Gazebo

1.9.0

Generated by Doxygen 1.8.2

Thu Jul 25 2013 15:25:51

Contents

Chapter 1

Gazebo API Reference

This documentation provides useful information about the Gazebo API. The code reference is divided into the groups below. Should you find problems with this documentation - typos, unclear phrases, or insufficient detail - please create a `new bitbucket issue`. Include sufficient detail to quickly locate the problematic documentation, and set the issue's fields accordingly: Assignee - blank; Kind - bug; Priority - minor; Version - blank.

Class `List` - Index of all classes in Gazebo, organized alphabetically

`Hierarchy` - Index of classes, organized hierachically according to their inheritance

`Modules Common`: Classes and files used ubiquitously across Gazebo

`Events`: For creating and destroying Gazebo events

`Math`: A set of classes that encapsulate math related properties and functions.

`Messages`: All messages and helper functions.

`Physics`: Classes for physics and dynamics

`Rendering`: A set of rendering related class, functions, and definitions.

`Sensors`: A set of sensor classes, functions, and definitions.

`Transport`: Handles transportation of messages.

Links `Website`: The main gazebo website, which contains news, downloads, and contact information.

`Wiki`: A collection of user supported documentation.

`Tutorials`: Tutorials that describe how to use Gazebo and implement your own simulations.

`Download`: How to download and install Gazebo

Chapter 2

Todo List

Member gazebo::physics::Joint::GetForce (p. **408) (int _index) GAZEBO_DEPRECATED(1.5)**

: not yet implemented. Get the forces applied at this Joint. Note that the unit of force should be consistent with the rest of the simulation scales. E.g. if you are using metric units, the unit for force is Newtons. If using imperial units (sorry), then unit of force is lb-force not (lb-mass), etc.

Member gazebo::physics::Joint::GetForce (p. **409) (unsigned int _index)**

: not yet implemented. Get the forces applied at this Joint. Note that the unit of force should be consistent with the rest of the simulation scales. E.g. if you are using metric units, the unit for force is Newtons. If using imperial units (sorry), then unit of force is lb-force not (lb-mass), etc.

Member gazebo::sensors::CameraSensor::GetTopic (p. **195) () const**

to be implemented

Class gazebo::SystemPlugin (p. **829)**

how to make doxygen reference to the file gazebo.cc::g_plugins?

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

Chapter 6

Class Index

6.1 Class List

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

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

Chapter 8

Module Documentation

8.1 Common

Files

- file **CommonTypes.hh**

Namespaces

- namespace **gazebo::common**

  Common namespace.

Classes

- class **gazebo::common::Animation**

  Manages an animation, which is a collection of keyframes and the ability to interpolate between the keyframes.
- class **gazebo::common::AssertionInternalError**

  Class for generating Exceptions which come from gazebo assertions.
- class **gazebo::common::BVHLoader**

  Handles loading BVH animation files.
- class **gazebo::common::ColladaLoader**

  Class used to load Collada mesh files.
- class **gazebo::common::Color**

  Defines a color.
- class **gazebo::common::Console**

  Message, error, warning functionality.
- class **gazebo::common::Exception**

  Class for generating exceptions.
- class **gazebo::common::Image**

  Encapsulates an image.
- class **gazebo::common::InternalError**

  Class for generating Internal Gazebo Errors: those errors which should never happend and represent programming bugs.
- class **gazebo::common::KeyEvent**

Generic description of a keyboard event.

- class **gazebo::common::KeyFrame**

  A key frame in an animation.

- class **gazebo::common::Material**

  Encapsulates description of a material.

- class **gazebo::common::Mesh**

  A 3D mesh.

- class **gazebo::common::MeshCSG**

  Creates CSG meshes.

- class **gazebo::common::MeshLoader**

  Base class for loading meshes.

- class **gazebo::common::MeshManager**

  Maintains and manages all meshes.

- class **gazebo::common::ModelDatabase**

  Connects to model database, and has utility functions to find models.

- class **gazebo::ModelPlugin**

  *A plugin with access to **physics::Model** (p. 516).*

- class **gazebo::common::MouseEvent**

  Generic description of a mouse event.

- class **gazebo::common::NodeAnimation**

  Node animation.

- struct **gazebo::common::NodeAssignment**

  Vertex to node weighted assignement for skeleton animation visualization.

- class **gazebo::common::NodeTransform**

  ***NodeTransform** (p. 576) **Skeleton.hh** (p. 1132) common/common.hh*

- class **gazebo::common::NumericAnimation**

  A numeric animation.

- class **gazebo::common::NumericKeyFrame**

  *A keyframe for a **NumericAnimation** (p. 581).*

- class **gazebo::common::PID**

  *Generic **PID** (p. 613) controller class.*

- class **gazebo::PluginT< T >**

  A class which all plugins must inherit from.

- class **gazebo::common::PoseAnimation**

  A pose animation.

- class **gazebo::common::PoseKeyFrame**

  *A keyframe for a **PoseAnimation** (p. 635).*

- class **gazebo::SensorPlugin**

  A plugin with access to physics::Sensor.

- class **SingletonT< T >**

  Singleton template class.

- class **gazebo::common::Skeleton**

  A skeleton.

- class **gazebo::common::SkeletonAnimation**

  ***Skeleton** (p. 760) animation.*

- class **gazebo::common::SkeletonNode**

  A skeleton node.

- class **gazebo::common::STLLoader**

  Class used to load STL mesh files.
- class **gazebo::common::SubMesh**

  A child mesh.
- class **gazebo::common::SystemPaths**

  Functions to handle getting system paths, keeps track of:
- class **gazebo::SystemPlugin**

  A plugin loaded within the gzserver on startup.
- class **gazebo::common::Time**

  *A **Time** (p. 831) class, can be used to hold wall- or sim-time.*
- class **gazebo::common::Timer**

  A timer class, used to time things in real world walltime.
- class **gazebo::common::Video**

  Handle video encoding and decoding using libavcodec.
- class **gazebo::VisualPlugin**

  A plugin loaded within the gzserver on startup.
- class **gazebo::WorldPlugin**

  *A plugin with access to **physics::World** (p. 945).*

Macros

- #define **gzclr_end** "\033[0m"

  End marker.
- #define **gzclr_start**(clr) "\033[1;33m"

  Start marker.
- #define **gzdbg** (**gazebo::common::Console::Instance**()->ColorMsg("Dbg", 36))

  Output a debug message.
- #define **gzerr**

  Output an error message.
- #define **gzlog** (**gazebo::common::Console::Instance**()->Log())

  Output a message to a log file.
- #define **gzmsg** (**gazebo::common::Console::Instance**()->ColorMsg("Msg", 32))

  Output a message.
- #define **gzthrow**(msg)

  This macro logs an error to the throw stream and throws an exception that contains the file name and line number.
- #define **gzwarn**

  Output a warning message.

Enumerations

- enum **gazebo::PluginType** {
  gazebo::WORLD_PLUGIN, **gazebo::MODEL_PLUGIN**, **gazebo::SENSOR_PLUGIN**, **gazebo::SYSTEM_PLUGIN**,
  gazebo::VISUAL_PLUGIN }

  Used to specify the type of plugin.

Functions

- **gazebo::common::Console::NullStream::NullStream** ()

  constructor
- void **gazebo::common::add_search_path_suffix** (const std::string &_suffix)

  *add path prefix to **common::SystemPaths** (p. 824)*
- std::ostream & **gazebo::common::Console::ColorErr** (const std::string &_lbl, const std::string &_file, unsigned int _line, int _color)

  Use this to output an error to the terminal.
- std::ostream & **gazebo::common::Console::ColorMsg** (const std::string &_lbl, int _color)

  Use this to output a colored message to the terminal.
- void **gazebo::common::ModelDatabase::DownloadDependencies** (const std::string &_path)

  Download all dependencies for a give model path.
- std::string **gazebo::common::find_file** (const std::string &_file)

  *search for file in **common::SystemPaths** (p. 824)*
- std::string **gazebo::common::find_file** (const std::string &_file, bool _searchLocalPath)

  *search for file in **common::SystemPaths** (p. 824)*
- std::string **gazebo::common::find_file_path** (const std::string &_file)

  *search for a file in **common::SystemPaths** (p. 824)*
- void **gazebo::common::ModelDatabase::Fini** ()

  Finalize the model database.
- std::string **gazebo::common::ModelDatabase::GetDBConfig** (const std::string &_uri)

  Return the database.config file as a string.
- std::string **gazebo::common::ModelDatabase::GetManifest** (const std::string &_uri) **GAZEBO_DEPRECATED**(1.5)

  Deprecated.
- std::string **gazebo::common::ModelDatabase::GetModelConfig** (const std::string &_uri)

  Return the model.config file as a string.
- std::string **gazebo::common::ModelDatabase::GetModelFile** (const std::string &_uri)

  Get a model's SDF file based on a URI.
- std::string **gazebo::common::ModelDatabase::GetModelName** (const std::string &_uri)

  Get the name of a model based on a URI.
- std::string **gazebo::common::ModelDatabase::GetModelPath** (const std::string &_uri, bool _forceDownload=false)

  Get the local path to a model.
- std::map< std::string, std::string > **gazebo::common::ModelDatabase::GetModels** ()

  Returns the dictionary of all the model names.
- void **gazebo::common::ModelDatabase::GetModels** (boost::function< void(const std::map< std::string, std::string > &)> _func)

  Get the dictionary of all model names via a callback.
- bool **gazebo::common::Console::GetQuiet** () const

  Get whether quiet output is set.
- std::string **gazebo::common::ModelDatabase::GetURI** ()

  Returns the the global model database URI.
- bool **gazebo::common::ModelDatabase::HasModel** (const std::string &_modelName)

  Returns true if the model exists on the database.
- void **gazebo::common::Console::Init** (const std::string &_logFilename)

Load the message parameters.

- bool **gazebo::common::Console::IsInitialized** () const

  Return true if Init has been called.

- std::ofstream & **gazebo::common::Console::Log** ()

  Use this to output a colored message to the terminal.

- void **gazebo::common::Console::SetQuiet** (bool _q)

  Set quiet output.

- void **gazebo::common::ModelDatabase::Start** (bool _fetchImmediately=false)

  Start the model database.

Variables

- static std::string **gazebo::common::PixelFormatNames** [ ]

  String names for the pixel formats.

8.1.1 Detailed Description

8.1.2 Macro Definition Documentation

8.1.2.1 #define gzclr_end "\033[0m"

End marker.

8.1.2.2 #define gzclr_start(*clr*) "\033[1;33m"

Start marker.

8.1.2.3 #define gzdbg (gazebo::common::Console::Instance()->ColorMsg("Dbg", 36))

Output a debug message.

8.1.2.4 #define gzerr

Value:

```
(gazebo::common::Console::Instance()->ColorErr("Error", \
          __FILE__, __LINE__, 31))
```

Output an error message.

Referenced by gazebo::transport::Connection::AsyncRead(), gazebo::event::Events::ConnectWorldUpdateStart(), gazebo::PluginT< ModelPlugin >::Create(), sdf::Param::Get(), sdf::Element::Get(), gazebo::physics::ScrewJoint< T >::Load(), sdf::ParamT< std::string >::Set(), and sdf::ParamT< std::string >::Update().

8.1.2.5 #define gzlog (gazebo::common::Console::Instance()->Log())

Output a message to a log file.

8.1.2.6 #define gzmsg (gazebo::common::Console::Instance()->ColorMsg("Msg", 32))

Output a message.

Referenced by sdf::ParamT< std::string >::Set().

8.1.2.7 #define gzthrow(*msg*)

Value:

```
{std::ostringstream throwStream;\
      throwStream << msg << std::endl << std::flush;\
      throw gazebo::common::Exception(__FILE__, __LINE__, throwStream.str()); }
```

This macro logs an error to the throw stream and throws an exception that contains the file name and line number.

Referenced by gazebo::transport::TopicManager::Advertise(), gazebo::PluginT< ModelPlugin >::Create(), gazebo::transport::CallbackHelperT< M >::GetMsgType(), and gazebo::transport::SubscribeOptions::Init().

8.1.2.8 #define gzwarn

Value:

```
(gazebo::common::Console::Instance()->ColorErr("Warning", \
          __FILE__, __LINE__, 33))
```

Output a warning message.

8.1.3 Enumeration Type Documentation

8.1.3.1 enum gazebo::PluginType

Used to specify the type of plugin.

Enumerator:

WORLD_PLUGIN A World plugin.

MODEL_PLUGIN A Model plugin.

SENSOR_PLUGIN A Sensor plugin.

SYSTEM_PLUGIN A System plugin.

