createDefaultStream() : Assimp::LogStream`
- `CurrentDirectory() : Assimp::IOSystem`
Here is a list of all class members with links to the classes they belong to:

- **d** -

  - d : **aiPlane**
  - d1 : **aiMatrix4x4t< TReal >**
  - d2 : **aiMatrix4x4t< TReal >**
  - d3 : **aiMatrix4x4t< TReal >**
  - d4 : **aiMatrix4x4t< TReal >**
  - data : **aiExportDataBlob**, **aiString**
  - debug() : **Assimp::Logger**
  - Debugging : **Assimp::Logger**
  - Decompose() : **aiMatrix4x4t< TReal >**
  - DecomposeNoScaling() : **aiMatrix4x4t< TReal >**
  - description : **aiExportFormatDesc**
  - detachStream() : **Assimp::DefaultLogger**, **Assimp::Logger**, **Assimp::NullLogger**
  - Determinant() : **aiMatrix3x3t< TReal >**, **aiMatrix4x4t< TReal >**
  - dir : **aiRay**
  - dismiss() : **Assimp::ScopeGuard< T >**
Here is a list of all class members with links to the classes they belong to:

- e -

- `elem_type`: aiMeshKey, aiQuatKey, aiVectorKey
- `Equal()`: aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >, aiQuaterniont< TReal >, aiVector2t< TReal >, aiVector3t< TReal >
- `Err`: Assimp::Logger
- `error()`: Assimp::Logger
- `ErrorSeverity`: Assimp::Logger
- `Exists()`: Assimp::IOSystem
- `Export()`: Assimp::Exporter
- `Exporter()`: Assimp::Exporter
- `ExportFormatEntry()`: Assimp::Exporter::ExportFormatEntry
- `ExportProperties()`: Assimp::ExportProperties
- `ExportToBlob()`: Assimp::Exporter
Here is a list of all class members with links to the classes they belong to:

- f -

- fileExtension : Assimp::aiExportFormatDesc
- FileSize() : Assimp::IOStream
- FileSizeProc : aiFile
- FindNode() : aiNode
- FloatPropertyMap : Assimp::ExportProperties
- Flush() : Assimp::IOStream
- FlushProc : aiFile
- FORBID_EMPTY : Assimp::BaseImporter
- fpExportFunc : Assimp::Exporter
- FreeBlob() : Assimp::Exporter
- FreeScene() : Assimp::Importer
- FromEulerAnglesXYZ() : aiMatrix4x4t< TReal >
- FromToMatrix() : aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >
Here is a list of all class members with links to the classes they belong to:

- **g** -

- **g**: aiColor3D, aiColor4t< TReal >, aiTexel
- **Get()**: aiMaterial, aiMetadata
- **get()**: Assimp::DefaultLogger
- **GetBlob()**: Assimp::Exporter
- **GetCameraMatrix()**: aiCamera
- **GetErrorString()**: Assimp::Exporter, Assimp::Importer
- **GetErrorText()**: Assimp::BaseImporter
- **GetExportFormatCount()**: Assimp::Exporter
- **GetExportFormatDescription()**: Assimp::Exporter
- **GetExtension()**: Assimp::BaseImporter
- **GetExtensionList()**: Assimp::BaseImporter, Assimp::Importer
- **GetImporter()**: Assimp::Importer
- **GetImporterCount()**: Assimp::Importer
- **GetImporterIndex()**: Assimp::Importer
- **GetImporterInfo()**: Assimp::Importer
- **GetInfo()**: Assimp::BaseImporter
- **GetIOHandler()**: Assimp::Exporter, Assimp::Importer
- **getLogSeverity()**: Assimp::Logger
- **GetMatrix()**: aiQuaternint< TReal >
- **GetMemoryRequirements()**: Assimp::Importer
- **GetNumColorChannels()**: aiMesh
- **GetNumUVChannels()**: aiMesh
- **GetOrphanedBlob()**: Assimp::Exporter
- GetOrphanedScene() : Assimp::Importer
- getOsSeparator() : Assimp::IOSystem
- GetProgressHandler() : Assimp::Importer
- GetPropertyBool() : Assimp::ExportProperties, Assimp::Importer
- GetPropertyFloat() : Assimp::ExportProperties, Assimp::Importer
- GetPropertyInteger() : Assimp::ExportProperties, Assimp::Importer
- GetPropertyMatrix() : Assimp::ExportProperties, Assimp::Importer
- GetPropertyString() : Assimp::ExportProperties, Assimp::Importer
- GetScene() : Assimp::Importer
- GetTexture() : aiMaterial
- GetTextureCount() : aiMaterial
Here is a list of all class members with links to the classes they belong to:

- **h** -

  - HasAnimations() : [aiScene](#)
  - HasBones() : [aiMesh](#)
  - HasCameras() : [aiScene](#)
  - HasFaces() : [aiMesh](#)
  - HasLights() : [aiScene](#)
  - HasMaterials() : [aiScene](#)
  - HasMeshes() : [aiScene](#)
  - HasNormals() : [aiAnimMesh](#) , [aiMesh](#)
  - HasPositions() : [aiAnimMesh](#) , [aiMesh](#)
  - HasPropertyBool() : [Assimp::ExportProperties](#)
  - HasPropertyFloat() : [Assimp::ExportProperties](#)
  - HasPropertyInteger() : [Assimp::ExportProperties](#)
  - HasPropertyMatrix() : [Assimp::ExportProperties](#)
  - HasPropertyString() : [Assimp::ExportProperties](#)
  - HasTangentsAndBitangents() : [aiAnimMesh](#) , [aiMesh](#)
  - HasTextureCoords() : [aiAnimMesh](#) , [aiMesh](#)
  - HasTextures() : [aiScene](#)
  - HasVertexColors() : [aiAnimMesh](#) , [aiMesh](#)
Here is a list of all class members with links to the classes they belong to:

- i -

- id : aiExportFormatDesc
- Importer : Assimp::BaseImporter, Assimp::Importer
- info() : Assimp::Logger
- Info : Assimp::Logger
- InternReadFile() : Assimp::BaseImporter
- Interpolate() : aiQuaternion< TReal >
- IntPropertyMap : Assimp::ExportProperties
- Inverse() : aiMatrix3x3< TReal >, aiMatrix4x4< TReal >
- IOStream() : Assimp::IOStream
- IOSystem() : Assimp::IOSystem
- IsBlack() : aiColor3D, aiColor4t< TReal >
- IsDefaultIOHandler() : Assimp::Exporter, Assimp::Importer
- IsDefaultProgressHandler() : Assimp::Importer
- IsIdentity() : aiMatrix4x4< TReal >
- isNullLogger() : Assimp::DefaultLogger
Here is a list of all class members with links to the classes they belong to:

- **k** -
  
  - **KeyType** : Assimp::ExportProperties
  - **kill()** : Assimp::DefaultLogger
Here is a list of all class members with links to the classes they belong to:

- l -

- length : aiString
- Length() : aiVector2t<TReal>, aiVector3t<TReal>
- lights : aiMemoryInfo
- Logger() : Assimp::Logger
- LogSeverity : Assimp::Logger
- LogStream() : Assimp::LogStream
Here is a list of all class members with links to the classes they belong to:

- **m** -
  - m : [aiMatrix3x3t](#) < TReal > , [aiMatrix4x4t](#) < TReal >
  - m_ErrorText : [Assimp::BaseImporter](#)
  - m_progress : [Assimp::BaseImporter](#)
  - m_Severity : [Assimp::Logger](#)
  - mAngleInnerCone : [aiLight](#)
  - mAngleOuterCone : [aiLight](#)
  - mAnimations : [aiScene](#)
  - mAnimMeshes : [aiMesh](#)
  - mAspect : [aiCamera](#)
  - materials : [aiMemoryInfo](#)
  - MatrixPropertyMap : [Assimp::ExportProperties](#)
  - mAttenuationConstant : [aiLight](#)
  - mAttenuationLinear : [aiLight](#)
  - mAttenuationQuadratic : [aiLight](#)
  - mAuthor : [aiImporterDesc](#)
  - MaxLenHint : [Assimp::Desc](#)
  - mBitangents : [aiAnimMesh](#) , [aiMesh](#)
  - mBones : [aiMesh](#)
  - mCameras : [aiScene](#)
  - mChannels : [aiAnimation](#)
  - mChildren : [aiNode](#)
  - mClipPlaneFar : [aiCamera](#)
  - mClipPlaneNear : [aiCamera](#)
- mColorAmbient: aiLight
- mColorDiffuse: aiLight
- mColors: aiAnimMesh, aiMesh
- mColorSpecular: aiLight
- mComments: aiImporterDesc
- mData: aiMatrix3x3t<TReal>, aiMatrix4x4t<TReal>, aiMetadataEntry
- mDescription: Assimp::Exporter::ExportFormatEntry
- mDirection: aiLight
- mDuration: aiAnimation
- mEnforcePP: Assimp::Exporter::ExportFormatEntry
- meshes: aiMemoryInfo
- mExportFunction: Assimp::Exporter::ExportFormatEntry
- mFaces: aiMesh
- mFileExtensions: aiImporterDesc
- mFlags: aiImporterDesc, aiScene
- mFloatProperties: Assimp::ExportProperties
- mHeight: aiTexture
- mHorizontalFOV: aiCamera
- mIndices: aiFace
- mIntProperties: Assimp::ExportProperties
- mKeys: aiMeshAnim, aiMetadata
- mLights: aiScene
- mLookAt: aiCamera
- mMaintainer: aiImporterDesc
- mMaterialIndex: aiMesh
- mMaterials: aiScene
- mMatrixProperties: Assimp::ExportProperties
- mMaxMajor: aiImporterDesc
- mMaxMinor: aiImporterDesc
- mMeshChannels: aiAnimation
- mMeshes: aiNode, aiScene
- mMetaData: aiNode
- mMinMajor: aiImporterDesc
- mMinMinor: aiImporterDesc
- mName: aiAnimation, aiBone, aiCamera, aiImporterDesc, aiLight, aiMesh, aiMeshAnim, aiNode
- mNodeName: aiNodeAnim
- mNormals: aiAnimMesh, aiMesh
- mNumAllocated: aiMaterial
- mNumAnimations: aiScene
- mNumAnimMeshes: aiMesh
- mNumBones: aiMesh
- mNumCameras: aiScene
- mNumChannels: aiAnimation
- mNumChildren: aiNode
- mNumFaces: aiMesh
- mNumIndices: aiFace
- mNumKeys: aiMeshAnim
- mNumLights: aiScene
- mNumMaterials: aiScene
- mNumMeshChannels: aiAnimation
- mNumMeshes: aiNode, aiScene
- mNumPositionKeys: aiNodeAnim
- mNumProperties: aiMaterial, aiMetadata
- mNumRotationKeys: aiNodeAnim
- mNumScalingKeys: aiNodeAnim
- mNumTextures: aiScene
- mNumUVComponents: aiMesh
- mNumVertices: aiAnimMesh, aiMesh
- mNumWeights: aiBone
- mOffsetMatrix: aiBone
- mParent: aiNode
- mPosition: aiCamera, aiLight
- mPositionKeys: aiNodeAnim
- mPostState: aiNodeAnim
- mPreState: aiNodeAnim
- mPrimitiveTypes: aiMesh
- mPrivate: aiScene
- mProperties: aiMaterial
- mRootNode: aiScene
- mRotation: aiUVTransform
- mRotationKeys: aiNodeAnim
- mScaling: aiUVTransform
- mScalingKeys: aiNodeAnim
- mStringProperties: Assimp::ExportProperties
- mTangents: aiAnimMesh, aiMesh
- mTextureCoords: aiAnimMesh, aiMesh
- mTextures: aiScene
- `mTicksPerSecond` : `aiAnimation`
- `mTime` : `aiMeshKey`, `aiQuatKey`, `aiVectorKey`
- `mTransformation` : `aiNode`
- `mTranslation` : `aiUVTransform`
- `mType` : `aiLight`, `aiMetadataEntry`
- `mUp` : `aiCamera`
- `mValue` : `aiMeshKey`, `aiQuatKey`, `aiVectorKey`
- `mValues` : `aiMetadata`
- `mVertexId` : `aiVertexWeight`
- `mVertices` : `aiAnimMesh`, `aiMesh`
- `mWeight` : `aiVertexWeight`
- `mWeights` : `aiBone`
- `mWidth` : `aiTexture`
Here is a list of all class members with links to the classes they belong to:

- **n** -

  - name: [aiExportDataBlob](#)
  - next: [aiExportDataBlob](#)
  - nodes: [aiMemoryInfo](#)
  - NORMAL: [Assimp::Logger](#)
  - Normalize(): [aiQuaternionTReal](#), [aiVector2TReal](#), [aiVector3TReal](#)
  - NormalizeSafe(): [aiVector3TReal](#)
Here is a list of all class members with links to the classes they belong to:

- O -

- OnDebug() : Assimp::Logger, Assimp::NullLogger
  - OnError() : Assimp::Logger, Assimp::NullLogger
  - OnInfo() : Assimp::Logger, Assimp::NullLogger
  - OnWarn() : Assimp::Logger, Assimp::NullLogger
  - Open() : Assimp::IOSystem
  - OpenProc : aiFileIO
  - operator aiColor4D() : aiTexel
  - operator aiMatrix3x3t< TOther >( ) : aiMatrix3x3t< TReal >
  - operator aiMatrix4x4t< TOther >( ) : aiMatrix4x4t< TReal >
  - operator aiVector2t< TOther >( ) : aiVector2t< TReal >
  - operator aiVector3t< TOther >( ) : aiVector3t< TReal >
  - operator T *( ) : Assimp::ScopeGuard< T >
  - operator!() : aiColor3D, aiColor4t< TReal >, aiFace, aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >, aiMeshKey, aiQuatKey, aiQuatKey, aiString, aiTexel, aiVector2t< TReal >, aiVector3t< TReal >, aiVectorKey
  - operator()() : Assimp::Interpolator< T >
  - operator*() : aiColor3D, aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >, aiQuatKey, aiQuatKey
  - operator*=() : aiColor4t< TReal >, aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >, aiVector2t< TReal >, aiVector3t< TReal >
TReal >
- operator-(): aiColor3D
- operator-(): aiColor4t< TReal >, aiVector2t< TReal >, aiVector3t< TReal >
- operator-(): Assimp::ScopeGuard< T >
- operator-(): aiColor4t< TReal >, aiVector2t< TReal >, aiVector3t< TReal >
- operator<(): aiColor3D, aiColor4t< TReal >, aiMeshKey, aiQuatKey, aiVector3t< TReal >, aiVectorKey
- operator-(): aiFace, aiString, aiVector2t< TReal >
- operator-(): aiColor3D, aiColor4t< TReal >, aiFace, aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >, aiMeshKey, aiQuaternion< TReal >, aiQuatKey, aiString, aiTexel, aiVector2t< TReal >, aiVector3t< TReal >, aiVectorKey
- operator-(): aiMeshKey, aiQuatKey, aiVectorKey
- operator-(): aiColor3D, aiColor4t< TReal >, aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >, aiVector2t< TReal >, aiVector3t< TReal >
Here is a list of all class members with links to the classes they belong to:

- **p** -
  
  - pcData : [aiTexture](#)
  - pimpl : [Assimp::Exporter](#)
  - Pimpl() : [Assimp::Importer](#)
  - pimpl : [Assimp::Importer](#)
  - PopDirectory() : [Assimp::IOSystem](#)
  - pos : [aiRay](#)
  - ProgressHandler() : [Assimp::ProgressHandler](#)
  - PushDirectory() : [Assimp::IOSystem](#)
Here is a list of all class members with links to the classes they belong to:

- r -

- r : aiColor3D, aiColor4t<TReal>, aiTexel
- Read() : Assimp::IOStream
- ReadFile() : Assimp::BaseImporter, Assimp::Importer
- ReadFileFromMemory() : Assimp::Importer
- ReadProc : aiFile
- RegisterExporter() : Assimp::Exporter
- RegisterLoader() : Assimp::Importer
- RegisterPPStep() : Assimp::Importer
- RemoveProperty() : aiMaterial
- Rotate() : aiQuaternions<TReal>
- Rotation() : aiMatrix3x3t<TReal>, aiMatrix4x4t<TReal>
- RotationX() : aiMatrix4x4t<TReal>
- RotationY() : aiMatrix4x4t<TReal>
- RotationZ() : aiMatrix3x3t<TReal>, aiMatrix4x4t<TReal>
Here is a list of all class members with links to the classes they belong to:

- **S** -

- Scaling() : [aiMatrix4x4t](../Classes/aiMatrix4x4t)\(<\mathrm{T\text{Real}}>\>
- ScopeGuard() : [Assimp::ScopeGuard](../Namespaces/Assimp)\(<\mathrm{T}>\>
- SearchFileHeaderForToken() : [Assimp::BaseImporter](../Classes/Assimp::BaseImporter)
- Seek() : [Assimp::IOStream](../Classes/Assimp::IOStream)
- SeekProc : [aiFile](../Classes/aiFile)
- sentinel : [aiPropertyStore](../Classes/aiPropertyStore)
- Set() : [aiMetadata](../Classes/aiMetadata) , [aiString](../Classes/aiString) , [aiVector2t](../Classes/aiVector2t)\(<\mathrm{T\text{Real}}>\), [aiVector3t](../Classes/aiVector3t)\(<\mathrm{T\text{Real}}>\>
- set() : [Assimp::DefaultLogger](../Namespaces/Assimp::DefaultLogger)
- SetExtraVerbose() : [Assimp::Importer](../Classes/Assimp::Importer)
- SetIOHandler() : [Assimp::Exporter](../Classes/Assimp::Exporter) , [Assimp::Importer](../Classes/Assimp::Importer)
- setLogSeverity() : [Assimp::Logger](../Namespaces/Assimp::Logger)
- SetProgressHandler() : [Assimp::Importer](../Classes/Assimp::Importer)
- SetPropertyBool() : [Assimp::ExportProperties](../Classes/Assimp::ExportProperties) , [Assimp::Importer](../Classes/Assimp::Importer)
- SetPropertyFloat() : [Assimp::ExportProperties](../Classes/Assimp::ExportProperties) , [Assimp::Importer](../Classes/Assimp::Importer)
- SetPropertyInteger() : [Assimp::ExportProperties](../Classes/Assimp::ExportProperties) , [Assimp::Importer](../Classes/Assimp::Importer)
- SetPropertyMatrix() : [Assimp::ExportProperties](../Classes/Assimp::ExportProperties) , [Assimp::Importer](../Classes/Assimp::Importer)
- SetPropertyString() : [Assimp::ExportProperties](../Classes/Assimp::ExportProperties) , [Assimp::Importer](../Classes/Assimp::Importer)
- SetupProperties() : [Assimp::BaseImporter](../Classes/Assimp::BaseImporter)
- SimpleExtensionCheck() : [Assimp::BaseImporter](../Classes/Assimp::BaseImporter)
- size : [aiExportDataBlob](../Classes/aiExportDataBlob)
- SquareLength() : [aiVector2t](../Classes/aiVector2t)\(<\mathrm{T\text{Real}}>\) , [aiVector3t](../Classes/aiVector3t)\(<\mathrm{T\text{Real}}>\>
- StackSize() : [Assimp::IOSystem](../Classes/Assimp::IOSystem)
- StringPropertyMap : [Assimp::ExportProperties](../Classes/Assimp::ExportProperties)
SymMul() : aiVector2t< TReal >, aiVector3t< TReal >
Here is a list of all class members with links to the classes they belong to:

- **t** -

- Tell() : [Assimp::IOStream](#)
- TellProc : [aiFile](#)
- TextFileMode : [Assimp::BaseImporter](#)
- TextFileToBuffer() : [Assimp::BaseImporter](#)
- textures : [aiMemoryInfo](#)
- total : [aiMemoryInfo](#)
- Translation() : [aiMatrix3x3t< TReal >](#), [aiMatrix4x4t< TReal >](#)
- Transpose() : [aiMatrix3x3t< TReal >](#), [aiMatrix4x4t< TReal >](#)

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by [doxygen](#) 1.8.11
Here is a list of all class members with links to the classes they belong to:

- u -

- UnregisterExporter() : Assimp::Exporter
- UnregisterLoader() : Assimp::Importer
- UnregisterPPStep() : Assimp::Importer
- Update() : Assimp::ProgressHandler
- UpdateFileRead() : Assimp::ProgressHandler
- UpdatePostProcess() : Assimp::ProgressHandler
- user : aiLogStream
- UserData : aiFile, aiFileIO
Here is a list of all class members with links to the classes they belong to:

- **v** -
  - **v**: aiVector2t\< TReal \>, aiVector3t\< TReal \>
  - ValidateFlags(): Assimp::Importer
  - VERBOSE: Assimp::Logger
Here is a list of all class members with links to the classes they belong to:

- **W** -

- **w**: `aiQuaternion< TReal >`
- **Warn**: `Assimp::Logger`
- **warn()**: `Assimp::Logger`
- **Write()**: `Assimp::IOStream`
- **write()**: `Assimp::LogStream`
- **WriteProc**: `aiFile`
Here is a list of all class members with links to the classes they belong to:

- **X** -

  - x : `aiQuaternion< TReal >`, `aiVector2t< TReal >`, `aiVector3t< TReal >`
Here is a list of all class members with links to the classes they belong to:

- **y** -

  - y: `aiQuaternion< TReal >`, `aiVector2t< TReal >`, `aiVector3t< TReal >`
Here is a list of all class members with links to the classes they belong to:

- **Z** -

  - z: `aiQuaternion< TReal >`, `aiVector3< TReal >`
Here is a list of all class members with links to the classes they belong to:

- ~
  - ~aiAnimation() : aiAnimation
  - ~aiAnimMesh() : aiAnimMesh
  - ~aiBone() : aiBone
  - ~aiExportDataBlob() : aiExportDataBlob
  - ~aiFace() : aiFace
  - ~aiMaterial() : aiMaterial
  - ~aiMesh() : aiMesh
  - ~aiMeshAnim() : aiMeshAnim
  - ~aiMetadata() : aiMetadata
  - ~aiNode() : aiNode
  - ~aiNodeAnim() : aiNodeAnim
  - ~aiScene() : aiScene
  - ~aiTexture() : aiTexture
  - ~BaseImporter() : Assimp::BaseImporter
  - ~Exporter() : Assimp::Exporter
  - ~Importer() : Assimp::Importer
  - ~IOStream() : Assimp::IOStream
  - ~IOSystem() : Assimp::IOSystem
  - ~Logger() : Assimp::Logger
  - ~LogStream() : Assimp::LogStream
  - ~ProgressHandler() : Assimp::ProgressHandler
  - ~ScopeGuard() : Assimp::ScopeGuard< T >
Assimp  v3.1.1 (June 2014)

- a -

- AddBinaryProperty() : aiMaterial
- AddProperty() : aiMaterial
- aiAnimation() : aiAnimation
- aiAnimMesh() : aiAnimMesh
- aiBone() : aiBone
- aiCamera() : aiCamera
- aiColor3D() : aiColor3D
- aiColor4t() : aiColor4t<TReal>
- aiExportDataBlob() : aiExportDataBlob
- aiFace() : aiFace
- aiLight() : aiLight
- aiMaterial() : aiMaterial
- aiMatrix3x3t() : aiMatrix3x3t<TReal>
- aiMatrix4x4t() : aiMatrix4x4t<TReal>
- aiMemoryInfo() : aiMemoryInfo
- aiMesh() : aiMesh
- aiMeshAnim() : aiMeshAnim
- aiMeshKey() : aiMeshKey
- aiMetadata() : aiMetadata
- aiNode() : aiNode
- aiNodeAnim() : aiNodeAnim
- aiPlane() : aiPlane
- aiQuatKey() : aiQuatKey
- aiRay()
- aiScene()
- aiString()
- aiTexture()
- aiUVTransform()
- aiVector2t()
- aiVector3t()
- aiVectorKey()
- aiVertexWeight()
- Append()
- ApplyCustomizedPostProcessing() : Assimp::Importer
- ApplyPostProcessing() : Assimp::Importer
- attachStream() : Assimp::DefaultLogger, Assimp::Logger, Assimp::NullLogger
- b -

- BaseImporter() : Assimp::BaseImporter
Assimp v3.1.1 (June 2014)

- C -

- C_Str() : aiString
- CanRead() : Assimp::BaseImporter
- CheckFormat() : aiTexture
- CheckMagicToken() : Assimp::BaseImporter
- Clear() : aiMaterial, aiString
- Close() : Assimp::IOSystem
- ComparePaths() : Assimp::IOSystem
- Conjugate() : aiQuaternion<TReal>
- ConvertToUTF8() : Assimp::BaseImporter
- ConvertUTF8toISO8859_1() : Assimp::BaseImporter
- CopyPropertyList() : aiMaterial
- CopyVector() : Assimp::BaseImporter
- create() : Assimp::DefaultLogger
- createDefaultStream() : Assimp::LogStream
- CurrentDirectory() : Assimp::IOSystem

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
- d -

- debug() : `Assimp::Logger`
- Decompose() : `aiMatrix4x4t< TReal >`
- DecomposeNoScaling() : `aiMatrix4x4t< TReal >`
- detachStream() : `Assimp::DefaultLogger`, `Assimp::Logger`, `Assimp::NullLogger`
- Determinant() : `aiMatrix3x3t< TReal >`, `aiMatrix4x4t< TReal >`
- dismiss() : `Assimp::ScopeGuard< T >`
- e -

- **Equal()**: `aiMatrix3x3t<TReal>`, `aiMatrix4x4t<TReal>`, `aiQuaterniont<TReal>`, `aiVector2t<TReal>`, `aiVector3t<TReal>`
- **error()**: `Assimp::Logger`
- **Exists()**: `Assimp::IOSystem`
- **Export()**: `Assimp::Exporter`
- **Exporter()**: `Assimp::Exporter`
- **ExportFormatEntry()**: `Assimp::Exporter::ExportFormatEntry`
- **ExportProperties()**: `Assimp::ExportProperties`
- **ExportToBlob()**: `Assimp::Exporter`
- f -

- FileSize() : Assimp::IOStream
- FindNode() : aiNode
- Flush() : Assimp::IOStream
- FreeBlob() : Assimp::Exporter
- FreeScene() : Assimp::Importer
- FromEulerAnglesXYZ() : aiMatrix4x4t< TReal >
- FromToMatrix() : aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >
- g -

- Get() : aiMaterial , aiMetadata
- get() : Assimp::DefaultLogger
- GetBlob() : Assimp::Exporter
- GetCameraMatrix() : aiCamera
- GetErrorString() : Assimp::Exporter , Assimp::Importer
- GetErrorText() : Assimp::BaseImporter
- GetExportFormatCount() : Assimp::Exporter
- GetExportFormatDescription() : Assimp::Exporter
- GetExtension() : Assimp::BaseImporter
- GetExtensionList() : Assimp::BaseImporter , Assimp::Importer
- GetImporter() : Assimp::Importer
- GetImporterCount() : Assimp::Importer
- GetImporterIndex() : Assimp::Importer
- GetImporterInfo() : Assimp::Importer
- GetInfo() : Assimp::BaseImporter
- GetIOHandler() : Assimp::Exporter , Assimp::Importer
- getLogSeverity() : Assimp::Logger
- GetMatrix() : aiQuaternions< TReal >
- GetMemoryRequirements() : Assimp::Importer
- GetNumColorChannels() : aiMesh
- GetNumUVChannels() : aiMesh
- GetOrphanedBlob() : Assimp::Exporter
- GetOrphanedScene() : Assimp::Importer
- getOsSeparator() : Assimp::IOSystem
- GetProgressHandler(): Assimp::Importer
- GetPropertyBool(): Assimp::ExportProperties, Assimp::Importer
- GetPropertyFloat(): Assimp::ExportProperties, Assimp::Importer
- GetPropertyInteger(): Assimp::ExportProperties, Assimp::Importer
- GetPropertyMatrix(): Assimp::ExportProperties, Assimp::Importer
- GetPropertyString(): Assimp::ExportProperties, Assimp::Importer
- GetScene(): Assimp::Importer
- GetTexture(): aiMaterial
- GetTextureCount(): aiMaterial
- h -

- HasAnimations() : aiScene
- HasBones() : aiMesh
- HasCameras() : aiScene
- HasFaces() : aiMesh
- HasLights() : aiScene
- HasMaterials() : aiScene
- HasMeshes() : aiScene
- HasNormals() : aiAnimMesh, aiMesh
- HasPositions() : aiAnimMesh, aiMesh
- HasPropertyBool() : Assimp::ExportProperties
- HasPropertyFloat() : Assimp::ExportProperties
- HasPropertyInteger() : Assimp::ExportProperties
- HasPropertyMatrix() : Assimp::ExportProperties
- HasPropertyString() : Assimp::ExportProperties
- HasTangentsAndBitangents() : aiAnimMesh, aiMesh
- HasTextureCoords() : aiAnimMesh, aiMesh
- HasTextures() : aiScene
- HasVertexColors() : aiAnimMesh, aiMesh
- i -

- Importer() : Assimp::Importer
- info() : Assimp::Logger
- InternReadFile() : Assimp::BaseImporter
- Interpolate() : aiQuaterniont<TReal>
- Inverse() : aiMatrix3x3t<TReal>, aiMatrix4x4t<TReal>
- IOStream() : Assimp::IOStream
- IOSystem() : Assimp::IOSystem
- IsBlack() : aiColor3D, aiColor4t<TReal>
- IsDefaultIOHandler() : Assimp::Exporter, Assimp::Importer
- IsDefaultProgressHandler() : Assimp::Importer
- IsIdentity() : aiMatrix4x4t<TReal>
- isNullLogger() : Assimp::DefaultLogger
- k -

- kill() : [Assimp::DefaultLogger](#)

---

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by [doxygen](#) 1.8.11*
- I -

- Length() : aiVector2t< TReal >, aiVector3t< TReal >
- Logger() : Assimp::Logger
- LogStream() : Assimp::LogStream
- n -

- Normalize() : `aiQuaternion< TReal >`, `aiVector2< TReal >`, `aiVector3< TReal >`
- NormalizeSafe() : `aiVector3< TReal >`

---

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11*
- O -

- OnDebug() : **Assimp::Logger**, **Assimp::NullLogger**
- OnError() : **Assimp::Logger**, **Assimp::NullLogger**
- OnInfo() : **Assimp::Logger**, **Assimp::NullLogger**
- OnWarn() : **Assimp::Logger**, **Assimp::NullLogger**
- Open() : **Assimp::IOSystem**
- operator aiColor4D() : **aiTexel**
- operator aiMatrix3x3t<TOther>() : **aiMatrix3x3t<TReal>**
- operator aiMatrix4x4t<TOther>() : **aiMatrix4x4t<TReal>**
- operator aiVector2t<TOther>() : **aiVector2t<TReal>**
- operator aiVector3t<TOther>() : **aiVector3t<TReal>**
- operator T *() : **Assimp::ScopeGuard<T>**
- operator!() : **aiColor3D**, **aiColor4t<TReal>**, **aiFace**, **aiMatrix3x3t<TReal>**, **aiMatrix4x4t<TReal>**, **aiMeshKey**, **aiQuatKey**, **aiString**, **aiTexel**, **aiVector2t<TReal>**, **aiVector3t<TReal>**, **aiVectorKey**
- operator()() : **Assimp::Interpolator<T>**
- operator*() : **aiColor3D**, **aiMatrix3x3t<TReal>**, **aiMatrix4x4t<TReal>**, **aiQuatKey**
- operator*() : **aiColor4t<TReal>**, **aiMatrix3x3t<TReal>**, **aiMatrix4x4t<TReal>**, **aiVector2t<TReal>**, **aiVector3t<TReal>**
- operator+() : **aiColor3D**
- operator+() : **aiColor4t<TReal>**, **aiVector2t<TReal>**, **aiVector3t<TReal>**
- operator-() : **aiColor3D**
- operator-=(aiColor4t<TReal>, aiVector2t<TReal>, aiVector3t<TReal>)
- operator->(Assimp::ScopeGuard<T>)
- operator/=(aiColor4t<TReal>, aiVector2t<TReal>, aiVector3t<TReal>)
- operator<(aiColor3D, aiColor4t<TReal>, aiMeshKey, aiQuatKey, aiVector3t<TReal>, aiVectorKey)
- operator==(aiFace, aiString, aiVector2t<TReal>)
- operator==(aiColor3D, aiColor4t<TReal>, aiFace, aiMatrix3x3t<TReal>, aiMatrix4x4t<TReal>, aiMeshKey, aiQuatKey, aiQuatKey, aiString, aiTexel, aiVector2t<TReal>, aiVector3t<TReal>, aiVectorKey)
- operator[](aiMeshKey, aiQuatKey, aiVectorKey)
- operator[](aiColor3D, aiColor4t<TReal>, aiMatrix3x3t<TReal>, aiMatrix4x4t<TReal>, aiVector2t<TReal>, aiVector3t<TReal>)

---

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11*
- p -

- Pimpl() : Assimp::Importer
- PopDirectory() : Assimp::IOSystem
- ProgressHandler() : Assimp::ProgressHandler
- PushDirectory() : Assimp::IOSystem
- r -

- Read() : Assimp::IOStream
- ReadFile() : Assimp::BaseImporter, Assimp::Importer
- ReadFileFromMemory() : Assimp::Importer
- RegisterExporter() : Assimp::Exporter
- RegisterLoader() : Assimp::Importer
- RegisterPPStep() : Assimp::Importer
- RemoveProperty() : aiMaterial
- Rotate() : aiQuaternion< TReal >
- Rotation() : aiMatrix3x3< TReal >, aiMatrix4x4< TReal >
- RotationX() : aiMatrix4x4< TReal >
- RotationY() : aiMatrix4x4< TReal >
- RotationZ() : aiMatrix3x3< TReal >, aiMatrix4x4< TReal >
- s -

- Scaling() : `aiMatrix4x4f< TReal >`
- ScopeGuard() : `Assimp::ScopeGuard< T >`
- SearchFileHeaderForToken() : `Assimp::BaseImporter`
- Seek() : `Assimp::IOStream`
- Set() : `aiMetadata`, `aiString`, `aiVector2f< TReal >`, `aiVector3f< TReal >`
- set() : `Assimp::DefaultLogger`
- SetExtraVerbose() : `Assimp::Importer`
- SetIOHandler() : `Assimp::Exporter`, `Assimp::Importer`
- setLogSeverity() : `Assimp::Logger`
- SetProgressHandler() : `Assimp::Importer`
- SetPropertyBool() : `Assimp::ExportProperties`, `Assimp::Importer`
- SetPropertyFloat() : `Assimp::ExportProperties`, `Assimp::Importer`
- SetPropertyInteger() : `Assimp::ExportProperties`, `Assimp::Importer`
- SetPropertyMatrix() : `Assimp::ExportProperties`, `Assimp::Importer`
- SetPropertyString() : `Assimp::ExportProperties`, `Assimp::Importer`
- SetupProperties() : `Assimp::BaseImporter`
- SimpleExtensionCheck() : `Assimp::BaseImporter`
- SquareLength() : `aiVector2f< TReal >`, `aiVector3f< TReal >`
- StackSize() : `Assimp::IOStream`
- SymMul() : `aiVector2f< TReal >`, `aiVector3f< TReal >`
Assimp v3.1.1 (June 2014)

- t -

- Tell() : Assimp::IOStream
- TextFileToBuffer() : Assimp::BaseImporter
- Translation() : aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >
- Transpose() : aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
- u -

- UnregisterExporter() : Assimp::Exporter
- UnregisterLoader() : Assimp::Importer
- UnregisterPPStep() : Assimp::Importer
- Update() : Assimp::ProgressHandler
- UpdateFileRead() : Assimp::ProgressHandler
- UpdatePostProcess() : Assimp::ProgressHandler
- V -

- ValidateFlags() : Assimp::Importer
- w -

- `warn()` : Assimp::Logger
- `Write()` : Assimp::IOStream
- `write()` : Assimp::LogStream
- ~ -

- ~aiAnimation() : aiAnimation
- ~aiAnimMesh() : aiAnimMesh
- ~aiBone() : aiBone
- ~aiExportDataBlob() : aiExportDataBlob
- ~aiFace() : aiFace
- ~aiMaterial() : aiMaterial
- ~aiMesh() : aiMesh
- ~aiMeshAnim() : aiMeshAnim
- ~aiMetadata() : aiMetadata
- ~aiNode() : aiNode
- ~aiNodeAnim() : aiNodeAnim
- ~aiScene() : aiScene
- ~aiTexture() : aiTexture
- ~BaseImporter() : Assimp::BaseImporter
- ~Exporter() : Assimp::Exporter
- ~Importer() : Assimp::Importer
- ~IOStream() : Assimp::IOStream
- ~IOSystem() : Assimp::IOSystem
- ~Logger() : Assimp::Logger
- ~LogStream() : Assimp::LogStream
- ~ProgressHandler() : Assimp::ProgressHandler
- ~ScopeGuard() : Assimp::ScopeGuard<T>
- a -

- a : aiColor4t< TReal >, aiPlane, aiTexel
- a1 : aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >
- a2 : aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >
- a3 : aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >
- a4 : aiMatrix4x4t< TReal >
- achFormatHint : aiTexture
- animations : aiMemoryInfo
- b -

- b : `aiColor3D`, `aiColor4t<TReal>`, `aiPlane`, `aiTexel`
- b1 : `aiMatrix3x3t<TReal>`, `aiMatrix4x4t<TReal>`
- b2 : `aiMatrix3x3t<TReal>`, `aiMatrix4x4t<TReal>`
- b3 : `aiMatrix3x3t<TReal>`, `aiMatrix4x4t<TReal>`
- b4 : `aiMatrix4x4t<TReal>"
- C -

- c : aiColor4t< TReal > , aiPlane
- c1 : aiMatrix3x3t< TReal > , aiMatrix4x4t< TReal >
- c2 : aiMatrix3x3t< TReal > , aiMatrix4x4t< TReal >
- c3 : aiMatrix3x3t< TReal > , aiMatrix4x4t< TReal >
- c4 : aiMatrix4x4t< TReal >
- callback : aiLogStream
- cameras : aiMemoryInfo
- CloseProc : aiFileIO
- d -

- d: aiPlane
- d1: aiMatrix4x4t<TReal>
- d2: aiMatrix4x4t<TReal>
- d3: aiMatrix4x4t<TReal>
- d4: aiMatrix4x4t<TReal>
- data: aiExportDataBlob, aiString
- description: aiExportFormatDesc
- dir: aiRay
- f -

- fileExtension : aiExportFormatDesc
- FileSizeProc : aiFile
- FlushProc : aiFile
- g -

- **g**: `aiColor3D`, `aiColor4t<TReal>`, `aiTexel`

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by `doxygen 1.8.11`
- i -

- id : aiExportFormatDesc
- l -

- length : [aiString](#)
- lights : [aiMemoryInfo](#)
- m -

- m : aiMatrix3x3t< TReal >, aiMatrix4x4t< TReal >
- m_ErrorText : Assimp::BaseImporter
- m_progress : Assimp::BaseImporter
- m_Severity : Assimp::Logger
- mAngleInnerCone : aiLight
- mAngleOuterCone : aiLight
- mAnimations : aiScene
- mAnimMeshes : aiMesh
- mAspect : aiCamera
- materials : aiMemoryInfo
- mAttenuationConstant : aiLight
- mAttenuationLinear : aiLight
- mAttenuationQuadratic : aiLight
- mAuthor : aiImporterDesc
- MaxLenHint : Assimp::Importer
- mBitangents : aiAnimMesh, aiMesh
- mBones : aiMesh
- mCameras : aiScene
- mChannels : aiAnimation
- mChildren : aiNode
- mClipPlaneFar : aiCamera
- mClipPlaneNear : aiCamera
- mColorAmbient : aiLight
- mColorDiffuse : aiLight
• mColors : aiAnimMesh, aiMesh
• mColorSpecular : aiLight
• mComments : aiImporterDesc
• mData : aiMatrix3x3t<TReal>, aiMatrix4x4t<TReal>, aiMetadataEntry
• mDescription : Assimp::Exporter::ExportFormatEntry
• mDirection : aiLight
• mDuration : aiAnimation
• mEnforcePP : Assimp::Exporter::ExportFormatEntry
• meshes : aiMemoryInfo
• mExportFunction : Assimp::Exporter::ExportFormatEntry
• mFaces : aiMesh
• mFileExtensions : aiImporterDesc
• mFlags : aiImporterDesc, aiScene
• mFloatProperties : Assimp::ExportProperties
• mHeight : aiTexture
• mHorizontalFOV : aiCamera
• mIndices : aiFace
• mIntProperties : Assimp::ExportProperties
• mKeys : aiMeshAnim, aiMetadata
• mLights : aiScene
• mLookAt : aiCamera
• mMaintainer : aiImporterDesc
• mMaterialIndex : aiMesh
• mMaterials : aiScene
• mMatrixProperties : Assimp::ExportProperties
• mMaxMajor : aiImporterDesc
• mMaxMinor : aiImporterDesc
• mMeshChannels : aiAnimation
• mMeshes : aiNode, aiScene
• mMetaData : aiNode
• mMinMajor : aiImporterDesc
• mMinMinor : aiImporterDesc
• mName : aiAnimation, aiBone, aiCamera, aiImporterDesc, aiLight, aiMesh, aiMeshAnim, aiNode
• mNodeName : aiNodeAnim
• mNormals : aiAnimMesh, aiMesh
• mNumAllocated : aiMaterial
• mNumAnimations : aiScene
• mNumAnimMeshes : aiMesh
- mNumBones: aiMesh
- mNumCameras: aiScene
- mNumChannels: aiAnimation
- mNumChildren: aiNode
- mNumFaces: aiMesh
- mNumIndices: aiFace
- mNumKeys: aiMeshAnim
- mNumLights: aiScene
- mNumMaterials: aiScene
- mNumMeshChannels: aiAnimation
- mNumMeshes: aiNode, aiScene
- mNumPositionKeys: aiNodeAnim
- mNumProperties: aiMaterial, aiMetadata
- mNumRotationKeys: aiNodeAnim
- mNumScalingKeys: aiNodeAnim
- mNumTextures: aiScene
- mNumUVComponents: aiMesh
- mNumVertices: aiAnimMesh, aiMesh
- mNumWeights: aiBone
- mOffsetMatrix: aiBone
- mParent: aiNode
- mPosition: aiCamera, aiLight
- mPositionKeys: aiNodeAnim
- mPostState: aiNodeAnim
- mPreState: aiNodeAnim
- mPrimitiveTypes: aiMesh
- mPrivate: aiScene
- mProperties: aiMaterial
- mRootNode: aiScene
- mRotation: aiUVTransform
- mRotationKeys: aiNodeAnim
- mScaling: aiUVTransform
- mScalingKeys: aiNodeAnim
- mStringProperties: Assimp::ExportProperties
- mTangents: aiAnimMesh, aiMesh
- mTextureCoords: aiAnimMesh, aiMesh
- mTextures: aiScene
- mTicksPerSecond: aiAnimation
- mTime: aiMeshKey, aiQuatKey, aiVectorKey
- mTransformation : aiNode
- mTranslation : aiUVTransform
- mType : aiLight, aiMetadataEntry
- mUp : aiCamera
- mValue : aiMeshKey, aiQuatKey, aiVectorKey
- mValues : aiMetadata
- mVertexId : aiVertexWeight
- mVertices : aiAnimMesh, aiMesh
- mWeight : aiVertexWeight
- mWeights : aiBone
- mWidth : aiTexture
- n -

- name: aiExportDataBlob
- next: aiExportDataBlob
- nodes: aiMemoryInfo

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
- o -

- OpenProc: aiFileIO
- p -

- pcData : aiTexture
- pimpl : Assimp::Exporter, Assimp::Importer
- pos : aiRay
- r -

- r : aiColor3D, aiColor4t<TReal>, aiTexel
- ReadProc : aiFile
- S -

- SeekProc : aiFile
- sentinel : aiPropertyStore
- size : aiExportDataBlob

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by **doxygen** 1.8.11
- t -

- TellProc : aiFile
- textures : aiMemoryInfo
- total : aiMemoryInfo
- u -

- user : aiLogStream
- UserData : aiFile, aiFileIO
- v -

- $v : ai\text{Vector2t}<T\text{Real}>$, $ai\text{Vector3t}<T\text{Real}>$
- W -

- w : aiQuaternion< TReal >
- WriteProc : aiFile

Generated on Sun Feb 21 2016 19:42:29 for Assimp by `doxygen` 1.8.11
- X -

- x : aiQuaternion< TReal >, aiVector2t< TReal >, aiVector3t< TReal >
- y -

- y: aiQuaternion< TReal >, aiVector2t< TReal >, aiVector3t< TReal >

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
- Z -

- **z**: `aiQuaternion< TReal >`, `aiVector3t< TReal >`
- elem_type: `aiMeshKey`, `aiQuatKey`, `aiVectorKey`
- FloatPropertyMap: `Assimp::ExportProperties`
- fpExportFunc: `Assimp::Exporter`
- IntPropertyMap: `Assimp::ExportProperties`
- KeyType: `Assimp::ExportProperties`
- MatrixPropertyMap: `Assimp::ExportProperties`
- StringPropertyMap: `Assimp::ExportProperties`
## Assimp v3.1.1 (June 2014)

- ErrorSeverity: [Assimp::Logger](assimp::Logger)
- LogSeverity: [Assimp::Logger](assimp::Logger)
- TextFileMode: [Assimp::BaseImporter](assimp::BaseImporter)

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by [doxygen](https://www.doxygen.org) 1.8.11*
• ALLOW_EMPTY : Assimp::BaseImporter
• Debugging : Assimp::Logger
• Err : Assimp::Logger
• FORBID_EMPTY : Assimp::BaseImporter
• Info : Assimp::Logger
• NORMAL : Assimp::Logger
• VERBOSE : Assimp::Logger
• Warn : Assimp::Logger
- Importer: **Assimp::BaseImporter**
Assimp v3.1.1 (June 2014)

Here is a list of all files with brief descriptions:

- **ai_assert.h**

- **AndroidJNIIOSystem.h**

- **anim.h** Defines the data structures in which the imported animations are returned

- **BaseImporter.h**

- **camera.h** Defines the aiCamera data structure

- **cexport.h** Defines the C-API for the Assimp export interface

- **cfileio.h** Defines generic C routines to access memory-mapped files

- **cinport.h** Defines the C-API to the Open Asset Import Library

- **color4.h** RGBA color structure, including operators when compiling in C++

- **color4.inl** Inline implementation of aiColor4t<TReal> operators

- **config.h** Defines constants for configurable properties for the library

- **DefaultLogger.hpp**

- **defs.h** Assimp build configuration setup

- **dox.h** General documentation built from a doxygen comment

- **Exporter.hpp** Defines the CPP-API for the Assimp export interface

- **Importer.hpp** Defines the C++-API to the Open Asset Import Library

- **importerdesc.h** aiImporterFlags, aiImporterDesc implementation

- **IOStream.hpp** File I/O wrappers for C++

- **IOSystem.hpp** File system wrapper for C++

- **light.h** Defines the aiLight data structure

- **Logger.hpp** Abstract base class 'Logger', base of the logging system
<table>
<thead>
<tr>
<th>Header File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogStream.hpp</td>
<td>Abstract base class 'LogStream', representing an output log stream</td>
</tr>
<tr>
<td>material.h</td>
<td>Defines the material system of the library</td>
</tr>
<tr>
<td>material.inl</td>
<td>Defines the C++ getters for the material system</td>
</tr>
<tr>
<td>matrix3x3.h</td>
<td>Definition of a 3x3 matrix, including operators when compiling in C++</td>
</tr>
<tr>
<td>matrix3x3.inl</td>
<td>Inline implementation of the 3x3 matrix operators</td>
</tr>
<tr>
<td>matrix4x4.h</td>
<td>4x4 matrix structure, including operators when compiling in C++</td>
</tr>
<tr>
<td>matrix4x4.inl</td>
<td>Inline implementation of the 4x4 matrix operators</td>
</tr>
<tr>
<td>mesh.h</td>
<td>Declares the data structures in which the imported geometry is returned by ASSIMP: aiMesh, aiFace and aiBone data structures</td>
</tr>
<tr>
<td>metadata.h</td>
<td>Defines the data structures for holding node meta information</td>
</tr>
<tr>
<td>NullLogger.hpp</td>
<td>Dummy logger</td>
</tr>
<tr>
<td>packpack1.h</td>
<td>Definitions for import post processing steps</td>
</tr>
<tr>
<td>ProgressHandler.hpp</td>
<td>Abstract base class 'ProgressHandler'</td>
</tr>
<tr>
<td>pstdint.h</td>
<td>Quaternion structure, including operators when compiling in C++</td>
</tr>
<tr>
<td>quaternion.h</td>
<td>Inline implementation of aiQuaternion&lt;TReal&gt; operators</td>
</tr>
<tr>
<td>quaternion.inl</td>
<td>Defines the data structures in which the imported scene is returned</td>
</tr>
<tr>
<td>scene.h</td>
<td>Defines texture helper structures for the library</td>
</tr>
<tr>
<td>texture.h</td>
<td>Basic data types and primitives, such as vectors or colors</td>
</tr>
<tr>
<td>types.h</td>
<td>2D vector structure, including operators when compiling in C++</td>
</tr>
<tr>
<td>vector2.h</td>
<td>Inline implementation of aiVector2t&lt;TReal&gt; operators</td>
</tr>
<tr>
<td>vector2.inl</td>
<td>3D vector structure, including operators when compiling in C++</td>
</tr>
<tr>
<td>vector3.h</td>
<td></td>
</tr>
</tbody>
</table>
vector3.inl

Inline implementation of aiVector3<TReal> operators

version.h

Functions to query the version of the Assimp runtime, check compile flags, ..
### Assimp v3.1.1 (June 2014)

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Namespaces</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td>File Members</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- [include](#)
- [assimp](#)

**Macros**

ai_assert.h File Reference
Macros

#define ai_assert(expression)
Macro Definition Documentation

#define ai_assert ( expression )
Assimp v3.1.1 (June 2014)

- include
- assimp
- port
- AndroidJNI

AndroidJNIIOSystem.h File Reference

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
anim.h File Reference

Defines the data structures in which the imported animations are returned.

More...
Classes

struct aiAnimation

An animation consists of keyframe data for a number of nodes. More...

struct aiMeshAnim

Describes vertex-based animations for a single mesh or a group of meshes. More...

struct aiMeshKey

Binds a anim mesh to a specific point in time. More...

struct aiNodeAnim

Describes the animation of a single node. More...

struct aiQuatKey

A time-value pair specifying a rotation for the given time. More...

struct aiVectorKey

A time-value pair specifying a certain 3D vector for the given time. More...

struct Assimp::Interpolator< T >

CPP-API: Utility class to simplify interpolations of various data types. More...
Namespaces

**Assimp**

Assimp's CPP-API and all internal APIs.
Enumerations
Detailed Description

Defines the data structures in which the imported animations are returned.
### Enumeration Type Documentation

**enum aiAnimBehaviour**

Defines how an animation channel behaves outside the defined time range.

This corresponds to `aiNodeAnim::mPreState` and `aiNodeAnim::mPostState`.

#### Enumerator

<table>
<thead>
<tr>
<th>aiAnimBehaviour_DEFAULT</th>
<th>The value from the default node transformation is taken.</th>
</tr>
</thead>
<tbody>
<tr>
<td>aiAnimBehaviour_CONSTANT</td>
<td>The nearest key value is used without interpolation.</td>
</tr>
<tr>
<td>aiAnimBehaviour_LINEAR</td>
<td>The value of the nearest two keys is linearly extrapolated for the current time value.</td>
</tr>
<tr>
<td>aiAnimBehaviour_REPEAT</td>
<td>The animation is repeated.</td>
</tr>
<tr>
<td>_aiAnimBehaviour_Force32Bit</td>
<td>This value is not used, it is just here to force the compiler to map this enum to a 32 Bit integer.</td>
</tr>
</tbody>
</table>
Assimp v3.1.1 (June 2014)

- code

Classes | Namespaces | Macros
BaseImporter.h File Reference
Classes

class Assimp::BaseImporter

FOR IMPORTER PLUGINS ONLY: The BaseImporter defines a common interface for all importer worker classes. More...

struct Assimp::ScopeGuard< T >
Namespaces

Assimp
Assimp's CPP-API and all internal APIs.
Macros

#define AI_MAKE_MAGIC(string)
Macro Definition Documentation

#define AI_MAKE_MAGIC ( string )

Value:
((uint32_t)((string[0] << 24) + \
• `include`
• `assimp`

**Classes**
camera.h File Reference

Defines the `aiCamera` data structure. [More...](#)
Classes

struct aiCamera
    Helper structure to describe a virtual camera. [More...](#)
Detailed Description

Defines the aiCamera data structure.
Assimp v3.1.1 (June 2014)

• include
• assimp

**Classes | Functions**
cexport.h File Reference

Defines the C-API for the Assimp export interface. More...
Classes

struct aiExportDataBlob
    Describes a blob of exported scene data. More...

struct aiExportFormatDesc
    Describes a file format which Assimp can export to. More...
**Functions**

**ASSIMP_API** void **aiCopyScene** (const **aiScene** *pIn, **aiScene** **pOut)**
Create a modifiable copy of a scene. [More...](#)

**ASSIMP_API** **aiExportScene** (const **aiScene** *pScene, const char *pFormatId, const char *pFileName, unsigned int pPreprocessing)
Exports the given scene to a chosen file format and writes the result file(s) to disk. [More...](#)

**ASSIMP_API** **aiExportSceneEx** (const **aiScene** *pScene, const char *pFormatId, const char *pFileName, **aiFileIO** *pIO, unsigned int pPreprocessing)
Exports the given scene to a chosen file format using custom IO logic supplied by you. [More...](#)

**ASSIMP_API** const **aiExportSceneToBlob** (const **aiScene** *pScene, const char *pFormatId, unsigned int pPreprocessing)
Exports the given scene to a chosen file format. [More...](#)

**ASSIMP_API** void **aiFreeScene** (const **aiScene** *pIn)
Frees a scene copy created using **aiCopyScene()** [More...](#)

**ASSIMP_API** size_t **aiGetExportFormatCount** (void)
Returns the number of export file formats available in the current **Assimp** build. [More...](#)

**ASSIMP_API** const **aiExportFormatDesc** **aiGetExportFormatDescription** (size_t pIndex)
Returns a description of the nth export file format. [More...](#)
ASSIMP_API void aiReleaseExportBlob (const aiExportDataBlob *pData)
Releases the memory associated with the given exported data. More...

ASSIMP_API void aiReleaseExportFormatDescription (const aiExportFormatDesc *desc)
Release a description of the nth export file format. More...
Detailed Description

Defines the C-API for the Assimp export interface.
Function Documentation

**ASSIMP_API** void aiCopyScene ( const aiScene * pIn,  
                                           aiScene **           pOut  
                                     )

Create a modifiable copy of a scene.

This is useful to import files via Assimp, change their topology and export them again. Since the scene returned by the various importer functions is const, a modifiable copy is needed.

Parameters

- **pIn**: Valid scene to be copied
- **pOut**: Receives a modifiable copy of the scene. Use aiFreeScene() to delete it again.

**ASSIMP_API** aiReturn aiExportScene ( const aiScene * pScene,  
                                                    const char *    pFormatId,  
                                                    const char *    pFileName,  
                                                    unsigned int    pPreprocessing  
                                            )

Exports the given scene to a chosen file format and writes the result file(s) to disk.

Parameters

- **pScene**: The scene to export. Stays in possession of the caller, is not changed by the function. The scene is expected to conform to Assimp's Importer output format as specified in the Data Structures Page. In short, this means the model data should use a right-handed coordinate systems, face winding should be counter-clockwise and the UV coordinate origin is assumed to be in the upper left. If your input data uses different conventions, have a look at the last parameter.
pFormatId
ID string to specify to which format you want to export to. Use aiGetExportFormatCount() / aiGetExportFormatDescription() to learn which export formats are available.

pFileName
Output file to write
Accepts any choice of the aiPostProcessSteps enumerated flags, but in reality only a subset of them makes sense here. Specifying 'preprocessing' flags is useful if the input scene does not conform to Assimp's default conventions as specified in the Data Structures Page. In short, this means the geometry data should use a right-handed coordinate systems, face winding should be counter-clockwise and the UV coordinate origin is assumed to be in the upper left. The aiProcess_MakeLeftHanded, aiProcess_FlipUVs and aiProcess_FlipWindingOrder flags are used in the import side to allow users to have those defaults automatically adapted to their conventions. Specifying those flags for exporting has the opposite effect, respectively. Some other of the aiPostProcessSteps enumerated values may be useful as well, but you'll need to try out what their effect on the exported file is. Many formats impose their own restrictions on the structure of the geometry stored therein, so some preprocessing may have little or no effect at all, or may be redundant as exporters would apply them anyhow. A good example is triangulation - whilst you can enforce it by specifying the aiProcess_Triangulate flag, most export formats support only triangulate data so they would run the step anyway.

If assimp detects that the input scene was directly taken from the importer side of the library (i.e. not copied using aiCopyScene and potentially modified afterwards), any postprocessing steps already applied to the scene will not be applied again, unless they show non-idempotent behaviour (aiProcess_MakeLeftHanded, aiProcess_FlipUVs and aiProcess_FlipWindingOrder).

Returns
a status code indicating the result of the export
Note
Use \texttt{aiCopyScene()} to get a modifiable copy of a previously imported scene.

\begin{verbatim}
ASSIMP\_API \texttt{aiReturn} aiExportSceneEx ( const \texttt{aiScene} * \texttt{pScene},
const char * \texttt{pFormatId},
const char * \texttt{pFileName},
\texttt{aiFileIO} * \texttt{pIO},
unsigned int \texttt{pPreprocessing} )
\end{verbatim}

Exports the given scene to a chosen file format using custom IO logic supplied by you.

Parameters

\textbf{pScene} \hfill The scene to export. Stays in possession of the caller, is not changed by the function.

\textbf{pFormatId} \hfill ID string to specify to which format you want to export to. Use \texttt{aiGetExportFormatCount()} / \texttt{aiGetExportFormatDescription()} to learn which export formats are available.

\textbf{pFileName} \hfill Output file to write custom IO implementation to be used. Use this if you use your own storage methods. If none is supplied, a default implementation using standard file IO is used. Note that \texttt{aiExportSceneToBlob} is provided as convenience function to export to memory buffers.

\textbf{pPreprocessing} \hfill Please see the documentation for \texttt{aiExportScene}

Returns

a status code indicating the result of the export

Note
Include \texttt{<aiFileIO.h>} for the definition of \texttt{aiFileIO}.
Use \texttt{aiCopyScene()} to get a modifiable copy of a previously imported scene.
Exports the given scene to a chosen file format.

Returns the exported data as a binary blob which you can write into a file or something. When you're done with the data, use `aiReleaseExportBlob()` to free the resources associated with the export.

### Parameters

- **pScene**
  - The scene to export. Stays in possession of the caller, is not changed by the function.

- **pFormatId**
  - ID string to specify to which format you want to export to. Use `aiGetExportFormatCount()` / `aiGetExportFormatDescription()` to learn which export formats are available.

- **pPreprocessing**
  - Please see the documentation for `aiExportScene`

### Returns

- the exported data or NULL in case of error

### Functions

- **ASSIMP_API void aiFreeScene ( const aiScene * pIn )**
  - Frees a scene copy created using `aiCopyScene()`

- **ASSIMP_API size_t aiGetExportFormatCount ( void )**
  - Returns the number of export file formats available in the current Assimp build.
  - Use `aiGetExportFormatDescription()` to retrieve infos of a specific export format.
Returns a description of the nth export file format.

Use `aiGetExportFormatCount()` to learn how many export formats are supported. The description must be released by calling `aiReleaseExportFormatDescription` afterwards.

Parameters

    pIndex     Index of the export format to retrieve information for. Valid range is 0 to `aiGetExportFormatCount()`

Returns

    A description of that specific export format. NULL if pIndex is out of range.

ASSIMP_API void aiReleaseExportBlob ( const aiExportDataBlob * pData )

Releases the memory associated with the given exported data.

Use this function to free a data blob returned by `aiExportScene()`.

Parameters

    pData the data blob returned by `aiExportSceneToBlob`

ASSIMP_API void
aiReleaseExportFormatDescription ( const aiExportFormatDesc * desc )

Release a description of the nth export file format.

Must be returned by `aiGetExportFormatDescription`

Parameters

    desc Pointer to the description
• include
• assimp

Classes | Typedefs
cfileio.h File Reference

Defines generic C routines to access memory-mapped files. More...
Classes

struct aiFile
C-API: File callbacks. More...

struct aiFileIO
C-API: File system callbacks. More...
**Typedefs**

typedef void(* aiFileCloseProc) (aiFileIO *, aiFile *)

typedef void(* aiFileFlushProc) (aiFile *)

typedef aiFile *(* aiFileOpenProc) (aiFileIO *, const char *, const char *)

typedef size_t(* aiFileReadProc) (aiFile *, char *, size_t, size_t)

typedef aiReturn(* aiFileSeek) (aiFile *, size_t, aiOrigin)

typedef size_t(* aiFileTellProc) (aiFile *)

typedef size_t(* aiFileWriteProc) (aiFile *, const char *, size_t, size_t)

typedef char * aiUserData
Detailed Description

Defines generic C routines to access memory-mapped files.
**Typedef Documentation**

typedef void(* aiFileCloseProc) (aiFileIO *, aiFile *)
typedef void(* aiFileFlushProc) (aiFile *)
typedef aiFile*(aiFileOpenProc) (aiFileIO *, const char *, const char *)
typedef size_t(* aiFileReadProc) (aiFile *, char *, size_t, size_t)
typedef aiReturn (aiFileSeek) (aiFile *, size_t, aiOrigin)
typedef size_t(* aiFileTellProc) (aiFile *)
typedef size_t(* aiFileWriteProc) (aiFile *, const char *, size_t, size_t)
typedef char* aiUserData

---

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11*
**Assimp v3.1.1 (June 2014)**

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Namespaces</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td>File Members</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- [include](#)
- [assimp](#)

[Classes](#) | [Macros](#) | [Typedefs](#) | [Functions](#)

cimport.h File Reference

Defines the C-API to the Open Asset Import Library. [More...](#)
Classes

struct aiLogStream

C-API: Represents a log stream. More...

struct aiPropertyStore

C-API: Represents an opaque set of settings to be used during importing. More...
Macros

#define AI_FALSE 0

#define AI_TRUE 1
Typedefs

typedef int aiBool
   Our own C boolean type. More...

typedef void(* aiLogStreamCallback) (const char *, char *)
Functions

ASSIMP_API const aiScene* aiApplyPostProcessing (const aiScene *pScene, unsigned int pFlags)

Apply post-processing to an already-imported scene. More...

ASSIMP_API void aiAttachLogStream (const aiLogStream *stream)

Attach a custom log stream to the libraries' logging system. More...

ASSIMP_API aiPropertyStore aiCreatePropertyStore (void)

Create an empty property store. More...

ASSIMP_API aiCreateQuaternionFromMatrix (aiQuaternion *quat, const aiMatrix3x3 *mat)

Construct a quaternion from a 3x3 rotation matrix. More...

ASSIMP_API void *aiDecomposeMatrix (const aiMatrix4x4 *mat, aiVector3D *scaling, aiQuaternion *rotation, aiVector3D *position)

Decompose a transformation matrix into its rotational, translational and scaling components. More...

ASSIMP_API void aiDetachAllLogStreams (void)

Detach all active log streams from the libraries' logging system. More...

ASSIMP_API aiReturn aiDetachLogStream (const aiLogStream *stream)

Detach a custom log stream from the libraries' logging system.
ASSIMP_API void aiEnableVerboseLogging (aiBool d)
Enable verbose logging. More...

ASSIMP_API const char * aiGetErrorString ()
Returns the error text of the last failed import process. More...

ASSIMP_API void aiGetExtensionList (aiString *szOut)
Get a list of all file extensions supported by ASSIMP. More...

ASSIMP_API size_t aiGetImportFormatCount (void)
Returns the number of import file formats available in the current Assimp build. More...

ASSIMP_API const aiImporterDesc * aiGetImportFormatDescription (size_t pIndex)
Returns a description of the nth import file format. More...

ASSIMP_API aiGetMemoryRequirements (const aiScene *pIn, aiMemoryInfo *in)
Get the approximated storage required by an imported asset. More...

ASSIMP_API aiGetPredefinedLogStream (aiDefaultLogStream pStreams, aiLogStream const char *file)
Get one of the predefine log streams. More...
ASSIMP_API aiIdentityMatrix3 (aiMatrix3x3 *mat)
    void
Get a 3x3 identity matrix. More...

ASSIMP_API aiIdentityMatrix4 (aiMatrix4x4 *mat)
    void
Get a 4x4 identity matrix. More...

ASSIMP_API
const aiScene aiImportFile (const char *pFile, unsigned int pFlags)
    *
Reads the given file and returns its content. More...

ASSIMP_API
const aiScene aiImportFileEx (const char *pFile, unsigned int pFlags, aiFileIO *pFS)
    *
Reads the given file using user-defined I/O functions and returns its content. More...

ASSIMP_API
const aiScene aiImportFileExWithProperties (const char *pFile, unsigned int pFlags, aiFileIO *pFS, const aiPropertyStore *pProps)
    *
Same as aiImportFileEx, but adds an extra parameter containing importer settings. More...

ASSIMP_API
const aiScene aiImportFileFromMemory (const char *pBuffer, unsigned int pLength, unsigned int pFlags, const char *pHint)
    *
Reads the given file from a given memory buffer,. More...

ASSIMP_API aiImportFileFromMemoryWithProperties (const char *pBuffer, const aiScene unsigned int pLength, unsigned int pFlags, const char *pHint, *
    const aiPropertyStore *pProps)
    *
Same as aiImportFileFromMemory, but adds an extra parameter containing importer settings. More...
ASSIMP_API aiBool aiIsExtensionSupported (const char *szExtension)

Returns whether a given file extension is supported by ASSIMP. More...

ASSIMP_API void aiMultiplyMatrix3 (aiMatrix3x3 *dst, const aiMatrix3x3 *src)

Multiply two 3x3 matrices. More...

ASSIMP_API void aiMultiplyMatrix4 (aiMatrix4x4 *dst, const aiMatrix4x4 *src)

Multiply two 4x4 matrices. More...

ASSIMP_API void aiReleaseImport (const aiScene *pScene)

Releases all resources associated with the given import process. More...

ASSIMP_API void aiReleasePropertyStore (aiPropertyStore *p)

Delete a property store. More...

ASSIMP_API aiSetImportPropertyFloat (aiPropertyStore *store, const char *szName, float value)

Set a floating-point property. More...

ASSIMP_API aiSetImportPropertyInteger (aiPropertyStore *store, const char *szName, int value)

Set an integer property. More...

ASSIMP_API aiSetImportPropertyMatrix (aiPropertyStore *store, const char *szName, const aiMatrix4x4 *mat)

Set a matrix property. More...
ASSIMP_API aiSetImportPropertyString (aiPropertyStore *store, const char *szName, const aiString *st)

Set a string property. More...

ASSIMP_API aiTransformVecByMatrix3 (aiVector3D *vec, const aiMatrix3x3 *mat)

Transform a vector by a 3x3 matrix. More...

ASSIMP_API aiTransformVecByMatrix4 (aiVector3D *vec, const aiMatrix4x4 *mat)

Transform a vector by a 4x4 matrix. More...

ASSIMP_API void aiTransposeMatrix3 (aiMatrix3x3 *mat)

Transpose a 3x3 matrix. More...

ASSIMP_API void aiTransposeMatrix4 (aiMatrix4x4 *mat)

Transpose a 4x4 matrix. More...
**Detailed Description**

Defines the C-API to the Open Asset Import Library.
Macro Definition Documentation

#define AI_FALSE 0
#define AI_TRUE 1
Typedef Documentation

typedef int aiBool

Our own C boolean type.

typedef void(* aiLogStreamCallback) (const char *, char *)
Function Documentation

**ASSIMP_API** const *aiScene* aiApplyPostProcessing ( const *aiScene* *pScene,
                                  unsigned int pFlags )

Apply post-processing to an already-imported scene.

This is strictly equivalent to calling `aiImportFile()/aiImportFileEx` with the same flags. However, you can use this separate function to inspect the imported scene first to fine-tune your post-processing setup.

Parameters
- pScene Scene to work on.
- pFlags Provide a bitwise combination of the `aiPostProcessSteps` flags.

Returns
A pointer to the post-processed data. Post processing is done in-place, meaning this is still the same `aiScene` which you passed for pScene. However, *if* post-processing failed, the scene could now be NULL. That's quite a rare case, post processing steps are not really designed to 'fail'. To be exact, the `aiProcess_ValidateDataStructure` flag is currently the only post processing step which can actually cause the scene to be reset to NULL.

**ASSIMP_API** void aiAttachLogStream ( const *aiLogStream* *stream )

Attach a custom log stream to the libraries' logging system.

Attaching a log stream can slightly reduce Assimp's overall import performance. Multiple log-streams can be attached.

Parameters
- stream Describes the new log stream.

Note
To ensure proper destruction of the logging system, you need to manually call `aiDetachLogStream()` on every single log stream you attach.
Alternatively (for the lazy folks) `aiDetachAllLogStreams` is provided.

**ASSIMP_API** `aiPropertyStore* aiCreatePropertyStore ( void )`

Create an empty property store.

Property stores are used to collect import settings.

Returns

New property store. Property stores need to be manually destroyed using the `aiReleasePropertyStore` API function.

**ASSIMP_API** `void aiCreateQuaternionFromMatrix ( aiQuaternion * quat, const aiMatrix3x3 * mat )`

Construct a quaternion from a 3x3 rotation matrix.

Parameters

- `quat` Receives the output quaternion.
- `mat` Matrix to 'quaternionize'.

See also

`aiQuaternion(const aiMatrix3x3& pRotMatrix)`

**ASSIMP_API** `void aiDecomposeMatrix ( const aiMatrix4x4 * mat, aiVector3D * scaling, aiQuaternion * rotation, aiVector3D * position )`

Decompose a transformation matrix into its rotational, translational and scaling components.

Parameters

- `mat` Matrix to decompose
- `scaling` Receives the scaling component
rotation Receives the rotational component
position Receives the translational component.

See also
aiMatrix4x4::Decompose (aiVector3D&, aiQuaternion&, aiVector3D&) const;

ASSIMP_API void aiDetachAllLogStreams ( void )

Detach all active log streams from the libraries' logging system.
This ensures that the logging system is terminated properly and all resources allocated by it are actually freed. If you attached a stream, don't forget to detach it again.

See also
aiAttachLogStream
aiDetachLogStream

ASSIMP_API aiReturn aiDetachLogStream ( const aiLogStream* stream )

Detach a custom log stream from the libraries' logging system.
This is the counterpart of aiAttachLogStream. If you attached a stream, don't forget to detach it again.

Parameters
stream The log stream to be detached.

Returns
AI_SUCCESS if the log stream has been detached successfully.

See also
aiDetachAllLogStreams

ASSIMP_API void aiEnableVerboseLogging ( aiBool d )

Enable verbose logging.
Verbose logging includes debug-related stuff and detailed import statistics. This can have severe impact on import performance and memory consumption. However, it might be useful to find out why a file didn't read correctly.

Parameters

    d AI_TRUE or AI_FALSE, your decision.

**ASSIMP_API** const char* aiGetErrorString()

Returns the error text of the last failed import process.

Returns

A textual description of the error that occurred at the last import process.
NULL if there was no error. There can't be an error if you got a non-NULL aiScene from aiImportFile/aiImportFileEx/aiApplyPostProcessing.

**ASSIMP_API** void aiGetExtensionList( aiString * szOut )

Get a list of all file extensions supported by ASSIMP.

If a file extension is contained in the list this does, of course, not mean that ASSIMP is able to load all files with this extension.

Parameters

    szOut String to receive the extension list. Format of the list:
        "*.3ds;*.obj;*.dae". NULL is not a valid parameter.

**ASSIMP_API** size_t aiGetImportFormatCount( void )

Returns the number of import file formats available in the current Assimp build.

Use aiGetImportFormatDescription() to retrieve infos of a specific import format.

**ASSIMP_API** const aiImporterDesc* aiGetImportFormatDescription( size_t pIndex )

Returns a description of the nth import file format.
Use `aiGetImportFormatCount()` to learn how many import formats are supported.

Parameters

- `pIndex` Index of the import format to retrieve information for. Valid range is 0 to `aiGetImportFormatCount()`

Returns

- A description of that specific import format. NULL if `pIndex` is out of range.

```
ASSIMP_API void aiGetMemoryRequirements ( const aiScene * pIn,
                                          aiMemoryInfo * in
)
```

Get the approximated storage required by an imported asset.

Parameters

- `pIn` Input asset.
- `in` Data structure to be filled.

```
ASSIMP_API aiLogStream aiGetPredefinedLogStream
     ( aiDefaultLogStream pStreams,
       const char * file
     )
```

Get one of the predefine log streams.

This is the quick'n'easy solution to access Assimp's log system. Attaching a log stream can slightly reduce Assimp's overall import performance.

Usage is rather simple (this will stream the log to a file, named log.txt, and the stdout stream of the process:

```
1 struct aiLogStream c;
2 c = aiGetPredefinedLogStream(aiDefaultLogStream_FILE,"log.txt");
3 aiAttachLogStream(&c);
4 c = aiGetPredefinedLogStream(aiDefaultLogStream_STDOUT,NULL);
```
5 aiAttachLogStream(&c);

Parameters
  pStreams One of the aiDefaultLogStream enumerated values.
  file Solely for the aiDefaultLogStream_FILE flag: specifies the file to write to. Pass NULL for all other flags.

Returns
  The log stream. callback is set to NULL if something went wrong.

ASSIMP_API void aiIdentityMatrix3 ( aiMatrix3x3 * mat )

Get a 3x3 identity matrix.

Parameters
  mat Matrix to receive its personal identity

ASSIMP_API void aiIdentityMatrix4 ( aiMatrix4x4 * mat )

Get a 4x4 identity matrix.

Parameters
  mat Matrix to receive its personal identity

ASSIMP_API const aiScene* aiImportFile ( const char * pFile,
                                          unsigned int pFlags )

Reads the given file and returns its content.

If the call succeeds, the imported data is returned in an aiScene structure. The data is intended to be read-only, it stays property of the ASSIMP library and will be stable until aiReleaseImport() is called. After you're done with it, call aiReleaseImport() to free the resources associated with this file. If the import fails, NULL is returned instead. Call aiGetErrorString() to retrieve a human-readable error text.

Parameters
pFile  Path and filename of the file to be imported, expected to be a null-terminated c-string. NULL is not a valid value.

Optional post processing steps to be executed after a successful import. Provide a bitwise combination of the aiPostProcessSteps flags.

Returns
Pointer to the imported data or NULL if the import failed.

ASSIMP_API const aiScene* aiImportFileEx ( const char * pFile, unsigned int pFlags, aiFileIO * pFS )

Reads the given file using user-defined I/O functions and returns its content.

If the call succeeds, the imported data is returned in an aiScene structure. The data is intended to be read-only, it stays property of the ASSIMP library and will be stable until aiReleaseImport() is called. After you're done with it, call aiReleaseImport() to free the resources associated with this file. If the import fails, NULL is returned instead. Call aiGetErrorString() to retrieve a human-readable error text.

Parameters

pFile  Path and filename of the file to be imported, expected to be a null-terminated c-string. NULL is not a valid value.

Optional post processing steps to be executed after a successful import. Provide a bitwise combination of the aiPostProcessSteps flags.

pFlags aiFileIO structure. Will be used to open the model file itself and any other files the loader needs to open. Pass NULL to use the default implementation.

pFS

Returns
Pointer to the imported data or NULL if the import failed.

Note
Include <aiFileIO.h> for the definition of aiFileIO.
ASSIMP_API const aiScene* aiImportFileExWithProperties
(const char * pFile,
unsigned int pFlags,
aiFileIO * pFS,
const aiPropertyStore * pProps)

Same as aiImportFileEx, but adds an extra parameter containing importer settings.

Parameters

pFile Path and filename of the file to be imported, expected to be a null-terminated c-string. NULL is not a valid value.

Optional post processing steps to be executed after a successful import. Provide a bitwise combination of the aiPostProcessSteps flags.

aiFileIO structure. Will be used to open the model file itself and any other files the loader needs to open. Pass NULL to use the default implementation.

pProps aiPropertyStore instance containing import settings.

Returns
Pointer to the imported data or NULL if the import failed.

Note
Include <aiFileIO.h> for the definition of aiFileIO.

See also
aiImportFileEx

ASSIMP_API const aiScene* aiImportFileFromMemory
(const char * pBuffer,
unsigned int pLength,
unsigned int pFlags,
Reads the given file from a given memory buffer.

If the call succeeds, the contents of the file are returned as a pointer to an aiScene object. The returned data is intended to be read-only, the importer keeps ownership of the data and will destroy it upon destruction. If the import fails, NULL is returned. A human-readable error description can be retrieved by calling aiGetErrorString().

Parameters

pBuffer Pointer to the file data
pLength Length of pBuffer, in bytes
pFlags Optional post processing steps to be executed after a successful import. Provide a bitwise combination of the aiPostProcessSteps flags. If you wish to inspect the imported scene first in order to fine-tune your post-processing setup, consider to use aiApplyPostProcessing().

pHint An additional hint to the library. If this is a non empty string, the library looks for a loader to support the file extension specified by pHint and passes the file to the first matching loader. If this loader is unable to completely the request, the library continues and tries to determine the file format on its own, a task that may or may not be successful. Check the return value, and you'll know ...

Returns

A pointer to the imported data, NULL if the import failed.

Note

This is a straightforward way to decode models from memory buffers, but it doesn't handle model formats that spread their data across multiple files or even directories. Examples include OBJ or MD3, which outsource parts of their material info into external scripts. If you need full functionality, provide a custom IOSystem to make Assimp find these files and use the regular aiImportFileEx() / aiImportFileExWithProperties() API.

ASSIMP_API const aiScene* ( const char * pBuffer,

const char * pHint )
aiImportFileFromMemoryWithProperties

unsigned int pLength,
unsigned int pFlags,
const char * pHint,
const aiPropertyStore * pProps
)

Same as aiImportFileFromMemory, but adds an extra parameter containing importer settings.

Parameters

pBuffer Pointer to the file data
pLength Length of pBuffer, in bytes
pFlags Optional post processing steps to be executed after a successful import. Provide a bitwise combination of the aiPostProcessSteps flags. If you wish to inspect the imported scene first in order to fine-tune your post-processing setup, consider to use aiApplyPostProcessing().

pHint An additional hint to the library. If this is a non empty string, the library looks for a loader to support the file extension specified by pHint and passes the file to the first matching loader. If this loader is unable to completely the request, the library continues and tries to determine the file format on its own, a task that may or may not be successful. Check the return value, and you'll know ...

pProps aiPropertyStore instance containing import settings.