VISUAL_PLUGIN A Visual plugin.

8.1.4 Function Documentation

8.1.4.1 gazebo::common::Console::NullStream::NullStream ( ) [inline]

constructor

8.1.4.2 void gazebo::common::add_search_path_suffix (const std::string & *_suffix*)

add path prefix to **common::SystemPaths** (p. 824)

8.1.4.3 std::ostream& gazebo::common::Console::ColorErr (const std::string & *_lbl,* const std::string & *_file,* unsigned int *_line,* int *_color*)

Use this to output an error to the terminal.

Parameters

in	*_lbl*	Text label
in	*_file*	File containing the error
in	*_line*	Line containing the error
in	*_color*	**Color** (p. 213) to make the label

Returns

Reference to an output stream

8.1.4.4 std::ostream& gazebo::common::Console::ColorMsg (const std::string & *_lbl,* int *_color*)

Use this to output a colored message to the terminal.

Parameters

in	*_lbl*	Text label
in	*_color*	**Color** (p. 213) to make the label

Returns

Reference to an output stream

8.1.4.5 void gazebo::common::ModelDatabase::DownloadDependencies (const std::string & *_path*)

Download all dependencies for a give model path.

Look's in the model's manifest file (_path/model.config) for all models listed in the <depend> block, and downloads the models if necessary.

Parameters

in	*_path*	Path to a model.

8.1.4.6 std::string gazebo::common::find_file (const std::string & *_file*)

search for file in **common::SystemPaths** (p. 824)

Parameters

in	*_file*	Name of the file to find.

8.1.4.7 std::string gazebo::common::find_file (const std::string & _file, bool _searchLocalPath)

search for file in **common::SystemPaths** (p. 824)

Parameters

in	*_file*	Name of the file to find.
in	*_searchLocal-Path*	True to search in the current working directory.

8.1.4.8 std::string gazebo::common::find_file_path (const std::string & _file)

search for a file in **common::SystemPaths** (p. 824)

Parameters

in	*_file*	the file name to look for

Returns

The path containing the file

8.1.4.9 void gazebo::common::ModelDatabase::Fini ( )

Finalize the model database.

8.1.4.10 std::string gazebo::common::ModelDatabase::GetDBConfig (const std::string & _uri)

Return the database.config file as a string.

Returns

The database config file from the model database.

8.1.4.11 std::string gazebo::common::ModelDatabase::GetManifest (const std::string & _uri)

Deprecated.

See Also

ModelDatabase::GetModelConfig (p. 34)
ModelDatabase::GetDBConfig (p. 34)

8.1.4.12 std::string gazebo::common::ModelDatabase::GetModelConfig (const std::string & _uri)

Return the model.config file as a string.

Returns

The model config file from the model database.

8.1.4.13 std::string gazebo::common::ModelDatabase::GetModelFile (const std::string & _uri)

Get a model's SDF file based on a URI.

Get a model file based on a URI. If the model is on a remote server, then the model fetched and installed locally.

Parameters

in	*_uri*	The URI of the model

Returns

The full path and filename to the SDF file

8.1.4.14 std::string gazebo::common::ModelDatabase::GetModelName (const std::string & _uri)

Get the name of a model based on a URI.

The URI must be fully qualified: `http://gazebosim.org/gazebo_models/ground_plane` or model-://gazebo_models

Parameters

in	*_uri*	the model uri

Returns

the model's name.

8.1.4.15 std::string gazebo::common::ModelDatabase::GetModelPath (const std::string & _uri, bool _forceDownload = `false`)

Get the local path to a model.

Get the path to a model based on a URI. If the model is on a remote server, then the model fetched and installed locally.

Parameters

in	*_uri*	the model uri
in	*_forceDownload*	True to skip searching local paths.

Returns

path to a model directory

8.1.4.16 std::map<std::string, std::string> gazebo::common::ModelDatabase::GetModels ( )

Returns the dictionary of all the model names.

This is a blocking call. Which means it will wait for the **ModelDatabase** (p. 530) to download the model list.

Returns

a map of model names, indexed by their full URI.

8.1.4.17 void gazebo::common::ModelDatabase::GetModels (boost::function< void(const std::map< std::string, std::string > &)> _func)

Get the dictionary of all model names via a callback.

This is the non-blocking version of **ModelDatabase::GetModels** (p. 35)

Parameters

in	*_func*	Callback function that receives the list of models.

8.1.4.18 bool gazebo::common::Console::GetQuiet ( ) const

Get whether quiet output is set.

Returns

True to if quiet output is set

8.1.4.19 std::string gazebo::common::ModelDatabase::GetURI ( )

Returns the the global model database URI.

Returns

the URI.

8.1.4.20 bool gazebo::common::ModelDatabase::HasModel (const std::string & _modelName)

Returns true if the model exists on the database.

Parameters

in	*_modelName*	URI of the model (eg: model://my_model_name).

Returns

True if the model was found.

8.1.4.21 void gazebo::common::Console::Init (const std::string & _logFilename)

Load the message parameters.

8.1.4.22 bool gazebo::common::Console::IsInitialized ( ) const

Return true if Init has been called.

Returns

True is initialized.

8.1.4.23 std::ofstream& gazebo::common::Console::Log ( )

Use this to output a colored message to the terminal.

Parameters

in	*_lbl*	Text label

Returns

Reference to an output stream

8.1.4.24 void gazebo::common::Console::SetQuiet (bool *_q*)

Set quiet output.

Parameters

in	*q*	True to prevent warning

8.1.4.25 void gazebo::common::ModelDatabase::Start (bool *_fetchImmediately* = `false`)

Start the model database.

Parameters

in	*_fetchImmediately*	True to fetch the models without waiting.

8.1.5 Variable Documentation

8.1.5.1 std::string gazebo::common::PixelFormatNames[] [static]

Initial value:

```
=
    {
      "UNKNOWN_PIXEL_FORMAT",
      "L_INT8",
      "L_INT16",
      "RGB_INT8",
      "RGBA_INT8",
      "BGRA_INT8",
      "RGB_INT16",
      "RGB_INT32",
      "BGR_INT8",
      "BGR_INT16",
      "BGR_INT32",
      "R_FLOAT16",
      "RGB_FLOAT16",
      "R_FLOAT32",
      "RGB_FLOAT32",
      "BAYER_RGGB8",
      "BAYER_RGGR8",
      "BAYER_GBRG8",
      "BAYER_GRBG8"
    }
```

String names for the pixel formats.

See Also

Image::PixelFormat (p. 380).

8.2 Events

Namespaces

- namespace **gazebo::event**

  Event *(p. 299) namespace.*

Classes

- class **gazebo::event::Connection**

  A class that encapsulates a connection.
- class **gazebo::event::Event**

  Base class for all events.
- class **gazebo::event::Events**

  An ***Event*** *(p. 299) class to get notifications for simulator events.*
- class **gazebo::event::EventT< T >**

  A class for event processing.

Functions

- virtual **gazebo::event::EventT< T >::~EventT** ()

  Destructor.
- ConnectionPtr **gazebo::event::EventT< T >::Connect** (const boost::function< T > &_subscriber)

  Connect a callback to this event.
- unsigned int **gazebo::event::EventT< T >::ConnectionCount** () const

  Get the number of connections.
- virtual void **gazebo::event::EventT< T >::Disconnect** (ConnectionPtr _c)

  Disconnect a callback to this event.
- virtual void **gazebo::event::EventT< T >::Disconnect** (int _id)

  Disconnect a callback to this event.

8.2.1 Detailed Description

8.2.2 Function Documentation

8.2.2.1 template<typename T > gazebo::event::EventT< T >::~EventT ( ) [virtual]

Destructor.

Destructor. Deletes all the associated connections.

8.2.2.2 template<typename T> ConnectionPtr gazebo::event::EventT< T >::Connect (const boost::function< T > & *_subscriber*)

Connect a callback to this event.

Adds a connection.

Parameters

in	*_subscriber*	Pointer to a callback function

Returns

A **Connection** (p. 226) object, which will automatically call Disconnect when it goes out of scope

Parameters

in	*_subscriber*	the subscriber to connect

Referenced by gazebo::event::Events::ConnectAddEntity(), gazebo::physics::Collision::ConnectContact(), gazebo::event::Events::ConnectCreateEntity(), gazebo::rendering::Events::ConnectCreateScene(), gazebo::event::Events::ConnectDeleteEntity(), gazebo::event::Events::ConnectDiagTimerStart(), gazebo::event::Events::ConnectDiagTimerStop(), gazebo::physics::Link::ConnectEnabled(), gazebo::physics::Joint::ConnectJointUpdate(), gazebo::rendering::DepthCamera::ConnectNewDepthFrame(), gazebo::rendering::Camera::ConnectNewImageFrame(), gazebo::rendering::GpuLaser::ConnectNewLaserFrame(), gazebo::physics::MultiRayShape::ConnectNewLaserScans(), gazebo::rendering::DepthCamera::ConnectNewRGBPointCloud(), gazebo::event::Events::ConnectPause(), gazebo::event::Events::ConnectPostRender(), gazebo::event::Events::ConnectPreRender(), gazebo::rendering::Events::ConnectRemoveScene(), gazebo::event::Events::ConnectRender(), gazebo::event::Events::ConnectSetSelectedEntity(), gazebo::event::Events::ConnectSigInt(), gazebo::event::Events::ConnectStep(), gazebo::event::Events::ConnectStop(), gazebo::transport::Connection::ConnectToShutdown(), gazebo::sensors::ForceTorqueSensor::ConnectUpdate(), gazebo::sensors::SonarSensor::ConnectUpdate(), gazebo::sensors::Sensor::ConnectUpdated(), gazebo::event::Events::ConnectWorldCreated(), gazebo::event::Events::ConnectWorldUpdateBegin(), gazebo::event::Events::ConnectWorldUpdateEnd(), and gazebo::event::Events::ConnectWorldUpdateStart().

8.2.2.3 template<typename T > unsigned int gazebo::event::EventT< T >::ConnectionCount ( ) const

Get the number of connections.

Returns

Number of connection to this **Event** (p. 299).

8.2.2.4 template<typename T > void gazebo::event::EventT< T >::Disconnect (ConnectionPtr _c) [virtual]

Disconnect a callback to this event.

Removes a connection.

Parameters

in	*_c*	The connection to disconnect
in	*_c*	the connection

Implements **gazebo::event::Event** (p. 301).

References NULL.

Referenced by gazebo::event::Events::DisconnectAddEntity(), gazebo::physics::Collision::DisconnectContact(), gazebo::event::Events::DisconnectCreateEntity(), gazebo::rendering::Events::DisconnectCreateScene(), gazebo::event::Events::DisconnectDeleteEntity(), gazebo::event::Events::DisconnectDiagTimerStart(), gazebo::event::Events::DisconnectDiagTimerStop(), gazebo::physics::Link::DisconnectEnabled(), gazebo::physics::Joint::DisconnectJoint-

Update(), gazebo::rendering::DepthCamera::DisconnectNewDepthFrame(), gazebo::rendering::Camera::DisconnectNewImageFrame(), gazebo::rendering::GpuLaser::DisconnectNewLaserFrame(), gazebo::physics::MultiRayShape::DisconnectNewLaserScans(), gazebo::rendering::DepthCamera::DisconnectNewRGBPointCloud(), gazebo::event::Events::DisconnectPause(), gazebo::event::Events::DisconnectPostRender(), gazebo::event::Events::DisconnectPreRender(), gazebo::rendering::Events::DisconnectRemoveScene(), gazebo::event::Events::DisconnectRender(), gazebo::event::Events::DisconnectSetSelectedEntity(), gazebo::transport::Connection::DisconnectShutdown(), gazebo::event::Events::DisconnectSigInt(), gazebo::event::Events::DisconnectStep(), gazebo::event::Events::DisconnectStop(), gazebo::sensors::ForceTorqueSensor::DisconnectUpdate(), gazebo::sensors::SonarSensor::DisconnectUpdate(), gazebo::sensors::Sensor::DisconnectUpdated(), gazebo::event::Events::DisconnectWorldCreated(), and gazebo::event::Events::DisconnectWorldUpdateEnd().

8.2.2.5 template<typename T > void gazebo::event::EventT< T >::Disconnect (int *_id*) [virtual]

Disconnect a callback to this event.

Removes a connection.

Parameters

in	*_id*	The id of the connection to disconnect
in	*_id*	the connection index

Implements **gazebo::event::Event** (p. 301).

8.3 Math

A set of classes that encapsulate math related properties and functions.

Files

- file **MathTypes.hh**

  Forward declarations for the math classes.