Returns

A pointer to the imported data, NULL if the import failed.

Note

This is a straightforward way to decode models from memory buffers, but it doesn't handle model formats that spread their data across multiple files or even directories. Examples include OBJ or MD3, which outsource parts of their material info into external scripts. If you need full functionality, provide a custom IOSystem to make Assimp find these files and use the
regular aiImportFileEx() / aiImportFileExWithProperties() API.

See also
aiImportFileFromMemory

ASSIMP_API aiBool aiIsExtensionSupported ( const char * szExtension )

Returns whether a given file extension is supported by ASSIMP.

Parameters

  szExtension Extension for which the function queries support for. Must include a leading dot '. '. Example: " .3ds" , " .md3"

Returns
  AI_TRUE if the file extension is supported.

ASSIMP_API void aiMultiplyMatrix3 ( aiMatrix3x3 * dst, const aiMatrix3x3 * src )

Multiply two 3x3 matrices.

Parameters

  dst First factor, receives result.
  src Matrix to be multiplied with ' dst '.

ASSIMP_API void aiMultiplyMatrix4 ( aiMatrix4x4 * dst, const aiMatrix4x4 * src )

Multiply two 4x4 matrices.

Parameters

  dst First factor, receives result.
  src Matrix to be multiplied with ' dst '.

ASSIMP_API void aiReleaseImport ( const aiScene * pScene )
Releases all resources associated with the given import process.

Call this function after you're done with the imported data.

Parameters

   pScene The imported data to release. NULL is a valid value.

**ASSIMP_API** void aiReleasePropertyStore ( aiPropertyStore * p )

Delete a property store.

Parameters

   p Property store to be deleted.

**ASSIMP_API** void aiSetImportPropertyFloat ( aiPropertyStore * store, 
   const char * szName, 
   float value 
)

Set a floating-point property.

This is the C-version of Assimp::Importer::SetPropertyFloat(). In the C interface, properties are always shared by all imports. It is not possible to specify them per import.

Parameters

   store Store to modify. Use aiCreatePropertyStore to obtain a store.
   Name of the configuration property to be set. All supported public
   szName properties are defined in the config.h header file
   (AI_CONFIG_XXX).
   value New value for the property

**ASSIMP_API** void aiSetImportPropertyInteger ( aiPropertyStore * store, 
   const char * szName, 
   int value 
)
Set an integer property.

This is the C-version of `Assimp::Importer::SetPropertyInteger()`. In the C interface, properties are always shared by all imports. It is not possible to specify them per import.

Parameters

store Store to modify. Use `aiCreatePropertyStore` to obtain a store.

Name of the configuration property to be set. All supported public `szName` properties are defined in the `config.h` header file (AI_CONFIG_XXX).

value New value for the property

```c
ASSIMP_API void aiSetImportPropertyMatrix ( aiPropertyStore * store,
                                          const char * szName,
                                          const aiMatrix4x4 * mat
                                      )
```

Set a matrix property.

This is the C-version of `Assimp::Importer::SetPropertyMatrix()`. In the C interface, properties are always shared by all imports. It is not possible to specify them per import.

Parameters

store Store to modify. Use `aiCreatePropertyStore` to obtain a store.

Name of the configuration property to be set. All supported public `szName` properties are defined in the `config.h` header file (AI_CONFIG_XXX).

mat New value for the property

```c
ASSIMP_API void aiSetImportPropertyString ( aiPropertyStore * store,
                                             const char * szName,
                                             const aiString * st
                                        )
```

Set a string property.
This is the C-version of Assimp::Importer::SetPropertyString(). In the C interface, properties are always shared by all imports. It is not possible to specify them per import.

Parameters
  store Store to modify. Use aiCreatePropertyStore to obtain a store.
  Name of the configuration property to be set. All supported public
  szName properties are defined in the config.h header file
  (AI_CONFIG_XXX).
  st New value for the property

ASSIMP_API void aiTransformVecByMatrix3 ( aiVector3D * vec,      
                                          const aiMatrix3x3 * mat
)

Transform a vector by a 3x3 matrix.

Parameters
  vec Vector to be transformed.
  mat Matrix to transform the vector with.

ASSIMP_API void aiTransformVecByMatrix4 ( aiVector3D * vec,      
                                          const aiMatrix4x4 * mat
)

Transform a vector by a 4x4 matrix.

Parameters
  vec Vector to be transformed.
  mat Matrix to transform the vector with.

ASSIMP_API void aiTransposeMatrix3 ( aiMatrix3x3 * mat )

Transpose a 3x3 matrix.

Parameters
  mat Pointer to the matrix to be transposed
ASSIMP_API void aiTransposeMatrix4 ( aiMatrix4x4 * mat )

Transpose a 4x4 matrix.

Parameters

mat Pointer to the matrix to be transposed
RGB color structure, including operators when compiling in C++.
Classes

class aiColor4t<TReal>

Represents a color in Red-Green-Blue space including an alpha component. More...
typedef aiColor4t< float > aiColor4D
Variables

class aiColor4t PACK_STRUCT
Detailed Description

RGBA color structure, including operators when compiling in C++.
typedef aiColor4t<float> aiColor4D
Variable Documentation

struct aiColor3D PACK_STRUCT

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp v3.1.1 (June 2014)

- include
- assimp

Macros | Functions

color4.inl File Reference

Inline implementation of aiColor4t<TReal> operators. More...
Macros

#define AI_COLOR4D_INL_INC
Functions

template<typename TReal >
  AI_FORCE_INLINE operator* (const aiColor4t<TReal>& v1, const aiColor4t<TReal>& v2)

template<typename TReal >
  AI_FORCE_INLINE aiColor4t<TReal> operator* (TReal f, const aiColor4t<TReal>& v)

template<typename TReal >
  AI_FORCE_INLINE aiColor4t<TReal> operator* (const aiColor4t<TReal>& v, TReal f)

template<typename TReal >
  AI_FORCE_INLINE aiColor4t<TReal> operator+ (const aiColor4t<TReal>& v1, const aiColor4t<TReal>& v2)

template<typename TReal >
  AI_FORCE_INLINE aiColor4t<TReal> operator+ (TReal f, const aiColor4t<TReal>& v)

template<typename TReal >
  AI_FORCE_INLINE aiColor4t<TReal> operator- (const aiColor4t<TReal>& v1, const aiColor4t<TReal>& v2)

template<typename TReal >
  AI_FORCE_INLINE aiColor4t<TReal> operator- (const aiColor4t<TReal>& v, TReal f)
template<typename TReal >
   AI_FORCE_INLINE aiColor4t<TReal> operator- (TReal f, const aiColor4t<TReal> &v)

template<typename TReal >
   AI_FORCE_INLINE aiColor4t<TReal> operator/ (const aiColor4t<TReal> &v1, const aiColor4t<TReal> &v2)

template<typename TReal >
   AI_FORCE_INLINE aiColor4t<TReal> operator/ (const aiColor4t<TReal> &v, TReal f)

template<typename TReal >
   AI_FORCE_INLINE aiColor4t<TReal> operator/ (TReal f, const aiColor4t<TReal> &v)
Detailed Description

Inline implementation of aiColor4t<TReal> operators.
Macro Definition Documentation

#define AI_COLOR4D_INL_INC
Function Documentation

template<typename TReal >
   AI_FORCE_INLINE aiColor4t<TReal>
operator*(
   const aiColor4t<TReal> &v1,
   const aiColor4t<TReal> &v2)

template<typename TReal >
   AI_FORCE_INLINE aiColor4t<TReal>
operator*(
   TReal f,
   const aiColor4t<TReal> &v)

template<typename TReal >
   AI_FORCE_INLINE aiColor4t<TReal>
operator*(
   const aiColor4t<TReal> &v,
   TReal f)

( TReal f,
   const aiColor4t<TReal> &v)

( TReal f,
   const aiColor4t<TReal> &v)
template<
typename TReal>

\texttt{AI\_FORCE\_INLINE aiColor4t<TReal>}
operator-\texttt{(const aiColor4t<TReal> &v1, const aiColor4t<TReal> &v2)}

\texttt{template<
typename TReal>

\texttt{AI\_FORCE\_INLINE aiColor4t<TReal>}
operator-\texttt{(const aiColor4t<TReal> &v, TReal f)}

\texttt{template<
typename TReal>

\texttt{AI\_FORCE\_INLINE aiColor4t<TReal>}
operator/\texttt{(const aiColor4t<TReal> &v1, const aiColor4t<TReal> &v2)}

\texttt{template<
typename TReal>

\texttt{AI\_FORCE\_INLINE aiColor4t<TReal>}
operator/\texttt{(const aiColor4t<TReal> &v, TReal f)}

\texttt{template<
typename TReal>

\texttt{AI\_FORCE\_INLINE aiColor4t<TReal>}
operator/\texttt{(TReal f, const aiColor4t<TReal> &v)}

\texttt{template<
typename TReal>

\texttt{AI\_FORCE\_INLINE aiColor4t<TReal>}
operator/\texttt{(TReal f, const aiColor4t<TReal> &v)}
Assimp v3.1.1 (June 2014)

- include
- assimp

Macros | Enumerations
config.h File Reference

Defines constants for configurable properties for the library. More...
Macros

#define AI_CONFIG_ANDROID_JNI_ASSIMP_MANAGER_SUPPORT "AI_CONFIG_ANDROID_JNI_ASSIMP_MANAGER_SUPPORT"
Specifies whether the Android JNI asset extraction is supported. More...

#define AI_CONFIG_EXPORT_XFILE_64BIT "EXPORT_XFILE_64BIT"
Specifies the xfile use double for real values of float. More...

#define AI_CONFIG_FAVOUR_SPEED "FAVOUR_SPEED"
A hint to assimp to favour speed against import quality. More...

#define AI_CONFIG_GLOB_MEASURE_TIME "GLOB_MEASURE_TIME"
Enables time measurements. More...

#define AI_CONFIG_IMPORT_AC_EVAL_SUBDIVISION "IMPORT_AC_EVAL_SUBDIVISION"
Configures whether the AC loader evaluates subdivision surfaces (indicated by the 'subdiv' attribute in the file).

#define AI_CONFIG_IMPORT_AC_SEPARATE_BFCULL "IMPORT_AC_SEPARATE_BFCULL"
Configures the AC loader to collect all surfaces which have the "Backface cull" flag set in separate meshes.

#define AI_CONFIG_IMPORT_ASE_RECONSTRUCT_NORMALS "IMPORT_ASE_RECONSTRUCT_NORMALS"
Configures the ASE loader to always reconstruct normal vectors basing on the smoothing groups loaded from the file.

#define AI_CONFIG_IMPORT_COLLADA_IGNORE_UP_DIRECTION "IMPORT_COLLADA_IGNORE_UP_DIRECTION"
Specifies whether the Collada loader will ignore the provided up direction.

#define AI_CONFIG_IMPORT_FBX_OPTIMIZE_EMPTY_ANIMATION_CURVES "IMPORT_FBX_OPTIMIZE_EMPTY_ANIMATION_CURVES"
Specifies whether the importer will drop empty animation curves or animation curves which match the bind pose transformation over their entire defined range. More...

#define AI_CONFIG_IMPORT_FBX_PRESERVE_PIVOTS "IMPORT_FBX_PRESERVE_PIVOTS"
Set whether the fbx importer will preserve pivot points for transformatio
#define AI_CONFIG_IMPORT_FBX_READ_ALL_GEOMETRY_LAYERS "IMPORT_FBX_READ_ALL_GEOMETRY_LAYERS"
Set whether the fbx importer will merge all geometry layers present in the source.

#define AI_CONFIG_IMPORT_FBX_READ_ALL_MATERIALS "IMPORT_FBX_READ_ALL_MATERIALS"
Set whether the fbx importer will read all materials present in the source.

#define AI_CONFIG_IMPORT_FBX_READ_ANIMATIONS "IMPORT_FBX_READ_ANIMATIONS"
Set whether the fbx importer will read animations. More...

#define AI_CONFIG_IMPORT_FBX_READ_CAMERAS "IMPORT_FBX_READ_CAMERAS"
Set whether the fbx importer will read cameras. More...

#define AI_CONFIG_IMPORT_FBX_READ_LIGHTS "IMPORT_FBX_READ_LIGHTS"
Set whether the fbx importer will read light sources. More...

#define AI_CONFIG_IMPORT_FBX_READ_MATERIALS "IMPORT_FBX_READ_MATERIALS"
Set whether the fbx importer will read materials. More...

#define AI_CONFIG_IMPORT_FBX_READ_TEXTURES "IMPORT_FBX_READ_TEXTURES"
Set whether the fbx importer will read embedded textures. More...

#define AI_CONFIG_IMPORT_FBX_STRICT_MODE "IMPORT_FBX_STRICT_MODE"
Set whether the fbx importer will act in strict mode in which only FBX 2013 is supported and any other sub-formats are rejected.

#define AI_CONFIG_IMPORT_GLOBAL_KEYFRAME "IMPORT_GLOBAL_KEYFRAME"
Set the vertex animation keyframe to be imported. More...

#define AI_CONFIG_IMPORT_IFC_CUSTOM_TRIANGULATION "IMPORT_IFC_CUSTOM_TRIANGULATION"
Specifies whether the IFC loader will use its own, custom triangulation algorithm to triangulate wall and floor meshes.

#define AI_CONFIG_IMPORT_IFC_SKIP_CURVE_REPRESENTATIONS "IMPORT_IFC_SKIP_CURVE_REPRESENTATIONS"
Specifies whether the IFC loader skips over shape representations of type 'Curve2D'.

#define AI_CONFIG_IMPORT_IFC_SKIP_SPACE_REPRESENTATIONS "IMPORT_IFC_SKIP_SPACE_REPRESENTATIONS"
Specifies whether the IFC loader skips over IfcSpace elements. More...

#define AI_CONFIG_IMPORT_IRR_ANIM_FPS "IMPORT_IRR_ANIM_FPS"
Defines the output frame rate of the IRR loader. More...

#define AI_CONFIG_IMPORT_LWO_ONE_LAYER_ONLY "IMPORT_LWO_ONE_LAYER_ONLY"
Configures the LWO loader to load just one layer from the model. More.

#define AI_CONFIG_IMPORT_LWS_ANIM_END "IMPORT_LWS_ANIM_END"

#define AI_CONFIG_IMPORT_LWS_ANIM_START "IMPORT_LWS_ANIM_START"
Defines the begin of the time range for which the LWS loader evaluates

#define AI_CONFIG_IMPORT_MD2_KEYFRAME "IMPORT_MD2_KEYFRAME"

#define AI_CONFIG_IMPORT_MD3_HANDLE_MULTIPART "IMPORT_MD3_HANDLE_MULTIPART"
Configures the M3D loader to detect and process multi-part Quake player models.

#define AI_CONFIG_IMPORT_MD3_KEYFRAME "IMPORT_MD3_KEYFRAME"

#define AI_CONFIG_IMPORT_MD3_SHADER_SRC "IMPORT_MD3_SHADER_SRC"
Specify the Quake 3 shader file to be used for a particular MD3 file. More...

#define AI_CONFIG_IMPORT_MD3_SKIN_NAME "IMPORT_MD3_SKIN_NAME"
Tells the MD3 loader which skin files to load. More...

#define AI_CONFIG_IMPORT_MD5_NO_ANIM_AUTOLOAD "IMPORT_MD5_NO_ANIM_AUTOLOAD"
Configures the MD5 loader to not load the MD5ANIM file for a MD5M

#define AI_CONFIG_IMPORT_MDC_KEYFRAME "IMPORT_MDC_KEYFRAME"
#define **AI_CONFIG_IMPORT_MDL_COLORMAP**  "IMPORT_MDL_COLC"
Sets the colormap (= palette) to be used to decode embedded textures in MDL (Quake or 3DGS) files.

#define **AI_CONFIG_IMPORT_MDL_KEYFRAME**  "IMPORT_MDL_KEYF"

#define **AI_CONFIG_IMPORT_NO_SKELETON_MESHES**  "IMPORT_NO_SKELETON_MESHES"
Global setting to disable generation of skeleton dummy meshes. [More...](#)

#define **AI_CONFIG_IMPORT_OGRE_MATERIAL_FILE**  "IMPORT_OGRE_MATERIAL_FILE"
Ogre Importer will try to find referenced materials from this file. [More...](#)

#define **AI_CONFIG_IMPORT_OGRE_TEXTURETYPE_FROM_FILENAME**
Ogre Importer detect the texture usage from its filename. [More...](#)

#define **AI_CONFIG_IMPORT_SMD_KEYFRAME**  "IMPORT_SMD_KEYF"

#define **AI_CONFIG_IMPORT_TER_MAKE_UVS**  "IMPORT_TER_MAKE_UVS"
Configures the terragen import plugin to compute uv's for terrains, if not given.

#define **AI_CONFIG_IMPORT_UNREAL_HANDLE_FLAGS**  "UNREAL_HANDLE_FLAGS"
Configures the UNREAL 3D loader to separate faces with different surface flags (e.g. normals).

#define **AI_CONFIG_IMPORT_UNREAL_KEYFRAME**  "IMPORT_UNREAL_KEYFRAME"

#define **AI_CONFIG_PP_CT_MAX_SMOOTHING_ANGLE**  "PP_CT_MAX_SMOOTHING_ANGLE"
Specifies the maximum angle that may be between two vertex tangents that their tangents and bi-tangents are smoothed.

#define **AI_CONFIG_PP_CT_TEXTURE_CHANNEL_INDEX**  "PP_CT_TEXTURE_CHANNEL_INDEX"
Source UV channel for tangent space computation. [More...](#)

#define **AI_CONFIG_PP_DB_ALL_OR_NONE**  "PP_DB_ALL_OR_NONE"
Require all bones qualify for deboning before removing any. [More...](#)
#define  AI_CONFIG_PP_DB_THRESHOLD    "PP_DB_THRESHOLD"
Lower the deboning threshold in order to remove more bones.  More...

#define  AI_CONFIG_PP_FD_REMOVE    "PP_FD_REMOVE"
Configures the aiProcess_FindDegenerates step to remove degenerated primitives.

#define  AI_CONFIG_PP_FID_ANIM_ACCURACY    "PP_FID_ANIM_ACCURACY"
Input parameter to the aiProcess_FindInvalidData step: Specifies the floating-point accuracy for animation values.

#define  AI_CONFIG_PP_GSN_MAX_SMOOTHING_ANGLE    "PP_GSN_MAX_SMOOTHING_ANGLE"
Specify the maximum angle that may be between two face normals at the same vertex position that their are smoothed together.

#define  AI_CONFIG_PP_ICL_PTCACHE_SIZE    "PP_ICL_PTCACHE_SIZE"
Set the size of the post-transform vertex cache to optimize the vertices for.

#define  AI_CONFIG_PP_LBW_MAX_WEIGHTS    "PP_LBW_MAX_WEIGHTS"
Set the maximum number of bones affecting a single vertex.  More...

#define  AI_CONFIG_PP_OG_EXCLUDE_LIST    "PP_OG_EXCLUDE_LIST"
Configures the aiProcess_OptimizeGraph step to preserve nodes matching a name in a given list.

#define  AI_CONFIG_PP_PTV_ADD_ROOT_TRANSFORMATION    "PP_PTV_ADD_ROOT_TRANSFORMATION"
Configures the aiProcess_PreTransformVertices step to use a user-defined matrix as the scene root node transformation before transforming vertices.  More...

#define  AI_CONFIG_PP_PTV_KEEP_HIERARCHY    "PP_PTV_KEEP_HIERARCHY"
Configures the aiProcess_PreTransformVertices step to keep the scene hierarchy.

#define  AI_CONFIG_PP_PTV_NORMALIZE    "PP_PTV_NORMALIZE"
Configures the aiProcess_PreTransformVertices step to normalize all vertex components into the [-1,1] range.

#define  AI_CONFIG_PP_PTV_ROOT_TRANSFORMATION    "PP_PTV_ROOT_TRANSFORMATION"
Configures the aiProcess_PreTransformVertices step to use a user-defined matrix as the scene root node transformation before transforming vertices.
vertices. More...

#define AI_CONFIG_PP_RRM_EXCLUDE_LIST "PP_RRM_EXCLUDE_LIST"
Configures the aiProcess_RemoveRedundantMaterials step to keep mate

#define AI_CONFIG_PP_RVC_FLAGS "PP_RVC_FLAGS"
Input parameter to the aiProcess_RemoveComponent step: Specifies the

#define AI_CONFIG_PP_SBBC_MAX_BONES "PP_SBBC_MAX_BONES"
Maximum bone count per mesh for the SplitbyBoneCount step. More...

#define AI_CONFIG_PP_SBP_REMOVE "PP_SBP_REMOVE"
Input parameter to the aiProcess_SortByPType step: Specifies which pri

#define AI_CONFIG_PP_SLM_TRIANGLE_LIMIT "PP_SLM_TRIANGLE_LIMIT"
Set the maximum number of triangles in a mesh. More...

#define AI_CONFIG_PP_SLM_VERTEX_LIMIT "PP_SLM_VERTEX_LIMIT"
Set the maximum number of vertices in a mesh. More...

#define AI_CONFIG_PP_TUV_EVALUATE "PP_TUV_EVALUATE"
Input parameter to the aiProcess_TransformUVCoords step: Specifies w

#define AI_DEBONE_THRESHOLD 1.0f

#define AI_LMW_MAX_WEIGHTS 0x4

#define AI_SBBC_DEFAULT_MAX_BONES 60

#define AI_SLM_DEFAULT_MAX_TRIANGLES 1000000

#define AI_SLM_DEFAULT_MAX_VERTICES 1000000
#define AI_UVTRAFO_ALL (AI_UVTRAFO_SCALING | AI_UVTRAFO_ROTATION)

#define AI_UVTRAFO_ROTATION 0x2

#define AI_UVTRAFO_SCALING 0x1

#define AI_UVTRAFO_TRANSLATION 0x4

#define aiComponent_COLORSn(n) (1u << (n+20u))

#define aiComponent_TEXCOORDSn(n) (1u << (n+25u))

#define PP_ICL_PTCACHE_SIZE 12

Default value for the AI_CONFIG_PP_ICL_PTCACHE_SIZE property.
Enumerations
Detailed Description

Defines constants for configurable properties for the library.

Typically these properties are set via `Assimp::Importer::SetPropertyFloat`, `Assimp::Importer::SetPropertyInteger` or `Assimp::Importer::SetPropertyString`, depending on the data type of a property. All properties have a default value. See the doc for the mentioned methods for more details.

The corresponding functions for use with the plain-c API are: `aiSetImportPropertyInteger`, `aiSetImportPropertyFloat`, `aiSetImportPropertyString`
Macro Definition Documentation

#define AI_CONFIG_ANDROID_JNI_ASSIMP_MANAGER_SUPPORT "AI_CONFIG_ANDROID_JNI_ASSIMP_MANAGER_SUPPORT"

Specifies whether the Android JNI asset extraction is supported.

Turn on this option if you want to manage assets in native Android application without having to keep the internal directory and asset manager pointer.

#define AI_CONFIG_EXPORT_XFILE_64BIT "EXPORT_XFILE_64BIT"

Specifies the xfile use double for real values of float.

Property type: Bool. Default value: false.

#define AI_CONFIG_FAVOUR_SPEED "FAVOUR_SPEED"

A hint to assimp to favour speed against import quality.

Enabling this option may result in faster loading, but it needn't. It represents just a hint to loaders and post-processing steps to use faster code paths, if possible. This property is expected to be an integer, != 0 stands for true. The default value is 0.

#define AI_CONFIG_GLOB_MEASURE_TIME "GLOB_MEASURE_TIME"

Enables time measurements.

If enabled, measures the time needed for each part of the loading process (i.e. IO time, importing, postprocessing, ..) and dumps these timings to the DefaultLogger. See the Performance Page for more information on this topic.

Property type: bool. Default value: false.

#define AI_CONFIG_IMPORT_AC_EVAL_SUBDIVISION "IMPORT_AC_EVAL_SUBDIVISION"
Configures whether the AC loader evaluates subdivision surfaces (indicated by the presence of the 'subdiv' attribute in the file).

By default, Assimp performs the subdivision using the standard Catmull-Clark algorithm

- Property type: bool. Default value: true.

#define AI_CONFIG_IMPORT_AC_SEPARATE_BFCULL "IMPORT_AC_SEPARATE_BFCULL"

Configures the AC loader to collect all surfaces which have the "Backface cull" flag set in separate meshes.

Property type: bool. Default value: true.

#define AI_CONFIG_IMPORT_ASE_RECONSTRUCT_NORMALS "IMPORT_ASE_RECONSTRUCT_NORMALS"

Configures the ASE loader to always reconstruct normal vectors basing on the smoothing groups loaded from the file.

Some ASE files have carry invalid normals, other don't.

- Property type: bool. Default value: true.

#define AI_CONFIG_IMPORT_COLLADA_IGNORE_UP_DIRECTION "IMPORT_COLLADA_IGNORE_UP_DIRECTION"

Specifies whether the Collada loader will ignore the provided up direction.

If this property is set to true, the up direction provided in the file header will be ignored and the file will be loaded as is. Property type: Bool. Default value: false.

#define AI_CONFIG_IMPORT_FBX_OPTIMIZE_EMPTY_ANIMATION_CURVES

Specifies whether the importer will drop empty animation curves or animation
curves which match the bind pose transformation over their entire defined range.

The default value is true (1) Property type: bool

#define  
AI_CONFIG_IMPORT_FBX_PRESERVE_PIVOTS "IMPORT_FBX_PRESERVE_PIVOTS"

Set whether the fbx importer will preserve pivot points for transformations (as extra nodes).

If set to false, pivots and offsets will be evaluated whenever possible.

The default value is true (1) Property type: bool

#define  
AI_CONFIG_IMPORT_FBX_READ_ALL_GEOMETRY LAYERS "IMPORT_FBX_READ_ALL_GEOMETRY_LAYERS"

Set whether the fbx importer will merge all geometry layers present in the source file or take only the first.

The default value is true (1) Property type: bool

#define  
AI_CONFIG_IMPORT_FBX_READ_ALL_MATERIALS "IMPORT_FBX_READ_ALL_MATERIALS"

Set whether the fbx importer will read all materials present in the source file or take only the referenced materials.

This is void unless IMPORT_FBX_READ_MATERIALS=1.

The default value is false (0) Property type: bool

#define  
AI_CONFIG_IMPORT_FBX_READ_ANIMATIONS "IMPORT_FBX_READ_ANIMATIONS"

Set whether the fbx importer will read animations.

The default value is true (1) Property type: bool

#define
AI_CONFIG_IMPORT_FBX_READ_CAMERAS "IMPORT_FBX_READ_CAMERAS"

Set whether the fbx importer will read cameras.
The default value is true (1) Property type: bool

#define
AI_CONFIG_IMPORT_FBX_READ_LIGHTS "IMPORT_FBX_READ_LIGHTS"

Set whether the fbx importer will read light sources.
The default value is true (1) Property type: bool

#define
AI_CONFIG_IMPORT_FBX_READ_MATERIALS "IMPORT_FBX_READ_MATERIALS"

Set whether the fbx importer will read materials.
The default value is true (1) Property type: bool

#define
AI_CONFIG_IMPORT_FBX_READ_TEXTURES "IMPORT_FBX_READ_TEXTURES"

Set whether the fbx importer will read embedded textures.
The default value is true (1) Property type: bool

#define
AI_CONFIG_IMPORT_FBX_STRICT_MODE "IMPORT_FBX_STRICT_MODE"

Set whether the fbx importer will act in strict mode in which only FBX 2013 is supported and any other sub formats are rejected.

FBX 2013 is the primary target for the importer, so this format is best supported and well-tested.
The default value is false (0) Property type: bool

#define
AI_CONFIG_IMPORT_GLOBAL_KEYFRAME "IMPORT_GLOBAL_KEYFRAME"

Set the vertex animation keyframe to be imported.

ASSIMP does not support vertex keyframes (only bone animation is supported). The library reads only one frame of models with vertex animations. By default this is the first frame.

Note
The default value is 0. This option applies to all importers. However, it is also possible to override the global setting for a specific loader. You can use the AI_CONFIG_IMPORT_XXX_KEYFRAME options (where XXX is a placeholder for the file format for which you want to override the global setting). Property type: integer.

#define AI_CONFIG_IMPORT_IFC_CUSTOM_TRIANGULATION "IMPORT_IFC_CUSTOM_TRIANGULATION"

Specifies whether the IFC loader will use its own, custom triangulation algorithm to triangulate wall and floor meshes.

If this property is set to false, walls will be either triangulated by aiProcess_Triangulate or will be passed through as huge polygons with faked holes (i.e. holes that are connected with the outer boundary using a dummy edge). It is highly recommended to set this property to true if you want triangulated data because aiProcess_Triangulate is known to have problems with the kind of polygons that the IFC loader spits out for complicated meshes. Property type: Bool. Default value: true.

#define AI_CONFIG_IMPORT_IFC_SKIP_CURVE_REPRESENTATIONS "IMPORT_IFC_SKIP_CURVE_REPRESENTATIONS"

Specifies whether the IFC loader skips over shape representations of type 'Curve2D'.

A lot of files contain both a faceted mesh representation and an outline with a presentation type of 'Curve2D'. Currently Assimp doesn't convert those, so turning this option off just clutters the log with errors. Property type: Bool. Default value: true.
#define AI_CONFIG_IMPORT_IFC_SKIP_SPACE_REPRESENTATIONS "IMPORT_IFC_SKIP_SPACE_REPRESENTATIONS"
Specifies whether the IFC loader skips over IfcSpace elements.

IfcSpace elements (and their geometric representations) are used to represent, well, free space in a building storey.
Property type: Bool. Default value: true.

#define AI_CONFIG_IMPORT_IRR_ANIM_FPS "IMPORT_IRR_ANIM_FPS"
Defines the output frame rate of the IRR loader.

IRR animations are difficult to convert for Assimp and there will always be a loss of quality. This setting defines how many keys per second are returned by the converter.
Property type: integer. Default value: 100

#define AI_CONFIG_IMPORT_LWO_ONE_LAYER_ONLY "IMPORT_LWO_ONE_LAYER_ONLY"
Configures the LWO loader to load just one layer from the model.

LWO files consist of layers and in some cases it could be useful to load only one of them. This property can be either a string - which specifies the name of the layer - or an integer - the index of the layer. If the property is not set the whole LWO model is loaded. Loading fails if the requested layer is not available. The layer index is zero-based and the layer name may not be empty.
Property type: Integer. Default value: all layers are loaded.

#define AI_CONFIG_IMPORT_LWS_ANIM_END "IMPORT_LWS_ANIM_END"
#define AI_CONFIG_IMPORT_LWS_ANIM_START "IMPORT_LWS_ANIM_START"
Defines the begin of the time range for which the LWS loader evaluates animations and computes aiNodeAnim's.
**Assimp** provides full conversion of LightWave's envelope system, including pre and post conditions. The loader computes linearly subsampled animation channels with the frame rate given in the LWS file. This property defines the start time. Note: animation channels are only generated if a node has at least one envelope with more than one key assigned. This property is given in frames, '0' is the first frame. By default, if this property is not set, the importer takes the animation start from the input LWS file ('FirstFrame' line)


See also

**AI_CONFIG_IMPORT_LWS_ANIM_END** - end of the imported time range

```c
#define AI_CONFIG_IMPORT_MD2_KEYFRAME "IMPORT_MD2_KEYFRAME"
#define AI_CONFIG_IMPORT_MD3_HANDLE_MULTIPART "IMPORT_MD3_HANDLER_MULTIPART"
```

Configures the M3D loader to detect and process multi-part Quake player models.

These models usually consist of 3 files, lower.md3, upper.md3 and head.md3. If this property is set to true, **Assimp** will try to load and combine all three files if one of them is loaded. Property type: bool. Default value: true.

```c
#define AI_CONFIG_IMPORT_MD3_KEYFRAME "IMPORT_MD3_KEYFRAME"
#define AI_CONFIG_IMPORT_MD3_SHADER_SRC "IMPORT_MD3_SHADER_SRC"
```

Specify the Quake 3 shader file to be used for a particular MD3 file.

This can also be a search path.

By default **Assimp**'s behaviour is as follows: If a MD3 file `any_path/models/any_q3_subdir/model_name/file_name.md3` is loaded, the library tries to locate the corresponding shader file in `any_path/scripts/model_name.shader`. This property overrides this behaviour. It can either specify a full path to the shader to be loaded or alternatively the path
(relative or absolute) to the directory where the shaders for all MD3s to be loaded reside. **Assimp** attempts to open
IMPORT_MD3_SHADER_SRC/model_name.shader first, IMPORT_MD3_SHADER_SRC/file_name.shader is the fallback file. Note that IMPORT_MD3_shader_SRC should have a terminal (back)slash. Property type: String. Default value: n/a.

```c
#define AI_CONFIG_IMPORT_MD3_SKIN_NAME "IMPORT_MD3_SKIN_NAME"
```

Tells the MD3 loader which skin files to load.

When loading MD3 files, **Assimp** checks whether a file [md3_file_name]_[skin_name].skin is existing. These files are used by Quake III to be able to assign different skins (e.g. red and blue team) to models. 'default', 'red', 'blue' are typical skin names. Property type: String. Default value: "default".

```c
#define AI_CONFIG_IMPORT_MD5_NO_ANIM_AUTOLOAD "IMPORT_MD5_NO_ANIM_AUTOLOAD"
```

Configures the MD5 loader to not load the MD5ANIM file for a MD5MESH file automatically.

The default strategy is to look for a file with the same name but the MD5ANIM extension in the same directory. If it is found, it is loaded and combined with the MD5MESH file. This configuration option can be used to disable this behaviour.

- Property type: bool. Default value: false.

```c
#define AI_CONFIG_IMPORT_MDC_KEYFRAME "IMPORT_MDC_KEYFRAME"
#define AI_CONFIG_IMPORT_MDLO_COLORMAP "IMPORT_MDLO_COLORMAP"
```

Sets the colormap (= palette) to be used to decode embedded textures in MDL (Quake or 3DGS) files.

This must be a valid path to a file. The file is 768 (256*3) bytes large and
contains RGB triplets for each of the 256 palette entries. The default value is colormap.lmp. If the file is not found, a default palette (from Quake 1) is used. Property type: string.

#define AI_CONFIG_IMPORT_MDL_KEYFRAME "IMPORT_MDL_KEYFRAME"
#define AI_CONFIG_IMPORT_NO_SKELETON_MESHES "IMPORT_NO_SKELETON_MESHES"

Global setting to disable generation of skeleton dummy meshes.

Skeleton dummy meshes are generated as a visualization aid in cases which the input data contains no geometry, but only animation data. Property data type: bool. Default value: false

#define AI_CONFIG_IMPORT_OGRE_MATERIAL_FILE "IMPORT_OGRE_MATERIAL_FILE"

Ogre Importer will try to find referenced materials from this file.

Ogre meshes reference with material names, this does not tell Assimp the file where it is located in. Assimp will try to find the source file in the following order: <material-name>.material, <mesh-filename-base>.material and lastly the material name defined by this config property. Property type: String. Default value: Scene.material.

#define AI_CONFIG_IMPORT_OGRE_TEXTURETYPE_FROM_FILENAME "IMPORT_OGRE_TEXTURETYPE_FROM_FILENAME"

Ogre Importer detect the texture usage from its filename.

Ogre material texture units do not define texture type, the textures usage depends on the used shader or Ogre's fixed pipeline. If this config property is true Assimp will try to detect the type from the textures filename postfix: _n, _nrm, _nrml, _normal, _normals and _normalmap for normal map, _s, _spec, _specular and _specularmap for specular map, _l, _light, _lightmap, _occ and _occlusion for light map, _disp and _displacement for displacement map. The matching is case insensitive. Postfix is taken between the last underscore and the last period. Default behavior is to detect type from lower cased texture unit name by
matching against: normalmap, specularmap, lightmap and displacementmap. For both cases if no match is found aiTextureType_DIFFUSE is used. Property type: Bool. Default value: false.

#define 
AI_CONFIG_IMPORT_SMD_KEYFRAME "IMPORT_SMD_KEYFRAME"
#define 
AI_CONFIG_IMPORT_TER_MAKE_UVS "IMPORT_TER_MAKE_UVS"

Configures the terragen import plugin to compute uv's for terrains, if not given. Furthermore a default texture is assigned.

UV coordinates for terrains are so simple to compute that you'll usually want to compute them on your own, if you need them. This option is intended for model viewers which want to offer an easy way to apply textures to terrains.

- Property type: bool. Default value: false.

#define 
AI_CONFIG_IMPORT_UNREAL_HANDLE_FLAGS "UNREAL_HANDLE_FLAGS"
#define 
AI_CONFIG_IMPORT_UNREAL_KEYFRAME "IMPORT_UNREAL_KEYFRAME"
#define 
AI_CONFIG_PP_CT_MAX_SMOOTHING_ANGLE "PP_CT_MAX_SMOOTHING_ANGLE"

Specifies the maximum angle that may be between two vertex tangents that their tangents and bi-tangents are smoothed.

This applies to the CalcTangentSpace-Step. The angle is specified in degrees. The maximum value is 175. Property type: float. Default value: 45 degrees
#define AI_CONFIG_PP_CT_TEXTURE_CHANNEL_INDEX "PP_CT_TEXTURE_CHANNEL_INDEX"

Source UV channel for tangent space computation.

The specified channel must exist or an error will be raised. Property type: integer. Default value: 0

#define AI_CONFIG_PP_DB_ALL_OR_NONE "PP_DB_ALL_OR_NONE"

Require all bones qualify for deboning before removing any.

This is used by the aiProcess_Debone PostProcess-Step.

Note

The default value is 0 Property type: bool.

#define AI_CONFIG_PP_DB_THRESHOLD "PP_DB_THRESHOLD"

Lower the deboning threshold in order to remove more bones.

This is used by the aiProcess_Debone PostProcess-Step.

Note

The default value is AI_DEBONE_THRESHOLD Property type: float.

#define AI_CONFIG_PP_FD_REMOVE "PP_FD_REMOVE"

Configures the aiProcess_FindDegenerates step to remove degenerated primitives from the import - immediately.

The default behaviour converts degenerated triangles to lines and degenerated lines to points. See the documentation to the aiProcess_FindDegenerates step for a detailed example of the various ways to get rid of these lines and points if you don't want them. Property type: bool. Default value: false.

#define AI_CONFIG_PP_FID_ANIM_ACCURACY "PP_FID_ANIM_ACCURACY"
Input parameter to the \texttt{aiProcess\_FindInvalidData} step: Specifies the floating-point accuracy for animation values.

The step checks for animation tracks where all frame values are absolutely equal and removes them. This tweakable controls the epsilon for floating-point comparisons - two keys are considered equal if the invariant $\text{abs}(n0-n1) > \text{epsilon}$ holds true for all vector respectively quaternion components. The default value is 0.f - comparisons are exact then.

\begin{verbatim}
#define AI_CONFIG_PP_GSN_MAX_SMOOTHING_ANGLE "PP_GSN_MAX_SMOOTHING_ANGLE"
\end{verbatim}

Specifies the maximum angle that may be between two face normals at the same vertex position that their are smoothed together.

Sometimes referred to as 'crease angle'. This applies to the GenSmoothNormals-Step. The angle is specified in degrees, so 180 is PI. The default value is 175 degrees (all vertex normals are smoothed). The maximum value is 175, too. Property type: float. Warning: setting this option may cause a severe loss of performance. The performance is unaffected if the \texttt{AI\_CONFIG\_FAVOUR\_SPEED} flag is set but the output quality may be reduced.

\begin{verbatim}
#define AI_CONFIG_PP_ICL_PTCACHE_SIZE "PP_ICL_PTCACHE_SIZE"
\end{verbatim}

Set the size of the post-transform vertex cache to optimize the vertices for.

This configures the \texttt{aiProcess\_ImproveCacheLocality} step.

The size is given in vertices. Of course you can't know how the vertex format will exactly look like after the import returns, but you can still guess what your meshes will probably have.

\textbf{Note}

The default value is \texttt{PP\_ICL\_PTCACHE\_SIZE}. That results in slight performance improvements for most nVidia/AMD cards since 2002. Property type: integer.
#define AI_CONFIG_PP_LBW_MAX_WEIGHTS "PP_LBW_MAX_WEIGHTS"

Set the maximum number of bones affecting a single vertex.

This is used by the aiProcess_LimitBoneWeights PostProcess-Step.

Note
The default value is AI_LBW_MAX_WEIGHTS Property type: integer.

#define AI_CONFIG_PP_OG_EXCLUDE_LIST "PP_OG_EXCLUDE_LIST"

Configures the aiProcess_OptimizeGraph step to preserve nodes matching a name in a given list.

This is a list of 1 to n strings, '' serves as delimiter character. Identifiers containing whitespaces must be enclosed in single quotation marks. For example: "keep-me and_me_to anotherNodeToBeKept \'name with whitespace\'". If a node matches on of these names, it will not be modified or removed by the postprocessing step.

This option might be useful if you are using some magic node names to pass additional semantics through the content pipeline. This ensures they won't be optimized away, but a general optimization is still performed for nodes not contained in the list. Property type: String. Default value: n/a

Note
Linefeeds, tabs or carriage returns are treated as whitespace. Node names are case sensitive.

#define AI_CONFIG_PP_PTV_ADD_ROOT_TRANSFORMATION "PP_PTV_ADD_ROOT_TRANSFORMATION"

Configures the aiProcess_PreTransformVertices step to use a users defined matrix as the scene root node transformation before transforming vertices.

Property type: bool. Default value: false.

#define
AI_CONFIG_PP_PTV_KEEP_HIERARCHY "PP_PTV_KEEP_HIERARCHY"

Configures the aiProcess_PreTransformVertices step to keep the scene hierarchy.

Meshes are moved to worldspace, but no optimization is performed (read: meshes with equal materials are not joined. The total number of meshes won't change).

This option could be of use for you if the scene hierarchy contains important additional information which you intend to parse. For rendering, you can still render all meshes in the scene without any transformations. Property type: bool. Default value: false.

#define AI_CONFIG_PP_PTV_NORMALIZE "PP_PTV_NORMALIZE"

Configures the aiProcess_PreTransformVertices step to normalize all vertex components into the [-1,1] range.

That is, a bounding box for the whole scene is computed, the maximum component is taken and all meshes are scaled appropriately (uniformly of course!). This might be useful if you don't know the spatial dimension of the input data

#define AI_CONFIG_PP_PTV_ROOT_TRANSFORMATION "PP_PTV_ROOT_TRANSFORMATION"

Configures the aiProcess_PreTransformVertices step to use a users defined matrix as the scene root node transformation before transforming vertices.

This property correspond to the 'a1' component of the transformation matrix. Property type: aiMatrix4x4.

#define AI_CONFIG_PP_RRM_EXCLUDE_LIST "PP_RRM_EXCLUDE_LIST"

Configures the aiProcess_RemoveRedundantMaterials step to keep materials matching a name in a given list.

This is a list of 1 to n strings, '"' serves as delimiter character. Identifiers
containing whitespaces must be enclosed in *single* quotation marks. For example: "keep-me and_me_to anotherMaterialToBeKept \'name with whitespace\'". If a material matches on of these names, it will not be modified or removed by the postprocessing step nor will other materials be replaced by a reference to it.
This option might be useful if you are using some magic material names to pass additional semantics through the content pipeline. This ensures they won't be optimized away, but a general optimization is still performed for materials not contained in the list. Property type: String. Default value: n/a

Note
Linefeeds, tabs or carriage returns are treated as whitespace. Material names are case sensitive.

#define AI_CONFIG_PP_RVC_FLAGS "PP_RVC_FLAGS"

Input parameter to the *aiProcess_RemoveComponent* step: Specifies the parts of the data structure to be removed.

See the documentation to this step for further details. The property is expected to be an integer, a bitwise combination of the *aiComponent* flags defined above in this header. The default value is 0. Important: if no valid mesh is remaining after the step has been executed (e.g you thought it was funny to specify ALL of the flags defined above) the import FAILS. Mainly because there is no data to work on anymore ...

#define AI_CONFIG_PP_SBBC_MAX_BONES "PP_SBBC_MAX_BONES"

Maximum bone count per mesh for the SplitbyBoneCount step.

Meshes are split until the maximum number of bones is reached. The default value is *AI_SBBC_DEFAULT_MAX_BONES*, which may be altered at compile-time. Property data type: integer.

#define AI_CONFIG_PP_SBP_REMOVE "PP_SBP_REMOVE"

Input parameter to the *aiProcess_SortByPType* step: Specifies which primitive types are removed by the step.
This is a bitwise combination of the aiPrimitiveType flags. Specifying all of them is illegal, of course. A typical use would be to exclude all line and point meshes from the import. This is an integer property, its default value is 0.

```c
#define AI_CONFIG_PP_SLM_TRIANGLE_LIMIT "PP_SLM_TRIANGLE_LIMIT"
```

Set the maximum number of triangles in a mesh.

This is used by the "SplitLargeMeshes" PostProcess-Step to determine whether a mesh must be split or not.

Note
The default value is AI_SLM_DEFAULT_MAX_TRIANGLES Property
type: integer.

```c
#define AI_CONFIG_PP_SLM_VERTEX_LIMIT "PP_SLM_VERTEX_LIMIT"
```

Set the maximum number of vertices in a mesh.

This is used by the "SplitLargeMeshes" PostProcess-Step to determine whether a mesh must be split or not.

Note
The default value is AI_SLM_DEFAULT_MAX_VERTICES Property
type: integer.

```c
#define AI_CONFIG_PP_TUV_Evaluate "PP_TUV_Evaluate"
```

Input parameter to the aiProcess_TransformUVCoords step: Specifies which UV transformations are evaluated.

This is a bitwise combination of the AI_UVTRAFO_XXX flags (integer property, of course). By default all transformations are enabled (AI_UVTRAFO_ALL).

```c
#define AI_DEBONE_THRESHOLD 1.0f
#define AI_LMW_MAX_WEIGHTS 0x4
```
#define AI_SBBC_DEFAULT_MAX_BONES 60
#define AI_SLM_DEFAULT_MAX_TRIANGLES 1000000
#define AI_SLM_DEFAULT_MAX_VERTICES 1000000
#define AI_UVTRAFO_ALL (AI_UVTRAFO_SCALING | AI_UVTRAFO_ROTATION | AI_UVTRAFO_TRANSLATION)
#define AI_UVTRAFO_ROTATION 0x2
#define AI_UVTRAFO_SCALING 0x1
#define AI_UVTRAFO_TRANSLATION 0x4
#define aiComponent_COLORSn (n) (1u << (n+20u))
#define aiComponent_TEXCOORDSn (n) (1u << (n+25u))
#define PP_ICL_PTCACHE_SIZE 12

Default value for the AI_CONFIG_PP_ICL_PTCACHE_SIZE property.
# Enumeration Type Documentation

```c
enum aiComponent
```

Enumerates components of the `aiScene` and `aiMesh` data structures that can be excluded from the import using the `aiProcess_RemoveComponent` step.

See the documentation to `aiProcess_RemoveComponent` for more details.

## Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>aiComponent_NORMALS</code></td>
<td>Normal vectors.</td>
</tr>
<tr>
<td><code>aiComponent_TANGENTS_AND_BITANGENTS</code></td>
<td>Tangents and bitangents go always together ...</td>
</tr>
<tr>
<td><code>aiComponent_COLORS</code></td>
<td>ALL color sets Use <code>aiComponent_COLORn(N)</code> to specify the N'th set.</td>
</tr>
<tr>
<td><code>aiComponent_TEXCOORDS</code></td>
<td>ALL texture UV sets <code>aiComponent_TEXCOORDn(N)</code> to specify the N'th set.</td>
</tr>
<tr>
<td><code>aiComponent_BONEWEIGHTS</code></td>
<td>Removes all bone weights from all meshes.</td>
</tr>
<tr>
<td></td>
<td>The scenegraph nodes corresponding to the bones are not removed. Use the <code>aiProcess_OptimizeGraph</code> step to do this</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>aiComponent_ANIMATIONS</td>
<td>Removes all node animation: (aiScene::mAnimations). The corresponding scenegraph nodes are NOT removed. Use aiProcess_OptimizeGraph step to do this.</td>
</tr>
<tr>
<td>aiComponent_TEXTURES</td>
<td>Removes all embedded textures (aiScene::mTextures).</td>
</tr>
<tr>
<td>aiComponent_LIGHTS</td>
<td>Removes all light sources (aiScene::mLights).</td>
</tr>
<tr>
<td>aiComponent_CAMERAS</td>
<td>Removes all cameras (aiScene::mCameras).</td>
</tr>
<tr>
<td>aiComponent_MESHES</td>
<td>Removes all meshes (aiScene::mMeshes).</td>
</tr>
<tr>
<td>aiComponent_MATERIALS</td>
<td>Removes all materials. One default material will be generated, so aiScene::mNumMaterials will</td>
</tr>
</tbody>
</table>
This value is not used.

_aiComponent_Force32Bit

It is just there to force the compiler to map this enum to 32 Bit integer.
Assimp v3.1.1 (June 2014)

- include
- assimp

Classes | Namespaces | Macros
DefaultLogger.hpp File Reference
Classes

class Assimp::DefaultLogger
CPP-API: Primary logging facility of Assimp. More...
Namespaces

Assimp
Assimp's CPP-API and all internal APIs.
Macros

#define ASSIMP_DEFAULT_LOG_NAME "AssimpLog.txt"
default name of logfile More...
Macro Definition Documentation

#define ASSIMP_DEFAULT_LOG_NAME "AssimpLog.txt"

default name of logfile
Assimp v3.1.1 (June 2014)

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Namespaces</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td>File Members</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- include
- assimp

**Macros**
defs.h File Reference

**Assimp** build configuration setup. [More...](#)
Macros

#define AI_DEG_TO_RAD(x) ((x)*0.0174532925f)
#define AI_FORCE_INLINE inline
#define AI_MATH_HALF_PI (AI_MATH_PI * 0.5)
#define AI_MATH_HALF_PI_F (AI_MATH_PI_F * 0.5f)
#define AI_MATH_PI (3.141592653589793238462643383279 )
#define AI_MATH_PI_F (3.1415926538f)
#define AI_MATH_TWO_PI (AI_MATH_PI * 2.0)
#define AI_MATH_TWO_PI_F (AI_MATH_PI_F * 2.0f)
#define AI_MAX_ALLOC(type) ((256U * 1024 * 1024) / sizeof(type))
#define AI_RAD_TO_DEG(x) ((x)*57.2957795f)
#define AI_WONT_RETURN
#define AI_WONT_RETURN_SUFFIX
#define ASSIMP_API __attribute__((visibility("default")))
#define ASSIMP_API_WINONLY
#define ASSIMP_BUILD_DEBUG
#define ASSIMP_BUILD_NEED_UNZIP

#define ASSIMP_BUILD_NEED_UNZIP

#define ASSIMP_BUILD_NEED_Z_INFLATE

#define ASSIMP_BUILD_NEED_Z_INFLATE

#define ASSIMP_BUILD_NEED_Z_INFLATE

#define ASSIMP_BUILD_NEED_Z_INFLATE

#define ASSIMP_BUILD_SINGLETHREADED

#define C_ENUM

#define C_STRUCT
Detailed Description

Assimp build configuration setup.

See the notes in the comment blocks to find out how to customize your Assimp build.
Macro Definition Documentation

#define AI_DEG_TO_RAD ( x ) ((x)*0.0174532925f)
#define AI_FORCE_INLINE inline
#define AI_MATH_HALF_PI (AI_MATH_PI * 0.5)
#define AI_MATH_HALF_PI_F (AI_MATH_PI_F * 0.5f)
#define AI_MATH_PI (3.141592653589793238462643383279 )
#define AI_MATH_PI_F (3.1415926538f)
#define AI_MATH_TWO_PI (AI_MATH_PI * 2.0)
#define AI_MATH_TWO_PI_F (AI_MATH_PI_F * 2.0f)
#define AI_MAX_ALLOC (type) ((256U * 1024 * 1024) / sizeof(type))
#define AI_RAD_TO_DEG ( x ) ((x)*57.2957795f)
#define AI_WONT_RETURN
#define AI_WONT_RETURN_SUFFIX
#define ASSIMP_API __attribute__((visibility("default")))
#define ASSIMP_API_WINONLY
#define ASSIMP_BUILD_DEBUG
#define ASSIMP_BUILD_NEED_UNZIP
#define ASSIMP_BUILD_NEED_UNZIP
#define ASSIMP_BUILD_NEED_Z_INFLATE
#define ASSIMP_BUILD_NEED_Z_INFLATE
#define ASSIMP_BUILD_NEED_Z_INFLATE
#define ASSIMP_BUILD_NEED_Z_INFLATE
#define ASSIMP_BUILD_SINGLETHREADED
#define C_ENUM
#define C_STRUCT
dox.h File Reference

General documentation built from a doxygen comment. More...
Detailed Description

General documentation built from a doxygen comment.
Exporter.hpp File Reference

Defines the CPP-API for the Assimp export interface. More...
Classes

class  Assimp::Exporter

struct  Assimp::Exporter::ExportFormatEntry
        Internal description of an Assimp export format option. More...

class  Assimp::ExportProperties
Namespaces

Assimp
Assimp's CPP-API and all internal APIs.
Variables

class ASSIMP_API Assimp::ExportProperties

CPP-API: The Exporter class forms an C++ interface to the export functionality of the Open Asset Import Library. More...
Detailed Description

Defines the CPP-API for the Assimp export interface.
Main Page | Related Pages | Namespaces | Classes | Files
---|---|---|---|---
File List | File Members

- include
- assimp

Classes | Namespaces | Macros
Importer.hpp File Reference

Defines the C++-API to the Open Asset Import Library. More...
Classes

class Assimp::Importer

CPP-API: The Importer class forms an C++ interface to the functionality of the Open Asset Import Library. More...
Namespaces

**Assimp**

**Assimp**'s CPP-API and all internal APIs.
Macros

#define AI_PROPERTY_WAS_NOT_EXISTING 0xffffffff
namespace Assimp More...
Detailed Description

Defines the C++-API to the Open Asset Import Library.
Macro Definition Documentation

#define AI_PROPERTY_WAS_NOT_EXISTING  0xffffffff

namespace Assimp

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Main Page | Related Pages | Namespaces | Classes | Files
---|---|---|---|---
File List | File Members |  

- **include**
- **assimp**

Classes | Enumerations | Functions

importerdesc.h File Reference

**aiImporterFlags, aiImporterDesc** implementation. [More...](#)
Classes

struct aiImporterDesc

Meta information about a particular importer. More...
Enumerations
**Functions**

```c
ASSIMP_API const C_STRUCT aiImporterDesc * aiGetImporterDesc (const char *extension)
```

Returns the Importer description for a given extension. [More...](#)
Detailed Description

aiImporterFlags, aiImporterDesc implementation.
### Enumeration Type Documentation

enum **aiImporterFlags**

Mixed set of flags for **aiImporterDesc**, indicating some features common to many importers.

#### Enumerator

<table>
<thead>
<tr>
<th>aiImporterFlags_SupportTextFlavour</th>
<th>Indicates that there is a textual encoding of the file format; and that it is supported.</th>
</tr>
</thead>
<tbody>
<tr>
<td>aiImporterFlags_SupportBinaryFlavour</td>
<td>Indicates that there is a binary encoding of the file format; and that it is supported.</td>
</tr>
<tr>
<td>aiImporterFlags_SupportCompressedFlavour</td>
<td>Indicates that there is a compressed encoding of the file format; and that it is supported.</td>
</tr>
<tr>
<td>aiImporterFlags_LimitedSupport</td>
<td>Indicates that the importer reads only a very particular subset of the file format.</td>
</tr>
<tr>
<td>aiImporterFlags_Experimental</td>
<td>This happens commonly for declarative or procedural formats which cannot easily be mapped to <strong>aiScene</strong></td>
</tr>
<tr>
<td>aiImporterFlags_Experimental</td>
<td>Indicates that the importer is highly experimental and should be used with care.</td>
</tr>
</tbody>
</table>
This only happens for trunk (i.e. SVN) versions, experimental code is not included in releases.
Function Documentation

ASSIMP_API const C_STRUCT aiImporterDesc* aiGetImporterDesc(const char * extension)

Returns the Importer description for a given extension.
Will return a NULL-pointer if no assigned importer desc. was found for the
given extension

Parameters
extension [in] The extension to look for

Returns
A pointer showing to the ImporterDesc,

See also
aiImporterDesc.
Assimp v3.1.1 (June 2014)

- include
- assimp

Classes | Namespaces
IOStream.hpp File Reference

File I/O wrappers for C++. More...
Classes

class Assimp::IOStream
CPP-API: Class to handle file I/O for C++. More...
Namespaces

Assimp
Assimp's CPP-API and all internal APIs.
Detailed Description

File I/O wrappers for C++.
File system wrapper for C++.  

**More...**
Classes

class Assimp::IOSystem

CPP-API: Interface to the file system. More...
Namespaces

Assimp
Assimp's CPP-API and all internal APIs.
Detailed Description

File system wrapper for C++.

Inherit this class to supply custom file handling logic to the Import library.

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp v3.1.1 (June 2014)

- include
- assimp

Classes | Enumerations

light.h File Reference

Defines the aiLight data structure. More...
Classes

struct aiLight

Helper structure to describe a light source. More...
Enumerations
Detailed Description

Defines the aiLight data structure.
**Enumeration Type Documentation**

enum **aiLightSourceType**

Enumerates all supported types of light sources.

**Enumerator**

**aiLightSource_UNDEFINED**

A directional light source has a well-defined direction but is infinitely far away.

**aiLightSource_DIRECTIONAL**

That's quite a good approximation for sunlight.

**aiLightSource_POINT**

A point light source has a well-defined position in space but no direction - it emits light in all directions.

A normal bulb is a point light.

**aiLightSource_SPOT**

A spot light source emits light in a specific angle.

It has a position and a direction it is pointing to. A good example for a spot light is a light spot in sport arenas.

**aiLightSource_AMBIENT**

The generic light level of the world, including the bounces of all other lightsources.

Typically, there's at most one ambient light in a scene. This light type doesn't have a valid position, direction, or other properties, just a color.
This value is not used.

_aiLightSource_Force32Bit

It is just there to force the compiler to map this enum to a 32 Bit integer.
Abstract base class 'Logger', base of the logging system. More...
Classes

class Assimp::Logger

CPP-API: Abstract interface for logger implementations. More...
Namespaces

Assimp

Assimp's CPP-API and all internal APIs.
Macros

#define MAX_LOG_MESSAGE_LENGTH 1024u
Detailed Description

Abstract base class 'Logger', base of the logging system.
Macro Definition Documentation

#define MAX_LOG_MESSAGE_LENGTH 1024u

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
### Assimp v3.1.1 (June 2014)

- [include](#)
- [assimp](#)

**Classes** | **Namespaces**
---|---
LogStream.hpp File Reference

Abstract base class 'LogStream', representing an output log stream. [More...](#)
Classes

class Assimp::LogStream

CPP-API: Abstract interface for log stream implementations. More...
Namespaces

Assimp

Assimp's CPP-API and all internal APIs.
Detailed Description

Abstract base class 'LogStream', representing an output log stream.
defines the material system of the library. More...
Classes

struct aiMaterial
Data structure for a material. More...

struct aiUVTransform
Defines how an UV channel is transformed. More...
Macros

#define AI_DEFAULT_MATERIAL_NAME "DefaultMaterial"
#define AI_MATKEY_BLEND_FUNC "$mat.blend",0,0
#define AI_MATKEY_BUMPSCALING "$mat.bumpscaling",0,0
#define AI_MATKEY_COLOR_AMBIENT "$clr.ambient",0,0
#define AI_MATKEY_COLOR_DIFFUSE "$clr.diffuse",0,0
#define AI_MATKEY_COLOR_EMISSIVE "$clr.emissive",0,0
#define AI_MATKEY_COLOR_REFLECTIVE "$clr.reflective",0,0
#define AI_MATKEY_COLOR_SPECULAR "$clr.specular",0,0
#define AI_MATKEY_COLOR_TRANSPARENT "$clr.transparent",0,0
#define AI_MATKEY_ENABLE_WIREFRAME "$mat.wireframe",0,0
#define bg.global",0,0
#define _AI_MATKEY_MAPPING_BASE,type,N
#define _AI_MATKEY_MAPPINGMODE_U_BASE,type,N
#define AI_MATKEY_MAPPING(type, N) _AI_MATKEY_MAPPING_BASE, type, N
#define AI_MATKEY_MAPPINGMODE_U(type, N) _AI_MATKEY_MAPPINGMODE_U_BASE, type, N
#define AI_MATKEY_MAPPINGMODE_V(type,
#define N) _AI_MATKEY_MAPPINGMODE_V_BASE,type,N

#define AI_MATKEY_NAME "mat.name",0,0

#define AI_MATKEY_OPACITY "$mat.opacity",0,0

#define AI_MATKEY_REFLECTIVITY "$mat.reflectivity",0,0

#define AI_MATKEY_REFRACTIVITY "$mat.refracti",0,0

#define AI_MATKEY_SHADING_MODEL "$mat.shadingm",0,0

#define AI_MATKEY_SHININESS "$mat.shininess",0,0

#define AI_MATKEY_SHININESS_STRENGTH "$mat.shinpercent",0,0

#define AI_MATKEY_TEXBLEND(type, N) _AI_MATKEY_TEXBLEND_BASE,type,N

#define AI_MATKEY_TEXFLAGS(type, N) _AI_MATKEY_TEXFLAGS_BASE,type,N

#define AI_MATKEY_TEXMAP_AXIS(type, N) _AI_MATKEY_TEXMAP_AXIS_BASE,type,N

#define AI_MATKEY_TEXOP(type, N) _AI_MATKEY_TEXOP_BASE,type,N

#define AI_MATKEY_TEXTURE(type, N) _AI_MATKEY_TEXTURE_BASE,type,N

#define AI_MATKEY_TWOSIDED "$mat.twosided",0,0
#define AI_MATKEY_UVTRANSFORM(type, N) _AI_MATKEY_UVTRANSFORM_BASE,type,N

#define AI_MATKEY_UVWSRC(type, N) _AI_MATKEY_UVWSRC_BASE,type,N

#define AI_TEXTURE_TYPE_MAX aiTextureType_UNKNOWN
Enumerations
Functions

ASSIMP_API aiReturn aiGetMaterialColor (const aiMaterial *pMat, const char *pKey, unsigned int type, unsigned int index, aiColor4D *pOut)

Retrieve a color property with a specific key from a material. More...

ASSIMP_API aiReturn aiGetMaterialFloat (const aiMaterial *pMat, const char *pKey, unsigned int type, unsigned int index, float *pOut)

Retrieve a single float property with a specific key from the material. More...

ASSIMP_API aiReturn aiGetMaterialFloatArray (const aiMaterial *pMat, const char *pKey, unsigned int type, unsigned int index, float *pOut, unsigned int *pMax)

Retrieve an array of float values with a specific key from the material. More...

ASSIMP_API aiReturn aiGetMaterialInteger (const aiMaterial *pMat, const char *pKey, unsigned int type, unsigned int index, int *pOut)

Retrieve an integer property with a specific key from a material. More...

ASSIMP_API aiReturn aiGetMaterialIntegerArray (const aiMaterial *pMat, const char *pKey, unsigned int type, unsigned int index, int *pOut, unsigned int *pMax)

__cplusplus

ASSIMP_API aiReturn aiGetMaterialProperty (const aiMaterial *pMat, const char *pKey, unsigned int type, unsigned int index, const aiMaterialProperty **pPropOut)

Retrieve a material property with a specific key from the material. More...

ASSIMP_API aiGetMaterialString (const aiMaterial *pMat, const char *pKey,
aiReturn unsigned int type, unsigned int index, aiString *pOut)
Retrieve a string from the material property table. More...

ASSIMP_API aiGetMaterialTexture (const aiMaterial *mat, aiTextureType
type, unsigned int index, aiString *path, aiTextureMapping
*map=NULL, unsigned int *uvindex=NULL, float
*blend=NULL, aiTextureOp *op=NULL, aiTextureMapMode
*mapmode=NULL, unsigned int *flags=NULL)
Helper function to get all values pertaining to a particular texture
slot from a material structure. More...

ASSIMP_API aiGetMaterialTextureCount (const aiMaterial *pMat,
unsigned int
aiTextureType type)
Get the number of textures for a particular texture type. More...

ASSIMP_API aiGetMaterialUVTransform (const aiMaterial *pMat, const char
*pKey, unsigned int type, unsigned int index, aiUVTransform
*pOut)
Retrieve a aiUVTransform value from the material property
table. More...
Variables

struct aiUVTransform PACK_STRUCT
Detailed Description

Defines the material system of the library.
#define AI_DEFAULT_MATERIAL_NAME "DefaultMaterial"
#define AI_MATKEY_BLEND_FUNC "$mat.blend",0,0
#define AI_MATKEY_BUMPSCALING "$mat.bumpscaling",0,0
#define AI_MATKEY_COLOR_AMBIENT "$clr.ambient",0,0
#define AI_MATKEY_COLOR_DIFFUSE "$clr.diffuse",0,0
#define AI_MATKEY_COLOR_EMISSIVE "$clr.emissive",0,0
#define AI_MATKEY_COLOR_REFLECTIVE "$clr.reflective",0,0
#define AI_MATKEY_COLOR_SPECULAR "$clr.specular",0,0
#define AI_MATKEY_COLOR_TRANSPARENT "$clr.transparent",0,0
#define AI_MATKEY_ENABLE_WIREFRAME "$mat.wireframe",0,0
#define AI_MATKEY_GLOBAL_BACKGROUND_IMAGE "?bg.global",0,0
#define AI_MATKEY_NAME "?mat.name",0,0
#define AI_MATKEY_OPACITY "$mat.opacity",0,0
#define AI_MATKEY_REFLECTIVITY "$mat.reflectivity",0,0
#define AI_MATKEY_REFRACTI "$mat.refracti",0,0
#define AI_MATKEY_SHADING_MODEL "$mat.shadingm",0,0
#define AI_MATKEY_SHININESS "$mat.shininess",0,0
#define AI_MATKEY_MAPPING ( type,
                        N
                     ) _AI_MATKEY_MAPPING_BASE,type,N
#define AI_MATKEY_MAPPINGMODE_U ( type,
                               N
                           ) _AI_MATKEY_MAPPINGMODE_U
#define AI_MATKEY_MAPPINGMODE_V ( type,
                               N
                          ) _AI_MATKEY_MAPPINGMODE_V
#define AI_MATKEY_NAME "?mat.name",0,0
#define AI_MATKEY_OPACITY "$mat.opacity",0,0
#define AI_MATKEY_REFLECTIVITY "$mat.reflectivity",0,0
#define AI_MATKEY_REFRACTI "$mat.refracti",0,0
#define AI_MATKEY_SHADING_MODEL "$mat.shadingm",0,0
#define AI_MATKEY_SHININESS "$mat.shininess",0,0
#define AI_MATKEY_SHININESS_STRENGTH "$mat.shinpercent",0,0
#define AI_MATKEY_TEXBLEND ( type, 
    N 
 )  _AI_MATKEY_TEXBLEND_BASE,type,N
#define AI_MATKEY_TEXFLAGS ( type, 
    N 
 )  _AI_MATKEY_TEXFLAGS_BASE,type,N
#define AI_MATKEY_TEXMAP_AXIS ( type, 
    N 
 )  _AI_MATKEY_TEXMAP_AXIS_BASE,type,N
#define AI_MATKEY_TEXOP ( type, 
    N 
 )  _AI_MATKEY_TEXOP_BASE,type,N
#define AI_MATKEY_TEXTURE ( type, 
    N 
 )  _AI_MATKEY_TEXTURE_BASE,type,N
#define AI_MATKEY_TWOSIDED "$mat.twosided",0,0
#define AI_MATKEY_UVTRANSFORM ( type, 
    N 
 )  _AI_MATKEY_UVTRANSFORM_BASE,type,N
#define AI_MATKEY_UVWSRC ( type, 
    N 
 )  _AI_MATKEY_UVWSRC_BASE,type,N
#define AI_TEXTURE_TYPE_MAX aiTextureType_UNKNOWN
Enumeration Type Documentation

enum aiBlendMode

Defines alpha-blend flags.

If you're familiar with OpenGL or D3D, these flags aren't new to you. They define *how* the final color value of a pixel is computed, basing on the previous color at that pixel and the new color value from the material. The blend formula is:

\[ 1 \text{ SourceColor} \times \text{SourceBlend} + \text{DestColor} \times \text{DestBlend} \]

where DestColor is the previous color in the framebuffer at this position and SourceColor is the material color before the transparency calculation. This corresponds to the `AI_MATKEY_BLEND_FUNC` property.

**Enumerator**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>aiBlendMode_Default</td>
<td>[ 1 \text{ SourceColor} \times \text{SourceAlpha} + \text{DestColor} \times (1-\text{SourceAlpha}) ]</td>
</tr>
<tr>
<td>aiBlendMode_Additive</td>
<td>Additive blending.</td>
</tr>
<tr>
<td>_aiBlendMode_Force32Bit</td>
<td>[ 1 \text{ SourceColor} \times 1 + \text{DestColor} \times 1 ]</td>
</tr>
</tbody>
</table>

enum aiShadingMode

Defines all shading models supported by the library.

The list of shading modes has been taken from Blender. See Blender documentation for more information. The API does not distinguish between "specular" and "diffuse" shaders (thus the specular term for diffuse shading
models like Oren-Nayar remains undefined). Again, this value is just a hint. **Assimp** tries to select the shader whose most common implementation matches the original rendering results of the 3D modeller which wrote a particular model as closely as possible.

**Enumerator**

<table>
<thead>
<tr>
<th>Shader Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aiShadingMode_Flat</td>
<td>Flat shading. Shading is done on per-face base, diffuse only. Also known as 'faceted shading'.</td>
</tr>
<tr>
<td>aiShadingMode_Gouraud</td>
<td>Simple Gouraud shading.</td>
</tr>
</tbody>
</table>
| aiShadingMode_Phong | Phong-Shading -.
| aiShadingMode_Blinn | Phong-Blinn-Shading. |
| aiShadingMode_Toon | Toon-Shading per pixel. Also known as 'comic' shader. |
| aiShadingMode_OrenNayar | OrenNayar-Shading per pixel. Extension to standard Lambertian shading, taking the roughness of the material into account |
| aiShadingMode_Minnaert | Minnaert-Shading per pixel. Extension to standard Lambertian shading, taking the "darkness" of the material into account |
CookTorrance-Shading per pixel.

aiShadingMode_CookTorrance
Special shader for metallic surfaces.

aiShadingMode_NoShading
No shading at all.

aiShadingMode_Fresnel
Fresnel shading.

_aiShadingMode_Force32Bit

enum aiTextureFlags

Defines some mixed flags for a particular texture.

Usually you'll instruct your cg artists how textures have to look like ... and how they will be processed in your application. However, if you use Assimp for completely generic loading purposes you might also need to process these flags in order to display as many 'unknown' 3D models as possible correctly.

This corresponds to the AI_MATKEY_TEXFLAGS property.

**Enumerator**

aiTextureFlags_Invert
The texture's color values have to be inverted (componentwise 1-n)

aiTextureFlags_UseAlpha
Explicit request to the application to process the alpha channel of the texture.

Mutually exclusive with aiTextureFlags_IgnoreAlpha. These flags are set if the library can say for sure that the alpha channel is used/is not used. If the model format does not define this, it is left to the application to decide whether the texture alpha channel - if any - is
evaluated or not.

Explicit request to the application to ignore the alpha channel of the texture.

`aiTextureFlags_IgnoreAlpha`

Mutually exclusive with `aiTextureFlags_UseAlpha`.

`_aiTextureFlags_Force32Bit`

`enum aiTextureMapMode`

Defines how UV coordinates outside the [0...1] range are handled.

Commonly referred to as 'wrapping mode'.

**Enumerator**

`aiTextureMapMode_Wrap`

A texture coordinate u|v is translated to u%1|v%1.

`aiTextureMapMode_Clamp`

Texture coordinates outside [0...1] are clamped to the nearest valid value.

`aiTextureMapMode_Decal`

If the texture coordinates for a pixel are outside [0...1] the texture is not applied to that pixel.

`aiTextureMapMode_Mirror`

A texture coordinate u|v becomes u%1|v%1 if (u-(u%1))%2 is zero and 1-(u%1)|1-(v%1) otherwise.

`_aiTextureMapMode_Force32Bit`

`enum aiTextureMapping`

Defines how the mapping coords for a texture are generated.
Real-time applications typically require full UV coordinates, so the use of the aiProcess_GenUVCoords step is highly recommended. It generates proper UV channels for non-UV mapped objects, as long as an accurate description how the mapping should look like (e.g. spherical) is given. See the `AI_MATKEY_MAPPING` property for more details.

**Enumerator**

The mapping coordinates are taken from an UV channel.

- `aiTextureMapping_UV` The `AI_MATKEY_UVWSRC` key specifies from which UV channel the texture coordinates are to be taken from (remember, meshes can have more than one UV channel).

- `aiTextureMapping_SPHERE` Spherical mapping.

- `aiTextureMapping_CYLINDER` Cylindrical mapping.

- `aiTextureMapping_BOX` Cubic mapping.

- `aiTextureMapping_PLANE` Planar mapping.

- `aiTextureMapping_OTHER` Undefined mapping.

- `-aiTextureMapping_Force32Bit enum aiTextureOp`

Defines how the Nth texture of a specific type is combined with the result of all previous layers.
Example (left: key, right: value):

1 DiffColor0 - gray
2 DiffTextureOp0 - aiTextureOpMultiply
3 DiffTexture0 - tex1.png
4 DiffTextureOp0 - aiTextureOpAdd
5 DiffTexture1 - tex2.png

Written as equation, the final diffuse term for a specific pixel would be:

1 diffFinal = DiffColor0 * sampleTex(DiffTexture0,UV0) +
2 sampleTex(DiffTexture1,UV0) * diffContrib;

where 'diffContrib' is the intensity of the incoming light for that pixel.

**Enumerator**

aiTextureOp_Multiply \[ T = T1 \times T2. \]

aiTextureOp_Add \[ T = T1 + T2. \]

aiTextureOp_Subtract \[ T = T1 - T2. \]

aiTextureOp_Divide \[ T = T1 / T2. \]

aiTextureOp_SmoothAdd \[ T = (T1 + T2) - (T1 \times T2) \]

aiTextureOp_SignedAdd \[ T = T1 + (T2-0.5) \]

_aiTextureOp_Force32Bit

enum aiTextureType

Defines the purpose of a texture.
This is a very difficult topic. Different 3D packages support different kinds of textures. For very common texture types, such as bumpmaps, the rendering results depend on implementation details in the rendering pipelines of these applications. Assimp loads all texture references from the model file and tries to determine which of the predefined texture types below is the best choice to match the original use of the texture as closely as possible.

In content pipelines you'll usually define how textures have to be handled, and the artists working on models have to conform to this specification, regardless which 3D tool they're using.

**Enumerator**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aiTextureType_NONE</td>
<td>Dummy value. No texture, but the value to be used as 'texture semantic' (#aiMaterialProperty::mSemantic) for all material properties not related to textures.</td>
</tr>
<tr>
<td>aiTextureType_DIFFUSE</td>
<td>The texture is combined with the result of the diffuse lighting equation.</td>
</tr>
<tr>
<td>aiTextureType_SPECULAR</td>
<td>The texture is combined with the result of the specular lighting equation.</td>
</tr>
<tr>
<td>aiTextureType_AMBIENT</td>
<td>The texture is combined with the result of the ambient lighting equation.</td>
</tr>
<tr>
<td>aiTextureType_EMISSIVE</td>
<td>The texture is added to the result of the lighting calculation.</td>
</tr>
<tr>
<td></td>
<td>It isn't influenced by incoming light.</td>
</tr>
</tbody>
</table>
aiTextureType_HEIGHT
The texture is a height map.
By convention, higher gray-scale values stand for higher elevations from the base height.

aiTextureType_NORMALS
The texture is a (tangent space) normal-map.
Again, there are several conventions for tangent-space normal maps. Assimp does (intentionally) not distinguish here.

aiTextureType_SHININESS
The texture defines the glossiness of the material.
The glossiness is in fact the exponent of the specular (phong) lighting equation. Usually there is a conversion function defined to map the linear color values in the texture to a suitable exponent. Have fun.

aiTextureType_OPACITY
The texture defines per-pixel opacity.
Usually 'white' means opaque and 'black' means 'transparency'. Or quite the opposite. Have fun.

aiTextureType_DISPLACEMENT
Displacement texture.
The exact purpose and format is application-dependent. Higher color values stand for higher vertex displacements.

Lightmap texture (aka Ambient Occlusion)
<table>
<thead>
<tr>
<th>Texture Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aiTextureType_LIGHTMAP</td>
<td>Both 'Lightmaps' and dedicated 'ambient occlusion maps' are covered by this material property. The texture contains a scaling value for the final color value of a pixel. Its intensity is not affected by incoming light.</td>
</tr>
<tr>
<td>aiTextureType_REFLECTION</td>
<td>Reflection texture.</td>
</tr>
<tr>
<td>aiTextureType_UNKNOWN</td>
<td>Contains the color of a perfect mirror reflection. Rarely used, almost never for real-time applications.</td>
</tr>
<tr>
<td>aiTextureType_UNKNOWN</td>
<td>Unknown texture.</td>
</tr>
<tr>
<td>_aiTextureType_Force32Bit</td>
<td>A texture reference that does not match any of the definitions above is considered to be 'unknown'. It is still imported, but is excluded from any further postprocessing.</td>
</tr>
</tbody>
</table>
Function Documentation

**ASSIMP_API aiReturn aiGetMaterialColor ( const aiMaterial * pMat,**

**const char * pKey,**

**unsigned int type,**

**unsigned int index,**

**aiColor4D * pOut**


__cplusplus

Retrieve a color value from the material property table

See the sample for aiGetMaterialFloat for more information

**aiReturn aiGetMaterialFloat ( const aiMaterial * pMat,**

**const char * pKey,**

**unsigned int type,**

**unsigned int index,**

**float * pOut**


inline

Retrieve a single float property with a specific key from the material.

Pass one of the AI_MATKEY_XXX constants for the last three parameters (the example reads the AI_MATKEY_SHININESS_STRENGTH property of the first diffuse texture)

1 float specStrength = 1.f; // default value, remains unmodified if we fail.
2 aiGetMaterialFloat(mat, AI_MATKEY_SHININESS_STRENGTH,
3 (float*)&specStrength);

Parameters

- **pMat** Pointer to the input material. May not be NULL
- **pKey** Key to search for. One of the AI_MATKEY_XXX constants.
pOut  Receives the output float.
type  (see the code sample above)
index (see the code sample above)

Returns
 specifies whether the key has been found. If not, the output float remains unmodified.