Namespaces

- namespace **gazebo::math**

  Math namespace.

Classes

- class **gazebo::math::Angle**

  An angle and related functions.
- class **gazebo::math::Box**

  Mathematical representation of a box and related functions.
- class **gazebo::math::Matrix3**

  A 3x3 matrix class.
- class **gazebo::math::Matrix4**

  A 3x3 matrix class.
- class **gazebo::math::Plane**

  A plane and related functions.
- class **gazebo::math::Pose**

  Encapsulates a position and rotation in three space.
- class **gazebo::math::Quaternion**

  A quaternion class.
- class **gazebo::math::Rand**

  Random number generator class.
- class **gazebo::math::RotationSpline**

  Spline *(p. 793) for rotations.*
- class **gazebo::math::Spline**

  Splines.
- class **gazebo::math::Vector2d**

  Generic double x, y vector.
- class **gazebo::math::Vector2i**

  Generic integer x, y vector.
- class **gazebo::math::Vector3**

  The ***Vector3*** *(p. 890) class represents the generic vector containing 3 elements.*
- class **gazebo::math::Vector4**

  double Generic x, y, z, w vector

Functions

- template<typename T >
  T **gazebo::math::clamp** (T _v, T _min, T _max)

  Simple clamping function.
- template<typename T >
  bool **gazebo::math::equal** (const T &_a, const T &_b, const T &_epsilon=1e-6)

  check if two values are equal, within a tolerance
- bool **gazebo::math::isnan** (float _v)

  check if a float is NaN
- bool **gazebo::math::isnan** (double _v)

  check if a double is NaN
- bool **gazebo::math::isPowerOfTwo** (unsigned int _x)

  is this a power of 2?
- template<typename T >
  T **gazebo::math::max** (const std::vector< T > &_values)

  get the maximum value of vector of values
- template<typename T >
  T **gazebo::math::mean** (const std::vector< T > &_values)

  get mean of vector of values
- template<typename T >
  T **gazebo::math::min** (const std::vector< T > &_values)

  get the minimum value of vector of values
- double **gazebo::math::parseFloat** (const std::string &_input)

  parse string into float
- int **gazebo::math::parseInt** (const std::string &_input)

  parse string into an integer
- template<typename T >
  T **gazebo::math::precision** (const T &_a, const unsigned int &_precision)

  get value at a specified precision
- template<typename T >
  T **gazebo::math::variance** (const std::vector< T > &_values)

  get variance of vector of values

Variables

- static const double **gazebo::math::NAN_D** = std::numeric_limits<double>::quiet_NaN()

  Returns the representation of a quiet not a number (NAN)
- static const int **gazebo::math::NAN_I** = std::numeric_limits<int>::quiet_NaN()

  Returns the representation of a quiet not a number (NAN)

8.3.1 Detailed Description

A set of classes that encapsulate math related properties and functions.

8.3.2 Function Documentation

8.3.2.1 template<typename T > T gazebo::math::clamp (T _v, T _min, T _max) [inline]

Simple clamping function.

Parameters

in	*_v*	value
in	*_min*	minimum
in	*_max*	maximum

References gazebo::math::max(), and gazebo::math::min().

8.3.2.2 template<typename T > bool gazebo::math::equal (const T & _a, const T & _b, const T & _epsilon = 1e-6) [inline]

check if two values are equal, within a tolerance

Parameters

in	*_a*	the first value
in	*_b*	the second value
in	*_epsilon*	the tolerance

Referenced by gazebo::math::Quaternion::Correct(), and gazebo::math::Quaternion::GetInverse().

8.3.2.3 bool gazebo::math::isnan (float _v) [inline]

check if a float is NaN

Parameters

in	*_v*	the value

Returns

true if _v is not a number, false otherwise

Referenced by gazebo::math::isnan().

8.3.2.4 bool gazebo::math::isnan (double _v) [inline]

check if a double is NaN

Parameters

in	*_v*	the value

Returns

true if _v is not a number, false otherwise

References gazebo::math::isnan().

8.3.2.5 bool gazebo::math::isPowerOfTwo (unsigned int *_x*) [inline]

is this a power of 2?

Parameters

in	*_x*	the number

Returns

true if _x is a power of 2, false otherwise

8.3.2.6 template<typename T > T gazebo::math::max (const std::vector< T > & *_values*) [inline]

get the maximum value of vector of values

Parameters

in	*_values*	the vector of values

Returns

maximum

References gazebo::math::min().

Referenced by gazebo::math::clamp(), and gazebo::math::min().

8.3.2.7 template<typename T > T gazebo::math::mean (const std::vector< T > & *_values*) [inline]

get mean of vector of values

Parameters

in	*_values*	the vector of values

Returns

the mean

8.3.2.8 template<typename T > T gazebo::math::min (const std::vector< T > & *_values*) [inline]

get the minimum value of vector of values

Parameters

in	*_values*	the vector of values

Returns

minimum

References gazebo::math::max().

Referenced by gazebo::math::clamp(), and gazebo::math::max().

8.3.2.9 double gazebo::math::parseFloat (const std::string & *_input*) [inline]

parse string into float

Parameters

_input	the string

Returns

a floating point number (can be NaN) or 0 with a message in the error stream

References gazebo::math::NAN_D.

8.3.2.10 int gazebo::math::parseInt (const std::string & *_input*) [inline]

parse string into an integer

Parameters

in	*_input*	the string

Returns

an integer, 0 or 0 and a message in the error stream

References gazebo::math::NAN_I.

8.3.2.11 template<typename T > T gazebo::math::precision (const T & *_a,* const unsigned int & *_precision*) [inline]

get value at a specified precision

Parameters

in	*_a*	the number
in	*_precision*	the precision

Returns

the value for the specified precision

8.3.2.12 template<typename T > T gazebo::math::variance (const std::vector< T > & _values) [inline]

get variance of vector of values

Parameters

in	*_values*	the vector of values

Returns

the squared deviation

8.3.3 Variable Documentation

8.3.3.1 const double gazebo::math::NAN_D = std::numeric_limits<double>::quiet_NaN() [static]

Returns the representation of a quiet not a number (NAN)

Referenced by gazebo::math::parseFloat().

8.3.3.2 const int gazebo::math::NAN_I = std::numeric_limits<int>::quiet_NaN() [static]

Returns the representation of a quiet not a number (NAN)

Referenced by gazebo::math::parseInt().

8.4 Messages

All messages and helper functions.

Namespaces

- namespace **gazebo::msgs**

  Messages namespace.

Classes

- class **google::protobuf::compiler::cpp::GazeboGenerator**

  *Google protobuf message generator for **gazebo::msgs*** (p. *91*).
- class **gazebo::msgs::MsgFactory**

  A factory that generates protobuf message based on a string type.

Macros

- #define **GZ_REGISTER_STATIC_MSG**(_msgtype, _classname)

  Static message registration macro.

Functions

- msgs::Vector3d **gazebo::msgs::Convert** (const math::Vector3 &_v)

  *Convert a **math::Vector3*** (p. *890*) *to a msgs::Vector3d.*
- msgs::Quaternion **gazebo::msgs::Convert** (const math::Quaternion &_q)

  *Convert a **math::Quaternion*** (p. *654*) *to a msgs::Quaternion.*
- msgs::Pose **gazebo::msgs::Convert** (const math::Pose &_p)

  *Convert a **math::Pose*** (p. *626*) *to a msgs::Pose.*
- msgs::Color **gazebo::msgs::Convert** (const common::Color &_c)

  *Convert a **common::Color*** (p. *213*) *to a msgs::Color.*
- msgs::Time **gazebo::msgs::Convert** (const common::Time &_t)

  *Convert a **common::Time*** (p. *831*) *to a msgs::Time.*
- msgs::PlaneGeom **gazebo::msgs::Convert** (const math::Plane &_p)

  *Convert a **math::Plane*** (p. *617*) *to a msgs::PlaneGeom.*
- math::Vector3 **gazebo::msgs::Convert** (const msgs::Vector3d &_v)

  Convert a msgs::Vector3d to a math::Vector.
- math::Quaternion **gazebo::msgs::Convert** (const msgs::Quaternion &_q)

  *Convert a msgs::Quaternion to a **math::Quaternion*** (p. *654*).
- math::Pose **gazebo::msgs::Convert** (const msgs::Pose &_p)

  *Convert a msgs::Pose to a **math::Pose*** (p. *626*).
- common::Color **gazebo::msgs::Convert** (const msgs::Color &_c)

  *Convert a msgs::Color to a **common::Color*** (p. *213*).
- common::Time **gazebo::msgs::Convert** (const msgs::Time &_t)

  *Convert a msgs::Time to a **common::Time*** (p. *831*).
- math::Plane **gazebo::msgs::Convert** (const msgs::PlaneGeom &_p)

Convert a msgs::PlaneGeom to a common::Plane.

- msgs::Request * **gazebo::msgs::CreateRequest** (const std::string &_request, const std::string &_data="")

  Create a request message.
- msgs::Fog **gazebo::msgs::FogFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::Fog from a fog SDF element.
- msgs::Geometry **gazebo::msgs::GeometryFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::Geometry from a geometry SDF element.
- msgs::Header * **gazebo::msgs::GetHeader** (google::protobuf::Message &_message)

  Get the header from a protobuf message.
- msgs::GUI **gazebo::msgs::GUIFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::GUI from a GUI SDF element.
- void **gazebo::msgs::Init** (google::protobuf::Message &_message, const std::string &_id="")

  Initialize a message.
- msgs::Light **gazebo::msgs::LightFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::Light from a light SDF element.
- msgs::MeshGeom **gazebo::msgs::MeshFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::MeshGeom from a mesh SDF element.
- msgs::Scene **gazebo::msgs::SceneFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::Scene from a scene SDF element.
- void **gazebo::msgs::Set** (common::Image &_img, const msgs::Image &_msg)

  *Convert a msgs::Image to a **common::Image** (p. 379).*
- void **gazebo::msgs::Set** (msgs::Image *_msg, const common::Image &_i)

  *Set a msgs::Image from a **common::Image** (p. 379).*
- void **gazebo::msgs::Set** (msgs::Vector3d *_pt, const math::Vector3 &_v)

  *Set a msgs::Vector3d from a **math::Vector3** (p. 890).*
- void **gazebo::msgs::Set** (msgs::Vector2d *_pt, const math::Vector2d &_v)

  *Set a msgs::Vector2d from a **math::Vector3** (p. 890).*
- void **gazebo::msgs::Set** (msgs::Quaternion *_q, const math::Quaternion &_v)

  *Set a msgs::Quaternion from a **math::Quaternion** (p. 654).*
- void **gazebo::msgs::Set** (msgs::Pose *_p, const math::Pose &_v)

  *Set a msgs::Pose from a **math::Pose** (p. 626).*
- void **gazebo::msgs::Set** (msgs::Color *_c, const common::Color &_v)

  *Set a msgs::Color from a **common::Color** (p. 213).*
- void **gazebo::msgs::Set** (msgs::Time *_t, const common::Time &_v)

  *Set a msgs::Time from a **common::Time** (p. 831).*
- void **gazebo::msgs::Set** (msgs::PlaneGeom *_p, const math::Plane &_v)

  *Set a msgs::Plane from a **math::Plane** (p. 617).*
- void **gazebo::msgs::Stamp** (msgs::Header *_header)

  Time stamp a header.
- void **gazebo::msgs::Stamp** (msgs::Time *_time)

  Set the time in a time message.
- msgs::TrackVisual **gazebo::msgs::TrackVisualFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::TrackVisual from a track visual SDF element.
- msgs::Visual **gazebo::msgs::VisualFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::Visual from a visual SDF element.

8.4.1 Detailed Description

All messages and helper functions.

8.4.2 Macro Definition Documentation

8.4.2.1 #define GZ_REGISTER_STATIC_MSG(_msgtype, _classname)

Value:

```
boost::shared_ptr<google::protobuf::Message> New##_classname() \
    { \
      return boost::shared_ptr<gazebo::msgs::_classname>(\
          new gazebo::msgs::_classname); \
    } \
    class Msg##_classname \
    { \
      public: Msg##_classname() \
      { \
        gazebo::msgs::MsgFactory::RegisterMsg(_msgtype, New##_classname);\
      } \
    }; \
    static Msg##_classname GzMsgInitializer;
```

Static message registration macro.

Use this macro to register messages.

Parameters

in	*_msgtype*	Message type name.
in	*_classname*	Class name for message.

8.4.3 Function Documentation

8.4.3.1 msgs::Vector3d gazebo::msgs::Convert (const math::Vector3 & _v)

Convert a **math::Vector3** (p. 890) to a msgs::Vector3d.