\textbf{ASSIMP\_API aiReturn aiGetMaterialFloatArray ( const aiMaterial \* pMat,}
\texttt{const char \* pKey,}
\texttt{unsigned int type,}
\texttt{unsigned int index,}
\texttt{float \* pOut,}
\texttt{unsigned int \* pMax })

Retrieve an array of float values with a specific key from the material.

Pass one of the \texttt{AI\_MATKEY\_XXX} constants for the last three parameters (the example reads the \texttt{AI\_MATKEY\_UVTRANSFORM} property of the first diffuse texture)

1 aiUVTransform trafo;
2 unsigned int max = sizeof(aiUVTransform);
3 if (AI\_SUCCESS != aiGetMaterialFloatArray(mat,}
\texttt{AI\_MATKEY\_UVTRANSFORM(aiTextureType\_DIFFUSE,0),}
\texttt{(float*)&trafo, &max) || sizeof(aiUVTransform) != max)
4 {}
5 // error handling
6 }

Parameters
 pMat  Pointer to the input material. May not be NULL
 pKey  Key to search for. One of the \texttt{AI\_MATKEY\_XXX} constants.
 pOut  Pointer to a buffer to receive the result.
 pMax  Specifies the size of the given buffer, in float's. Receives the number of values (not bytes!) read.
type (see the code sample above)
index (see the code sample above)

Returns
Specifies whether the key has been found. If not, the output arrays remains
unmodified and \( pMax \) is set to 0.

```c
aiReturn aiGetMaterialInteger ( const aiMaterial * pMat,
                                 const char * pKey,
                                 unsigned int type,
                                 unsigned int index,
                                 int * pOut
                               )
```

Retrieve an integer property with a specific key from a material.

See the sample for aiGetMaterialFloat for more information.

```c
ASSIMP_API aiReturn aiGetMaterialIntegerArray ( const aiMaterial * pMat,
                                               const char * pKey,
                                               unsigned int type,
                                               unsigned int index,
                                               int * pOut,
                                               unsigned int * pMax
                               )
```

Retrieve an array of integer values with a specific key from a material.

See the sample for aiGetMaterialFloatArray for more information.

```c
ASSIMP_API aiReturn aiGetMaterialProperty
               ( const aiMaterial * pMat,
                 const char * pKey,
                 unsigned int type,
```
Retrieve a material property with a specific key from the material.

Parameters

- **pMat**: Pointer to the input material. May not be NULL.
- **pKey**: Key to search for. One of the AI_MATKEY_XXX constants.
- **type**: Specifies the type of the texture to be retrieved (e.g. diffuse, specular, height map ...)
- **index**: Index of the texture to be retrieved.
- **pPropOut**: Pointer to receive a pointer to a valid aiMaterialProperty structure or NULL if the key has not been found.

```c
ASSIMP_API aiReturn aiGetMaterialProperty(
    const aiMaterial* pMat,
    const aiMaterialProperty** pPropOut,
    const char* pKey,
    unsigned int type,
    unsigned int index,
    aiString* pOut)
```

Retrieve a string from the material property table.

See the sample for `aiGetMaterialFloat` for more information.

```c
ASSIMP_API aiReturn aiGetMaterialString(
    const aiMaterial* pMat,
    const char* pKey,
    unsigned int type,
    unsigned int index,
    aiString* pOut)
```

```c
ASSIMP_API aiReturn aiGetMaterialTexture(
    const aiMaterial* mat,
    aiTextureType type,
    unsigned int index,
    aiString* path,
    aiTextureMapping* mapping = NULL,
    unsigned int* uvindex = NULL,
    float* blend = NULL,
    ...)
aiTextureOp * op = NULL,
aiTextureMapMode * mapmode = NULL,
unsigned int * flags = NULL
)

Helper function to get all values pertaining to a particular texture slot from a material structure.

This function is provided just for convenience. You could also read the texture by parsing all of its properties manually. This function bundles all of them in a huge function monster.

Parameters

[in] mat Pointer to the input material. May not be NULL
[in] type Specifies the texture stack to read from (e.g. diffuse, specular, height map ...).
[in] index Index of the texture. The function fails if the requested index is not available for this texture type.
aiGetMaterialTextureCount() can be used to determine the number of textures in a particular texture stack.
[out] path Receives the output path This parameter must be non-null.
[out] mapping The texture mapping mode to be used. Pass NULL if you're not interested in this information.
For UV-mapped textures: receives the index of the UV source channel. Unmodified otherwise. Pass NULL if you're not interested in this information.
[out] uvindex Receives the blend factor for the texture Pass NULL if you're not interested in this information.
[out] blend Receives the texture blend operation to be perform between this texture and the previous texture. Pass NULL if you're not interested in this information.
[out] op Receives the mapping modes to be used for the texture. Pass NULL if you're not interested in this information. Otherwise, pass a pointer to an array of two aiTextureMapMode's (one for each axis, UV order).
[out] mapmode Receives the the texture flags.
[out] flags
Returns

AI_SUCCESS on success, otherwise something else. Have fun.

`ASSIMP_API` `unsigned int aiGetMaterialTextureCount (const aiMaterial * pMat, aiTextureType type)`

Get the number of textures for a particular texture type.

Parameters

- `[in]` `pMat` Pointer to the input material. May not be NULL
- `type` Texture type to check for

Returns

Number of textures for this type.

Note

A texture can be easily queried using `aiGetMaterialTexture()`

`ASSIMP_API` `aiReturn aiGetMaterialUVTransform ( const aiMaterial * pMat, const char * pKey, unsigned int type, unsigned int index, aiUVTransform * pOut )`

Retrieve a `aiUVTransform` value from the material property table.

See the sample for `aiGetMaterialFloat` for more information
Variable Documentation

struct aiUVTransform PACK_STRUCT

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Main Page  Related Pages  Namespaces  Classes  Files
File List  File Members

- include
- assimp

Macros
material.inl File Reference

Defines the C++ getters for the material system. More...
Macros

#define AI_MATERIAL_INL_INC
Detailed Description

Defines the C++ getters for the material system.
#define AI_MATERIAL_INL_INC
Definition of a 3x3 matrix, including operators when compiling in C++. More...
Classes

class aiMatrix3x3t< TReal >

   Represents a row-major 3x3 matrix. More...

class aiMatrix4x4t< TReal >

   Represents a row-major 4x4 matrix, use this for homogeneous coordinates. More...

class aiVector2t< TReal >

   Represents a two-dimensional vector. More...
**Typedefs**

typedef aiMatrix3x3t< float > aiMatrix3x3
Variables

class aiMatrix3x3t PACK_STRUCT
Detailed Description

Definition of a 3x3 matrix, including operators when compiling in C++.
typedef Documentation

typedef aiMatrix3x3t<float> aiMatrix3x3
Variable Documentation

class aiMatrix3x3t PACK_STRUCT

Generated on Sun Feb 21 2016 19:42:29 for Assimp by _doxygen_ 1.8.11
## Assimp v3.1.1 (June 2014)

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Namespaces</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td>File Members</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- [include](#)
- [assimp](#)

**Macros**

**matrix3x3.inl File Reference**

Inline implementation of the 3x3 matrix operators. [More...](#)
Macros

#define AI_MATRIX3x3_INL_INC
Detailed Description

Inline implementation of the 3x3 matrix operators.
Macro Definition Documentation

#define AI_MATRIX3x3_INL_INC

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
include
assimp

Classes | Typedefs | Variables

matrix4x4.h File Reference

4x4 matrix structure, including operators when compiling in C++ More...
Classes

class aiMatrix3x3t< TReal >

Represents a row-major 3x3 matrix. More...

class aiMatrix4x4t< TReal >

Represents a row-major 4x4 matrix, use this for homogeneous coordinates. More...

class aiQuaternions< TReal >

Represents a quaternion in a 4D vector. More...
Typedefs

typedef aiMatrix4x4t< float > aiMatrix4x4
Variables

class aiMatrix4x4t PACK_STRUCT
**Detailed Description**

4x4 matrix structure, including operators when compiling in C++
Typedef Documentation

typedef aiMatrix4x4t<float> aiMatrix4x4
Variable Documentation

class aiMatrix4x4t PACK_STRUCT

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Macros
matrix4x4.inl File Reference

Inline implementation of the 4x4 matrix operators. More...
Macros

#define AI_MATRIX4x4_INL_INC
Detailed Description

Inline implementation of the 4x4 matrix operators.
Macro Definition Documentation

#define AI_MATRIX4x4_INL_INC

Generated on Sun Feb 21 2016 19:42:29 for Assimp by _doxygen_ 1.8.11
Declares the data structures in which the imported geometry is returned by ASSIMP: aiMesh, aiFace and aiBone data structures. More...
Classes

struct aiAnimMesh
NOT CURRENTLY IN USE. More...

struct aiBone
A single bone of a mesh. More...

struct aiFace
A single face in a mesh, referring to multiple vertices. More...

struct aiMesh
A mesh represents a geometry or model with a single material. More...

struct aiVertexWeight
A single influence of a bone on a vertex. More...
Macros

#define AI_MAX_BONE_WEIGHTS 0x7fffffff
Maximum number of indices per face (polygon). More...

#define AI_MAX_FACE_INDICES 0x7fff
Maximum number of indices per face (polygon). More...

#define AI_MAX_FACES 0x7fffffff
Maximum number of faces per mesh. More...

#define AI_MAX_NUMBER_OF_COLOR_SETS 0x8
Supported number of vertex color sets per mesh. More...

#define AI_MAX_NUMBER_OF_TEXTURECOORDS 0x8
Supported number of texture coord sets (UV(W) channels) per mesh. More...

#define AI_MAX_VERTICES 0x7fffffff
Maximum number of vertices per mesh. More...

#define AI_PRIMITIVE_TYPE_FOR_N_INDICES(n) ((n) > 3 ? aiPrimitiveType_POLYGON : (aiPrimitiveType)(1u << ((n)-1)))
enum aiPrimitiveType More...
Enumerations
Detailed Description

Declares the data structures in which the imported geometry is returned by ASSIMP: aiMesh, aiFace and aiBone data structures.
Macro Definition Documentation

#define AI_MAX_BONE_WEIGHTS 0x7fffffff
Maximum number of indices per face (polygon).

#define AI_MAX_FACE_INDICES 0x7fff
Maximum number of indices per face (polygon).

#define AI_MAX_FACES 0x7fffffff
Maximum number of faces per mesh.

#define AI_MAX_NUMBER_OF_COLOR_SETS 0x8
Supported number of vertex color sets per mesh.

#define AI_MAX_NUMBER_OF_TEXTURECOORDS 0x8
Supported number of texture coord sets (UV(W) channels) per mesh.

#define AI_MAX_VERTICES 0x7fffffff
Maximum number of vertices per mesh.

#define AI_PRIMITIVE_TYPE_FOR_N_INDICES(n) ((n) > 3 ? aiPrimitiveType_POLYGON : (aiPrimitiveType)(1u << ((n)-1)))

enum aiPrimitiveType
Enumeration Type Documentation

enum aiPrimitiveType

Enumerates the types of geometric primitives supported by Assimp.

See also
- aiFace Face data structure
- aiProcess_SortByPType Per-primitive sorting of meshes
- aiProcess_Triangulate Automatic triangulation
- AI_CONFIG_PP_SBP_REMOVE Removal of specific primitive types.

**Enumerator**

A point primitive.

aiPrimitiveType_POINT This is just a single vertex in the virtual world, aiFace contains just one index for such a primitive.

A line primitive.

aiPrimitiveType_LINE This is a line defined through a start and an end position. aiFace contains exactly two indices for such a primitive.

A triangular primitive.

aiPrimitiveType_TRIANGLE A triangle consists of three indices.

A higher-level polygon with more than 3 edges.

aiPrimitiveType_POLYGON A triangle is a polygon, but polygon in this context means "all polygons that are not triangles". The "Triangulate"-Step is provided for
your convenience, it splits all polygons in triangles (which are much easier to handle).

This value is not used.

_aiPrimitiveType_Force32Bit

It is just here to force the compiler to map this enum to a 32 Bit integer.
metadata.h File Reference

Defines the data structures for holding node meta information. More...
Classes

struct aiMetadata
  Container for holding metadata.  More...

struct aiMetadataEntry
  Metadata entry.  More...
Enumerations
Functions

`aiMetadataType GetAiType (bool)`
Helper functions to get the aiType enum entry for a type.
More...

`aiMetadataType GetAiType (int)`

`aiMetadataType GetAiType (uint64_t)`

`aiMetadataType GetAiType (float)`

`aiMetadataType GetAiType (aiString)`

`aiMetadataType GetAiType (aiVector3D)`
Detailed Description

Defines the data structures for holding node meta information.
Enumeration Type Documentation

enum aiMetadataType

Enum used to distinguish data types.

**Enumerator**

AI_BOOL
AI_INT
AI_UINT64
AI_FLOAT
AI_AISTRING
AI_AIVECTOR3D
FORCE_32BIT
Function Documentation

`aiMetadataType GetAiType ( bool ) inline`

Helper functions to get the aiType enum entry for a type.

`aiMetadataType GetAiType ( int ) inline`

`aiMetadataType GetAiType ( uint64_t ) inline`

`aiMetadataType GetAiType ( float ) inline`

`aiMetadataType GetAiType ( aiString ) inline`

`aiMetadataType GetAiType ( aiVector3D ) inline`
**NullLogger.hpp File Reference**

Dummy logger. [More...]( )
Classes

class Assimp::NullLogger

CPP-API: Empty logging implementation. More...
Namespaces

Assimp

Assimp's CPP-API and all internal APIs.
Detailed Description

Dummy logger.
### Assimp v3.1.1 (June 2014)

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Namespaces</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td>File Members</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- include
- assimp
- Compiler

poppack1.h File Reference

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11*
Assimp v3.1.1 (June 2014)

- include
- assimp

**Macros | Enumerations**

postprocess.h File Reference

Definitions for import post processing steps. More...
Macros

#define aiProcess_ConvertToLeftHanded
Shortcut flag for Direct3D-based applications. More...

#define aiProcessPreset_TargetRealtime_Fast
Default postprocess configuration optimizing the data for real-time rendering. More...

#define aiProcessPreset_TargetRealtime_MaxQuality
Default postprocess configuration optimizing the data for real-time rendering. More...

#define aiProcessPreset_TargetRealtime_Quality
Default postprocess configuration optimizing the data for real-time rendering. More...
Detailed Description

Definitions for import post processing steps.
Macro Definition Documentation

#define aiProcess_ConvertToLeftHanded

**Value:**

```
( \naiProcess_MakeLeftHanded | \naiProcess_FlipUVs | \naiProcess_FlipWindingOrder | \n0 )
```

Shortcut flag for Direct3D-based applications.

Supersedes the `aiProcess_MakeLeftHanded` and `aiProcess_FlipUVs` and `aiProcess_FlipWindingOrder` flags. The output data matches Direct3D's conventions: left-handed geometry, upper-left origin for UV coordinates and finally clockwise face order, suitable for CCW culling.

**Deprecated:**

#define aiProcessPreset_TargetRealtime_Fast

**Value:**

```
( \naiProcess_CalcTangentSpace | \naiProcess_GenNormals | \naiProcess_JoinIdenticalVertices | \naiProcess_Triangulate | \naiProcess_GenUVCoords | \naiProcess_SortByPType | \n0 )
```

Default postprocess configuration optimizing the data for real-time rendering.

Applications would want to use this preset to load models on end-user PCs, maybe for direct use in game.

If you're using DirectX, don't forget to combine this value with the `aiProcess_ConvertToLeftHanded` step. If you don't support UV transformations in your application apply the `aiProcess_TransformUVCoords` step, too.
Note
Please take the time to read the docs for the steps enabled by this preset. Some of them offer further configurable properties, while some of them might not be of use for you so it might be better to not specify them.

#define aiProcessPreset_TargetRealtime_MaxQuality

Value:
( \naiProcessPreset_TargetRealtime_Quality | \naiProcess_FindInstances | \naiProcess_ValidateDataStructure | \naiProcess_OptimizeMeshes | \n0 )

Default postprocess configuration optimizing the data for real-time rendering.

This preset enables almost every optimization step to achieve perfectly optimized data. It's your choice for level editor environments where import speed is not important.

If you're using DirectX, don't forget to combine this value with the aiProcess_ConvertToLeftHanded step. If you don't support UV transformations in your application, apply the aiProcess_TransformUVCoords step, too.

Note
Please take the time to read the docs for the steps enabled by this preset. Some of them offer further configurable properties, while some of them might not be of use for you so it might be better to not specify them.

#define aiProcessPreset_TargetRealtime_Quality

Value:
( \naiProcess_CalcTangentSpace | \naiProcess_GenSmoothNormals | \naiProcess_JoinIdenticalVertices | \naiProcess_ImproveCacheLocality | \naiProcess_LimitBoneWeights | \naiProcess_RemoveRedundantMaterials | \naiProcess_SplitLargeMeshes | \n
Default postprocess configuration optimizing the data for real-time rendering.

Unlike `aiProcessPreset_TargetRealtime_Fast`, this configuration performs some extra optimizations to improve rendering speed and to minimize memory usage. It could be a good choice for a level editor environment where import speed is not so important.

If you're using DirectX, don't forget to combine this value with the `aiProcess_ConvertToLeftHanded` step. If you don't support UV transformations in your application apply the `aiProcess_TransformUVCoords` step, too.

Note

Please take the time to read the docs for the steps enabled by this preset. Some of them offer further configurable properties, while some of them might not be of use for you so it might be better to not specify them.
## Enumeration Type Documentation

**enum aiPostProcessSteps**

Defines the flags for all possible post processing steps.

**Note**
Some steps are influenced by properties set on the `Assimp::Importer` itself

See also

- `Assimp::Importer::ReadFile()`
- `Assimp::Importer::SetPropertyInteger()`
- `aiImportFile`
- `aiImportFileEx`

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aiProcess_CalcTangentSpace</td>
<td>Calculates the tangents and bitangents for the imported meshes. Does nothing if a mesh does not have normals. You might want this post-processing step to be executed if you plan to use tangent space calculations such as normal mapping applied to the meshes. There's an importer property, which allows you to specify a maximum smoothing angle for the algorithm. However, usually you'll want to leave it at the default value.</td>
</tr>
<tr>
<td>aiProcess_JoinIdenticalVertices</td>
<td>Identifies and joins identical vertex data sets within all imported meshes. After this step is run, each mesh contains unique vertices, so a vertex may be used by multiple faces. You usually want to use this post-processing step. If your application deals with indexed geometry, this step is compulsory or you'll just waste rendering time. No vertices are referenced by more than one face and buffer is required for rendering.</td>
</tr>
<tr>
<td>aiProcess_CalcTangentSpace</td>
<td>Converts all the imported data to a left-handed coordinate space.</td>
</tr>
</tbody>
</table>


aiProcess_MakeLeftHanded

By default the data is returned in a right-handed coordinate space (which OpenGL prefers).
In this space, +X points to the right, +Z points away from the viewer.

You'll probably want to consider this flag if you use Direct3D for rendering. The aiProcess_ConvertToLeftHanded flag supersedes this setting and bundles all conversions typically required for D3D-based applications.

aiProcess_Triangulate

Triangulates all faces of all meshes.

By default the imported mesh data might contain faces with more than 3 indices. For rendering you'll usually want all faces to be triangles. This post-processing step splits up faces with more than 3 indices into triangles. Line and point primitives are you want 'triangles only' with no other kind of primitives, try the following solution:

- Specify both aiProcess_Triangulate
- Ignore all point and line meshes when you process assimp's output

aiProcess_RemoveComponent

Removes some parts of the data structure (animations, materials, light sources, cameras, textures, vertex components).

The components to be removed are specified in a separate importer property, AI_CONFIG_PP_RVC_FLAGS. This is quite useful if you don't need all parts of the output structure. Vertex colors are rarely used today for example. Calling this step to remove unneeded data from the pipeline as early as possible results in increased performance and a more optimized output data structure. This step is also useful if you want to force to recompute normals or tangents. The corresponding steps don't recompute them if they're already there (loaded from the source asset). By using this step you can make sure they are NOT there.

This flag is a poor one, mainly because it is confusing. In the following case: a 3D model has been exported from a CAD app, and it has per-face vertex colors. Vertex positions can't be shared, thus the model to optimize the data because of these nasty little vertex colors. Most apps don't even process them, so it's all for nothing. By using this flag, unneeded components are excluded.
as early as possible thus opening more room for internal optimizations.

aiProcess_GenNormals
Generates normals for all faces of all meshes. This is ignored if normals are already there at the time this flag is evaluated. Model importers try to load them from the source file, so they're usually already there. Face normals are shared between all points of a single face, so a single point can have multiple normals, which forces the library to duplicate vertices in some cases. aiProcess_JoinIdenticalVertices is senseless. This flag may not be specified together with aiProcess_GenSmoothNormals.

aiProcess_GenSmoothNormals
Generates smooth normals for all vertices. This is ignored if normals are already there at the time this flag is evaluated. Model importers try to load them from the source file, so they're usually already there. This flag may not be specified together with the property, AI_CONFIG_PP_GSN_MAX_SMOOTHING_ANGLE, maximum for the normal smoothing algorithm. Normals exceeding this limit are not smoothed, resulting in a 'hard' seam between two faces. Using a decent angle there (e.g. 80 degrees) results in very good visual appearance.

aiProcess_SplitLargeMeshes
Splits large meshes into smaller sub-meshes. This is quite useful for real-time rendering, where the number of triangles which can be maximally processed in a single draw-call is limited by the video driver/hardware. The maximum vertex buffer is usually limited too. Both requirements can be met with this step: you may specify both a triangle and vertex limit for a single mesh. The split limits can (and should!) be set through the properties, AI_CONFIG_PP_SLM_TRIANGLE_LIMIT and AI_SLM_DEFAULT_MAX_TRIANGLES, and AI_CONFIG_PP_SLM_MAX_VERTICES and AI_SLM_DEFAULT_MAX_VERTICES. Note that splitting is generally a time-consuming task, but only if there's something to split.
**aiProcess_PreTransformVertices**

Removes the node graph and pre-transform matrices of their nodes.

The output scene still contains nodes, however, each one referencing only one mesh, and for rendering, you can simply render all meshes with their local transformations and the node hierarchy. This step is intended for applications without a scenegraph. The step can cause some problems: if e.g., a mesh of the asset contains normals and another, using the same material index, does not, they will be brought together, but the first mesh's part of the normal list is zeroed. However, these artifacts are rare.

Note

The **AI_CONFIG_PP_PTV_NORMALIZE** constant normalizes the scene's spatial dimension to the -1..1 range.

**aiProcess_LimitBoneWeights**

Limits the number of bones simultaneously affecting a single vertex to a maximum value.

If any vertex is affected by more than the maximum number of bones, the least important vertex weights are removed and the remaining vertex weights are renormalized so that the weights still sum up to 1. The default bone weight limit is 4 (defined as **AI_LMW_MAX_WEIGHTS** in **config.h**), but you can use the importer property to supply your own limit.

If you intend to perform the skinning in hardware, this post-processing step might be of interest to you.

**aiProcess_Verify**

Validates the imported scene data structure. All animations and bones are linked correctly, all material references are correct, etc.

It is recommended that you capture Assimp easily and find out what's wrong if a file fails validation. The validator is quite strict and will find all inconsistencies in the data structure.
developers use it to debug their loaders.

- Error: There's something wrong with the added data and the resulting data cannot be used further. The data is unusable. The import fails. `aiGetErrorString()` or `aiGetErrorString()` carry the error message around.

- Warning: There are some minor issues (e.g., 1000000 animation keyframes with the same time), but further postprocessing and use of the data structure is still safe. Warning details are written to the log file, set in `aiScene::mFlags`.

This post-processing step is not time-consuming. Its use is not compulsory, but recommended.

Reorders triangles for better vertex cache locality.

The step tries to improve the ACMR (average cache miss ratio) for all meshes. The implementation runs in O(n) (see this paper).

If you intend to render huge models in hardware, this step might be of interest to you. The `AI_CONFIG_PP_ICL_PTCACHE_SIZE` importer property can be used to fine-tune the cache optimization.

Searches for redundant/unreferenced materials and removes them.

This is especially useful in combination with `aiProcess_OptimizeMeshes` flags. Both join small meshes with equal characteristics, but they can't do their work if two meshes have different materials. Because several material settings are lost during Assimp's import for redundant materials), huge models often have materials defined several times with exactly the same settings.

Several material settings not contributing in all comparisons (e.g., the material name) are ignored in all comparisons (e.g., the material name). Alternatively take a look at the `AI_CONFIG_PP_RRM_EXCLUDE_LIST`
This step tries to determine which meshes have normal vectors that are facing inwards and inverts them.

The algorithm is simple but effective: the mesh's bounding box of all vertices + their normals is compared against the volume of the bounding box of all vertices without their normals. This works well for most objects, problems might occur with planar surfaces. However, the step tries to filter such cases. The step inverts all in-facing normals. Generally it is recommended to enable this step, although the result is not always correct.

This step splits meshes with more than one primitive type into homogeneous sub-meshes.

The step is executed after the triangulation step. The step sets the aiMesh::mPrimitiveTypes. This is especially useful for real-time rendering where point and line primitives are often ignored or removed. You can use the AI_CONFIG_PP_SBP_REMOVE importer property to specify which primitive types you need. This can be used to easily exclude lines and points, which are rarely used, from the import.

This step searches all meshes for degenerate primitives and converts them to proper lines or points.

A face is 'degenerate' if one or more of its points are identical. To have the degenerate stuff not only detected and collapsed but removed, try one of the following procedures:

1. (if you support lines and points for rendering but don't want the degenerates)
   - Specify the aiProcess_FindDegenerate
   - Set the AT_CONFIG_PP_FD_REMOVE importer property to 1. This will cause the step to remove degenerate triangles from the import as soon as they're detected. They won't pass any further pipeline steps.

2. (if you don't support lines and points at all)
   - Specify the aiProcess_FindDegenerate
   - Specify the aiProcess_SortByPType separate meshes.

aiProcess_FixInfacingNormals

aiProcess_SortByPType

aiProcess_FindDegenerates
- Set the `AI_CONFIG_PP_SBP_REMOVE` 1 aiPrimitiveType_POINTS | aiPrimitiveType_LINES to cause SortByPType to reject point and line meshes from the scene.

**Note**
Degenerate polygons are not necessarily evil and that's why they're not removed by default. There are several file formats which don't support lines or points, and some exporters bypass the format specification and write them as degenerate triangles instead.

---

**aiProcess_FindInvalidData**

This step searches all meshes for invalid data, such as zeroed normal vectors or invalid UV coords and removes/fixes them. This is intended to get rid of some common exporter errors.

This is especially useful for normals. If they are invalid, the step will recognize this, they will be removed and can later be recomputed.

The step will also remove meshes that are infinitely small and reduce animation tracks consisting of hundreds if redundant keys by the flag.

**AI_CONFIG_PP_FID_ANIM_ACCURACY**

This property decides the accuracy of the check for duplicate animation tracks.

---

**aiProcess_GenUVCoords**

This step converts non-UV mappings (such as spherical or cylindrical mapping) to proper texture coordinate channels.

Most applications will support UV mapping, so you will probably want to specify this step in every case. Note that Assimp is not able to match the original mapping implementation of the 3D app which produced a model perfectly. It's always better to let the modelling app compute the UV channels - 3ds max, Maya, Blender, LightWave, and Modo do this for example.

**Note**
If this step is not requested, you'll need to process the property in order to display all assets properly.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aiProcess_TransformUVCoords</td>
<td>This step applies per-texture UV transformations and bakes them into stand-alone texture coordinate channels. UV transformations are specified per-texture and key for more information. This step processes all textures with transformed input UV coordinates and generates a new (pre-transformed) UV channel. Most applications won't support UV transformations, so you will probably want to specify this step. Note: UV transformations are usually implemented in real-time apps by transforming texture coordinates at vertex shader stage with a 3x3 (homogenous) transformation matrix.</td>
</tr>
<tr>
<td>aiProcess_FindInstances</td>
<td>This step searches for duplicate meshes and replaces them with references to the first mesh. This step takes a while, so don't use it if speed is a concern. Its main purpose is to workaround the fact that many export file formats don't support instanced meshes, so exporters need to duplicate meshes. This step removes the duplicates again. Please note that Assimp does not currently support per-node material assignment to meshes, which means that identical meshes with different materials are currently not supported.</td>
</tr>
<tr>
<td>aiProcess_OptimizeMeshes</td>
<td>A postprocessing step to reduce the number of meshes. This will, in fact, reduce the number of draw calls. This is a very effective optimization and is recommended to be used together with #aiProcess_OptimizeGraph, if possible. The flag is fully compatible with both #aiProcess_SplitLargeMeshes and #aiProcess_SortByPType.</td>
</tr>
<tr>
<td></td>
<td>A postprocessing step to optimize the scene hierarchy. Nodes without animations, bones, lights or cameras assigned are collapsed and joined.</td>
</tr>
</tbody>
</table>
aiProcess_OptimizeGraph

Node names can be lost during this step to pass additional information through list of node names you want to be kept won't be touched or modified.

Use this flag with caution. Most single node, so complex hierarchies useful for editor environments, but optimization if you just want to get your own format, and render it as fast as possible.

This flag is designed to be used with #aiProcess_OptimizeMeshes for best results.

@note 'Crappy' scenes with thousands of extremely small meshes packed in deeply nested nodes exist for almost all file formats. #aiProcess_OptimizeMeshes in combination with #aiProcess_OptimizeGraph usually fixes them all and makes them renderable.

aiProcess_FlipUVs

This step flips all UV coordinates along the y-axis and adjusts material settings and bitangents accordingly.

**Output UV coordinate system:**

1 0y|0y -------- 1x|0y
2 ||
3 ||
4 ||
5 0x|1y -------- 1x|1y

You'll probably want to consider this flag aiProcess_ConvertToLeftHanded flag supersedes this setting and bundles all conversions typically required for D3D-based applications.

This step adjusts the output face winding order.

The default face winding order is counterclockwise.
aiProcess_SplitByBoneCount
This step splits meshes with many bones or as many bones as a given limit.

aiProcess_Debone
This step removes bones losslessly or ac
In some cases (i.e. formats that r assign dummy bone weights to other animated meshes. Full, weight-base animating nodes is extremely cheap the data in that regard.

Use `<tt>`#AI_CONFIG_PP_DB_THRESHOLD
Use `<tt>`#AI_CONFIG_PP_DB_ALL_OR_NC only if all bones within the scene

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by [doxygen](http://www.stackoverflow.com) 1.8.11*
ProgressHandler.hpp File Reference

Abstract base class 'ProgressHandler'. More...
Classes

class Assimp::ProgressHandler

CPP-API: Abstract interface for custom progress report receivers. More...
Namespaces

Assimp
Assimp's CPP-API and all internal APIs.
Detailed Description

Abstract base class 'ProgressHandler'.
Assimp v3.1.1 (June 2014)

- include
- assimp
- Compiler

Macros | Typedefs
pstdint.h File Reference
Macros

#define __int8_t_defined

#define __PSTDINT_H_INCLUDED

#define INT16_MAX 0x7fff
#define INT16_MIN INT16_C(0x8000)

#define INT32_MAX (0x7fffffffL)
#define INT32_MIN INT32_C(0x80000000)

#define INT8_MAX 0x7f
#define INT8_MIN INT8_C(0x80)

#define INT_FAST16_MAX INT_FAST16_MIN
#define INT16_MAX
#define INT_LEAST16_MAX
#define INT16_MIN
#define INT_LEAST16_MIN

#define INT_FAST32_MAX INT_FAST32_MIN
#define INT32_MAX
#define INT_FAST32_MIN
#define INT_LEAST32_MAX
#define INT32_MIN
#define INT_LEAST32_MIN

#define INT_FAST8_MAX INT_FAST8_MIN
#define INT8_MAX
#define INT_FAST8_MIN
#define INT8_MIN
#define INT_LEAST8_MAX
#define INT8_C(0x80)

#define INT_LEAST16_MAX
#define INT16_MAX
#define INT_LEAST16_MIN INT16_MIN
#define INT_LEAST32_MAX INT32_MAX
#define INT_LEAST32_MIN INT32_MIN
#define INT_LEAST8_MAX INT8_MAX
#define INT_LEAST8_MIN INT8_MIN
#define INTMAX_C(v) INT32_C(v)
#define INTMAX_MAX INT32_MAX
#define PRINTF_INT16_DEC_WIDTH "5"
#define PRINTF_INT16_HEX_WIDTH "4"
#define PRINTF_INT32_DEC_WIDTH "10"
#define PRINTF_INT32_HEX_WIDTH "8"
#define PRINTF_INT64_DEC_WIDTH "20"
#define PRINTF_INT64_HEX_WIDTH "16"
#define PRINTF_INT8_DEC_WIDTH "3"
#define PRINTF_INT8_HEX_WIDTH "2"
#define PRINTF_INTMAX_DEC_WIDTH PRINTF_INT32_DEC_WIDTH
#define PRINTF_INTMAX_HEX_WIDTH PRINTF_INT32_HEX_WIDTH
#define PRINTF_INTMAX_MODIFIER PRINTF_INT32_MODIFIER
#define PRINTF_LEAST16_MODIFIER PRINTF_INT16_MODIFIER
#define PRINTF_LEAST32_MODIFIER PRINTF_INT32_MODIFIER
#define SIG_ATOMIC_MAX (((sig_atomic_t) 1) << (sizeof (sig_atomic_t)*CHAR_BIT-1)) - 1)
#define SIZE_MAX (~(size_t)0)
#define STDINT_H_UINTPTR_T_DEFINED
#define UINT16_MAX 0xffff
#define UINT32_MAX (0xffffffffUL)
#define UINT8_MAX 0xff
#define UINT_FAST16_MAX UINT_LEAST16_MAX
#define UINT_FAST32_MAX UINT_LEAST32_MAX
#define UINT_FAST8_MAX UINT_LEAST8_MAX
#define UINT_LEAST16_MAX UINT16_MAX
#define UINT_LEAST32_MAX UINT32_MAX
#define UINT_LEAST8_MAX UINT8_MAX
#define UINTMAX_C(v) UINT32_C(v)
```c
#define UINTMAX_MAX  UINT32_MAX
#define ULONG_LONG_MAX  UINT64_C (18446744073709551615)
```
**Typedefs**

typedef int_least16_t int_fast16_t

typedef int_least32_t int_fast32_t

typedef int_least8_t int_fast8_t

typedef int16_t int_least16_t

typedef int32_t int_least32_t

typedef int8_t int_least8_t

typedef int32_t intmax_t

typedef ptrdiff_t intptr_t

typedef uint_least16_t uint_fast16_t

typedef uint_least32_t uint_fast32_t

typedef uint_least8_t uint_fast8_t

typedef uint16_t uint_least16_t

typedef uint32_t uint_least32_t

typedef uint8_t uint_least8_t

typedef uint32_t uintmax_t
Macro Definition Documentation

#define __int8_t_defined
#define _PSTDINT_H_INCLUDED
#define INT16_MAX 0x7fff
#define INT16_MIN INT16_C(0x8000)
#define INT32_MAX (0x7fffffffL)
#define INT32_MIN INT32_C(0x80000000)
#define INT8_MAX 0x7f
#define INT8_MIN INT8_C(0x80)
#define INT_FAST16_MAX INT_LEAST16_MAX
#define INT_FAST16_MIN INT_LEAST16_MIN
#define INT_FAST32_MAX INT_LEAST32_MAX
#define INT_FAST32_MIN INT_LEAST32_MIN
#define INT_FAST8_MAX INT_LEAST8_MAX
#define INT_FAST8_MIN INT_LEAST8_MIN
#define INT_LEAST16_MAX INT16_MAX
#define INT_LEAST16_MIN INT16_MIN
#define INT_LEAST32_MAX INT32_MAX
#define INT_LEAST32_MIN INT32_MIN
#define INT_LEAST8_MAX INT8_MAX
#define INT_LEAST8_MIN INT8_MIN
#define INTMAX_C(v) INT32_C(v)
#define INTMAX_MAX INT32_MAX
#definePRINTF_INT16_DEC_WIDTH "5"
#definePRINTF_INT16_HEX_WIDTH "4"
#definePRINTF_INT32_DEC_WIDTH "10"
#definePRINTF_INT32_HEX_WIDTH "8"
#definePRINTF_INT64_DEC_WIDTH "20"
#definePRINTF_INT64_HEX_WIDTH "16"
# define PRINTF_INT8_DEC_WIDTH "3"
# define PRINTF_INT8_HEX_WIDTH "2"
# define PRINTF_INTMAX_DEC_WIDTH PRINTF_INT32_DEC_WIDTH
# define PRINTF_INTMAX_HEX_WIDTH PRINTF_INT32_HEX_WIDTH
# define PRINTF_INTMAX_MODIFIER PRINTF_INT32_MODIFIER
# define PRINTF_LEAST16_MODIFIER PRINTF_INT16_MODIFIER
# define PRINTF_LEAST32_MODIFIER PRINTF_INT32_MODIFIER
# define SIG_ATOMIC_MAX (((sig_atomic_t) 1) << (sizeof (sig_atomic_t)*CHAR_BIT-1)) - 1)
# define SIZE_MAX (~(size_t)0)
# define STDINT_H_UINTPTR_T_DEFINED
# define UINT16_MAX 0xffff
# define UINT32_MAX (0xffffffffUL)
# define UINT8_MAX 0xff
# define UINT_FAST16_MAX UINT_LEAST16_MAX
# define UINT_FAST32_MAX UINT_LEAST32_MAX
# define UINT_FAST8_MAX UINT_LEAST8_MAX
# define UINT_LEAST16_MAX UINT16_MAX
# define UINT_LEAST32_MAX UINT32_MAX
# define UINT_LEAST8_MAX UINT8_MAX
# define UINTMAX_C ( v ) UINT32_C(v)
# define UINTMAX_MAX UINT32_MAX
# define ULONG_LONG_MAX UINT64_C(18446744073709551615)
typedef int_least16_t int_fast16_t
typedef int_least32_t int_fast32_t
typedef int_least8_t int_fast8_t
typedef int16_t int_least16_t
typedef int32_t int_least32_t
typedef int8_t int_least8_t
typedef int32_t intmax_t
typedef ptrdiff_t intptr_t
typedef uint_least16_t uint_fast16_t
typedef uint_least32_t uint_fast32_t
typedef uint_least8_t uint_fast8_t
typedef uint16_t uint_least16_t
typedef uint32_t uint_least32_t
typedef uint8_t uint_least8_t
typedef uint32_t uintmax_t
include
assimp
Compiler

Macros
pushpack1.h File Reference
Macros

#define AI_PUSHPACK_IS_DEFINED
Macro Definition Documentation

#define AI_PUSHPACK_IS_DEFINED
**Classes** | **Typedefs**
quaternion.h File Reference

Quaternion structure, including operators when compiling in C++.  
[More...](#)
Classes

class aiMatrix3x3t<TReal>
  Represents a row-major 3x3 matrix. More...

class aiQuaterniont<TReal>
  Represents a quaternion in a 4D vector. More...

class aiVector3t<TReal>
  Represents a three-dimensional vector. More...
Typedefs

typedef aiQuaternion< float > aiQuaternion
Detailed Description

Quaternion structure, including operators when compiling in C++.
typedef aiQuaternion<br>&lt;float&gt; aiQuaternion
Macros

quaternion.inl File Reference

Inline implementation of aiQuaternion<TRel> operators.  More...
Macros

#define AI_QUATERNION_INL_INC
Detailed Description

Inline implementation of aiQuaternion<TReal> operators.
Macro Definition Documentation

#define AI_QUATERNION_INL_INC
include
assimp

Classes | Macros

scene.h File Reference

Defines the data structures in which the imported scene is returned. More...
**Classes**

**struct**  **aiNode**
A node in the imported hierarchy.  [More...](#)

**struct**  **aiScene**
The root structure of the imported data.  [More...](#)
Macros

#define **AI_SCENE_FLAGS_INCOMPLETE** 0x1
Specifies that the scene data structure that was imported is not complete.  More...

#define **AI_SCENE_FLAGS_NON_VERBOSE_FORMAT** 0x8
This flag is currently only set by the aiProcess_JoinIdenticalVertices step. More...

#define **AI_SCENE_FLAGS_TERRAIN** 0x10
Denotes pure height-map terrain data. More...

#define **AI_SCENE_FLAGS_VALIDATED** 0x2
This flag is set by the validation postprocess-step (aiPostProcess_ValidateDS) if the validation is successful. More...

#define **AI_SCENE_FLAGS_VALIDATION_WARNING** 0x4
This flag is set by the validation postprocess-step (aiPostProcess_ValidateDS) if the validation is successful but some issues have been found. More...
**Detailed Description**

Defines the data structures in which the imported scene is returned.
# Macro Definition Documentation

```c
#define AI_SCENE_FLAGS_INCOMPLETE  0x1
```

Specifies that the scene data structure that was imported is not complete.

This flag bypasses some internal validations and allows the import of animation skeletons, material libraries or camera animation paths using Assimp. Most applications won't support such data.