Parameters

in	*_v*	The vector to convert

Returns

A msgs::Vector3d object

8.4.3.2 msgs::Quaternion gazebo::msgs::Convert (const math::Quaternion & _q)

Convert a **math::Quaternion** (p. 654) to a msgs::Quaternion.

Parameters

in	*_q*	The quaternion to convert

Returns

A msgs::Quaternion object

8.4.3.3 msgs::Pose gazebo::msgs::Convert (const math::Pose & _p)

Convert a **math::Pose** (p. 626) to a msgs::Pose.

Parameters

in	_p	The pose to convert

Returns

A msgs::Pose object

8.4.3.4 msgs::Color gazebo::msgs::Convert (const common::Color & _c)

Convert a **common::Color** (p. 213) to a msgs::Color.

Parameters

in	_c	The color to convert

Returns

A msgs::Color object

8.4.3.5 msgs::Time gazebo::msgs::Convert (const common::Time & _t)

Convert a **common::Time** (p. 831) to a msgs::Time.

Parameters

in	_t	The time to convert

Returns

A msgs::Time object

8.4.3.6 msgs::PlaneGeom gazebo::msgs::Convert (const math::Plane & _p)

Convert a **math::Plane** (p. 617) to a msgs::PlaneGeom.

Parameters

in	_p	The plane to convert

Returns

A msgs::PlaneGeom object

8.4.3.7 math::Vector3 gazebo::msgs::Convert (const msgs::Vector3d & _v)

Convert a msgs::Vector3d to a math::Vector.

Parameters

in	*_v*	The plane to convert

Returns

A **math::Vector3** (p. 890) object

8.4.3.8 math::Quaternion gazebo::msgs::Convert (const msgs::Quaternion & _q)

Convert a msgs::Quaternion to a **math::Quaternion** (p. 654).

Parameters

in	*_q*	The quaternion to convert

Returns

A **math::Quaternion** (p. 654) object

8.4.3.9 math::Pose gazebo::msgs::Convert (const msgs::Pose & _p)

Convert a msgs::Pose to a **math::Pose** (p. 626).

Parameters

in	*_q*	The pose to convert

Returns

A **math::Pose** (p. 626) object

8.4.3.10 common::Color gazebo::msgs::Convert (const msgs::Color & _c)

Convert a msgs::Color to a **common::Color** (p. 213).

Parameters

in	*_c*	The color to convert

Returns

A **common::Color** (p. 213) object

8.4.3.11 common::Time gazebo::msgs::Convert (const msgs::Time & _t)

Convert a msgs::Time to a **common::Time** (p. 831).

Parameters

in	*_t*	The time to convert

Returns

A **common::Time** (p. 831) object

8.4.3.12 math::Plane gazebo::msgs::Convert (const msgs::PlaneGeom & _p)

Convert a msgs::PlaneGeom to a common::Plane.

Parameters

in	*_p*	The plane to convert

Returns

A common::Plane object

8.4.3.13 msgs::Request* gazebo::msgs::CreateRequest (const std::string & _request, const std::string & _data = " ")

Create a request message.

Parameters

in	*_request*	Request string
in	*_data*	Optional data string

Returns

A Request message

8.4.3.14 msgs::Fog gazebo::msgs::FogFromSDF (sdf::ElementPtr _sdf)

Create a msgs::Fog from a fog SDF element.

Parameters

in	*_sdf*	The sdf element

Returns

The new msgs::Fog object

8.4.3.15 msgs::Geometry gazebo::msgs::GeometryFromSDF (sdf::ElementPtr _sdf)

Create a msgs::Geometry from a geometry SDF element.

Parameters

in	*_sdf*	The sdf element

Returns

The new msgs::Geometry object

8.4.3.16 msgs::Header* gazebo::msgs::GetHeader (google::protobuf::Message & _message)

Get the header from a protobuf message.

Parameters

in	*_message*	A google protobuf message

Returns

A pointer to the message's header

8.4.3.17 msgs::GUI gazebo::msgs::GUIFromSDF (sdf::ElementPtr _sdf)

Create a msgs::GUI from a GUI SDF element.

Parameters

in	*_sdf*	The sdf element

Returns

The new msgs::GUI object

8.4.3.18 void gazebo::msgs::Init (google::protobuf::Message & _message, const std::string & _id = " ")

Initialize a message.

Parameters

in	*_message*	Message to initialize
in	*_id*	Optional string id

Referenced by gazebo::physics::HingeJoint< T >::Init().

8.4.3.19 msgs::Light gazebo::msgs::LightFromSDF (sdf::ElementPtr _sdf)

Create a msgs::Light from a light SDF element.

Parameters

in	*_sdf*	The sdf element

Returns

The new msgs::Light object

8.4.3.20 msgs::MeshGeom gazebo::msgs::MeshFromSDF (sdf::ElementPtr _sdf)

Create a msgs::MeshGeom from a mesh SDF element.

Parameters

in	*_sdf*	The sdf element

Returns

The new msgs::MeshGeom object

8.4.3.21 msgs::Scene gazebo::msgs::SceneFromSDF (sdf::ElementPtr _sdf)

Create a msgs::Scene from a scene SDF element.

Parameters

in	*_sdf*	The sdf element

Returns

The new msgs::Scene object

8.4.3.22 void gazebo::msgs::Set (common::Image & _img, const msgs::Image & _msg)

Convert a msgs::Image to a **common::Image** (p. 379).

Parameters

out	*_img*	The **common::Image** (p. 379) container
in	*_msg*	The Image message to convert

8.4.3.23 void gazebo::msgs::Set (msgs::Image * _msg, const common::Image & _i)

Set a msgs::Image from a **common::Image** (p. 379).

Parameters

out	*_msg*	A msgs::Image pointer
in	*_i*	A **common::Image** (p. 379) reference

8.4.3.24 void gazebo::msgs::Set (msgs::Vector3d * _pt, const math::Vector3 & _v)

Set a msgs::Vector3d from a **math::Vector3** (p. 890).

Parameters

out	*_pt*	A msgs::Vector3d pointer
in	*_v*	A **math::Vector3** (p. 890) reference

8.4.3.25 void gazebo::msgs::Set (msgs::Vector2d * _pt, const math::Vector2d & _v)

Set a msgs::Vector2d from a **math::Vector3** (p. 890).

Parameters

out	*_pt*	A msgs::Vector2d pointer
in	*_v*	A **math::Vector2d** (p. 873) reference

8.4.3.26 void gazebo::msgs::Set (msgs::Quaternion * _q, const math::Quaternion & _v)

Set a msgs::Quaternion from a **math::Quaternion** (p. 654).

Parameters

out	*_q*	A msgs::Quaternion pointer
in	*_v*	A **math::Quaternion** (p. 654) reference

8.4.3.27 void gazebo::msgs::Set (msgs::Pose * _p, const math::Pose & _v)

Set a msgs::Pose from a **math::Pose** (p. 626).

Parameters

out	*_p*	A msgs::Pose pointer
in	*_v*	A **math::Pose** (p. 626) reference

8.4.3.28 void gazebo::msgs::Set (msgs::Color * _c, const common::Color & _v)

Set a msgs::Color from a **common::Color** (p. 213).

Parameters

out	*_p*	A msgs::Color pointer
in	*_v*	A **common::Color** (p. 213) reference

8.4.3.29 void gazebo::msgs::Set (msgs::Time * *_t*, const common::Time & *_v*)

Set a msgs::Time from a **common::Time** (p. 831).

Parameters

out	*_p*	A msgs::Time pointer
in	*_v*	A **common::Time** (p. 831) reference

8.4.3.30 void gazebo::msgs::Set (msgs::PlaneGeom * *_p*, const math::Plane & *_v*)

Set a msgs::Plane from a **math::Plane** (p. 617).

Parameters

out	*_p*	A msgs::Plane pointer
in	*_v*	A **math::Plane** (p. 617) reference

8.4.3.31 void gazebo::msgs::Stamp (msgs::Header * *_header*)

Time stamp a header.

Parameters

in	*_header*	Header to stamp

8.4.3.32 void gazebo::msgs::Stamp (msgs::Time * *_time*)

Set the time in a time message.

Parameters

in	*_time*	A Time message

8.4.3.33 msgs::TrackVisual gazebo::msgs::TrackVisualFromSDF (sdf::ElementPtr *_sdf*)

Create a msgs::TrackVisual from a track visual SDF element.

Parameters

in	*_sdf*	The sdf element

Returns

The new msgs::TrackVisual object

8.4.3.34 msgs::Visual gazebo::msgs::VisualFromSDF (sdf::ElementPtr *_sdf*)

Create a msgs::Visual from a visual SDF element.

Parameters

in	*_sdf*	The sdf element

Returns

The new msgs::Visual object

8.5 Classes for physics and dynamics

Files

- file **PhysicsTypes.hh**

  default namespace for gazebo

Namespaces

- namespace **gazebo::physics**

  namespace for physics

Classes

- class **gazebo::physics::Actor**

  Actor (p. *115) class enables GPU based mesh model / skeleton scriptable animation.*
- class **gazebo::physics::BallJoint**< **T** >

  Base (p. *141) class for a ball joint.*
- class **gazebo::physics::Base**

  Base (p. *141) class for most physics classes.*
- class **gazebo::physics::BoxShape**

  Box geometry primitive.
- class **gazebo::physics::Collision**

  Base (p. *141) class for all collision entities.*
- class **gazebo::physics::CollisionState**

  Store state information of a ***physics::Collision*** (p. *199) object.*
- class **gazebo::physics::Contact**

  A contact between two collisions.
- class **gazebo::physics::ContactManager**

  Aggregates all the contact information generated by the collision detection engine.
- class **gazebo::physics::CylinderShape**

  Cylinder collision.
- class **gazebo::physics::Entity**

  Base (p. *141) class for all physics objects in Gazebo.*
- class **gazebo::physics::Gripper**

  A gripper abstraction.
- class **gazebo::physics::HeightmapShape**

  HeightmapShape (p. *371) collision shape builds a heightmap from an image.*
- class **gazebo::physics::Hinge2Joint**< **T** >

  A two axis hinge joint.
- class **gazebo::physics::HingeJoint**< **T** >

  A single axis hinge joint.
- class **gazebo::physics::Inertial**

  A class for inertial information about a link.
- class **gazebo::physics::Joint**

  Base (p. *141) class for all joints.*
- class **gazebo::physics::JointController**

*A class for manipulating **physics::Joint*** (p. *401*).

- class **gazebo::physics::JointState**

  *keeps track of state of a **physics::Joint*** (p. *401*)
- class **gazebo::physics::JointWrench**

  Wrench information from a joint.
- class **gazebo::physics::Link**

  Link (p. *439*) *class defines a rigid body entity, containing information on inertia, visual and collision properties of a rigid body.*
- class **gazebo::physics::LinkState**

  *Store state information of a **physics::Link*** (p. *439*) *object.*
- class **Logplay**

  Open and playback log files that were recorded using LogRecord.
- class **gazebo::util::LogPlay**
- class **gazebo::physics::MeshShape**

  Triangle mesh collision shape.
- class **gazebo::physics::Model**

  A model is a collection of links, joints, and plugins.
- class **gazebo::physics::ModelState**

  *Store state information of a **physics::Model*** (p. *516*) *object.*
- class **gazebo::physics::MultiRayShape**

  Laser collision contains a set of ray-collisions, structured to simulate a laser range scanner.
- class **gazebo::physics::PhysicsEngine**

  Base (p. *141*) *class for a physics engine.*
- class **gazebo::physics::PhysicsFactory**

  The physics factory instantiates different physics engines.
- class **gazebo::physics::PlaneShape**

  Collision (p. *199*) *for an infinite plane.*
- class **gazebo::physics::RayShape**

  Base (p. *141*) *class for Ray collision geometry.*
- class **gazebo::physics::Road**

  *for building a **Road*** (p. *697*) *from SDF*
- class **gazebo::physics::ScrewJoint< T >**

  A screw joint, which has both prismatic and rotational DOFs.
- class **gazebo::physics::Shape**

  Base (p. *141*) *class for all shapes.*
- class **gazebo::physics::SliderJoint< T >**

  A slider joint.
- class **gazebo::physics::SphereShape**

  Sphere collision shape.
- class **gazebo::physics::State**

  State (p. *797*) *of an entity.*
- class **gazebo::physics::SurfaceParams**

  SurfaceParams (p. *820*) *defines various Surface contact parameters.*
- class **gazebo::physics::UniversalJoint< T >**

  A universal joint.
- class **gazebo::physics::World**

  The world provides access to all other object within a simulated environment.
- class **gazebo::physics::WorldState**

  *Store state information of a **physics::World*** (p. *945*) *object.*

Macros

- #define **GZ_REGISTER_PHYSICS_ENGINE**(name, classname)

  Static physics registration macro.