```c
#define AI_SCENE_FLAGS_NON_VERBOSE_FORMAT  0x8
```

This flag is currently only set by the aiProcess_JoinIdenticalVertices step.

It indicates that the vertices of the output meshes aren't in the internal verbose format anymore. In the verbose format all vertices are unique, no vertex is ever referenced by more than one face.

```c
#define AI_SCENE_FLAGS_TERRAIN  0x10
```

Denotes pure height-map terrain data.

Pure terrains usually consist of quads, sometimes triangles, in a regular grid. The x,y coordinates of all vertex positions refer to the x,y coordinates on the terrain height map, the z-axis stores the elevation at a specific point.

TER (Terragen) and HMP (3D Game Studio) are height map formats.

Note: Assimp is probably not the best choice for loading huge terrains - fully triangulated data takes extremely much free store and should be avoided as long as possible (typically you'll do the triangulation when you actually need to render it).

```c
#define AI_SCENE_FLAGS_VALIDATED  0x2
```

This flag is set by the validation postprocess-step (aiPostProcess.ValidateDS) if the validation is successful.
In a validated scene you can be sure that any cross references in the data structure (e.g. vertex indices) are valid.

#define AI_SCENE_FLAGS_VALIDATION_WARNING 0x4

This flag is set by the validation postprocess-step (aiPostProcess_ValidateDS) if the validation is successful but some issues have been found.

This can for example mean that a texture that does not exist is referenced by a material or that the bone weights for a vertex don't sum to 1.0 ... . In most cases you should still be able to use the import. This flag could be useful for applications which don't capture Assimp's log output.
texture.h File Reference

Defines texture helper structures for the library. More...
Classes

struct aiTexel
Helper structure to represent a texel in a ARGB8888 format. More...

struct aiTexture
Helper structure to describe an embedded texture. More...
Macros

#define AI_MAKE_EMBEDDED_TEXNAME(_n_) "*" # _n_

Used to build the reserved path name used by the material system to reference textures that are embedded into their corresponding model files. More...
Variables

struct aiTexel PACK_STRUCT
Detailed Description

Defines texture helper structures for the library.

Used for file formats which embed their textures into the model file. Supported are both normal textures, which are stored as uncompressed pixels, and "compressed" textures, which are stored in a file format such as PNG or TGA.
Macro Definition Documentation

#define AI_MAKE_EMBEDDED_TEXNAME(_n_) "*" #_n_

Used to build the reserved path name used by the material system to reference textures that are embedded into their corresponding model files.

The parameter specifies the index of the texture (zero-based, in the aiScene::mTextures array)
Variable Documentation

struct aiTexel PACK_STRUCT
Basic data types and primitives, such as vectors or colors. More...
Classes

struct **aiColor3D**
Represents a color in Red-Green-Blue space. [More...](#)

struct **aiMemoryInfo**
Stores the memory requirements for different components (e.g. [More...](#))

struct **aiPlane**
Represents a plane in a three-dimensional, euclidean space. [More...](#)

struct **aiRay**
Represents a ray. [More...](#)

struct **aiString**
Represents an UTF-8 string, zero byte terminated. [More...](#)
Namespaces

**Assimp**

Assimp's CPP-API and all internal APIs.
Enumerations
Variables

const size_t MAXLEN = 1024

Maximum dimension for strings, ASSIMP strings are zero terminated. More...

struct aiPlane PACK_STRUCT
Detailed Description

Basic data types and primitives, such as vectors or colors.
Enumeration Type Documentation

enum aiDefaultLogStream

Enumerates predefined log streaming destinations.

Logging to these streams can be enabled with a single call to 
#LogStream::createDefaultStream.

**Enumerator**

aiDefaultLogStream_FILE Stream the log to a file.

aiDefaultLogStream_STDOUT Stream the log to std::cout.

aiDefaultLogStream_STDERR Stream the log to std::cerr.

aiDefaultLogStream_DEBUGGER MSVC only: Stream the log the the 
debugger (this relies on OutputDebugString 
from the Win32 SDK)

enum aiOrigin

Seek origins (for the virtual file system API).

Much cooler than using SEEK_SET, SEEK_CUR or SEEK_END.

**Enumerator**

aiOrigin_SET Beginning of the file.

aiOrigin_CUR Current position of the file pointer.
End of the file, offsets must be negative.

enum aiReturn

Standard return type for some library functions.

Rarely used, and if, mostly in the C API.

**Enumerator**

aiReturn_SUCCESS Indicates that a function was successful.

aiReturn_FAILURE Indicates that a function failed.

aiReturn_OUTOFMEMORY Indicates that not enough memory was available to perform the requested operation.
Variable Documentation

const size_t MAXLEN = 1024

Maximum dimension for strings, ASSIMP strings are zero terminated.

struct aiColor3D PACK_STRUCT

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
• `include`
• `assimp`

**Classes** | **Typedefs** | **Variables**

**vector2.h** File Reference

2D vector structure, including operators when compiling in C++  More...
Classes

class aiVector2t<TReal>
    Represents a two-dimensional vector. More...
typedef aiVector2t< float > aiVector2D
Variables

class aiVector2t PACK_STRUCT
Detailed Description

2D vector structure, including operators when compiling in C++
Typedef Documentation

typedef aiVector2t<float> aiVector2D
Variable Documentation

class aiVector2t PACK_STRUCT

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
### vector2.inl File Reference

Inline implementation of \texttt{aiVector2t<TReal>} operators.

- **include**
- **assimp**

**Macros** | **Functions**
---

More...
Macros

#define AI_VECTOR2D_INL_INC
Functions

template<typename TReal>
TReal operator* (const aiVector2t<TReal> &v1, const aiVector2t<TReal> &v2)

template<typename TReal>
aiVector2t<TReal> operator* (TReal f, const aiVector2t<TReal> &v)

template<typename TReal>
aiVector2t<TReal> operator* (const aiVector2t<TReal> &v, TReal f)

template<typename TReal>
aiVector2t<TReal> operator+ (const aiVector2t<TReal> &v1, const aiVector2t<TReal> &v2)

template<typename TReal>
aiVector2t<TReal> operator- (const aiVector2t<TReal> &v1, const aiVector2t<TReal> &v2)

template<typename TReal>
aiVector2t<TReal> operator- (const aiVector2t<TReal> &v)

template<typename TReal>
aiVector2t<TReal> operator/ (const aiVector2t<TReal> &v, TReal f)

template<typename TReal>
aiVector2t<TReal> operator/ (const aiVector2t<TReal> &v, const aiVector2t<TReal> &v2)
Detailed Description

Inline implementation of aiVector2t<TReal> operators.
Macro Definition Documentation

#define AI_VECTOR2D_INL_INC
Function Documentation

template<typename TReal >
TReal operator* ( const aiVector2t< TReal > & v1,
    const aiVector2t< TReal > & v2 inline
)

template<typename TReal >
aiVector2t<TReal> operator* ( TReal f,
    const aiVector2t< TReal > & v inline
)

template<typename TReal >
aiVector2t<TReal> operator* ( const aiVector2t< TReal > & v,
    TReal f inline
)

template<typename TReal >
aiVector2t<TReal> operator+ ( const aiVector2t< TReal > & v1,
    const aiVector2t< TReal > & v2 inline
)

template<typename TReal >
aiVector2t<TReal> operator- ( const aiVector2t< TReal > & v1,
    const aiVector2t< TReal > & v2 inline
)

template<typename TReal >
aiVector2t<TReal> operator- ( const aiVector2t< TReal > & v ) inline

template<typename TReal >
aiVector2t<TReal> operator/ ( const aiVector2t< TReal > & v,
    TReal f inline
)

template<typename TReal >
aiVector2t<TReal> operator/ ( const aiVector2t< TReal > & v,
    const aiVector2t< TReal > & v2 inline
)
3D vector structure, including operators when compiling in C++  More...
Classes

class \texttt{aiMatrix3x3t< TReal >}

Represents a row-major 3x3 matrix. \texttt{More...}

class \texttt{aiMatrix4x4t< TReal >}

Represents a row-major 4x4 matrix, use this for homogeneous coordinates. \texttt{More...}

class \texttt{aiVector3t< TReal >}

Represents a three-dimensional vector. \texttt{More...}
Typedefs

typedef aiVector3t< float > aiVector3D
Variables

class aiVector3t PACK_STRUCT
Detailed Description

3D vector structure, including operators when compiling in C++
Typedef Documentation

typedef aiVector3t<float> aiVector3D
Variable Documentation

class aiVector3t PACK_STRUCT
Macros | Functions

vector3.inl File Reference

Inline implementation of aiVector3t<TReal> operators. More...
Macros

#define AI_VECTOR3D_INL_INC
Functions

template<typename TReal >
aiVector3t < TReal > operator* (const aiMatrix3x3t < TReal > &pMatrix, const aiVector3t < TReal > &pVector)
Transformation of a vector by a 3x3 matrix. More...

template<typename TReal >
aiVector3t < TReal > operator* (const aiMatrix4x4t < TReal > &pMatrix, const aiVector3t < TReal > &pVector)
Transformation of a vector by a 4x4 matrix. More...

template<typename TReal >
AI_FORCE_INLINE operator* (const aiVector3t < TReal > &v1, const TReal &v2)

template<typename TReal >
AI_FORCE_INLINE operator* (TReal f, const aiVector3t < TReal > &v)

template<typename TReal >
AI_FORCE_INLINE operator* (const aiVector3t < TReal > &v, TReal f)

template<typename TReal >
AI_FORCE_INLINE operator+ (const aiVector3t < TReal > &v1, const aiVector3t < TReal > &v2)

template<typename TReal >
AI_FORCE_INLINE operator- (const aiVector3t < TReal > &v1, const aiVector3t < TReal > &v2)
AI_FORCE_INLINE aiVector3t<TReal> operator-(const aiVector3t<TReal> &v)

template<typename TReal>
AI_FORCE_INLINE aiVector3t<TReal> operator/(const aiVector3t<TReal> &v, TReal f)

template<typename TReal>
AI_FORCE_INLINE operator/(const aiVector3t<TReal> &v, const aiVector3t<TReal> &v2)

template<typename TReal>
AI_FORCE_INLINE operator^(const aiVector3t<TReal> &v1, const aiVector3t<TReal> &v2)
Detailed Description

Inline implementation of aiVector3t<TReal> operators.
Macro Definition Documentation

#define AI_VECTOR3D_INL_INC
Function Documentation

template<
typename TReal>

\texttt{aiVector3t<TReal> operator\ast ( const aiMatrix3x3t<TReal> \& pMatrix, const aiVector3t<TReal> \& pVector \text{ inline })}

Transformation of a vector by a 3x3 matrix.

template<
typename TReal>

\texttt{aiVector3t<TReal> operator\ast ( const aiMatrix4x4t<TReal> \& pMatrix, const aiVector3t<TReal> \& pVector \text{ inline })}

Transformation of a vector by a 4x4 matrix.

template<
typename TReal>

\texttt{AI\_FORCE\_INLINE TReal operator\ast ( const aiVector3t<TReal> \& v1, const aiVector3t<TReal> \& v2 )}

\texttt{operator\ast}

\texttt{AI\_FORCE\_INLINE aiVector3t<TReal>( TReal f, const aiVector3t<TReal> \& v )}

\texttt{operator\ast}

\texttt{AI\_FORCE\_INLINE aiVector3t<TReal>( const aiVector3t<TReal> \& v, TReal f )}

\texttt{operator\ast}

\texttt{AI\_FORCE\_INLINE aiVector3t<TReal>( const aiVector3t<TReal> \& v1, \text{ inline } )}
const aiVector3t<TReal> & v2)

template<typename TReal >
AI_FORCE_INLINE aiVector3t<TReal>
operator-(
const aiVector3t<TReal> & v1,
const aiVector3t<TReal> & v2)

template<typename TReal >
AI_FORCE_INLINE aiVector3t<TReal>
operator/(TReal f)

template<typename TReal >
AI_FORCE_INLINE aiVector3t<TReal>
operator/(const aiVector3t<TReal> & v,
const aiVector3t<TReal> & v2)

template<typename TReal >
AI_FORCE_INLINE aiVector3t<TReal>
operator^(const aiVector3t<TReal> & v1,
const aiVector3t<TReal> & v2)

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp  v3.1.1 (June 2014)

- include
- assimp

**Macros** | **Functions**

version.h File Reference

Functions to query the version of the Assimp runtime, check compile flags, ...

[More...]
## Macros

#define **ASSIMP_CFLAGS_DEBUG** 0x4

Assimp was compiled as a debug build. [More...](#)

#define **ASSIMP_CFLAGS_NOBOOST** 0x8

Assimp was compiled with ASSIMP_BUILD_BOOST_WORKAROUND defined. [More...](#)

#define **ASSIMP_CFLAGS_SHARED** 0x1

Assimp was compiled as a shared object (Windows: DLL) [More...](#)

#define **ASSIMP_CFLAGS_SINGLETHREADED** 0x10

Assimp was compiled with ASSIMP_BUILD_SINGLETHREADED defined. [More...](#)

#define **ASSIMP_CFLAGS_STLPORT** 0x2

Assimp was compiled against STLport. [More...](#)
Functions

**ASSIMP_API**

unsigned int **aiGetCompileFlags** (void)

Returns assimp's compile flags. [More...](#)

**ASSIMP_API**

const char * **aiGetLegalString** (void)

Returns a string with legal copyright and licensing information about Assimp. [More...](#)

**ASSIMP_API**

unsigned int **aiGetVersionMajor** (void)

Returns the current major version number of Assimp. [More...](#)

**ASSIMP_API**

unsigned int **aiGetVersionMinor** (void)

Returns the current minor version number of Assimp. [More...](#)

**ASSIMP_API**

unsigned int **aiGetVersionRevision** (void)

Returns the repository revision of the Assimp runtime. [More...](#)
Detailed Description

Functions to query the version of the Assimp runtime, check compile flags, ...
Macro Definition Documentation

#define ASSIMP_CFLAGS_DEBUG 0x4

Assimp was compiled as a debug build.

#define ASSIMP_CFLAGS_NOBOOST 0x8

Assimp was compiled with ASSIMP_BUILD_BOOST_WORKAROUND defined.

#define ASSIMP_CFLAGS_SHARED 0x1

Assimp was compiled as a shared object (Windows: DLL)

#define ASSIMP_CFLAGS_SINGLETTHREADED 0x10

Assimp was compiled with ASSIMP_BUILD_SINGLETTHREADED defined.

#define ASSIMP_CFLAGS_STLPORT 0x2

Assimp was compiled against STLport.
Function Documentation

**ASSIMP_API** unsigned int aiGetCompileFlags ( void )

Returns assimp's compile flags.

Returns
Any bitwise combination of the ASSIMP_CFLAGS_xxx constants.

**ASSIMP_API** const char* aiGetLegalString ( void )

Returns a string with legal copyright and licensing information about Assimp.

The string may include multiple lines.

Returns
Pointer to static string.

**ASSIMP_API** unsigned int aiGetVersionMajor ( void )

Returns the current major version number of Assimp.

Returns
Major version of the Assimp runtime the application was linked/built against

**ASSIMP_API** unsigned int aiGetVersionMinor ( void )

Returns the current minor version number of Assimp.

Returns
Minor version of the Assimp runtime the application was linked/built against

**ASSIMP_API** unsigned int aiGetVersionRevision ( void )

Returns the repository revision of the Assimp runtime.
Returns

SVN Repository revision number of the Assimp runtime the application was linked/built against.
Here is a list of all file members with links to the files they belong to:

- __int8_t_defined : [pstdint.h](#)
- _aiAnimBehaviour_Force32Bit : [anim.h](#)
- _aiBlendMode_Force32Bit : [material.h](#)
- _aiComponent_Force32Bit : [config.h](#)
- _aiLightSource_Force32Bit : [light.h](#)
- _aiPrimitiveType_Force32Bit : [mesh.h](#)
- _aiShadingMode_Force32Bit : [material.h](#)
- _aiTextureFlags_Force32Bit : [material.h](#)
- _aiTextureMapMode_Force32Bit : [material.h](#)
- _aiTextureMapping_Force32Bit : [material.h](#)
- _aiTextureOp_Force32Bit : [material.h](#)
- _aiTextureType_Force32Bit : [material.h](#)
- _PSTDINT_H_INCLUDED : [pstdint.h](#)
Here is a list of all file members with links to the files they belong to:

- a -

- AI_AISTRING : [metadata.h](metadata.h)
- AI_AIVECTOR3D : [metadata.h](metadata.h)
- ai_assert : [ai_assert.h](ai_assert.h)
- AI_BOOL : [metadata.h](metadata.h)
- AI_COLOR4D_INL_INC : [color4.inl](color4.inl)
- AI_CONFIG_ANDROID_JNI_ASSIMP_MANAGER_SUPPORT : [config.h](config.h)
- AI_CONFIG_EXPORT_XFILE_64BIT : [config.h](config.h)
- AI_CONFIG_FAVOUR_SPEED : [config.h](config.h)
- AI_CONFIG_GLOB_MEASURE_TIME : [config.h](config.h)
- AI_CONFIG_IMPORT_AC_EVAL_SUBDIVISION : [config.h](config.h)
- AI_CONFIG_IMPORT_AC_SEPARATE_BFCULL : [config.h](config.h)
- AI_CONFIG_IMPORT_ASE_RECONSTRUCT_NORMALS : [config.h](config.h)
- AI_CONFIG_IMPORT_COLLADA_IGNORE_UP_DIRECTION : [config.h](config.h)
- AI_CONFIG_IMPORT_FBX_OPTIMIZE_EMPTY_ANIMATION_CURVES : [config.h](config.h)
- AI_CONFIG_IMPORT_FBX_PRESERVE_PIVOTS : [config.h](config.h)
- AI_CONFIG_IMPORT_FBX_READ_ALL_GEOMETRY_LAYERS : [config.h](config.h)
- AI_CONFIG_IMPORT_FBX_READ_ALL_MATERIALS : [config.h](config.h)
- AI_CONFIG_IMPORT_FBX_READ_ANIMATIONS : [config.h](config.h)
- AI_CONFIG_IMPORT_FBX_READ_CAMERAS : [config.h](config.h)
- AI_CONFIG_IMPORT_FBX_READ_LIGHTS : [config.h](config.h)
- AI_CONFIG_IMPORT_FBX_READ_MATERIALS : [config.h](config.h)
- AI_CONFIG_IMPORT_FBX_READ_TEXTURES : [config.h](config.h)
- AI_CONFIG_IMPORT_FBX.Strict_Mode: config.h
- AI_CONFIG_IMPORTGLOBAL_KEYFRAME: config.h
- AI_CONFIG_IMPORT_IFC_CUSTOM_TRIANGULATION: config.h
- AI_CONFIG_IMPORT_IFC_SKIP_CURVE_REPRESENTATIONS: config.h
- AI_CONFIG_IMPORT_IFC_SKIP_SPACE_REPRESENTATIONS: config.h
- AI_CONFIG_IMPORT_IRR_ANIM_FPS: config.h
- AI_CONFIG_IMPORT_LWO_ONE_LAYER_ONLY: config.h
- AI_CONFIG_IMPORT_LWS_ANIM_END: config.h
- AI_CONFIG_IMPORT_LWS_ANIM_START: config.h
- AI_CONFIG_IMPORT_MD2_KEYFRAME: config.h
- AI_CONFIG_IMPORT_MD3_HANDLE_MULTIPART: config.h
- AI_CONFIG_IMPORT_MD3_KEYFRAME: config.h
- AI_CONFIG_IMPORT_MD3_SHADER_SRC: config.h
- AI_CONFIG_IMPORT_MD3_SKIN_NAME: config.h
- AI_CONFIG_IMPORT_MD5_NO_ANIM_AUTOLOAD: config.h
- AI_CONFIG_IMPORT_MDC_KEYFRAME: config.h
- AI_CONFIG_IMPORT_MDL_COLORMAP: config.h
- AI_CONFIG_IMPORT_MDL_KEYFRAME: config.h
- AI_CONFIG_IMPORT_NO_SKELETON_MESHES: config.h
- AI_CONFIG_IMPORT_OGRE_MATERIAL_FILE: config.h
- AI_CONFIG_IMPORT_OGRE_TEXTURETYPE_FROM_FILENAME: config.h
- AI_CONFIG_IMPORT_SMD_KEYFRAME: config.h
- AI_CONFIG_IMPORT_TER_MAKE_UVS: config.h
- AI_CONFIG_IMPORT_UNREAL_HANDLE_FLAGS: config.h
- AI_CONFIG_IMPORT_UNREAL_KEYFRAME: config.h
- AI_CONFIG_PP_CT_MAX_SMOOTHING_ANGLE: config.h
- AI_CONFIG_PP_CT_TEXTURE_CHANNEL_INDEX: config.h
- AI_CONFIG_PP_DB_ALL_OR_NONE: config.h
- AI_CONFIG_PP_DB_THRESHOLD: config.h
- AI_CONFIG_PP_FD_REMOVE: config.h
- AI_CONFIG_PP_FID_ANIM_ACCURACY: config.h
- AI_CONFIG_PP_GSN_MAX_SMOOTHING_ANGLE: config.h
- AI_CONFIG_PP_ICL_PTCACHE_SIZE: config.h
- AI_CONFIG_PP_LBW_MAX_WEIGHTS: config.h
- AI_CONFIG_PP_OG_EXCLUDE_LIST: config.h
- AI_CONFIG_PP_PTV_ADD_ROOT_TRANSFORMATION: config.h
- AI_CONFIG_PP_PTV_KEEP_HIERARCHY : config.h
- AI_CONFIG_PP_PTV_NORMALIZE : config.h
- AI_CONFIG_PP_PTV_ROOT_TRANSFORMATION : config.h
- AI_CONFIG_PP_RRM_EXCLUDE_LIST : config.h
- AI_CONFIG_PP_RVC_FLAGS : config.h
- AI_CONFIG_PP_SBBC_MAX_BONES : config.h
- AI_CONFIG_PP_SBP_REMOVE : config.h
- AI_CONFIG_PP_SLM_TRIANGLE_LIMIT : config.h
- AI_CONFIG_PP_SLM_VERTEX_LIMIT : config.h
- AI_CONFIG_PP_TUV_EVALUATE : config.h
- AI_DEBONE_THRESHOLD : config.h
- AI_DEFAULT_MATERIAL_NAME : material.h
- AI_DEG_TO_RAD : defs.h
- AI_FALSE : cimport.h
- AI_FLOAT : metadata.h
- AI_FORCE_INLINE : defs.h
- AI_INT : metadata.h
- AI_LMW_MAX_WEIGHTS : config.h
- AI_MAKE_EMBEDDED_TEXNAME : texture.h
- AI_MAKE_MAGIC : BaseImporter.h
- AI_MATERIAL_INL_INC : material.inl
- AI_MATH_HALF_PI : defs.h
- AI_MATH_HALF_PI_F : defs.h
- AI_MATH_PI : defs.h
- AI_MATH_PI_F : defs.h
- AI_MATH_TWO_PI : defs.h
- AI_MATH_TWO_PI_F : defs.h
- AI_MATKEY_BLEND_FUNC : material.h
- AI_MATKEY_BUMPSCALING : material.h
- AI_MATKEY_COLOR_AMBIENT : material.h
- AI_MATKEY_COLOR_DIFFUSE : material.h
- AI_MATKEY_COLOR_EMISSIVE : material.h
- AI_MATKEY_COLOR_REFLECTIVE : material.h
- AI_MATKEY_COLOR_SPECULAR : material.h
- AI_MATKEY_COLOR_TRANSPARENT : material.h
- AI_MATKEY_ENABLE_WIREFRAME : material.h
- AI_MATKEY_GLOBAL_BACKGROUND_IMAGE : material.h
- AI_MATKEY_MAPPING : material.h
- AI_MATKEY_MAPPINGMODE_U : material.h
- AI_TRUE : cimport.h
- AI_UINT64 : metadata.h
- AI_UVTRAFO_ALL : config.h
- AI_UVTRAFO_ROTATION : config.h
- AI_UVTRAFO_SCALING : config.h
- AI_UVTRAFO_TRANSLATION : config.h
- AI_VECTOR2D_INL_INC : vector2.inl
- AI_VECTOR3D_INL_INC : vector3.inl
- AI_WONT_RETURN : defs.h
- AI_WONT_RETURN_SUFFIX : defs.h
- aiAnimBehaviour : anim.h
- aiAnimBehaviour_CONSTANT : anim.h
- aiAnimBehaviour_DEFAULT : anim.h
- aiAnimBehaviour_LINEAR : anim.h
- aiAnimBehaviour_REPEAT : anim.h
- aiApplyPostProcessing() : cimport.h
- aiAttachLogStream() : cimport.h
- aiBlendMode : material.h
- aiBlendMode_Additive : material.h
- aiBlendMode_Default : material.h
- aiBool : cimport.h
- aiColor4D : color4.h
- aiComponent : config.h
- aiComponent_ANIMATIONS : config.h
- aiComponent_BONEWEIGHTS : config.h
- aiComponent_CAMERAS : config.h
- aiComponent_COLORS : config.h
- aiComponent_COLORSn : config.h
- aiComponent_LIGHTS : config.h
- aiComponent_MATERIAlS : config.h
- aiComponent_MEShES : config.h
- aiComponent_NORMALS : config.h
- aiComponent_TANGENTS_AND_BITANGENTS : config.h
- aiComponent_TEXCOORDS : config.h
- aiComponent_TEXCOORDSn : config.h
- aiComponent_TEXTURES : config.h
- aiCopyScene() : cexport.h
- aiCreatePropertyStore() : cimport.h
- aiCreateQuaternionFromMatrix() : cimport.h
- `aiDecomposeMatrix()` : `cimport.h`
- `aiDefaultLogStream` : `types.h`
- `aiDefaultLogStream_DEBUGGER` : `types.h`
- `aiDefaultLogStream_FILE` : `types.h`
- `aiDefaultLogStream_STDERR` : `types.h`
- `aiDefaultLogStream_STDOUT` : `types.h`
- `aiDetachAllLogStreams()` : `cimport.h`
- `aiDetachLogStream()` : `cimport.h`
- `aiEnableVerboseLogging()` : `cimport.h`
- `aiExportScene()` : `cexport.h`
- `aiExportSceneEx()` : `cexport.h`
- `aiExportSceneToBlob()` : `cexport.h`
- `aiFileCloseProc` : `cfileio.h`
- `aiFileFlushProc` : `cfileio.h`
- `aiFileOpenProc` : `cfileio.h`
- `aiFileReadProc` : `cfileio.h`
- `aiFileSeek` : `cfileio.h`
- `aiFileTellProc` : `cfileio.h`
- `aiFileWriteProc` : `cfileio.h`
- `aiFreeScene()` : `cexport.h`
- `aiGetCompileFlags()` : `version.h`
- `aiGetErrorString()` : `cimport.h`
- `aiGetExportFormatCount()` : `cexport.h`
- `aiGetExportFormatDescription()` : `cexport.h`
- `aiGetExtensionList()` : `cimport.h`
- `aiGetImporterDesc()` : `importerdesc.h`
- `aiGetImportFormatCount()` : `cimport.h`
- `aiGetImportFormatDescription()` : `cimport.h`
- `aiGetLegalString()` : `version.h`
- `aiGetMaterialColor()` : `material.h`
- `aiGetMaterialFloat()` : `material.h`
- `aiGetMaterialFloatArray()` : `material.h`
- `aiGetMaterialInteger()` : `material.h`
- `aiGetMaterialIntegerArray()` : `material.h`
- `aiGetMaterialProperty()` : `material.h`
- `aiGetMaterialString()` : `material.h`
- `aiGetMaterialTexture()` : `material.h`
- `aiGetMaterialTextureCount()` : `material.h`
- `aiGetMaterialUVTransform()` : `material.h`
- aiGetMemoryRequirements(): *cimport.h*
- aiGetPredefinedLogStream(): *cimport.h*
- aiGetVersionMajor(): *version.h*
- aiGetVersionMinor(): *version.h*
- aiGetVersionRevision(): *version.h*
- aiIdentityMatrix3(): *cimport.h*
- aiIdentityMatrix4(): *cimport.h*
- aiImporterFlags: *importerdesc.h*
- aiImporterFlags_Experimental: *importerdesc.h*
- aiImporterFlags_LimitedSupport: *importerdesc.h*
- aiImporterFlags_SupportBinaryFlavour: *importerdesc.h*
- aiImporterFlags_SupportCompressedFlavour: *importerdesc.h*
- aiImporterFlags_SupportTextFlavour: *importerdesc.h*
- aiImportFile(): *cimport.h*
- aiImportFileEx(): *cimport.h*
- aiImportFileExWithProperties(): *cimport.h*
- aiImportFromFileMemory(): *cimport.h*
- aiImportFromFileMemoryWithProperties(): *cimport.h*
- aiIsExtensionSupported(): *cimport.h*
- aiLightSource_AMBIENT: *light.h*
- aiLightSource_DIRECTIONAL: *light.h*
- aiLightSource_POINT: *light.h*
- aiLightSource_SPOT: *light.h*
- aiLightSource_UNDEFINED: *light.h*
- aiLightSourceType: *light.h*
- aiLogStreamCallback: *cimport.h*
- aiMatrix3x3: *matrix3x3.h*
- aiMatrix4x4: *matrix4x4.h*
- aiMetadataType: *metadata.h*
- aiMultiplyMatrix3(): *cimport.h*
- aiMultiplyMatrix4(): *cimport.h*
- aiOrigin: *types.h*
- aiOrigin_CUR: *types.h*
- aiOrigin_END: *types.h*
- aiOrigin_SET: *types.h*
- aiPostProcessSteps: *postprocess.h*
- aiPrimitiveType: *mesh.h*
- aiPrimitiveType_LINE: *mesh.h*
- aiPrimitiveType_POINT: *mesh.h*
- aiPrimitiveType_POLYGON: mesh.h
- aiPrimitiveType_TRIANGLE: mesh.h
- aiProcess_CalcTangentSpace: postprocess.h
- aiProcess_ConvertToLeftHanded: postprocess.h
- aiProcess_Debone: postprocess.h
- aiProcess_FindDegenerates: postprocess.h
- aiProcess_FindInstances: postprocess.h
- aiProcess_FindInvalidData: postprocess.h
- aiProcess_FixInfacingNormals: postprocess.h
- aiProcess_FlipUVs: postprocess.h
- aiProcess_FlipWindingOrder: postprocess.h
- aiProcess_GenNormals: postprocess.h
- aiProcess_GenSmoothNormals: postprocess.h
- aiProcess_GenUVCoords: postprocess.h
- aiProcess_ImproveCacheLocality: postprocess.h
- aiProcess_JoinIdenticalVertices: postprocess.h
- aiProcess_LimitBoneWeights: postprocess.h
- aiProcess_MakeLeftHanded: postprocess.h
- aiProcess_OptimizeGraph: postprocess.h
- aiProcess_OptimizeMeshes: postprocess.h
- aiProcess_PreTransformVertices: postprocess.h
- aiProcess_RemoveComponent: postprocess.h
- aiProcess_RemoveRedundantMaterials: postprocess.h
- aiProcess_SortByPType: postprocess.h
- aiProcess_SplitByBoneCount: postprocess.h
- aiProcess_SplitLargeMeshes: postprocess.h
- aiProcess_TransformUVCoords: postprocess.h
- aiProcess_Triangulate: postprocess.h
- aiProcess_VerifyDataStructure: postprocess.h
- aiProcessPreset_TargetRealtime_Fast: postprocess.h
- aiProcessPreset_TargetRealtime_MaxQuality: postprocess.h
- aiProcessPreset_TargetRealtime_Quality: postprocess.h
- aiQuaternion: quaternion.h
- aiReleaseExportBlob(): cexport.h
- aiReleaseExportFormatDescription(): cexport.h
- aiReleaseImport(): cimport.h
- aiReleasePropertyStore(): cimport.h
- aiReturn: types.h
- aiReturn_FAILURE: types.h
- aiReturn_OUTOFMEMORY : types.h
- aiReturn_SUCCESS : types.h
- aiSetImportPropertyFloat() : cimport.h
- aiSetImportPropertyInteger() : cimport.h
- aiSetImportPropertyMatrix() : cimport.h
- aiSetImportPropertyString() : cimport.h
- aiShadingMode : material.h
- aiShadingMode_Blinn : material.h
- aiShadingMode_CookTorrance : material.h
- aiShadingMode_Flat : material.h
- aiShadingMode_Fresnel : material.h
- aiShadingMode_Gouraud : material.h
- aiShadingMode_Minnaert : material.h
- aiShadingMode_NoShading : material.h
- aiShadingMode_OrenNayar : material.h
- aiShadingMode_Phung : material.h
- aiShadingMode_Toon : material.h
- aiTextureFlags : material.h
- aiTextureFlags_IgnoreAlpha : material.h
- aiTextureFlags_Invert : material.h
- aiTextureFlags_UseAlpha : material.h
- aiTextureMapMode : material.h
- aiTextureMapMode_Clamp : material.h
- aiTextureMapMode_Decal : material.h
- aiTextureMapMode_Mirror : material.h
- aiTextureMapMode_Wrap : material.h
- aiTextureMapping : material.h
- aiTextureMapping_BOX : material.h
- aiTextureMapping_CYLINDER : material.h
- aiTextureMapping_OTHER : material.h
- aiTextureMapping_PLANE : material.h
- aiTextureMapping_SPHERE : material.h
- aiTextureMapping_UV : material.h
- aiTextureOp : material.h
- aiTextureOp_Add : material.h
- aiTextureOp_Divide : material.h
- aiTextureOp_Multiply : material.h
- aiTextureOp_SignedAdd : material.h
- aiTextureOp_SmoothAdd : material.h
- aiTextureOp_Subtract: `material.h`
- aiTextureType: `material.h`
- aiTextureType_AMBIENT: `material.h`
- aiTextureType_DIFFUSE: `material.h`
- aiTextureType_DISPLACEMENT: `material.h`
- aiTextureType_EMISSIVE: `material.h`
- aiTextureType_HEIGHT: `material.h`
- aiTextureType_LIGHTMAP: `material.h`
- aiTextureType_NONE: `material.h`
- aiTextureType_NORMALS: `material.h`
- aiTextureType_OPACITY: `material.h`
- aiTextureType_REFLECTION: `material.h`
- aiTextureType_SHININESS: `material.h`
- aiTextureType_SPECULAR: `material.h`
- aiTextureType_UNKNOWN: `material.h`
- aiTransformVecByMatrix3(): `cimport.h`
- aiTransformVecByMatrix4(): `cimport.h`
- aiTransposeMatrix3(): `cimport.h`
- aiTransposeMatrix4(): `cimport.h`
- aiUserData: `cfileio.h`
- aiVector2D: `vector2.h`
- aiVector3D: `vector3.h`
- ASSIMP_API: `defs.h`
- ASSIMP_API_WINONLY: `defs.h`
- ASSIMP_BUILD_DEBUG: `defs.h`
- ASSIMP_BUILD_NEED_UNZIP: `defs.h`
- ASSIMP_BUILD_NEED_Z_INFLATE: `defs.h`
- ASSIMP_BUILD_SINGLETHREADED: `defs.h`
- ASSIMP_CFLAGS_DEBUG: `version.h`
- ASSIMP_CFLAGS_NOBOOST: `version.h`
- ASSIMP_CFLAGS_SHARED: `version.h`
- ASSIMP_CFLAGS_SINGLETHEADED: `version.h`
- ASSIMP_CFLAGS_STLPORT: `version.h`
- ASSIMP_DEFAULT_LOG_NAME: `DefaultLogger.hpp`
Here is a list of all file members with links to the files they belong to:

- C -

- **C_ENUM**: [defs.h](#)
- **C_STRUCT**: [defs.h](#)
Here is a list of all file members with links to the files they belong to:

- **f** -

  - FORCE_32BIT : [metadata.h](#)

---

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by [doxygen](#) 1.8.11*
Here is a list of all file members with links to the files they belong to:

- **g** -

  - GetAiType() : [metadata.h](#)
Here is a list of all file members with links to the files they belong to:

- i -

- INT16_MAX : [pstdint.h](#)
- INT16_MIN : [pstdint.h](#)
- INT32_MAX : [pstdint.h](#)
- INT32_MIN : [pstdint.h](#)
- INT8_MAX : [pstdint.h](#)
- INT8_MIN : [pstdint.h](#)
- INT_FAST16_MAX : [pstdint.h](#)
- INT_FAST16_MIN : [pstdint.h](#)
- int_fast16_t : [pstdint.h](#)
- INT_FAST32_MAX : [pstdint.h](#)
- INT_FAST32_MIN : [pstdint.h](#)
- int_fast32_t : [pstdint.h](#)
- INT_FAST8_MAX : [pstdint.h](#)
- INT_FAST8_MIN : [pstdint.h](#)
- int_fast8_t : [pstdint.h](#)
- INT_LEAST16_MAX : [pstdint.h](#)
- INT_LEAST16_MIN : [pstdint.h](#)
- int_least16_t : [pstdint.h](#)
- INT_LEAST32_MAX : [pstdint.h](#)
- INT_LEAST32_MIN : [pstdint.h](#)
- int_least32_t : [pstdint.h](#)
- INT_LEAST8_MAX : [pstdint.h](#)
- INT_LEAST8_MIN : [pstdint.h](#)
- int_least8_t : [pstdint.h](#)
- INTMAX_C : [pstdint.h](#)
- INTMAX_MAX : [pstdint.h](#)
- intmax_t: `pstdint.h`
- intptr_t: `pstdint.h`
Here is a list of all file members with links to the files they belong to:

- \texttt{m} -

- \texttt{MAX\_LOG\_MESSAGE\_LENGTH} : \texttt{Logger.hpp}
- \texttt{MAXLEN} : \texttt{types.h}
Here is a list of all file members with links to the files they belong to:

- operator+() : [vector3.inl](#), [color4.inl](#), [vector2.inl](#)
- operator-() : [color4.inl](#), [vector2.inl](#), [vector3.inl](#)
- operator/() : [vector2.inl](#), [color4.inl](#), [vector3.inl](#), [vector2.inl](#), [color4.inl](#), [vector3.inl](#)
- operator^() : [vector3.inl](#)
Here is a list of all file members with links to the files they belong to:

- **p** -

- PACK_STRUCT: [color4.h](#), [material.h](#), [matrix4x4.h](#), [vector3.h](#), [texture.h](#), [matrix3x3.h](#), [types.h](#), [vector2.h](#)
- PP_ICL_PTCACHE_SIZE: [config.h](#)
- PRINTF_INT16_DEC_WIDTH: [pstdint.h](#)
- PRINTF_INT16_HEX_WIDTH: [pstdint.h](#)
- PRINTF_INT32_DEC_WIDTH: [pstdint.h](#)
- PRINTF_INT32_HEX_WIDTH: [pstdint.h](#)
- PRINTF_INT64_DEC_WIDTH: [pstdint.h](#)
- PRINTF_INT64_HEX_WIDTH: [pstdint.h](#)
- PRINTF_INT8_DEC_WIDTH: [pstdint.h](#)
- PRINTF_INT8_HEX_WIDTH: [pstdint.h](#)
- PRINTF_INTMAX_DEC_WIDTH: [pstdint.h](#)
- PRINTF_INTMAX_HEX_WIDTH: [pstdint.h](#)
- PRINTF_INTMAX_MODIFIER: [pstdint.h](#)
- PRINTF_LEAST16_MODIFIER: [pstdint.h](#)
- PRINTF_LEAST32_MODIFIER: [pstdint.h](#)

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by [doxygen](#) 1.8.11*
Here is a list of all file members with links to the files they belong to:

- S -

- SIG_ATOMIC_MAX : [pstdint.h](pstdint.h)
- SIZE_MAX : [pstdint.h](pstdint.h)
- STDINT_H_UINTPTR_T_DEFINED : [pstdint.h](pstdint.h)
Here is a list of all file members with links to the files they belong to:

- **u** -

- UINT16_MAX : [pstdint.h](pstdint.h)
- UINT32_MAX : [pstdint.h](pstdint.h)
- UINT8_MAX : [pstdint.h](pstdint.h)
- UINT_FAST16_MAX : [pstdint.h](pstdint.h)
- uint_fast16_t : [pstdint.h](pstdint.h)
- UINT_FAST32_MAX : [pstdint.h](pstdint.h)
- uint_fast32_t : [pstdint.h](pstdint.h)
- UINT_FAST8_MAX : [pstdint.h](pstdint.h)
- uint_fast8_t : [pstdint.h](pstdint.h)
- UINT_LEAST16_MAX : [pstdint.h](pstdint.h)
- uint_least16_t : [pstdint.h](pstdint.h)
- UINT_LEAST32_MAX : [pstdint.h](pstdint.h)
- uint_least32_t : [pstdint.h](pstdint.h)
- UINT_LEAST8_MAX : [pstdint.h](pstdint.h)
- uint_least8_t : [pstdint.h](pstdint.h)
- UINTMAX_C : [pstdint.h](pstdint.h)
- UINTMAX_MAX : [pstdint.h](pstdint.h)
- uintmax_t : [pstdint.h](pstdint.h)
- ULONG_LONG_MAX : [pstdint.h](pstdint.h)
- a -

- aiApplyPostProcessing() : cimport.h
- aiAttachLogStream() : cimport.h
- aiCopyScene() : cexport.h
- aiCreatePropertyStore() : cimport.h
- aiCreateQuaternionFromMatrix() : cimport.h
- aiDecomposeMatrix() : cimport.h
- aiDetachAllLogStreams() : cimport.h
- aiDetachLogStream() : cimport.h
- aiEnableVerboseLogging() : cimport.h
- aiExportScene() : cexport.h
- aiExportSceneEx() : cexport.h
- aiExportSceneToBlob() : cexport.h
- aiFreeScene() : cexport.h
- aiGetCompileFlags() : version.h
- aiGetErrorString() : cimport.h
- aiGetExportFormatCount() : cexport.h
- aiGetExportFormatDescription() : cexport.h
- aiGetExtensionList() : cimport.h
- aiGetImporterDesc() : importerdesc.h
- aiGetImportFormatCount() : cimport.h
- aiGetImportFormatDescription() : cimport.h
- aiGetLegalString() : version.h
- aiGetMaterialColor() : material.h
- aiGetMaterialFloat() : material.h
- aiGetMaterialFloatArray() : material.h
- aiGetMaterialInteger() : material.h
• aiGetMaterialIntegerArray() : material.h
• aiGetMaterialProperty() : material.h
• aiGetMaterialString() : material.h
• aiGetMaterialTexture() : material.h
• aiGetMaterialTextureCount() : material.h
• aiGetMaterialUVTransform() : material.h
• aiGetMemoryRequirements() : cimport.h
• aiGetPredefinedLogStream() : cimport.h
• aiGetVersionMajor() : version.h
• aiGetVersionMinor() : version.h
• aiGetVersionRevision() : version.h
• aiIdentityMatrix3() : cimport.h
• aiIdentityMatrix4() : cimport.h
• aiImportFile() : cimport.h
• aiImportFileEx() : cimport.h
• aiImportFileExWithProperties() : cimport.h
• aiImportFileFromMemory() : cimport.h
• aiImportFileFromMemoryWithProperties() : cimport.h
• aiIsExtensionSupported() : cimport.h
• aiMultiplyMatrix3() : cimport.h
• aiMultiplyMatrix4() : cimport.h
• aiReleaseExportBlob() : cexport.h
• aiReleaseExportFormatDescription() : cexport.h
• aiReleaseImport() : cimport.h
• aiReleasePropertyStore() : cimport.h
• aiSetImportPropertyFloat() : cimport.h
• aiSetImportPropertyInteger() : cimport.h
• aiSetImportPropertyMatrix() : cimport.h
• aiSetImportPropertyString() : cimport.h
• aiTransformVecByMatrix3() : cimport.h
• aiTransformVecByMatrix4() : cimport.h
• aiTransposeMatrix3() : cimport.h
• aiTransposeMatrix4() : cimport.h

- g -

• GetAiType() : metadata.h
- o -

- operator*(): color4.inl, vector3.inl, vector2.inl, color4.inl, vector2.inl, 
  vector3.inl, vector2.inl
- operator+(): color4.inl, vector2.inl, vector3.inl, color4.inl
- operator-(): color4.inl, vector2.inl, vector3.inl
- operator/(): color4.inl, vector2.inl, vector3.inl
- operator^(): vector3.inl

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
• MAXLEN: types.h
• PACK_STRUCT: color4.h, material.h, matrix3x3.h, matrix4x4.h, texture.h, types.h, vector2.h, vector3.h

Generated on Sun Feb 21 2016 19:42:29 for Assimp by 1.8.11
- a -

- aiBool : cimport.h
- aiColor4D : color4.h
- aiFileCloseProc : cfileio.h
- aiFileFlushProc : cfileio.h
- aiFileOpenProc : cfileio.h
- aiFileReadProc : cfileio.h
- aiFileSeek : cfileio.h
- aiFileTellProc : cfileio.h
- aiFileWriteProc : cfileio.h
- aiLogStreamCallback : cimport.h
- aiMatrix3x3 : matrix3x3.h
- aiMatrix4x4 : matrix4x4.h
- aiQuaternion : quaternion.h
- aiUserData : cfileio.h
- aiVector2D : vector2.h
- aiVector3D : vector3.h

- i -

- int_fast16_t : pstdint.h
- int_fast32_t : pstdint.h
- int_fast8_t : pstdint.h
- int_least16_t : pstdint.h
- int_least32_t : pstdint.h
- int_least8_t : pstdint.h
- intmax_t : pstdint.h
- u -
  - intptr_t : pstdint.h
  - uint_fast16_t : pstdint.h
  - uint_fast32_t : pstdint.h
  - uint_fast8_t : pstdint.h
  - uint_least16_t : pstdint.h
  - uint_least32_t : pstdint.h
  - uint_least8_t : pstdint.h
  - uintmax_t : pstdint.h
• aiAnimBehaviour : anim.h
• aiBlendMode : material.h
• aiComponent : config.h
• aiDefaultLogStream : types.h
• aiImporterFlags : importerdesc.h
• aiLightSourceType : light.h
• aiMetadataType : metadata.h
• aiOrigin : types.h
• aiPostProcessSteps : postprocess.h
• aiPrimitiveType : mesh.h
• aiReturn : types.h
• aiShadingMode : material.h
• aiTextureFlags : material.h
• aiTextureMapMode : material.h
• aiTextureMapping : material.h
• aiTextureOp : material.h
• aiTextureType : material.h
- _ _ -

- _aiAnimBehaviour_Force32Bit : anim.h
  _aiBlendMode_Force32Bit : material.h
  _aiComponent_Force32Bit : config.h
  _aiLightSource_Force32Bit : light.h
  _aiPrimitiveType_Force32Bit : mesh.h
  _aiShadingMode_Force32Bit : material.h
  _aiTextureFlags_Force32Bit : material.h
  _aiTextureMapMode_Force32Bit : material.h
  _aiTextureMapping_Force32Bit : material.h
  _aiTextureOp_Force32Bit : material.h
  _aiTextureType_Force32Bit : material.h

- a -

- AI_AISTRING : metadata.h
  AI_AIVECTOR3D : metadata.h
  AI_BOOL : metadata.h
  AI_FLOAT : metadata.h
  AI_INT : metadata.h
  AI_UINT64 : metadata.h
  aiAnimBehaviour_CONSTANT : anim.h
  aiAnimBehaviour_DEFAULT : anim.h
  aiAnimBehaviour_LINEAR : anim.h
  aiAnimBehaviour_REPEAT : anim.h
  aiBlendMode_Additive : material.h
  aiBlendMode_Default : material.h
- aiComponent_ANIMATIONS : config.h
- aiComponent_BONEWEIGHTS : config.h
- aiComponent_CAMERAS : config.h
- aiComponent_COLORS : config.h
- aiComponent_LIGHTS : config.h
- aiComponent_MATERIALS : config.h
- aiComponent_MESHES : config.h
- aiComponent_NORMALS : config.h
- aiComponent_TANGENTS_AND_BITANGENTS : config.h
- aiComponent_TEXCOORDS : config.h
- aiComponent_TEXTURES : config.h
- aiDefaultLogStream_DEBUGGER : types.h
- aiDefaultLogStream_FILE : types.h
- aiDefaultLogStream_STDERR : types.h
- aiDefaultLogStream_STDOUT : types.h
- aiImporterFlags_Experimental : importerdesc.h
- aiImporterFlags_LimitedSupport : importerdesc.h
- aiImporterFlags_SupportBinaryFlavour : importerdesc.h
- aiImporterFlags_SupportCompressedFlavour : importerdesc.h
- aiImporterFlags_SupportTextFlavour : importerdesc.h
- aiLightSource_AMBIENT : light.h
- aiLightSource_DIRECTIONAL : light.h
- aiLightSource_POINT : light.h
- aiLightSource_SPOT : light.h
- aiLightSource_UNDEFINED : light.h
- aiOrigin_CUR : types.h
- aiOrigin_END : types.h
- aiOrigin_SET : types.h
- aiPrimitiveType_LINE : mesh.h
- aiPrimitiveType_POINT : mesh.h
- aiPrimitiveType_POLYGON : mesh.h
- aiPrimitiveType_TRIANGLE : mesh.h
- aiProcess_CalcTangentSpace : postprocess.h
- aiProcess_Debone : postprocess.h
- aiProcess_FindDegenerates : postprocess.h
- aiProcess_FindInstances : postprocess.h
- aiProcess_FindInvalidData : postprocess.h
- aiProcess_FixInfacingNormals : postprocess.h
- aiProcess_FlipUvs : postprocess.h
• aiProcess_FlipWindingOrder : postprocess.h
• aiProcess_GenNormals : postprocess.h
• aiProcess_GenSmoothNormals : postprocess.h
• aiProcess_GenUVCoords : postprocess.h
• aiProcess_ImproveCacheLocality : postprocess.h
• aiProcess_JoinIdenticalVertices : postprocess.h
• aiProcess_LimitBoneWeights : postprocess.h
• aiProcess_MakeLeftHanded : postprocess.h
• aiProcess_OptimizeGraph : postprocess.h
• aiProcess_OptimizeMeshes : postprocess.h
• aiProcess_PreTransformVertices : postprocess.h
• aiProcess_RemoveComponent : postprocess.h
• aiProcess_RemoveRedundantMaterials : postprocess.h
• aiProcess_SortByPType : postprocess.h
• aiProcess_SplitByBoneCount : postprocess.h
• aiProcess_SplitLargeMeshes : postprocess.h
• aiProcess_TransformUVCoords : postprocess.h
• aiProcess_Triangulate : postprocess.h
• aiProcess.ValidateDataStructure : postprocess.h
• aiReturn_FAILURE : types.h
• aiReturn_OUTOFMEMORY : types.h
• aiReturn_SUCCESS : types.h
• aiShadingMode_Blinn : material.h
• aiShadingMode_CookTorrance : material.h
• aiShadingMode_Flat : material.h
• aiShadingMode_Fresnel : material.h
• aiShadingMode_Gouraud : material.h
• aiShadingMode_Minnaert : material.h
• aiShadingMode_NoShading : material.h
• aiShadingMode_OrenNayar : material.h
• aiShadingMode_Phong : material.h
• aiShadingMode_Toon : material.h
• aiTextureFlags_IgnoreAlpha : material.h
• aiTextureFlags_Invert : material.h
• aiTextureFlags_UseAlpha : material.h
• aiTextureMapMode_Clamp : material.h
• aiTextureMapMode_Decal : material.h
• aiTextureMapMode_Mirror : material.h
• aiTextureMapMode_Wrap : material.h
-  f  -

- aiTextureMapping_BOX : material.h
- aiTextureMapping_CYLINDER : material.h
- aiTextureMapping_OTHER : material.h
- aiTextureMapping_PLANE : material.h
- aiTextureMapping_SPHERE : material.h
- aiTextureMapping_UV : material.h
- aiTextureOp_Add : material.h
- aiTextureOp_Divide : material.h
- aiTextureOp_Multiply : material.h
- aiTextureOp_SignedAdd : material.h
- aiTextureOp_SmoothAdd : material.h
- aiTextureOp_Subtract : material.h
- aiTextureType_AMBIENT : material.h
- aiTextureType_DIFFUSE : material.h
- aiTextureType_DISPLACEMENT : material.h
- aiTextureType_EMISSIVE : material.h
- aiTextureType_HEIGHT : material.h
- aiTextureType_LIGHTMAP : material.h
- aiTextureTypeNONE : material.h
- aiTextureType_NORMALS : material.h
- aiTextureType_OPACITY : material.h
- aiTextureType_REFLECTION : material.h
- aiTextureType_SHININESS : material.h
- aiTextureType_SPECULAR : material.h
- aiTextureTypeUNKNOWN : material.h

- f -

- FORCE_32BIT : metadata.h
__int8_t_defined : pstdint.h
__PSTDINT_H_INCLUDED : pstdint.h
- a -

- ai_assert : ai_assert.h
- AI_COLOR4D_INL_INC : color4.inl
- AI_CONFIG_ANDROID_JNI_ASSIMP_MANAGER_SUPPORT : config.h
- AI_CONFIG_EXPORT_XFILE_64BIT : config.h
- AI_CONFIG_FAVOUR_SPEED : config.h
- AI_CONFIG_GLOB_MEASURE_TIME : config.h
- AI_CONFIG_IMPORT_AC_EVAL_SUBDIVISION : config.h
- AI_CONFIG_IMPORT_AC_SEPARATE_BFCULL : config.h
- AI_CONFIG_IMPORT_ASE_RECONSTRUCT_NORMALS : config.h
- AI_CONFIG_IMPORT_COLLADA_IGNORE_UP_DIRECTION : config.h
- AI_CONFIG_IMPORT_FBX_OPTIMIZE_EMPTY_ANIMATION_CURVES : config.h
- AI_CONFIG_IMPORT_FBX_PRESERVE_PIVOTS : config.h
- AI_CONFIG_IMPORT_FBX_READ_ALL_GEOMETRY_LAYERS : config.h
- AI_CONFIG_IMPORT_FBX_READ_ALL_MATERIALS : config.h
- AI_CONFIG_IMPORT_FBX_READ_ANIMATIONS : config.h
- AI_CONFIG_IMPORT_FBX_READ_CAMERAS : config.h
- AI_CONFIG_IMPORT_FBX_READ_LIGHTS : config.h
- AI_CONFIG_IMPORT_FBX_READ_MATERIALS : config.h
- AI_CONFIG_IMPORT_FBX_READ_TEXTURES : config.h
- AI_CONFIG_IMPORT_FBX STRICT_MODE : config.h
- AI_CONFIG_IMPORT_GLOBAL_KEYFRAME : config.h
- AI_CONFIG_IMPORT_IFC_CUSTOM_TRIANGULATION : config.h
- AI_CONFIG_IMPORT_IFC_SKIP_CURVE_REPRESENTATIONS: config.h
- AI_CONFIG_IMPORT_IFC_SKIP_SPACE_REPRESENTATIONS: config.h
- AI_CONFIG_IMPORT_IRR_ANIM_FPS: config.h
- AI_CONFIG_IMPORT_LWO_ONE_LAYER_ONLY: config.h
- AI_CONFIG_IMPORT_LWS_ANIM_END: config.h
- AI_CONFIG_IMPORT_LWS_ANIM_START: config.h
- AI_CONFIG_IMPORT_MD2_KEYFRAME: config.h
- AI_CONFIG_IMPORT_MD3_HANDLE_MULTIPART: config.h
- AI_CONFIG_IMPORT_MD3_KEYFRAME: config.h
- AI_CONFIG_IMPORT_MD3_SHADER_SRC: config.h
- AI_CONFIG_IMPORT_MD3_SKIN_NAME: config.h
- AI_CONFIG_IMPORT_MD5_NO_ANIM_AUTOLOAD: config.h
- AI_CONFIG_IMPORT_MDC_KEYFRAME: config.h
- AI_CONFIG_IMPORT_MDL_COLORMAP: config.h
- AI_CONFIG_IMPORT_MDL_KEYFRAME: config.h
- AI_CONFIG_IMPORT_NO_SKELETON_MESHES: config.h
- AI_CONFIG_IMPORT_OGRE_MATERIAL_FILE: config.h
- AI_CONFIG_IMPORT_OGRE_TEXTURETYPE_FROM_FILENAME: config.h
- AI_CONFIG_IMPORT_SMD_KEYFRAME: config.h
- AI_CONFIG_IMPORT_TER_MAKE_UVS: config.h
- AI_CONFIG_IMPORT_UNREAL_HANDLE_FLAGS: config.h
- AI_CONFIG_IMPORT_UNREAL_KEYFRAME: config.h
- AI_CONFIG_PP_CT_MAX_SMOOTHING_ANGLE: config.h
- AI_CONFIG_PP_CT_TEXTURE_CHANNEL_INDEX: config.h
- AI_CONFIG_PP_DB_ALL_OR_NONE: config.h
- AI_CONFIG_PP_DB_THRESHOLD: config.h
- AI_CONFIG_PP_FD_REMOVE: config.h
- AI_CONFIG_PP_FID_ANIM_ACCURACY: config.h
- AI_CONFIG_PP_GSN_MAX_SMOOTHING_ANGLE: config.h
- AI_CONFIG_PP_ICL_PTCACHE_SIZE: config.h
- AI_CONFIG_PP_LBW_MAX_WEIGHTS: config.h
- AI_CONFIG_PP_OG_EXCLUDE_LIST: config.h
- AI_CONFIG_PP_PTV_ADD_ROOT_TRANSFORMATION: config.h
- AI_CONFIG_PP_PTV_KEEP_HIERARCHY: config.h
- AI_CONFIG_PP_PTV_NORMALIZE: config.h
- AI_CONFIG_PP_PTV_ROOT_TRANSFORMATION: config.h
- AI_CONFIG_PP_RRM_EXCLUDE_LIST: config.h
- AI_CONFIG_PP_RVC_FLAGS: config.h
- AI_CONFIG_PP_SBBC_MAX_BONES: config.h
- AI_CONFIG_PP_SBP_REMOVE: config.h
- AI_CONFIG_PP_SLM_TRIANGLE_LIMIT: config.h
- AI_CONFIG_PP_SLM_VERTEX_LIMIT: config.h
- AI_CONFIG_PP_TUV_EVALUATE: config.h
- AI_DEBONE_THRESHOLD: config.h
- AI_DEFAULT_MATERIAL_NAME: material.h
- AI_DEG_TO_RAD: defs.h
- AI_FALSE: cimport.h
- AI_FORCEINLINE: defs.h
- AI_LMW_MAX_WEIGHTS: config.h
- AI_MAKE_EMBEDDED_TEXNAME: texture.h
- AI_MAKE_MAGIC: BaseImporter.h
- AI_MATERIAL_INL_INC: material.inl
- AI_MATH_HALF_PI: defs.h
- AI_MATH_HALF_PI_F: defs.h
- AI_MATH_PI: defs.h
- AI_MATH_PI_F: defs.h
- AI_MATH_TWO_PI: defs.h
- AI_MATH_TWO_PI_F: defs.h
- AI_MATKEY_BLEND_FUNC: material.h
- AI_MATKEY_BUMPSCALING: material.h
- AI_MATKEY_COLOR_AMBIENT: material.h
- AI_MATKEY_COLOR_DIFFUSE: material.h
- AI_MATKEY_COLOR_EMISSIVE: material.h
- AI_MATKEY_COLOR_REFLECTIVE: material.h
- AI_MATKEY_COLOR_SPECULAR: material.h
- AI_MATKEY_COLOR_TRANSPARENT: material.h
- AI_MATKEY_ENABLE_WIREFRAME: material.h
- AI_MATKEY_GLOBAL_BACKGROUND_IMAGE: material.h
- AI_MATKEY_MAPPABLE: material.h
- AI_MATKEY_MAPPABLEMODE_U: material.h
- AI_MATKEY_MAPPABLEMODE_V: material.h
- AI_MATKEY_NAME: material.h
- AI_MATKEY_OPACITY: material.h
- AI_MATKEY_REFLECTIVITY: material.h
- AI_MATKEY_REFRACTIVITY: material.h
- `AI_MATKEY_SHADING_MODEL`: `material.h`
- `AI_MATKEY_SHININESS`: `material.h`
- `AI_MATKEY_SHININESS_STRENGTH`: `material.h`
- `AI_MATKEY_TEXBLEND`: `material.h`
- `AI_MATKEY_TEXFLAGS`: `material.h`
- `AI_MATKEY_TEXMAP_AXIS`: `material.h`
- `AI_MATKEY_TEXOP`: `material.h`
- `AI_MATKEY_TEXTURE`: `material.h`
- `AI_MATKEY_TWOSIDED`: `material.h`
- `AI_MATKEY_UVTRANSFORM`: `material.h`
- `AI_MATKEY_UVWSRC`: `material.h`
- `AI_MATRIX3x3_INL_INC`: `matrix3x3.inl`
- `AI_MATRIX4x4_INL_INC`: `matrix4x4.inl`
- `AI_MAX_ALLOC`: `defs.h`
- `AI_MAX_BONE_WEIGHTS`: `mesh.h`
- `AI_MAX_FACE_INDICES`: `mesh.h`
- `AI_MAX_FACES`: `mesh.h`
- `AI_MAX_NUMBER_OF_COLOR_SETS`: `mesh.h`
- `AI_MAX_NUMBER_OF_TEXTURECOORDS`: `mesh.h`
- `AI_MAX_VERTICES`: `mesh.h`
- `AI_PRIMITIVE_TYPE_FOR_N_INDICES`: `mesh.h`
- `AI_PROPERTY_WAS_NOT_EXISTING`: `Importer.hpp`
- `AI_PUSHPACK_IS_DEFINED`: `pushpack1.h`
- `AI_QUATERNION_INL_INC`: `quaternion.inl`
- `AI_RAD_TO_DEG`: `defs.h`
- `AI_SBBC_DEFAULT_MAX_BONES`: `config.h`
- `AI_SCENE_FLAGS_INCOMPLETE`: `scene.h`
- `AI_SCENE_FLAGS_NON_VERBOSE_FORMAT`: `scene.h`
- `AI_SCENE_FLAGS_TERRAIN`: `scene.h`
- `AI_SCENE_FLAGS_VALIDATED`: `scene.h`
- `AI_SCENE_FLAGS_VALIDATION_WARNING`: `scene.h`
- `AI_SLM_DEFAULT_MAX_TRIANGLES`: `config.h`
- `AI_SLM_DEFAULT_MAX_VERTICES`: `config.h`
- `AI_TEXTURE_TYPE_MAX`: `material.h`
- `AI_TRUE`: `cimport.h`
- `AI_UVTRAFO_ALL`: `config.h`
- `AI_UVTRAFO_ROTATION`: `config.h`
- `AI_UVTRAFO_SCALING`: `config.h`
- `AI_UVTRAFO_TRANSLATION`: `config.h`
- AI_VECTOR2D_INL_INC: vector2.inl
- AI_VECTOR3D_INL_INC: vector3.inl
- AI_WONT_RETURN: defs.h
- AI_WONT_RETURN_SUFFIX: defs.h
- aiComponent(COLORS)n: config.h
- aiComponent_TEXCOORDSn: config.h
- aiProcess_ConvertToLeftHanded: postprocess.h
- aiProcessPreset_TargetRealtime_Fast: postprocess.h
- aiProcessPreset_TargetRealtime_MaxQuality: postprocess.h
- aiProcessPreset_TargetRealtime_Quality: postprocess.h
- ASSIMP_API: defs.h
- ASSIMP_API_WINONLY: defs.h
- ASSIMP_BUILD_DEBUG: defs.h
- ASSIMP_BUILD_NEED_UNZIP: defs.h
- ASSIMP_BUILD_NEED_Z_INFLATE: defs.h
- ASSIMP_BUILD_SINGLETHREADED: defs.h
- ASSIMP_CFLAGS_DEBUG: version.h
- ASSIMP_CFLAGS_NOBOOST: version.h
- ASSIMP_CFLAGS_SHARED: version.h
- ASSIMP_CFLAGS_SINGLETHEADED: version.h
- ASSIMP_CFLAGS_STLPORT: version.h
- ASSIMP_DEFAULT_LOG_NAME: DefaultLogger.hpp

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
- C -

- **C_ENUM** : [defs.h](#)
- **C_STRUCT** : [defs.h](#)

---

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by [doxygen](http://www.stackoverflow.com) 1.8.11*
- i -

- INT16_MAX : pstdint.h
- INT16_MIN : pstdint.h
- INT32_MAX : pstdint.h
- INT32_MIN : pstdint.h
- INT8_MAX : pstdint.h
- INT8_MIN : pstdint.h
- INT_FAST16_MAX : pstdint.h
- INT_FAST16_MIN : pstdint.h
- INT_FAST32_MAX : pstdint.h
- INT_FAST32_MIN : pstdint.h
- INT_FAST8_MAX : pstdint.h
- INT_FAST8_MIN : pstdint.h
- INT_LEAST16_MAX : pstdint.h
- INT_LEAST16_MIN : pstdint.h
- INT_LEAST32_MAX : pstdint.h
- INT_LEAST32_MIN : pstdint.h
- INT_LEAST8_MAX : pstdint.h
- INT_LEAST8_MIN : pstdint.h
- INTMAX_C : pstdint.h
- INTMAX_MAX : pstdint.h
- m -

- MAX_LOG_MESSAGE_LENGTH : Logger.hpp
- p -

- PP_ICL_PTCACHE_SIZE : config.h
- PRINTF_INT16_DEC_WIDTH : pstdint.h
- PRINTF_INT16_HEX_WIDTH : pstdint.h
- PRINTF_INT32_DEC_WIDTH : pstdint.h
- PRINTF_INT32_HEX_WIDTH : pstdint.h
- PRINTF_INT64_DEC_WIDTH : pstdint.h
- PRINTF_INT64_HEX_WIDTH : pstdint.h
- PRINTF_INT8_DEC_WIDTH : pstdint.h
- PRINTF_INT8_HEX_WIDTH : pstdint.h
- PRINTF_INTMAX_DEC_WIDTH : pstdint.h
- PRINTF_INTMAX_HEX_WIDTH : pstdint.h
- PRINTF_INTMAX_MODIFIER : pstdint.h
- PRINTF_LEAST16_MODIFIER : pstdint.h
- PRINTF_LEAST32_MODIFIER : pstdint.h
- s -

- SIG_ATOMIC_MAX : [psdint.h](#)
- SIZE_MAX : [psdint.h](#)
- STDINT_H_UINTPTR_T_DEFINED : [psdint.h](#)
- u -

- UINT16_MAX : pstdint.h
- UINT32_MAX : pstdint.h
- UINT8_MAX : pstdint.h
- UINT_FAST16_MAX : pstdint.h
- UINT_FAST32_MAX : pstdint.h
- UINT_FAST8_MAX : pstdint.h
- UINT_LEAST16_MAX : pstdint.h
- UINT_LEAST32_MAX : pstdint.h
- UINT_LEAST8_MAX : pstdint.h
- UINTMAX_C : pstdint.h
- UINTMAX_MAX : pstdint.h
- ULONG_LONG_MAX : pstdint.h
The assimp library returns the imported data in a collection of structures

This page lists some useful resources for assimp
Assimp v3.1.1 (June 2014)

- Assimp
- BaseImporter

Assimp::BaseImporter Member List

This is the complete list of members for Assimp::BaseImporter, including all inherited members.

- **ALLOW_EMPTY** enum value
- **BaseImporter()**
- **CanRead(const std::string &pFile, IOSystem *pIOHandler, bool checkSig)** const =0
- **CheckMagicToken(IOSystem *pIOHandler, const std::string &pFile, const void *magic, unsigned int num, unsigned int offset=0, unsigned int size=4)**
- **ConvertToUTF8(std::vector< char > &data)**
- **ConvertUTF8toISO8859_1(std::string &data)**
- **CopyVector(std::vector< T > * &vec, T ** &out, unsigned int &outLength)**
- **FORBID_EMPTY** enum value
- **GetErrorText() const**
- **GetExtension(const std::string &pFile)**
- **GetExtensionList(std::set< std::string > &extensions)**
- **GetInfo() const =0**
- **Importer** class

Assimp::BaseImporter

- ALLOW_EMPTY enum value
- BaseImporter()
- CanRead(const std::string &pFile, IOSystem *pIOHandler, bool checkSig) const =0
- CheckMagicToken(IOSystem *pIOHandler, const std::string &pFile, const void *magic, unsigned int num, unsigned int offset=0, unsigned int size=4)
- ConvertToUTF8(std::vector< char > &data)
- ConvertUTF8toISO8859_1(std::string &data)
- CopyVector(std::vector< T > * &vec, T ** &out, unsigned int &outLength)
- FORBID_EMPTY enum value
- GetErrorText() const
- GetExtension(const std::string &pFile)
- GetExtensionList(std::set< std::string > &extensions)
- GetInfo() const =0
- Importer class
**InternReadFile** (const std::string &pFile, aiScene *pScene, IOSystem *pIOHandler)=0

**m_ErrorText**

**m_progress**

**ReadFile** (const Importer *pImp, const std::string &pFile, IOSystem *pIOHandler)

**SearchFileHeaderForToken** (IOSystem *pIOSystem, const std::string &file, const char **tokens, unsigned int numTokens, unsigned int searchBytes=200, bool tokensSol=false)

**SetupProperties** (const Importer *pImp)

**SimpleExtensionCheck** (const std::string &pFile, const char *ext0, const char *ext1=NULL, const char *ext2=NULL)

**TextFileMode** enum name

**TextFileToBuffer** (IOStream *stream, std::vector<char> &data, TextFileMode mode=FORBID_EMPTY)

**~BaseImporter** ()
Assimp v3.1.1 (June 2014)

- Assimp
- DefaultLogger

Assimp::DefaultLogger Member List

This is the complete list of members for Assimp::DefaultLogger, including all inherited members.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachStream</td>
<td>(LogStream *pStream, unsigned int severity)</td>
</tr>
<tr>
<td>create</td>
<td>(const char *name=ASSIMP_DEFAULT_LOG_NAME, LogSeverity severity=NORMAL, unsigned int defStreams=aiDefaultLogStream_DEBUGGER</td>
</tr>
<tr>
<td>debug</td>
<td>(const char *message)</td>
</tr>
<tr>
<td>debug</td>
<td>(const std::string &amp;message)</td>
</tr>
<tr>
<td>Debugging</td>
<td>enum value</td>
</tr>
<tr>
<td>detachStream</td>
<td>(LogStream *pStream, unsigned int severity)</td>
</tr>
<tr>
<td>Err</td>
<td>enum value</td>
</tr>
<tr>
<td>error</td>
<td>(const char *message)</td>
</tr>
<tr>
<td>error</td>
<td>(const std::string &amp;message)</td>
</tr>
<tr>
<td>ErrorSeverity</td>
<td>enum name</td>
</tr>
<tr>
<td>get()</td>
<td></td>
</tr>
<tr>
<td>getLogSeverity() const</td>
<td></td>
</tr>
<tr>
<td>info</td>
<td>(const char *message)</td>
</tr>
<tr>
<td>info</td>
<td>(const std::string &amp;message)</td>
</tr>
<tr>
<td>Info</td>
<td>enum value</td>
</tr>
<tr>
<td>isNullLogger()</td>
<td></td>
</tr>
<tr>
<td>kill()</td>
<td></td>
</tr>
<tr>
<td>Logger()</td>
<td></td>
</tr>
</tbody>
</table>
Logger (LogSeverity severity)

LogSeverity enum name

m_Severity

NORMAL enum value

set (Logger *logger)

setLogSeverity (LogSeverity log_severity)

VERBOSE enum value

warn (const char *message)

warn (const std::string &message)

Warn enum value

~Logger ()
Assimp v3.1.1 (June 2014)

- Assimp
- Exporter

Assimp::Exporter Member List

This is the complete list of members for **Assimp::Exporter**, including all inherited members.