Typedefs

- typedef PhysicsEnginePtr(∗ **gazebo::physics::PhysicsFactoryFn**)(WorldPtr world)

Functions

- WorldPtr **gazebo::physics::create_world** (const std::string &_name="")

  Create a world given a name.
- bool **gazebo::physics::fini** ()

  *Finalize transport by calling **gazebo::transport::fini** (p. 78).*
- WorldPtr **gazebo::physics::get_world** (const std::string &_name="")

  Returns a pointer to a world by name.
- void **gazebo::physics::init_world** (WorldPtr _world)

  Init world given a pointer to it.
- void **gazebo::physics::init_worlds** ()

  initialize multiple worlds stored in static variable gazebo::g_worlds
- bool **gazebo::physics::load** ()

  *Setup **gazebo::SystemPlugin** (p. 829)'s and call **gazebo::transport::init** (p. 79).*
- void **gazebo::physics::load_world** (WorldPtr _world, **sdf::ElementPtr** _sdf)

  *Load world from **sdf::Element** (p. 280) pointer.*
- void **gazebo::physics::load_worlds** (**sdf::ElementPtr** _sdf)

  *load multiple worlds from single **sdf::Element** (p. 280) pointer*
- void **gazebo::physics::pause_world** (WorldPtr _world, bool _pause)

  *Pause world by calling **World::SetPaused** (p. 955).*
- void **gazebo::physics::pause_worlds** (bool pause)

  pause multiple worlds stored in static variable gazebo::g_worlds
- void **gazebo::physics::remove_worlds** ()

  remove multiple worlds stored in static variable gazebo::g_worlds
- void **gazebo::physics::run_world** (WorldPtr _world, unsigned int _iterations=0)

  *Run world by calling **World::Run()** (p. 955) given a pointer to it.*
- void **gazebo::physics::run_worlds** (unsigned int _iterations=0)

  Run multiple worlds stored in static variable gazebo::g_worlds.
- void **gazebo::physics::stop_world** (WorldPtr _world)

  *Stop world by calling **World::Stop()** (p. 956) given a pointer to it.*
- void **gazebo::physics::stop_worlds** ()

  stop multiple worlds stored in static variable gazebo::g_worlds
- bool **gazebo::physics::worlds_running** ()

  Return true if any world is running.

Variables

- static std::string **gazebo::physics::EntityTypename** [ ]

  String names for the different entity types.

8.5.1 Detailed Description

8.5.2 Macro Definition Documentation

8.5.2.1 #define GZ_REGISTER_PHYSICS_ENGINE(*name, classname*)

Value:

```
PhysicsEnginePtr New##classname(WorldPtr _world) \
    { \
      return PhysicsEnginePtr(new gazebo::physics::classname(_world)); \
    } \
    void Register##classname() \
    {\
      PhysicsFactory::RegisterPhysicsEngine(name, New##classname);\
    }
```

Static physics registration macro.

Use this macro to register physics engine with the server.

Parameters

in	*name*	Physics type name, as it appears in the world file.
in	*classname*	C++ class name for the physics engine.

8.5.3 Typedef Documentation

8.5.3.1 typedef PhysicsEnginePtr(∗ gazebo::physics::PhysicsFactoryFn)(WorldPtr world)

8.5.4 Function Documentation

8.5.4.1 WorldPtr gazebo::physics::create_world (const std::string & *_name* = " ")

Create a world given a name.

Parameters

in	*_name*	Name of the world to create.

Returns

Pointer to the new world.

8.5.4.2 bool gazebo::physics::fini ( )

Finalize transport by calling **gazebo::transport::fini** (p. 78).

8.5.4.3 WorldPtr gazebo::physics::get_world (const std::string & *_name* = " ")

Returns a pointer to a world by name.

Parameters

in	*_name*	Name of the world to get.

Returns

Pointer to the world.

8.5.4.4 void gazebo::physics::init_world (WorldPtr *_world*)

Init world given a pointer to it.

Parameters

in	*_world*	**World** (p. 945) to initialize.

8.5.4.5 void gazebo::physics::init_worlds ( )

initialize multiple worlds stored in static variable gazebo::g_worlds

8.5.4.6 bool gazebo::physics::load ( )

Setup **gazebo::SystemPlugin** (p. 829)'s and call **gazebo::transport::init** (p. 79).

8.5.4.7 void gazebo::physics::load_world (WorldPtr *_world,* sdf::ElementPtr *_sdf*)

Load world from **sdf::Element** (p. 280) pointer.

Parameters

in	*_world*	Pointer to a world.
in	*_sdf*	SDF values to load from.

8.5.4.8 void gazebo::physics::load_worlds (sdf::ElementPtr *_sdf*)

load multiple worlds from single **sdf::Element** (p. 280) pointer

Parameters

in	*_sdf*	SDF values used to create worlds.

8.5.4.9 void gazebo::physics::pause_world (WorldPtr *_world,* bool *_pause*)

Pause world by calling **World::SetPaused** (p. 955).

Parameters

in	*_world*	**World** (p. 945) to pause or unpause.
in	*_pause*	True to pause, False to unpause.

8.5.4.10 void gazebo::physics::pause_worlds (bool *pause*)

pause multiple worlds stored in static variable gazebo::g_worlds

Parameters

in	*_pause*	True to pause, False to unpause.

8.5.4.11 void gazebo::physics::remove_worlds ( )

remove multiple worlds stored in static variable gazebo::g_worlds

8.5.4.12 void gazebo::physics::run_world (WorldPtr *_world,* unsigned int *_iterations* = 0)

Run world by calling **World::Run()** (p. 955) given a pointer to it.

Parameters

in	*_world*	**World** (p. 945) to run.
in	*_iterations*	Number of iterations for each world to take. Zero indicates that each world should continue forever.

8.5.4.13 void gazebo::physics::run_worlds (unsigned int *_iterations* = 0)

Run multiple worlds stored in static variable gazebo::g_worlds.

Parameters

in	*_iterations*	Number of iterations for each world to take. Zero indicates that each world should continue forever.

8.5.4.14 void gazebo::physics::stop_world (WorldPtr *_world*)

Stop world by calling **World::Stop()** (p. 956) given a pointer to it.

Parameters

in	*_world*	**World** (p. 945) to stop.

8.5.4.15 void gazebo::physics::stop_worlds ( )

stop multiple worlds stored in static variable gazebo::g_worlds

8.5.4.16 bool gazebo::physics::worlds_running ( )

Return true if any world is running.

Returns

True if any world is running.

8.5.5 Variable Documentation

8.5.5.1 std::string gazebo::physics::EntityTypename[] [static]

Initial value:

```
= {
      "common",
      "entity",
      "model",
      "actor",
      "link",
      "collision",
      "light",
      "visual",
      "joint",
      "ball",
      "hinge2",
      "hinge",
      "slider",
      "universal",
      "shape",
      "box",
      "cylinder",
      "heightmap",
      "map",
      "multiray",
      "ray",
      "plane",
      "sphere",
      "trimesh"
    }
```

String names for the different entity types.

8.6 Rendering

A set of rendering related class, functions, and definitions.

Namespaces

- namespace **gazebo::rendering**

  Rendering namespace.

Classes

- class **gazebo::rendering::ArrowVisual**

  Basic arrow visualization.
- class **gazebo::rendering::AxisVisual**

  Basic axis visualization.
- class **gazebo::rendering::Camera**

  Basic camera sensor.
- class **gazebo::rendering::CameraVisual**

  Basic camera visualization.
- class **gazebo::rendering::COMVisual**

  Basic Center of Mass visualization.
- class **gazebo::rendering::ContactVisual**

  Contact visualization.
- class **gazebo::rendering::Conversions**

  Conversions *(*p. *254)* ***Conversions.hh*** *(*p. *1006)* ***rendering/Conversions.hh*** *(*p. *1006).*
- class **gazebo::rendering::DepthCamera**

  Depth camera used to render depth data into an image buffer.
- class **gazebo::rendering::DynamicLines**

  Class for drawing lines that can change.
- class **gazebo::rendering::DynamicRenderable**

  Abstract base class providing mechanisms for dynamically growing hardware buffers.
- class **gazebo::rendering::Events**

  Base class for rendering events.
- class **gazebo::rendering::FPSViewController**

  First Person Shooter style view controller.
- class **gazebo::rendering::GpuLaser**

  GPU based laser distance sensor.
- class **gazebo::rendering::Grid**

  Displays a grid of cells, drawn with lines.
- class **gazebo::rendering::GUIOverlay**

  A class that creates a CEGUI overlay on a render window.
- class **gazebo::rendering::Heightmap**

  Rendering a terrain using heightmap information.
- class **gazebo::rendering::JointVisual**

  Visualization for joints.
- class **gazebo::rendering::LaserVisual**

Visualization for laser data.

- class **gazebo::rendering::Light**

  A light source.

- class **gazebo::rendering::MovableText**

  Movable text.

- class **gazebo::rendering::OrbitViewController**

  Orbit view controller.

- class **gazebo::rendering::Projector**

  Projects a material onto surface, light a light projector.

- class **gazebo::rendering::RenderEngine**

  Adaptor to Ogre3d.

- class **gazebo::rendering::RFIDTagVisual**

  Visualization for RFID tags sensor.

- class **gazebo::rendering::RFIDVisual**

  Visualization for RFID sensor.

- class **Road**

  Used to render a strip of road.

- class **gazebo::rendering::Road2d**
- class **gazebo::rendering::RTShaderSystem**

  *Implements **Ogre** (p. 110)'s Run-Time Shader system.*

- class **gazebo::rendering::Scene**

  Representation of an entire scene graph.

- class **gazebo::rendering::SelectionObj**

  A graphical selection object.

- class **gazebo::rendering::SonarVisual**

  Visualization for sonar data.

- class **gazebo::rendering::UserCamera**

  A camera used for user visualization of a scene.

- class **gazebo::rendering::VideoVisual**

  A visual element that displays a video as a texture.

- class **gazebo::rendering::ViewController**

  Base class for view controllers.

- class **gazebo::rendering::Visual**

  A renderable object.

- class **gazebo::rendering::WindowManager**

  Class to mangage render windows.

- class **gazebo::rendering::WireBox**

  Draws a wireframe box.

- class **gazebo::rendering::WrenchVisual**

  Visualization for sonar data.

Functions

- rendering::ScenePtr **gazebo::rendering::create_scene** (const std::string &_name, bool _enableVisualizations)

  *create **rendering::Scene*** (p. *707) by name.*
- bool **gazebo::rendering::fini** ()

  teardown rendering engine.
- rendering::ScenePtr **gazebo::rendering::get_scene** (const std::string &_name="")

  *get pointer to **rendering::Scene*** (p. *707) by name.*
- bool **gazebo::rendering::init** ()

  init rendering engine.
- bool **gazebo::rendering::load** ()

  load rendering engine.
- void **gazebo::rendering::remove_scene** (const std::string &_name)

  *remove a **rendering::Scene*** (p. *707) by name*

8.6.1 Detailed Description

A set of rendering related class, functions, and definitions.

8.6.2 Function Documentation

8.6.2.1 rendering::ScenePtr gazebo::rendering::create_scene (const std::string & *_name,* bool *_enableVisualizations*)

create **rendering::Scene** (p. 707) by name.

Parameters

in	*_name*	Name of the scene to create.
in	*_enable-Visualizations*	True enables visualization elements such as laser lines.

8.6.2.2 bool gazebo::rendering::fini ( )

teardown rendering engine.

8.6.2.3 rendering::ScenePtr gazebo::rendering::get_scene (const std::string & *_name =* " ")

get pointer to **rendering::Scene** (p. 707) by name.

Parameters

in	*_name*	Name of the scene to retreive.

8.6.2.4 bool gazebo::rendering::init ( )

init rendering engine.

8.6.2.5 bool gazebo::rendering::load ( )

load rendering engine.

8.6.2.6 void gazebo::rendering::remove_scene (const std::string & *_name*)

remove a **rendering::Scene** (p. 707) by name

Parameters

in	*_name*	The name of the scene to remove.

8.7 Gazebo_parser

Namespaces

- namespace **sdf**

  namespace for Simulation Description Format parser

Classes

- class **sdf::Element**

  SDF *(*p. *728)* ***Element*** *(*p. *280) class.*
- class **sdf::SDF**

  Base ***SDF*** *(*p. *728) class.*

8.7.1 Detailed Description

8.8 Sensors

A set of sensor classes, functions, and definitions.

Files

- file **SensorTypes.hh**

  Forward declarations and typedefs for sensors.

Namespaces

- namespace **gazebo::sensors**

  Sensors namespace.

Classes

- class **gazebo::sensors::CameraSensor**

  Basic camera sensor.
- class **gazebo::sensors::ContactSensor**

  Contact sensor.
- class **gazebo::sensors::DepthCameraSensor**
- class **gazebo::sensors::ForceTorqueSensor**

  Sensor *(p. 731) for measure force and torque on a joint.*
- class **gazebo::sensors::GpuRaySensor**
- class **gazebo::sensors::ImuSensor**

  An IMU sensor.
- class **gazebo::sensors::MultiCameraSensor**

  Multiple camera sensor.
- class **gazebo::sensors::RaySensor**

  Sensor *(p. 731) with one or more rays.*
- class **gazebo::sensors::RFIDSensor**

  Sensor *(p. 731) class for RFID type of sensor.*
- class **gazebo::sensors::RFIDTag**

  RFIDTag *(p. 691) to interact with RFIDTagSensors.*
- class **gazebo::sensors::Sensor**

  Base class for sensors.
- class **SensorFactor**

  The sensor factory; the class is just for namespacing purposes.
- class **gazebo::sensors::SensorFactory**
- class **gazebo::sensors::SensorManager**

  Class to manage and update all sensors.
- class **gazebo::sensors::SonarSensor**

  Sensor *(p. 731) with sonar cone.*

Macros

- #define **GZ_REGISTER_STATIC_SENSOR**(name, classname)

  Static sensor registration macro.

Functions

- std::string **gazebo::sensors::create_sensor** (**sdf::ElementPtr** _elem, const std::string &_worldName, const std::string &_parentName)

  Create a sensor using SDF.
- bool **gazebo::sensors::fini** ()

  shutdown the sensor generation loop.
- SensorPtr **gazebo::sensors::get_sensor** (const std::string &_name)

  Get a sensor using by name.
- bool **gazebo::sensors::init** ()

  initialize the sensor generation loop.
- bool **gazebo::sensors::load** ()

  Load the sensor library.
- void **gazebo::sensors::remove_sensor** (const std::string &_sensorName)

  Remove a sensor by name.
- bool **gazebo::sensors::remove_sensors** ()

  Remove all sensors.
- void **gazebo::sensors::run** () **GAZEBO_DEPRECATED**(1.5)

  Deprecated.
- void **gazebo::sensors::run_once** (bool _force=false)

  Run the sensor generation one step.
- void **gazebo::sensors::run_threads** ()

  Run sensors in a threads. This is a non-blocking call.
- void **gazebo::sensors::stop** ()

  Stop the sensor generation loop.

8.8.1 Detailed Description

A set of sensor classes, functions, and definitions. GPU based laser sensor.

Depth camera sensor This sensor is used for simulating standard monocular cameras

This sensor cast rays into the world, tests for intersections, and reports the range to the nearest object. It is used by ranging sensor models (e.g., sonars and scanning laser range finders).

8.8.2 Macro Definition Documentation

8.8.2.1 #define GZ_REGISTER_STATIC_SENSOR(*name, classname*)

Value:

```
Sensor *New##classname() \
  { \
    return new gazebo::sensors::classname(); \
  } \
  void Register##classname() \
  {\
    SensorFactory::RegisterSensor(name, New##classname);\
  }
```

Static sensor registration macro.

Use this macro to register sensors with the server.

Parameters

name	Sensor type name, as it appears in the world file.
classname	C++ class name for the sensor.

8.8.3 Function Documentation

8.8.3.1 std::string gazebo::sensors::create_sensor (sdf::ElementPtr *_elem,* const std::string & *_worldName,* const std::string & *_parentName*)

Create a sensor using SDF.

Parameters

in	*_elem*	The SDF element that describes the sensor.
in	*_worldName*	Name of the world in which to create the sensor.
in	*_parentName*	The fully scoped parent name (model::link).

Returns

The name of the new sensor.

8.8.3.2 bool gazebo::sensors::fini ( )

shutdown the sensor generation loop.

Returns

True if successfully finalized, false if not

8.8.3.3 SensorPtr gazebo::sensors::get_sensor (const std::string & *_name*)

Get a sensor using by name.

The given name should have: world_name::model_name::link_name::sensor_name

Parameters

in	*_name*	Name of the sensor. This name should be fully scoped. This means _name = world_name::model_name::link_name::sensor_name. You may use the unscoped sensor name if that name is unique within the entire simulation. If the name is not unique a NULL pointer is returned.

Returns

Pointer to the sensor, NULL if the sensor could not be found.

8.8.3.4 bool gazebo::sensors::init ( )

initialize the sensor generation loop.

Returns

True if successfully initialized, false if not

8.8.3.5 bool gazebo::sensors::load ( )

Load the sensor library.

Returns

True if successfully loaded, false if not.

8.8.3.6 void gazebo::sensors::remove_sensor (const std::string & *_sensorName*)

Remove a sensor by name.

Parameters

in	*_sensorName*	Name of sensor to remove

8.8.3.7 bool gazebo::sensors::remove_sensors ( )

Remove all sensors.

Returns

True if all successfully removed, false if not

8.8.3.8 void gazebo::sensors::run ( )

Deprecated.

See Also

run_threads (p. 75)

8.8.3.9 void gazebo::sensors::run_once (bool *_force* = `false`)

Run the sensor generation one step.

Parameters

_force,:	If true, all sensors are forced to update. Otherwise a sensor will update based on it's Hz rate.

8.8.3.10 void gazebo::sensors::run_threads ( )

Run sensors in a threads. This is a non-blocking call.

8.8.3.11 void gazebo::sensors::stop ( )

Stop the sensor generation loop.

8.9 Transport

Handles transportation of messages.

Files

- file **TransportTypes.hh**

  Forward declarations for transport.

Classes

- class **gazebo::transport::CallbackHelper**

  A helper class to handle callbacks when messages arrive.
- class **gazebo::transport::CallbackHelperT< M >**

  Callback helper Template.
- class **gazebo::transport::Connection**

  Single TCP/IP connection manager.
- class **gazebo::transport::ConnectionManager**

  Manager of connections.
- class **gazebo::transport::IOManager**

  Manages boost::asio IO.
- class **gazebo::transport::Node**

  A node can advertise and subscribe topics, publish on advertised topics and listen to subscribed topics.
- class **gazebo::transport::Publication**

  A publication for a topic.
- class **gazebo::transport::PublicationTransport**

  transport/transport.hh
- class **gazebo::transport::Publisher**

  A publisher of messages on a topic.
- class **gazebo::transport::RawCallbackHelper**

  Used to connect publishers to subscribers, where the subscriber wants the raw data from the publisher.
- class **gazebo::transport::SubscribeOptions**

  Options for a subscription.
- class **gazebo::transport::Subscriber**

  A subscriber to a topic.
- class **gazebo::transport::SubscriptionTransport**

  transport/transport.hh
- class **gazebo::transport::TopicManager**

  Manages topics and their subscriptions.

Typedefs

- typedef boost::shared_ptr
  < CallbackHelper > **gazebo::transport::CallbackHelperPtr**

  *boost shared pointer to **transport::CallbackHelper** (p. 161)*

Functions

- void **gazebo::transport::clear_buffers** ()

  Clear any remaining communication buffers.
- void **gazebo::transport::fini** ()

  Cleanup the transport component.
- bool **gazebo::transport::get_master_uri** (std::string &_master_host, unsigned int &_master_port)

  Get the hostname and port of the master from the GAZEBO_MASTER_URI environment variable.
- void **gazebo::transport::get_topic_namespaces** (std::list< std::string > &_namespaces)

  Return all the namespace (world names) on the master.
- std::map< std::string, std::list< std::string > > **gazebo::transport::getAdvertisedTopics** ()

  Get a list of all the topics and their message types.
- std::list< std::string > **gazebo::transport::getAdvertisedTopics** (const std::string &_msgType)

  Get a list of all the unique advertised topic names.
- bool **gazebo::transport::getMinimalComms** ()

  Get whether minimal comms has been enabled.
- std::string **gazebo::transport::getTopicMsgType** (const std::string &_topicName)

  Get the message typename that is published on the given topic.
- bool **gazebo::transport::init** (const std::string &_master_host="", unsigned int _master_port=0)

  Initialize the transport system.
- bool **gazebo::transport::is_stopped** ()

  Is the transport system stopped?
- void **gazebo::transport::pause_incoming** (bool _pause)

  Pause or unpause incoming messages.
- template<typename M > void **gazebo::transport::publish** (const std::string &_topic, const google::protobuf::Message &_message)

  A convenience function for a one-time publication of a message.
- boost::shared_ptr< msgs::Response > **gazebo::transport::request** (const std::string &_worldName, const std::string &_request, const std::string &_data="")

  Send a request and receive a response.
- void **gazebo::transport::requestNoReply** (const std::string &_worldName, const std::string &_request, const std::string &_data="")

  Send a request and don't wait for a response.
- void **gazebo::transport::requestNoReply** (NodePtr _node, const std::string &_request, const std::string &_data="")

  Send a request and don't wait for a response.
- void **gazebo::transport::run** ()

  Run the transport component.
- void **gazebo::transport::setMinimalComms** (bool _enabled)

  Set whether minimal comms should be used.
- void **gazebo::transport::stop** ()

  Stop the transport component from running.

8.9.1 Detailed Description

Handles transportation of messages.

Remarks

Environment Variables:

- GAZEBO_IP_WHITE_LIST: Comma separated list of valid IPs. Leave this empty to accept connections from all addresses.
- GAZEBO_IP: IP address to export. This will override the default IP lookup.
- GAZEBO_HOSTNAME: Hostame to export. Setting this will override both GAZEBO_IP and the default IP lookup.

8.9.2 Typedef Documentation

8.9.2.1 typedef boost::shared_ptr<CallbackHelper> gazebo::transport::CallbackHelperPtr

boost shared pointer to **transport::CallbackHelper** (p. 161)

8.9.3 Function Documentation

8.9.3.1 void gazebo::transport::clear_buffers ( )

Clear any remaining communication buffers.

8.9.3.2 void gazebo::transport::fini ( )

Cleanup the transport component.

8.9.3.3 bool gazebo::transport::get_master_uri (std::string & *_master_host,* unsigned int & *_master_port*)

Get the hostname and port of the master from the GAZEBO_MASTER_URI environment variable.

Parameters

out	*_master_host*	The hostname of the master is set to this param
out	*_master_port*	The port of the master is set to this param

Returns

true if GAZEBO_MASTER_URI was successfully parsed; false otherwise (in which case output params are not set)

8.9.3.4 void gazebo::transport::get_topic_namespaces (std::list< std::string > & *_namespaces*)

Return all the namespace (world names) on the master.

Parameters

out	*_namespaces*	The list of namespace will be written here

8.9.3.5 std::map<std::string, std::list<std::string> > gazebo::transport::getAdvertisedTopics ( )

Get a list of all the topics and their message types.

Returns

A map where keys are message types, and values are a list of topic names.

8.9.3.6 std::list<std::string> gazebo::transport::getAdvertisedTopics (const std::string & *_msgType*)

Get a list of all the unique advertised topic names.

Parameters

in	*_msgType*	Type of message to filter the result on. If empty, then a list of all the topics is returned.

Returns

A list of the advertised topics that publish messages of the type specified by _msgType.

8.9.3.7 bool gazebo::transport::getMinimalComms ( )

Get whether minimal comms has been enabled.

Returns

True if minimal comms is enabled.

8.9.3.8 std::string gazebo::transport::getTopicMsgType (const std::string & *_topicName*)

Get the message typename that is published on the given topic.

Parameters

in	*_topicName*	Name of the topic to query.

Returns

The message type, or empty string if the topic is not valid.

8.9.3.9 bool gazebo::transport::init (const std::string & *_master_host* = " ", unsigned int *_master_port* = 0)

Initialize the transport system.

Parameters

in	*_master_host*	The hostname or IP of the master. Leave empty to use pull address from the GAZEBO_MASTER_URI env var.
in	*_master_port*	The port of the master. Leave empty to use pull address from the GAZEBO_MASTER_URI env var.

Returns

true if initialization succeeded; false otherwise

8.9.3.10 bool gazebo::transport::is_stopped ( )

Is the transport system stopped?

Returns

true if the transport system is stopped; false otherwise

8.9.3.11 void gazebo::transport::pause_incoming (bool *_pause*)

Pause or unpause incoming messages.