```cpp
Assimp::Exporter Export(const aiScene *pScene, const char *pFormatId, const char *pPath, unsigned int pPreprocessing=0u, const ExportProperties *pProperties=NULL)
Assimp::Exporter Export(const aiScene *pScene, const std::string &pFormatId, const std::string &pPath, unsigned int pPreprocessing=0u, const ExportProperties *pProperties=NULL)
Assimp::Exporter Exporter()
Assimp::Exporter ExportToBlob(const aiScene *pScene, const char *pFormatId, unsigned int pPreprocessing=0u, const ExportProperties *pProperties=NULL)
Assimp::Exporter ExportToBlob(const aiScene *pScene, const std::string &pFormatId, unsigned int pPreprocessing=0u, const ExportProperties *pProperties=NULL)
Assimp::Exporter fpExportFunc typedef
Assimp::Exporter FreeBlob()
Assimp::Exporter GetBlob()
Assimp::Exporter GetErrorString()
Assimp::Exporter GetExportFormatCount()
Assimp::Exporter GetExportFormatDescription(size_t pIndex)
```
<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GetIOHandler()</code> const</td>
<td><code>Assimp::Exporter</code></td>
</tr>
<tr>
<td><code>GetOrphanedBlob()</code> const</td>
<td><code>Assimp::Exporter</code></td>
</tr>
<tr>
<td><code>IsDefaultIOHandler()</code> const</td>
<td><code>Assimp::Exporter</code></td>
</tr>
<tr>
<td><code>pimpl</code></td>
<td><code>Assimp::Exporter</code> protected</td>
</tr>
<tr>
<td><code>RegisterExporter</code>(const ExportFormatEntry &amp;desc)</td>
<td><code>Assimp::Exporter</code></td>
</tr>
<tr>
<td><code>SetIOHandler</code>(IOSystem *pIOHandler)</td>
<td><code>Assimp::Exporter</code></td>
</tr>
<tr>
<td><code>UnregisterExporter</code>(const char *id)</td>
<td><code>Assimp::Exporter</code></td>
</tr>
<tr>
<td><code>~Exporter()</code></td>
<td><code>Assimp::Exporter</code></td>
</tr>
</tbody>
</table>

Generated on Sun Feb 21 2016 19:42:29 for Assimp by `doxygen` 1.8.11
Assimp v3.1.1 (June 2014)

- Assimp
- Exporter
- ExportFormatEntry

Assimp::Exporter::ExportFormatEntry Member List

This is the complete list of members for Assimp::Exporter::ExportFormatEntry, including all inherited members.

- `ExportFormatEntry` (const char *pId, const char *pDesc, const char *pExtension, fpExportFunc pFunction, unsigned int pEnforcePP=0u)
- `ExportFormatEntry`()
- `mDescription`
- `mEnforcePP`
- `mExportFunction`

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp v3.1.1 (June 2014)

- Assimp
- ExportProperties

Assimp::ExportProperties Member List

This is the complete list of members for `Assimp::ExportProperties`, including all inherited members.