When paused, messages are queued for later delivery

Parameters

in	*_pause*	If true, pause; otherwise unpause

8.9.3.12 template<typename M > void gazebo::transport::publish (const std::string & *_topic,* const google::protobuf::Message & *_message*)

A convenience function for a one-time publication of a message.

This is inefficient, compared to **Node::Advertise** (p. 566) followed by **Publisher::Publish** (p. 651). This function should only be used when sending a message very infrequently.

Parameters

in	*_topic*	The topic to advertise
in	*_message*	Message to be published

8.9.3.13 boost::shared_ptr<msgs::Response> gazebo::transport::request (const std::string & *_worldName,* const std::string & *_request,* const std::string & *_data* = " ")

Send a request and receive a response.

This call will block until a response is received.

Parameters

in	*_worldName*	The name of the world to which the request should be sent
in	*_request*	The type request.
in	*_data*	Optional data string.

Returns

The response to the request. Can be empty.

8.9.3.14 void gazebo::transport::requestNoReply (const std::string & _worldName, const std::string & _request, const std::string & _data = " ")

Send a request and don't wait for a response.

This is non-blocking.

Parameters

in	*_worldName*	The name of the world to which the request should be sent.
in	*_request*	The type request.
in	*_data*	Optional data string.

8.9.3.15 void gazebo::transport::requestNoReply (NodePtr _node, const std::string & _request, const std::string & _data = " ")

Send a request and don't wait for a response.

This is non-blocking.

Parameters

in	*_node*	Pointer to a node that provides communication.
in	*_request*	The type request.
in	*_data*	Optional data string.

8.9.3.16 void gazebo::transport::run ( )

Run the transport component.

Creates a thread to handle message passing. This call will block until the master can be contacted or until a retry limit is reached

8.9.3.17 void gazebo::transport::setMinimalComms (bool _enabled)

Set whether minimal comms should be used.

This will be used to reduce network traffic.

8.9.3.18 void gazebo::transport::stop ( )

Stop the transport component from running.

8.10 Utility

Files

- file **UtilTypes.hh**

Classes

- class **gazebo::util::DiagnosticManager**

  A diagnostic manager class.
- class **gazebo::util::DiagnosticTimer**

  A timer designed for diagnostics.

Macros

- #define **DIAG_TIMER_LAP**(_name, _prefix) ((void)0)
- #define **DIAG_TIMER_START**(_name) ((void) 0)
- #define **DIAG_TIMER_STOP**(_name) ((void) 0)

8.10.1 Detailed Description

8.10.2 Macro Definition Documentation

8.10.2.1 #define DIAG_TIMER_LAP(*_name, _prefix*) ((void)0)

8.10.2.2 #define DIAG_TIMER_START(*_name*) ((void) 0)

8.10.2.3 #define DIAG_TIMER_STOP(*_name*) ((void) 0)

Chapter 9

Namespace Documentation

9.1 boost Namespace Reference

9.2 gazebo Namespace Reference

Forward declarations for the common classes.

Namespaces

- namespace **common**

  Common namespace.
- namespace **event**

  Event *(p. 299) namespace.*
- namespace **math**

  Math namespace.
- namespace **msgs**

  Messages namespace.
- namespace **physics**

  namespace for physics
- namespace **rendering**

  Rendering namespace.
- namespace **sensors**

  Sensors namespace.
- namespace **transport**
- namespace **util**

Classes

- class **Master**

  A ROS Master-like manager that directs gztopic connections, enables each gazebo network client to locate one another for peer-to-peer communication.
- class **ModelPlugin**

  A plugin with access to ***physics::Model*** *(p. 516).*

- class **PluginT**

  A class which all plugins must inherit from.
- class **SensorPlugin**

  A plugin with access to physics::Sensor.
- class **Server**
- class **SystemPlugin**

  A plugin loaded within the gzserver on startup.
- class **VisualPlugin**

  A plugin loaded within the gzserver on startup.
- class **WorldPlugin**

  *A plugin with access to **physics::World** (p. 945).*

Typedefs

- typedef boost::shared_ptr < GUIPlugin > **GUIPluginPtr**
- typedef boost::shared_ptr < **ModelPlugin** > **ModelPluginPtr**
- typedef boost::shared_ptr < **SensorPlugin** > **SensorPluginPtr**
- typedef boost::shared_ptr < **SystemPlugin** > **SystemPluginPtr**
- typedef boost::shared_ptr < **VisualPlugin** > **VisualPluginPtr**
- typedef boost::shared_ptr < **WorldPlugin** > **WorldPluginPtr**

Enumerations

- enum **PluginType** { **WORLD_PLUGIN**, **MODEL_PLUGIN**, **SENSOR_PLUGIN**, **SYSTEM_PLUGIN**, **VISUAL_PLUGIN** }

  Used to specify the type of plugin.

Functions

- void **add_plugin** (const std::string &_filename)
- std::string **find_file** (const std::string &_file)

  Find a file in the gazebo search paths.
- void **fini** ()
- bool **init** ()
- bool **load** (int _argc=0, char **_argv=0)
- void **print_version** ()
- void **run** ()
- void **stop** ()

9.2.1 Detailed Description

Forward declarations for the common classes. Forward declarations for the util classes.

9.2.2 Typedef Documentation

9.2.2.1 typedef boost::shared_ptr<GUIPlugin> gazebo::GUIPluginPtr

9.2.2.2 typedef boost::shared_ptr<ModelPlugin> gazebo::ModelPluginPtr

9.2.2.3 typedef boost::shared_ptr<SensorPlugin> gazebo::SensorPluginPtr

9.2.2.4 typedef boost::shared_ptr<SystemPlugin> gazebo::SystemPluginPtr

9.2.2.5 typedef boost::shared_ptr<VisualPlugin> gazebo::VisualPluginPtr

9.2.2.6 typedef boost::shared_ptr<WorldPlugin> gazebo::WorldPluginPtr

9.2.3 Function Documentation

9.2.3.1 void gazebo::add_plugin (const std::string & *_filename*)

9.2.3.2 std::string gazebo::find_file (const std::string & *_file*)

Find a file in the gazebo search paths.

9.2.3.3 void gazebo::fini ( )

9.2.3.4 bool gazebo::init ( )

9.2.3.5 bool gazebo::load (int *_argc* = 0, char ** *_argv* = 0)

9.2.3.6 void gazebo::print_version ( )

9.2.3.7 void gazebo::run ( )

9.2.3.8 void gazebo::stop ( )

9.3 gazebo::common Namespace Reference

Common namespace.

Classes

- class **Animation**

  Manages an animation, which is a collection of keyframes and the ability to interpolate between the keyframes.
- class **AssertionInternalError**

  Class for generating Exceptions which come from gazebo assertions.
- class **BVHLoader**

  Handles loading BVH animation files.
- class **ColladaLoader**

  Class used to load Collada mesh files.
- class **Color**

Defines a color.

- class **Console**

  Message, error, warning functionality.
- class **Exception**

  Class for generating exceptions.
- class **Image**

  Encapsulates an image.
- class **InternalError**

  Class for generating Internal Gazebo Errors: those errors which should never happend and represent programming bugs.
- class **KeyEvent**

  Generic description of a keyboard event.
- class **KeyFrame**

  A key frame in an animation.
- class **Material**

  Encapsulates description of a material.
- class **Mesh**

  A 3D mesh.
- class **MeshCSG**

  Creates CSG meshes.
- class **MeshLoader**

  Base class for loading meshes.
- class **MeshManager**

  Maintains and manages all meshes.
- class **ModelDatabase**

  Connects to model database, and has utility functions to find models.
- class **MouseEvent**

  Generic description of a mouse event.
- class **NodeAnimation**

  Node animation.
- struct **NodeAssignment**

  Vertex to node weighted assignement for skeleton animation visualization.
- class **NodeTransform**

  NodeTransform *(*p. *576)* ***Skeleton.hh*** *(*p. *1132) common/common.hh*
- class **NumericAnimation**

  A numeric animation.
- class **NumericKeyFrame**

  A keyframe for a ***NumericAnimation*** *(*p. *581).*
- class **PID**

  Generic ***PID*** *(*p. *613) controller class.*
- class **PoseAnimation**

  A pose animation.
- class **PoseKeyFrame**

  A keyframe for a ***PoseAnimation*** *(*p. *635).*
- class **Skeleton**

  A skeleton.
- class **SkeletonAnimation**

  Skeleton *(*p. *760) animation.*

- class **SkeletonNode**

  A skeleton node.
- class **STLLoader**

  Class used to load STL mesh files.
- class **SubMesh**

  A child mesh.
- class **SystemPaths**

  Functions to handle getting system paths, keeps track of:
- class **Time**

  *A **Time*** (p. *831) class, can be used to hold wall- or sim-time.*
- class **Timer**

  A timer class, used to time things in real world walltime.
- class **UpdateInfo**

  Information for use in an update event.
- class **Video**

  Handle video encoding and decoding using libavcodec.

Typedefs

- typedef boost::shared_ptr < **Animation** > **AnimationPtr**
- typedef boost::shared_ptr < DiagnosticTimer > **DiagnosticTimerPtr**
- typedef std::map< unsigned int, **SkeletonNode** * > **NodeMap**
- typedef std::map< unsigned int, **SkeletonNode** * >::iterator **NodeMapIter**
- typedef boost::shared_ptr < **NumericAnimation** > **NumericAnimationPtr**
- typedef std::vector < common::Param * > **Param_V**
- typedef boost::shared_ptr < **PoseAnimation** > **PoseAnimationPtr**
- typedef std::map< double, std::vector< **NodeTransform** > > **RawNodeAnim**
- typedef std::vector < std::vector< std::pair < std::string, double > > > **RawNodeWeights**
- typedef std::map< std::string, **RawNodeAnim** > **RawSkeletonAnim**
- typedef std::map< std::string, std::string > **StrStr_M**

Functions

- void **add_search_path_suffix** (const std::string &_suffix)

  *add path prefix to **common::SystemPaths*** (p. *824)*
- std::string **find_file** (const std::string &_file)

  *search for file in **common::SystemPaths*** (p. *824)*

- std::string **find_file** (const std::string &_file, bool _searchLocalPath)
  *search for file in **common::SystemPaths*** (p. 824)
- std::string **find_file_path** (const std::string &_file)
  *search for a file in **common::SystemPaths*** (p. 824)

Variables

- static std::string **PixelFormatNames** [ ]
  String names for the pixel formats.

9.3.1 Detailed Description

Common namespace.

9.3.2 Typedef Documentation

9.3.2.1 typedef boost::shared_ptr<Animation> gazebo::common::AnimationPtr

9.3.2.2 typedef boost::shared_ptr<DiagnosticTimer> gazebo::common::DiagnosticTimerPtr

9.3.2.3 typedef std::map<unsigned int, SkeletonNode*> gazebo::common::NodeMap

9.3.2.4 typedef std::map<unsigned int, SkeletonNode*>::iterator gazebo::common::NodeMapIter

9.3.2.5 typedef boost::shared_ptr<NumericAnimation> gazebo::common::NumericAnimationPtr

9.3.2.6 typedef std::vector<common::Param*> gazebo::common::Param_V

9.3.2.7 typedef boost::shared_ptr<PoseAnimation> gazebo::common::PoseAnimationPtr

9.3.2.8 typedef std::map<double, std::vector<NodeTransform> > gazebo::common::RawNodeAnim

9.3.2.9 typedef std::vector<std::vector<std::pair<std::string, double> > > gazebo::common::RawNodeWeights

9.3.2.10 typedef std::map<std::string, RawNodeAnim> gazebo::common::RawSkeletonAnim

9.3.2.11 typedef std::map<std::string, std::string> gazebo::common::StrStr_M

9.4 gazebo::event Namespace Reference

Event (p. 299) namespace.

Classes

- class **Connection**
  A class that encapsulates a connection.
- class **Event**
  Base class for all events.

- class **Events**

  *An **Event** (p. 299) class to get notifications for simulator events.*
- class **EventT**

  A class for event processing.

Typedefs

- typedef std::vector
  < **ConnectionPtr** > **Connection_V**
- typedef boost::shared_ptr
  < **Connection** > **ConnectionPtr**

9.4.1 Detailed Description

Event (p. 299) namespace.

9.4.2 Typedef Documentation

9.4.2.1 typedef std::vector<ConnectionPtr> gazebo::event::Connection_V

9.4.2.2 typedef boost::shared_ptr<Connection> gazebo::event::ConnectionPtr

9.5 gazebo::math Namespace Reference

Math namespace.