```
ExportProperties() Assimp::ExportProperties
ExportProperties(const ExportProperties &other) Assimp::ExportProperties
FloatPropertyMap typedef Assimp::ExportProperties
GetPropertyBool(const char *szName, bool bErrorReturn=false) const Assimp::ExportProperties inline
GetPropertyFloat(const char *szName, float fErrorReturn=10e10f) const Assimp::ExportProperties
GetPropertyInteger(const char *szName, int iErrorReturn=0xffffffff) const Assimp::ExportProperties
GetPropertyMatrix(const char *szName, const aiMatrix4x4 &sErrorReturn=aiMatrix4x4()) const Assimp::ExportProperties
GetPropertyString(const char *szName, const std::string &sErrorReturn="") const Assimp::ExportProperties
HasPropertyBool(const char *szName) const Assimp::ExportProperties
HasPropertyFloat(const char *szName) const Assimp::ExportProperties
HasPropertyInteger(const char *szName) const Assimp::ExportProperties
HasPropertyMatrix(const char *szName) const Assimp::ExportProperties
```
const

HasPropertyString(const char *szName)
const

IntPropertyMap typedef

KeyType typedef

MatrixPropertyMap typedef

mFloatProperties

mIntProperties

mMatrixProperties

mStringProperties

SetPropertyBool(const char *szName, bool value)

SetPropertyFloat(const char *szName, float fValue)

SetPropertyInteger(const char *szName, int iValue)

SetPropertyMatrix(const char *szName, const aiMatrix4x4 &sValue)

SetPropertyString(const char *szName, const std::string &sValue)

StringPropertyMap typedef
Assimp v3.1.1 (June 2014)

- Assimp
- Importer

Assimp::Importer Member List

This is the complete list of members for Assimp::Importer, including all inherited members.

- **ApplyCustomizedPostProcessing**(BaseProcess *rootProcess, bool requestValidation)  
  Assimp::Importer
- **ApplyPostProcessing**(unsigned int pFlags)  
  Assimp::Importer
- **FreeScene**()  
  Assimp::Importer
- **GetErrorString**() const  
  Assimp::Importer
- **GetExtensionList**(aiString &szOut) const  
  Assimp::Importer
- **GetExtensionList**(std::string &szOut) const  
  Assimp::Importer
- **GetImporter**(size_t index) const  
  Assimp::Importer
- **GetImporter**(const char *szExtension) const  
  Assimp::Importer
- **GetImporterCount**() const  
  Assimp::Importer
- **GetImporterIndex**(const char *szExtension) const  
  Assimp::Importer
- **GetImporterInfo**(size_t index) const  
  Assimp::Importer
- **GetIOHandler**() const  
  Assimp::Importer
- **GetMemoryRequirements**(aiMemoryInfo &in) const  
  Assimp::Importer
- **GetOrphanedScene**()  
  Assimp::Importer
- **GetProgressHandler**() const  
  Assimp::Importer
- **GetPropertyBool**(const char *szName, bool bErrorReturn=false) const  
  Assimp::Importer
- **GetPropertyFloat**(const char *szName, float fErrorReturn=10e10f) const  
  Assimp::Importer
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GetPropertyInteger</code></td>
<td>const char *szName, int iErrorReturn=0xffffffff const</td>
</tr>
<tr>
<td><code>GetPropertyMatrix</code></td>
<td>const char *szName, const aiMatrix4x4 &amp;sErrorReturn=aiMatrix4x4() const</td>
</tr>
<tr>
<td><code>GetPropertyString</code></td>
<td>const char *szName, const std::string &amp;sErrorReturn=&quot;&quot; const</td>
</tr>
<tr>
<td><code>GetScene</code></td>
<td>const</td>
</tr>
<tr>
<td><code>Importer</code></td>
<td></td>
</tr>
<tr>
<td><code>Importer(Importer &amp;other)</code></td>
<td></td>
</tr>
<tr>
<td><code>IsDefaultIOHandler</code></td>
<td>const</td>
</tr>
<tr>
<td><code>IsDefaultProgressHandler</code></td>
<td>const</td>
</tr>
<tr>
<td><code>IsExtensionSupported</code></td>
<td>const char *szExtension</td>
</tr>
<tr>
<td><code>IsExtensionSupported</code></td>
<td>const std::string &amp;szExtension</td>
</tr>
<tr>
<td><code>MaxLenHint</code></td>
<td></td>
</tr>
<tr>
<td><code>pimpl</code></td>
<td></td>
</tr>
<tr>
<td><code>Pimpl()</code></td>
<td></td>
</tr>
<tr>
<td><code>ReadFile</code></td>
<td>const char *pFile, unsigned int pFlags</td>
</tr>
<tr>
<td><code>ReadFile</code></td>
<td>const std::string &amp;pFile, unsigned int pFlags</td>
</tr>
</tbody>
</table>
| `ReadFileFromMemory`                         | const void *pBuffer, size_t pLength, unsigned int pFlags, const char *pHint=""
| `RegisterLoader`                             | BaseImporter *pImp                                                          |
| `RegisterPPStep`                             | BaseProcess *pImp                                                           |
| `SetExtraVerbose`                            | bool bDo                                                                    |
| `SetIOHandler`                               | IOSystem *pIOHandler                                                        |
| `SetProgressHandler`                         | ProgressHandler *pHandler                                                   |
| `SetPropertyBool`                            | const char *szName, bool value                                               |
| `SetPropertyFloat`                           | const char *szName, float fValue                                             |
| `SetPropertyInteger`                         | const char *szName, int iValue                                               |
| `SetPropertyMatrix`                          | const char *szName, const                                                    |
aiMatrix4x4 &sValue)

SetPropertyString(const char *szName, const
std::string &sValue)

UnregisterLoader(BaseImporter *pImp)

UnregisterPPStep(BaseProcess *pImp)

ValidateFlags(unsigned int pFlags) const

~Importer()
Assimp v3.1.1 (June 2014)

- Assimp
- Interpolator

Assimp::Interpolator< T > Member List

This is the complete list of members for Assimp::Interpolator< T >, including all inherited members.

\texttt{operator()}(T &out, const T &a, const T &b, float d) const

\begin{verbatim}
Assimp::Interpolator< T > inline
\end{verbatim}

\textit{Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11}
Assimp v3.1.1 (June 2014)

- Assimp
- Iostream

Assimp::Iostream Member List

This is the complete list of members for Assimp::Iostream, including all inherited members.

- `FileSize()` const = 0
- `Flush()` = 0
- `Iostream(void)`
- `Read(void *pvBuffer, size_t pSize, size_t pCount)=0`
- `Seek(size_t pOffset, aiOrigin pOrigin)=0`
- `Tell()` const = 0
- `Write(const void *pvBuffer, size_t pSize, size_t pCount)=0`
- `~Iostream()`

```
Generated on Sun Feb 21 2016 19:42:29 for Assimp by 1.8.11
```
Assimp v3.1.1 (June 2014)

- Assimp
- IOSystem

Assimp::IOSystem Member List

This is the complete list of members for Assimp::IOSystem, including all inherited members.

<table>
<thead>
<tr>
<th>Function</th>
<th>Assimp::IOSystem</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Close (IOStream *pFile)=0</td>
<td>pure virtual</td>
<td></td>
</tr>
<tr>
<td>ComparePaths (const char *one, const char *second) const</td>
<td>virtual</td>
<td></td>
</tr>
<tr>
<td>ComparePaths (const std::string &amp;one, const std::string &amp;second) const</td>
<td>inline</td>
<td></td>
</tr>
<tr>
<td>CurrentDirectory () const</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exists (const std::string &amp;pFile) const</td>
<td>inlinevirtual</td>
<td></td>
</tr>
<tr>
<td>Exists (const char *pFile) const =0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getOsSeparator () const =0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOSystem ()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open (const char *pFile, const char *pMode=&quot;rb&quot;)=0</td>
<td>pure virtual</td>
<td></td>
</tr>
<tr>
<td>Open (const std::string &amp;pFile, const std::string &amp;pMode=std::string(&quot;rb&quot;))</td>
<td>inline</td>
<td></td>
</tr>
<tr>
<td>PopDirectory ()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PushDirectory (const std::string &amp;path)</td>
<td>inlinevirtual</td>
<td></td>
</tr>
<tr>
<td>StackSize () const</td>
<td></td>
<td></td>
</tr>
<tr>
<td>~IOSystem ()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Assimp v3.1.1 (June 2014)

- **Assimp**
- **Logger**

Assimp::Logger Member List

This is the complete list of members for **Assimp::Logger**, including all inherited members.

- `attachStream)(LogStream *pStream, unsigned int severity=Debugging|Err|Warn|Info)=0` - **Assimp::Logger** pure virtual
- `debug)(const char *message)` - **Assimp::Logger**
- `debug)(const std::string &message)` - **Assimp::Logger**
- `Debugging enum value` - **Assimp::Logger**
- `detachStream)(LogStream *pStream, unsigned int severity=Debugging|Err|Warn|Info)=0` - **Assimp::Logger** pure virtual
- `Err enum value` - **Assimp::Logger**
- `error)(const char *message)` - **Assimp::Logger**
- `error)(const std::string &message)` - **Assimp::Logger**
- `ErrorSeverity enum name` - **Assimp::Logger**
- `getLogSeverity() const` - **Assimp::Logger**
- `info)(const char *message)` - **Assimp::Logger**
- `info)(const std::string &message)` - **Assimp::Logger**
- `Info enum value` - **Assimp::Logger**
- `Logger()` - **Assimp::Logger**
- `Logger(LogSeverity severity)` - **Assimp::Logger** inline
- `LogSeverity enum name` - **Assimp::Logger**
- `m_Severity` - **Assimp::Logger** protected
Assimp::Logger

*NORMAL* enum value

**OnDebug**(const char *message)=0

**OnError**(const char *message)=0

**OnInfo**(const char *message)=0

**OnWarn**(const char *message)=0

**setLogSeverity**(LogSeverity log_severity)

**VERBOSE** enum value

**warn**(const char *message)

**warn**(const std::string &message)

**Warn** enum value

**~Logger**()

Assimp::Logger protected pure virtual

Assimp::Logger protected pure virtual

Assimp::Logger protected pure virtual

Assimp::Logger protected pure virtual

Assimp::Logger inline

Assimp::Logger

Assimp::Logger

Assimp::Logger inline

Assimp::Logger

Assimp::Logger inline virtual

Generated on Sun Feb 21 2016 19:42:29 for Assimp by **doxygen** 1.8.11
Assimp v3.1.1 (June 2014)

- **Assimp**
- **LogStream**

Assimp::LogStream Member List

This is the complete list of members for **Assimp::LogStream**, including all inherited members.

```cpp
createDefaultStream(aiDefaultLogStream stream, const char *name="AssimpLog.txt", IOSystem *io=NULL)  
Assimp::LogStream static

LogStream()  
Assimp::LogStream inlineprotected

write(const char *message)=0  
Assimp::LogStream pure virtual

~LogStream()  
Assimp::LogStream inlinevirtual
```

---

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by [doxygen](http://www.doxygen.org) 1.8.11*
Assimp v3.1.1 (June 2014)

- Assimp
- NullLogger

Assimp::NullLogger Member List

This is the complete list of members for Assimp::NullLogger, including all inherited members.

- `attachStream(LogStream *pStream, unsigned int severity)`
  - Assimp::NullLogger inline virtual
- `debug(const char *message)`
  - Assimp::Logger
- `debug(const std::string &message)`
  - Assimp::Logger inline
- `Debugging` enum value
  - Assimp::Logger
- `detachStream(LogStream *pStream, unsigned int severity)`
  - Assimp::NullLogger inline virtual
- `Err` enum value
  - Assimp::Logger
- `error(const char *message)`
  - Assimp::Logger
- `error(const std::string &message)`
  - Assimp::Logger inline
- `ErrorSeverity` enum name
  - Assimp::Logger
- `getLogSeverity()` const
  - Assimp::Logger inline
- `info(const char *message)`
  - Assimp::Logger inline
- `info(const std::string &message)`
  - Assimp::Logger inline
- `Info` enum value
  - Assimp::Logger
- `Logger()`
  - Assimp::Logger inline protected
- `Logger(LogSeverity severity)`
  - Assimp::Logger inline explicit protected
- `LogSeverity` enum name
  - Assimp::Logger
- `m_Severity`
  - Assimp::Logger protected
- `NORMAL` enum value
  - Assimp::Logger
- `OnDebug(const char *message)`
  - Assimp::NullLogger inline virtual
OnError(const char *message) Assimp::NullLogger inlinevirtual
OnInfo(const char *message) Assimp::NullLogger inlinevirtual
OnWarn(const char *message) Assimp::NullLogger inlinevirtual
setLogSeverity(LogSeverity log_severity) Assimp::Logger inline
VERBOSE enum value Assimp::Logger
warn(const char *message) Assimp::Logger
warn(const std::string &message) Assimp::Logger inline
Warn enum value Assimp::Logger
~Logger() Assimp::Logger inlinevirtual
Assimp v3.1.1 (June 2014)

Assimp
- ProgressHandler

Assimp::ProgressHandler Member List

This is the complete list of members for `Assimp::ProgressHandler`, including all inherited members.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ProgressHandler()</code></td>
<td>Assimp::ProgressHandler inlineprotected</td>
</tr>
<tr>
<td><code>Update(float percentage=-1.f)=0</code></td>
<td>Assimp::ProgressHandler pure virtual</td>
</tr>
<tr>
<td><code>UpdateFileRead(int currentStep, int numberOfSteps)</code></td>
<td>Assimp::ProgressHandler inlinevirtual</td>
</tr>
<tr>
<td><code>UpdatePostProcess(int currentStep, int numberOfSteps)</code></td>
<td>Assimp::ProgressHandler inlinevirtual</td>
</tr>
<tr>
<td><code>~ProgressHandler()</code></td>
<td>Assimp::ProgressHandler inlinevirtual</td>
</tr>
</tbody>
</table>

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11*
Assimp v3.1.1 (June 2014)

Assimp::ScopeGuard< T > Member List

This is the complete list of members for Assimp::ScopeGuard< T >, including all inherited members.

- `dismiss()`
- `operator T *()`
- `operator->()`
- `ScopeGuard(T *obj)`
- `~ScopeGuard()`

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
aiAnimation Member List

This is the complete list of members for `aiAnimation`, including all inherited members.

```cpp
aiAnimation() aiAnimation inline
mChannels aiAnimation
mDuration aiAnimation
mMeshChannels aiAnimation
mName aiAnimation
mNumChannels aiAnimation
mNumMeshChannels aiAnimation
mTicksPerSecond aiAnimation
~aiAnimation() aiAnimation inline
```
aiAnimMesh Member List

This is the complete list of members for aiAnimMesh, including all inherited members.

```cpp
aiAnimMesh() aiAnimMesh inline
HasNormals() const aiAnimMesh inline
HasPositions() const aiAnimMesh inline
HasTangentsAndBitangents() const aiAnimMesh inline
HasTextureCoords(unsigned int pIndex) const aiAnimMesh inline
HasVertexColors(unsigned int pIndex) const aiAnimMesh inline
mBitangents aiAnimMesh
mColors aiAnimMesh
mNormals aiAnimMesh
mNumVertices aiAnimMesh
mTangents aiAnimMesh
mTextureCoords aiAnimMesh
mVertices aiAnimMesh
~aiAnimMesh() aiAnimMesh inline
```

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
aiBone Member List

This is the complete list of members for aiBone, including all inherited members.

aiBone()     aiBone
aiBone(const aiBone &other) aiBone inline
mName        aiBone
mNumWeights  aiBone
mOffsetMatrix aiBone
mWeights     aiBone
~aiBone()     aiBone inline

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
aiCamera Member List

This is the complete list of members for aiCamera, including all inherited members.

aiCamera()           aiCamera inline
GetCameraMatrix(aiMatrix4x4 &out) const aiCamera inline
mAspect               aiCamera
mClipPlaneFar         aiCamera
mClipPlaneNear        aiCamera
mHorizontalFOV        aiCamera
mLookAt               aiCamera
mName                 aiCamera
mPosition             aiCamera
mUp                   aiCamera
aiColor3D Member List

This is the complete list of members for aiColor3D, including all inherited members.

aiColor3D()
aiColor3D(float _r, float _g, float _b)
aiColor3D(float _r)
aiColor3D(const aiColor3D &o)
b
g
IsBlack() const
operator!=(const aiColor3D &other) const
operator*(const aiColor3D &c) const
operator*(float f) const
operator+(const aiColor3D &c) const
operator-(const aiColor3D &c) const
operator<(const aiColor3D &other) const
operator==(const aiColor3D &other) const
operator[](unsigned int i) const
operator[](unsigned int i)
aiColor4t< TReal > Member List

This is the complete list of members for aiColor4t< TReal >, including all inherited members.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>aiColor4t()</td>
</tr>
<tr>
<td>b</td>
<td>aiColor4t(TReal _r)</td>
</tr>
<tr>
<td>c</td>
<td>aiColor4t(TReal _r)</td>
</tr>
<tr>
<td>g</td>
<td>aiColor4t(TReal _r)</td>
</tr>
<tr>
<td>IsBlack() const</td>
<td>aiColor4t&lt; TReal &gt; inline</td>
</tr>
<tr>
<td>operator!=</td>
<td>aiColor4t&lt; TReal &gt; inline</td>
</tr>
<tr>
<td>operator*=(TReal f)</td>
<td>aiColor4t&lt; TReal &gt; inline</td>
</tr>
<tr>
<td>operator+=(const aiColor4t &amp;o)</td>
<td>aiColor4t&lt; TReal &gt; inline</td>
</tr>
<tr>
<td>operator-=(const aiColor4t &amp;o)</td>
<td>aiColor4t&lt; TReal &gt; inline</td>
</tr>
</tbody>
</table>
operator/= (TReal f)
operator< (const aiColor4t &other) const
operator==(const aiColor4t &other) const
operator[](unsigned int i) const
operator[](unsigned int i)

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
aiExportDataBlob Member List

This is the complete list of members for aiExportDataBlob, including all inherited members.

- aiExportDataBlob()
- aiExportDataBlob data
- aiExportDataBlob name
- aiExportDataBlob next
- aiExportDataBlob size
- ~aiExportDataBlob()
aiExportFormatDesc Member List

This is the complete list of members for aiExportFormatDesc, including all inherited members.

- description         aiExportFormatDesc
- fileExtension      aiExportFormatDesc
- id                  aiExportFormatDesc

Generated on Sun Feb 21 2016 19:42:29 for Assimp by Doxygen 1.8.11
aiFace Member List

This is the complete list of members for aiFace, including all inherited members.

```c
aiFace()
aiFace(const aiFace &o)
mIndices
mNumIndices
operator!=(const aiFace &o) const
operator=(const aiFace &o)
operator==(const aiFace &o) const
~aiFace()
```

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11*
aiFile Member List

This is the complete list of members for aiFile, including all inherited members.

FileSizeProc aiFile
FlushProc aiFile
ReadProc aiFile
SeekProc aiFile
TellProc aiFile
UserData aiFile
WriteProc aiFile
aiFileIO Member List

This is the complete list of members for aiFileIO, including all inherited members.

CloseProc aiFileIO  
OpenProc aiFileIO  
UserData aiFileIO
aiImporterDesc Member List

This is the complete list of members for aiImporterDesc, including all inherited members.

- `mAuthor` aiImporterDesc
- `mComments` aiImporterDesc
- `mFileExtensions` aiImporterDesc
- `mFlags` aiImporterDesc
- `mMaintainer` aiImporterDesc
- `mMaxMajor` aiImporterDesc
- `mMaxMinor` aiImporterDesc
- `mMinMajor` aiImporterDesc
- `mMinMinor` aiImporterDesc
- `mName` aiImporterDesc

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
aiLight Member List

This is the complete list of members for `aiLight`, including all inherited members.

```plaintext
aiLight ()
mAngleInnerCone
mAngleOuterCone
mAttenuationConstant
mAttenuationLinear
mAttenuationQuadratic
mColorAmbient
mColorDiffuse
mColorSpecular
mDirection
mName
mPosition
mType
```

Generated on Sun Feb 21 2016 19:42:29 for Assimp by [Doxygen](http://www.doxygen.nl) 1.8.11
aiLogStream Member List

This is the complete list of members for aiLogStream, including all inherited members.

callback aiLogStream
user aiLogStream
This is the complete list of members for `aiMaterial`, including all inherited members.

- **AddBinaryProperty**(const void *pInput, unsigned int pSizeInBytes, const char *pKey, unsigned int type, unsigned int index, aiPropertyTypeInfo pType)
  
- **AddProperty**(const aiString *pInput, const char *pKey, unsigned int type=0, unsigned int index=0)
  
- **AddProperty**(const TYPE *pInput, unsigned int pNumValues, const char *pKey, unsigned int type=0, unsigned int index=0)
  
- **AddProperty**(const aiVector3D *pInput, unsigned int pNumValues, const char *pKey, unsigned int type=0, unsigned int index=0)
  
- **AddProperty**(const aiColor3D *pInput, unsigned int pNumValues, const char *pKey, unsigned int type=0, unsigned int index=0)
  
- **AddProperty**(const aiColor4D *pInput, unsigned int pNumValues, const char *pKey, unsigned int type=0, unsigned int index=0)
  
- **AddProperty**(const int *pInput, unsigned int pNumValues, const char *pKey, unsigned int type=0, unsigned int index=0)
  
- **AddProperty**(const float *pInput, unsigned int pNumValues, const char *pKey, unsigned int type=0, unsigned int index=0)
  
- **AddProperty**(const aiUVTransform *pInput, unsigned int pNumValues, const char *pKey, unsigned int type=0, unsigned int index=0)
  
- `aiMaterial()`
  
- `Clear()`
  
- **CopyPropertyList**(aiMaterial *pcDest, const aiMaterial *pcSrc)
Get(const char *pKey, unsigned int type, unsigned int idx, Type *pOut, unsigned int *pMax) const
Get(const char *pKey, unsigned int type, unsigned int idx, int *pOut, unsigned int *pMax) const
Get(const char *pKey, unsigned int type, unsigned int idx, float *pOut, unsigned int *pMax) const
Get(const char *pKey, unsigned int type, unsigned int idx, Type &pOut) const
Get(const char *pKey, unsigned int type, unsigned int idx, int &pOut) const
Get(const char *pKey, unsigned int type, unsigned int idx, float &pOut) const
Get(const char *pKey, unsigned int type, unsigned int idx, aiString &pOut) const
Get(const char *pKey, unsigned int type, unsigned int idx, aiColor3D &pOut) const
Get(const char *pKey, unsigned int type, unsigned int idx, aiColor4D &pOut) const
Get(const char *pKey, unsigned int type, unsigned int idx, aiUVTransform &pOut) const
GetTexture(aiTextureType type, unsigned int index, aiString *path, aiTextureMapping *mapping=NULL, unsigned int *uvindex=NULL, float *blend=NULL, aiTextureOp *op=NULL, aiTextureMapMode *mapmode=NULL) const
GetTextureCount(aiTextureType type) const
mNumAllocated
mNumProperties
mProperties
RemoveProperty(const char *pKey, unsigned int type=0, unsigned int index=0)
~aiMaterial()
This is the complete list of members for \texttt{aiMatrix3x3t<TReal>}, including all inherited members.

\begin{verbatim}
\textbf{aiMatrix3x3t()} \text{\texttt{aiMatrix3x3t(TReal_a1, TReal_a2, TReal_a3, TReal_b1, TReal_b2, TReal_b3, TReal_c1, TReal_c2, TReal_c3)}} \text{\texttt{aiMatrix3x3t(const aiMatrix4x4t<TReal> \&pMatrix)}}
\end{verbatim}

\textbf{Determinant()} const
**Equal** (const aiMatrix4x4t< TReal > &m, TReal epsilon=1e-6) const

**FromToMatrix** (const aiVector3t< TReal > &from, const aiVector3t< TReal > &to, aiMatrix3x3t &out)

**Inverse** ()

m
mData

operator aiMatrix3x3t< TOther >() const

operator!=(const aiMatrix4x4t< TReal > &m) const

operator*(const aiMatrix3x3t &m) const

operator*=(const aiMatrix3x3t &m)

operator==(const aiMatrix4x4t< TReal > &m) const

operator[](unsigned int p_iIndex)

operator[](unsigned int p_iIndex) const

**Rotation** (TReal a, const aiVector3t< TReal > &axis, aiMatrix3x3t &out)

**RotationZ** (TReal a, aiMatrix3x3t &out)

**Translation** (const aiVector2t< TReal > &v, aiMatrix3x3t &out)

**Transpose** ()
aiMatrix4x4t<TReal> Member List

This is the complete list of members for `aiMatrix4x4t<TReal>`, including all inherited members.

```cpp
aiMatrix4x4t<TReal>()

aiMatrix4x4t(TReal_a1, TReal_a2, TReal_a3, TReal_a4, TReal_b1, TReal_b2, TReal_b3, TReal_b4, TReal_c1, TReal_c2, TReal_c3, TReal_c4, TReal_d1, TReal_d2, TReal_d3, TReal_d4)

aiMatrix4x4t(const aiMatrix3x3t<TReal> &m)

aiMatrix4x4t(const aiVector3t<TReal> &scaling, const aiQuaterniont<TReal> &rotation, const aiVector3t<TReal> &position)

b1

b2

b3
```
b4
aiMatrix4x4t<TReal>

\( \text{Decompose} \) (aiVector3t<TReal> &scaling,
aiQuaterniont<TReal> &rotation,
aiVector3t<TReal> &position) const

\( \text{DecomposeNoScaling} \) (aiQuaterniont<TReal> &rotation,
aiVector3t<TReal> &position) const

\( \text{Determinant} \) () const

\( \text{Equal} \) (const aiMatrix4x4t &m, TReal epsilon=1e-6) const

\( \text{FromEulerAnglesXYZ} \) (TReal x, TReal y, TReal z)

\( \text{FromEulerAnglesXYZ} \) (const aiVector3t<TReal> &blubb)

\( \text{FromToMatrix} \) (const aiVector3t<TReal> &from,
const aiVector3t<TReal> &to, aiMatrix4x4t &out)

\( \text{Inverse} \) ()
IsIdentity() const

m

mData

operator aiMatrix4x4t< TOther >() const

operator!=(const aiMatrix4x4t &m) const

operator*(const aiMatrix4x4t &m) const

operator*=(const aiMatrix4x4t &m)

operator[](unsigned int p_iIndex)

operator[](unsigned int p_iIndex) const

Rotation(TReal a, const aiVector3t< TReal > &axis, aiMatrix4x4t &out)

RotationX(TReal a, aiMatrix4x4t &out)

RotationY(TReal a, aiMatrix4x4t &out)

RotationZ(TReal a, aiMatrix4x4t &out)

Scaling(const aiVector3t< TReal > &v, aiMatrix4x4t &out)

Translation(const aiVector3t< TReal > &v, aiMatrix4x4t &out)

Transpose()
aiMemoryInfo Member List

This is the complete list of members for aiMemoryInfo, including all inherited members.

aiMemoryInfo() aiMemoryInfo inline
animations aiMemoryInfo
cameras aiMemoryInfo
lights aiMemoryInfo
materials aiMemoryInfo
meshes aiMemoryInfo
nodes aiMemoryInfo
textures aiMemoryInfo
total aiMemoryInfo
aiMesh Member List

This is the complete list of members for aiMesh, including all inherited members.

**aiMesh()**

**GetNumColorChannels() const**

**GetNumUVChannels() const**

**HasBones() const**

**HasFaces() const**

**HasNormals() const**

**HasPositions() const**

**HasTangentsAndBitangents() const**

**HasTextureCoords(unsigned int pIndex) const**

**HasVertexColors(unsigned int pIndex) const**

**mAnimMeshes**

**mBitangents**

**mBones**

**mColors**

**mFaces**

**mMaterialIndex**

**mName**

**mNormals**

**mNumAnimMeshes**

**mNumBones**

**mNumFaces**

**mNumUVComponents**

**mNumVertices**

**mPrimitiveTypes**
<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>mTangents</td>
<td>aiMesh</td>
</tr>
<tr>
<td>mTextureCoords</td>
<td>aiMesh</td>
</tr>
<tr>
<td>mVertices</td>
<td>aiMesh</td>
</tr>
<tr>
<td>~aiMesh()</td>
<td>aiMesh inline</td>
</tr>
</tbody>
</table>

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11*
aiMeshAnim Member List

This is the complete list of members for aiMeshAnim, including all inherited members.

aiMeshAnim() aiMeshAnim inline
mKeys aiMeshAnim
mName aiMeshAnim
mNumKeys aiMeshAnim
~aiMeshAnim() aiMeshAnim inline
aiMeshKey Member List

This is the complete list of members for `aiMeshKey`, including all inherited members.

```
aiMeshKey() aiMeshKey inline
aiMeshKey(double time, const unsigned int value) aiMeshKey inline
elem_type typedef aiMeshKey
mTime aiMeshKey
mValue aiMeshKey
operator!=(const aiMeshKey &o) const aiMeshKey inline
operator<(const aiMeshKey &o) const aiMeshKey inline
operator==(const aiMeshKey &o) const aiMeshKey inline
operator>(const aiMeshKey &o) const aiMeshKey inline
```

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
aiMetadata Member List

This is the complete list of members for `aiMetadata`, including all inherited members.

- `aiMetadata()`
- `Get(unsigned index, T &value)`
- `Get(const aiString &key, T &value)`
- `Get(const std::string &key, T &value)`
- `mKeys`
- `mNumProperties`
- `mValues`
- `Set(unsigned index, const std::string &key, const T &value)`
- `~aiMetadata()`

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by Doxygen 1.8.11
aiMetadataEntry Member List

This is the complete list of members for aiMetadataEntry, including all inherited members.

mData aiMetadataEntry
mType aiMetadataEntry
aiNode Member List

This is the complete list of members for aiNode, including all inherited members.

aiNode()
aiNode(const std::string &name)
FindNode(const aiString &name) const
FindNode(const aiString &name)
FindNode(const char *name) const
FindNode(const char *name)

mChildren
mMeshes
mMetaData
mName
mNumChildren
mNumMeshes
mParent
mTransformation
~aiNode()
aiNodeAnim Member List

This is the complete list of members for aiNodeAnim, including all inherited members.

*aiNodeAnim*()  
*mNodeName*  
*mNumPositionKeys*  
*mNumRotationKeys*  
*mNumScalingKeys*  
*mPositionKeys*  
*mPostState*  
*mPreState*  
*mRotationKeys*  
*mScalingKeys*  
~*aiNodeAnim*()
aiPlane Member List

This is the complete list of members for `aiPlane`, including all inherited members.

```cpp
aiPlane() // aiPlane
aiPlane(float _a, float _b, float _c, float _d) // aiPlane inline
aiPlane(const aiPlane &o) // aiPlane inline
```
**aiPropertyStore Member List**

This is the complete list of members for `aiPropertyStore`, including all inherited members.

```
sentinel aiPropertyStore
```

---

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by [doxygen](http://www.doxygen.org) 1.8.11*
aiQuaternion< TReal > Member List

This is the complete list of members for aiQuaternion< TReal >, including all inherited members.

aiQuaternion()

aiQuaternion(TReal pw, TReal px, TReal py, TReal pz)

aiQuaternion(const aiMatrix3x3< TReal > &pRotMatrix)

aiQuaternion(TReal rotx, TReal roty, TReal rotz)

aiQuaternion(aiVector3< TReal > axis, TReal angle)

aiQuaternion(aiVector3< TReal > normalized)

Conjugate()

Equal(const aiQuaternion &o, TReal epsilon=1e-6) const

GetMatrix() const

Interpolate(aiQuaternion &pOut, const aiQuaternion &pStart, const aiQuaternion &pEnd, TReal pFactor)

Normalize()

operator!=(const aiQuaternion &o) const

operator*(const aiQuaternion &two) const
operator==(const aiQuaternion<TReal>& o) const

Rotate(const aiVector3<TReal>& in)

w

x

y

z
aiQuatKey Member List

This is the complete list of members for aiQuatKey, including all inherited members.

aiQuatKey() aiQuatKey inline
aiQuatKey(double time, const aiQuaternion &value) aiQuatKey inline
elem_type typedef aiQuatKey
mTime aiQuatKey
mValue aiQuatKey
operator!=(const aiQuatKey &o) const aiQuatKey inline
operator<(const aiQuatKey &o) const aiQuatKey inline
operator==(const aiQuatKey &o) const aiQuatKey inline
operator>(const aiQuatKey &o) const aiQuatKey inline
# aiRay Member List

This is the complete list of members for `aiRay`, including all inherited members.

```c
aiRay()
aiRay(const aiVector3D &pos, const aiVector3D &dir)
aiRay(const aiRay &o)
dir
pos
```

---

*Generated on Sun Feb 21 2016 19:42:29 for Assimp by Doxygen 1.8.11*
aiScene Member List

This is the complete list of members for aiScene, including all inherited members.

aiScene() aiScene
HasAnimations() const aiScene inline
HasCameras() const aiScene inline
HasLights() const aiScene inline
HasMaterials() const aiScene inline
HasMeshes() const aiScene inline
HasTextures() const aiScene inline
mAnimations aiScene
mCameras aiScene
mFlags aiScene
mLights aiScene
mMaterials aiScene
mMeshes aiScene
mNumAnimations aiScene
mNumCameras aiScene
mNumLights aiScene
mNumMaterials aiScene
mNumMeshes aiScene
mNumTextures aiScene
mPrivate aiScene
mRootNode aiScene
mTextures aiScene
~aiScene() aiScene
aiString Member List

This is the complete list of members for aiString, including all inherited members.

- aiString()
- aiString(const aiString &rOther)
- aiString(const std::string &pString)
- Append(const char *app)
- C_Str() const
- Clear()
- data
- length
- operator!=(const aiString &other) const
- operator=(const char *sz)
- operator=(const std::string &pString)
- operator==(const aiString &other) const
- Set(const std::string &pString)
- Set(const char *sz)
aiTexel Member List

This is the complete list of members for aiTexel, including all inherited members.

```c
aiTexel a
aiTexel b
aiTexel g
operator aiColor4D() const aiTexel inline
operator!=(const aiTexel &other) const aiTexel inline
operator==(const aiTexel &other) const aiTexel inline
aiTexel r
```
aiTexture Member List

This is the complete list of members for `aiTexture`, including all inherited members.

- `achFormatHint`  `aiTexture`
- `aiTexture()`  `aiTexture` inline
- `CheckFormat(const char *s)`  `aiTexture` inline
- `mHeight`  `aiTexture`
- `mWidth`  `aiTexture`
- `pcData`  `aiTexture`
- `~aiTexture()`  `aiTexture` inline
aiUVTransform Member List

This is the complete list of members for aiUVTransform, including all inherited members.

aiUVTransform()  aiUVTransform inline
mRotation       aiUVTransform
mScaling        aiUVTransform
mTranslation    aiUVTransform
Assimp v3.1.1 (June 2014)

aiVector2t\textless\ TReal\textgreater\ Member List

This is the complete list of members for \texttt{aiVector2t\textless\ TReal\textgreater}, including all inherited members.

\begin{verbatim}
aiVector2t() \hspace{1cm} aiVector2t< TReal > \hspace{1cm} inline
aiVector2t(TReal \_x, TReal \_y) \hspace{1cm} aiVector2t< TReal > \hspace{1cm} inline
aiVector2t(TReal _xyz) \hspace{1cm} aiVector2t< TReal > \hspace{1cm} inline
aiVector2t(const aiVector2t &o) \hspace{1cm} aiVector2t< TReal > \hspace{1cm} inline
Equal(const aiVector2t &other, TReal epsilon=1e-6) const \hspace{1cm} aiVector2t< TReal >
Length() const \hspace{1cm} aiVector2t< TReal >
Normalize() \hspace{1cm} aiVector2t< TReal >
operator aiVector2t< TOther >() const \hspace{1cm} aiVector2t< TReal >
operator!=(const aiVector2t &other) const \hspace{1cm} aiVector2t< TReal >
operator*=(TReal f) \hspace{1cm} aiVector2t< TReal >
operator+=(const aiVector2t &o) \hspace{1cm} aiVector2t< TReal >
operator-=(const aiVector2t &o) \hspace{1cm} aiVector2t< TReal >
operator/=(TReal f) \hspace{1cm} aiVector2t< TReal >
\end{verbatim}
operator=(TReal f) aiVector2t<TReal>
operator==(const aiVector2t &other) const aiVector2t<TReal>
operator[](unsigned int i) const aiVector2t<TReal>
operator[](unsigned int i) aiVector2t<TReal>
Set(TReal pX, TReal pY) aiVector2t<TReal>
SquareLength() const aiVector2t<TReal>
SymMul(const aiVector2t o) aiVector2t<TReal>
v aiVector2t<TReal>
x aiVector2t<TReal>
y aiVector2t<TReal>
aiVector3t<TReal> Member List

This is the complete list of members for `aiVector3t<TReal>`, including all inherited members.

- `aiVector3t()`
- `aiVector3t(TReal _x, TReal _y, TReal _z)`
- `aiVector3t(TReal _xyz)`
- `aiVector3t(const aiVector3t &o)`
- `Equal(const aiVector3t &other, TReal epsilon=1e-6) const` (inline)
- `Length() const` (inline)
- `Normalize()` (inline)
- `NormalizeSafe()` (inline)
- `operator aiVector3t< TOther >() const` (inline)
- `operator!=(const aiVector3t &other) const` (inline)
- `operator*=(TReal f)` (inline)
- `operator*=(const aiMatrix3x3t< TReal > &mat)` (inline)
- `operator*=(const aiMatrix4x4t< TReal > &mat)` (inline)
operator+= (const aiVector3t &o)
operator-= (const aiVector3t &o)
operator/(TReal f)
operator< (const aiVector3t &other) const
operator== (const aiVector3t &other) const
operator[](unsigned int i) const
operator[](unsigned int i)
Set (TReal pX, TReal pY, TReal pZ)
SquareLength() const
SymMul (const aiVector3t &o)

v
x
y
z
aiVectorKey Member List

This is the complete list of members for `aiVectorKey`, including all inherited members.

```cpp
aiVectorKey()    aiVectorKey inline
aiVectorKey(double time, const aiVector3D &value) aiVectorKey inline
elem_type typedef aiVectorKey
mTime          aiVectorKey
mValue         aiVectorKey
operator!=(const aiVectorKey &o) const    aiVectorKey inline
operator<(const aiVectorKey &o) const     aiVectorKey inline
operator==(const aiVectorKey &o) const     aiVectorKey inline
operator>(const aiVectorKey &o) const      aiVectorKey inline
```

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
aiVertexWeight Member List

This is the complete list of members for aiVertexWeight, including all inherited members.

aiVertexWeight()
aiVertexWeight(unsigned int pID, float pWeight)
mVertexId
mWeight

---

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp v3.1.1 (June 2014)

- include

include Directory Reference
Directories

directory  assimp

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
include
assimp

assimp Directory Reference
Directories

directory   Compiler

directory   port
Files

file ai_assert.h

file anim.h
Defines the data structures in which the imported animations are returned.

file camera.h
Defines the aiCamera data structure.

file cexport.h
Defines the C-API for the Assimp export interface.

file cfileio.h
Defines generic C routines to access memory-mapped files.

file cimport.h
Defines the C-API to the Open Asset Import Library.

file color4.h
RGBA color structure, including operators when compiling in C++.

file color4.inl
Inline implementation of aiColor4t<TReal> operators.
file `config.h`

Defines constants for configurable properties for the library.

file `DefaultLogger.hpp`

file `defs.h`

`Assimp` build configuration setup.

file `Exporter.hpp`

Defines the CPP-API for the `Assimp` export interface.

file `Importer.hpp`

Defines the C++-API to the Open Asset Import Library.

file `importerdesc.h`

`aiImporterFlags`, `aiImporterDesc` implementation.

file `IOStream.hpp`

File I/O wrappers for C++.

file `IOSystem.hpp`

File system wrapper for C++.

file `light.h`

Defines the `aiLight` data structure.
file Logger.hpp

Abstract base class 'Logger', base of the logging system.

file LogStream.hpp

Abstract base class 'LogStream', representing an output log stream.

file material.h

Defines the material system of the library.

file material.inl

Defines the C++ getters for the material system.

file matrix3x3.h

Definition of a 3x3 matrix, including operators when compiling in C++.

file matrix3x3.inl

Inline implementation of the 3x3 matrix operators.

file matrix4x4.h

4x4 matrix structure, including operators when compiling in C++

file matrix4x4.inl

Inline implementation of the 4x4 matrix operators.

file mesh.h
Declares the data structures in which the imported geometry is returned by ASSIMP: **aiMesh**, **aiFace** and **aiBone** data structures.

**file** metadata.h

Defines the data structures for holding node meta information.

**file** NullLogger.hpp

Dummy logger.

**file** postprocess.h

Definitions for import post processing steps.

**file** ProgressHandler.hpp

Abstract base class 'ProgressHandler'.

**file** quaternion.h

Quaternion structure, including operators when compiling in C++.

**file** quaternion.inl

Inline implementation of aiQuaternionT<TReal> operators.

**file** scene.h

Defines the data structures in which the imported scene is returned.

**file** texture.h
Defines texture helper structures for the library.

file types.h

Basic data types and primitives, such as vectors or colors.

file vector2.h

2D vector structure, including operators when compiling in C++

file vector2.inl

Inline implementation of aiVector2t<TReal> operators.

file vector3.h

3D vector structure, including operators when compiling in C++

file vector3.inl

Inline implementation of aiVector3t<TReal> operators.

file version.h

Functions to query the version of the Assimp runtime, check compile flags, ...

--------------------------------------------------------------------------------------------------------

Generated on Sun Feb 21 2016 19:42:29 for Assimp by doxygen 1.8.11
Assimp v3.1.1 (June 2014)

- include
- assimp
- port

port Directory Reference
Directories

directory  AndroidJNI
Assimp v3.1.1 (June 2014)

- **include**
- **assimp**
- **port**
- **AndroidJNI**

AndroidJNI Directory Reference
Files

file  AndroidJNIIOSystem.h
Assimp v3.1.1 (June 2014)

- code

code Directory Reference
Files

file  BaseImporter.h

Generated on Sun Feb 21 2016 19:42:29 for Assimp by  doxygen 1.8.11
Assimp v3.1.1 (June 2014)

- include
- assimp
- Compiler

Compiler Directory Reference
Files

file  poppack1.h

file  pstdint.h

file  pushpack1.h