Classes

- class **Angle**

  An angle and related functions.
- class **Box**

  Mathematical representation of a box and related functions.
- class **Matrix3**

  A 3x3 matrix class.
- class **Matrix4**

  A 3x3 matrix class.
- class **Plane**

  A plane and related functions.
- class **Pose**

  Encapsulates a position and rotation in three space.
- class **Quaternion**

  A quaternion class.
- class **Rand**

  Random number generator class.
- class **RotationSpline**

  ***Spline** (p. 793) for rotations.*

- class **Spline**

  Splines.
- class **Vector2d**

  Generic double x, y vector.
- class **Vector2i**

  Generic integer x, y vector.
- class **Vector3**

  *The **Vector3** (p. 890) class represents the generic vector containing 3 elements.*
- class **Vector4**

  double Generic x, y, z, w vector

Typedefs

- typedef boost::mt19937 **GeneratorType**
- typedef boost::normal_distribution < double > **NormalRealDist**
- typedef boost::variate_generator < **GeneratorType** &, **NormalRealDist** > **NRealGen**
- typedef boost::variate_generator < **GeneratorType** &, **UniformIntDist** > **UIntGen**
- typedef boost::uniform_int< int > **UniformIntDist**
- typedef boost::uniform_real < double > **UniformRealDist**
- typedef boost::variate_generator < **GeneratorType** &, **UniformRealDist** > **URealGen**

Functions

- template<typename T >
  T **clamp** (T _v, T _min, T _max)

  Simple clamping function.
- template<typename T >
  bool **equal** (const T &_a, const T &_b, const T &_epsilon=1e-6)

  check if two values are equal, within a tolerance
- bool **isnan** (float _v)

  check if a float is NaN
- bool **isnan** (double _v)

  check if a double is NaN
- bool **isPowerOfTwo** (unsigned int _x)

  is this a power of 2?
- template<typename T >
  T **max** (const std::vector< T > &_values)

get the maximum value of vector of values

- template<typename T >
  T **mean** (const std::vector< T > &_values)

  get mean of vector of values
- template<typename T >
  T **min** (const std::vector< T > &_values)

  get the minimum value of vector of values
- double **parseFloat** (const std::string &_input)

  parse string into float
- int **parseInt** (const std::string &_input)

  parse string into an integer
- template<typename T >
  T **precision** (const T &_a, const unsigned int &_precision)

  get value at a specified precision
- template<typename T >
  T **variance** (const std::vector< T > &_values)

  get variance of vector of values

Variables

- static const double **NAN_D** = std::numeric_limits<double>::quiet_NaN()

  Returns the representation of a quiet not a number (NAN)
- static const int **NAN_I** = std::numeric_limits<int>::quiet_NaN()

  Returns the representation of a quiet not a number (NAN)

9.5.1 Detailed Description

Math namespace.

9.5.2 Typedef Documentation

9.5.2.1 typedef boost::mt19937 gazebo::math::GeneratorType

9.5.2.2 typedef boost::normal_distribution<double> gazebo::math::NormalRealDist

9.5.2.3 typedef boost::variate_generator<GeneratorType&, NormalRealDist > gazebo::math::NRealGen

9.5.2.4 typedef boost::variate_generator<GeneratorType&, UniformIntDist > gazebo::math::UIntGen

9.5.2.5 typedef boost::uniform_int<int> gazebo::math::UniformIntDist

9.5.2.6 typedef boost::uniform_real<double> gazebo::math::UniformRealDist

9.5.2.7 typedef boost::variate_generator<GeneratorType&, UniformRealDist > gazebo::math::URealGen

9.6 gazebo::msgs Namespace Reference

Messages namespace.

Classes

- class **MsgFactory**

  A factory that generates protobuf message based on a string type.

Typedefs

- typedef boost::shared_ptr
  < google::protobuf::Message >(* **MsgFactoryFn**)()

Functions

- msgs::Vector3d **Convert** (const **math::Vector3** &_v)

  *Convert a **math::Vector3*** (p. *890) to a msgs::Vector3d.*
- msgs::Quaternion **Convert** (const **math::Quaternion** &_q)

  *Convert a **math::Quaternion*** (p. *654) to a msgs::Quaternion.*
- msgs::Pose **Convert** (const **math::Pose** &_p)

  *Convert a **math::Pose*** (p. *626) to a msgs::Pose.*
- msgs::Color **Convert** (const **common::Color** &_c)

  *Convert a **common::Color*** (p. *213) to a msgs::Color.*
- msgs::Time **Convert** (const **common::Time** &_t)

  *Convert a **common::Time*** (p. *831) to a msgs::Time.*
- msgs::PlaneGeom **Convert** (const **math::Plane** &_p)

  *Convert a **math::Plane*** (p. *617) to a msgs::PlaneGeom.*
- **math::Vector3 Convert** (const msgs::Vector3d &_v)

  Convert a msgs::Vector3d to a math::Vector.
- **math::Quaternion Convert** (const msgs::Quaternion &_q)

  *Convert a msgs::Quaternion to a **math::Quaternion*** (p. *654).*
- **math::Pose Convert** (const msgs::Pose &_p)

  *Convert a msgs::Pose to a **math::Pose*** (p. *626).*
- **common::Color Convert** (const msgs::Color &_c)

  *Convert a msgs::Color to a **common::Color*** (p. *213).*
- **common::Time Convert** (const msgs::Time &_t)

  *Convert a msgs::Time to a **common::Time*** (p. *831).*
- **math::Plane Convert** (const msgs::PlaneGeom &_p)

  Convert a msgs::PlaneGeom to a common::Plane.
- msgs::Request * **CreateRequest** (const std::string &_request, const std::string &_data="")

  Create a request message.
- msgs::Fog **FogFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::Fog from a fog SDF element.
- msgs::Geometry **GeometryFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::Geometry from a geometry SDF element.
- msgs::Header * **GetHeader** (google::protobuf::Message &_message)

  Get the header from a protobuf message.
- msgs::GUI **GUIFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::GUI from a GUI SDF element.
- void **Init** (google::protobuf::Message &_message, const std::string &_id="")

Initialize a message.

- msgs::Light **LightFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::Light from a light SDF element.
- msgs::MeshGeom **MeshFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::MeshGeom from a mesh SDF element.
- msgs::Scene **SceneFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::Scene from a scene SDF element.
- void **Set** (**common::Image** &_img, const msgs::Image &_msg)

  *Convert a msgs::Image to a **common::Image** (p. 379).*
- void **Set** (msgs::Image *_msg, const **common::Image** &_i)

  *Set a msgs::Image from a **common::Image** (p. 379).*
- void **Set** (msgs::Vector3d *_pt, const **math::Vector3** &_v)

  *Set a msgs::Vector3d from a **math::Vector3** (p. 890).*
- void **Set** (msgs::Vector2d *_pt, const **math::Vector2d** &_v)

  *Set a msgs::Vector2d from a **math::Vector3** (p. 890).*
- void **Set** (msgs::Quaternion *_q, const **math::Quaternion** &_v)

  *Set a msgs::Quaternion from a **math::Quaternion** (p. 654).*
- void **Set** (msgs::Pose *_p, const **math::Pose** &_v)

  *Set a msgs::Pose from a **math::Pose** (p. 626).*
- void **Set** (msgs::Color *_c, const **common::Color** &_v)

  *Set a msgs::Color from a **common::Color** (p. 213).*
- void **Set** (msgs::Time *_t, const **common::Time** &_v)

  *Set a msgs::Time from a **common::Time** (p. 831).*
- void **Set** (msgs::PlaneGeom *_p, const **math::Plane** &_v)

  *Set a msgs::Plane from a **math::Plane** (p. 617).*
- void **Stamp** (msgs::Header *_header)

  Time stamp a header.
- void **Stamp** (msgs::Time *_time)

  Set the time in a time message.
- msgs::TrackVisual **TrackVisualFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::TrackVisual from a track visual SDF element.
- msgs::Visual **VisualFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::Visual from a visual SDF element.

9.6.1 Detailed Description

Messages namespace.

9.6.2 Typedef Documentation

9.6.2.1 typedef boost::shared_ptr<google::protobuf::Message>(* gazebo::msgs::MsgFactoryFn)()

9.7 gazebo::physics Namespace Reference

namespace for physics

Classes

- class **Actor**

  Actor *(p. 115) class enables GPU based mesh model / skeleton scriptable animation.*
- class **BallJoint**

  Base *(p. 141) class for a ball joint.*
- class **Base**

  Base *(p. 141) class for most physics classes.*
- class **BoxShape**

  Box geometry primitive.
- class **Collision**

  Base *(p. 141) class for all collision entities.*
- class **CollisionState**

  Store state information of a ***physics::Collision*** *(p. 199) object.*
- class **Contact**

  A contact between two collisions.
- class **ContactManager**

  Aggregates all the contact information generated by the collision detection engine.
- class **ContactPublisher**

  A custom contact publisher created for each contact filter in the ***Contact*** *(p. 241) Manager.*
- class **CylinderShape**

  Cylinder collision.
- class **Entity**

  Base *(p. 141) class for all physics objects in Gazebo.*
- class **Gripper**

  A gripper abstraction.
- class **HeightmapShape**

  HeightmapShape *(p. 371) collision shape builds a heightmap from an image.*
- class **Hinge2Joint**

  A two axis hinge joint.
- class **HingeJoint**

  A single axis hinge joint.
- class **Inertial**

  A class for inertial information about a link.
- class **Joint**

  Base *(p. 141) class for all joints.*
- class **JointController**

  A class for manipulating ***physics::Joint*** *(p. 401).*
- class **JointState**

  keeps track of state of a ***physics::Joint*** *(p. 401)*
- class **JointWrench**

  Wrench information from a joint.
- class **Link**

  Link *(p. 439) class defines a rigid body entity, containing information on inertia, visual and collision properties of a rigid body.*
- class **LinkState**

  Store state information of a ***physics::Link*** *(p. 439) object.*
- class **MeshShape**

Triangle mesh collision shape.

- class **Model**

  A model is a collection of links, joints, and plugins.
- class **ModelState**

  Store state information of a ***physics::Model*** *(*p. *516) object.*
- class **MultiRayShape**

  Laser collision contains a set of ray-collisions, structured to simulate a laser range scanner.
- class **PhysicsEngine**

  Base *(*p. *141) class for a physics engine.*
- class **PhysicsFactory**

  The physics factory instantiates different physics engines.
- class **PlaneShape**

  Collision *(*p. *199) for an infinite plane.*
- class **RayShape**

  Base *(*p. *141) class for Ray collision geometry.*
- class **Road**

  for building a ***Road*** *(*p. *697) from SDF*
- class **ScrewJoint**

  A screw joint, which has both prismatic and rotational DOFs.
- class **Shape**

  Base *(*p. *141) class for all shapes.*
- class **SliderJoint**

  A slider joint.
- class **SphereShape**

  Sphere collision shape.
- class **State**

  State *(*p. *797) of an entity.*
- class **SurfaceParams**

  SurfaceParams *(*p. *820) defines various Surface contact parameters.*
- struct **TrajectoryInfo**
- class **UniversalJoint**

  A universal joint.
- class **World**

  The world provides access to all other object within a simulated environment.
- class **WorldState**

  Store state information of a ***physics::World*** *(*p. *945) object.*

Typedefs

- typedef std::vector< **ActorPtr** > **Actor_V**
- typedef boost::shared_ptr< **Actor** > **ActorPtr**
- typedef std::vector< **BasePtr** > **Base_V**
- typedef boost::shared_ptr< **Base** > **BasePtr**
- typedef boost::shared_ptr < **BoxShape** > **BoxShapePtr**
- typedef std::vector< **CollisionPtr** > **Collision_V**
- typedef boost::shared_ptr < **Collision** > **CollisionPtr**

- typedef boost::shared_ptr < **Contact** > **ContactPtr**
- typedef boost::shared_ptr < **CylinderShape** > **CylinderShapePtr**
- typedef boost::shared_ptr< **Entity** > **EntityPtr**
- typedef boost::shared_ptr < **HeightmapShape** > **HeightmapShapePtr**
- typedef boost::shared_ptr < **Inertial** > **InertialPtr**
- typedef std::vector< **JointPtr** > **Joint_V**
- typedef std::vector < **JointControllerPtr** > **JointController_V**
- typedef boost::shared_ptr < **JointController** > **JointControllerPtr**
- typedef boost::shared_ptr< **Joint** > **JointPtr**
- typedef std::map< std::string, **JointState** > **JointState_M**
- typedef std::vector< **LinkPtr** > **Link_V**
- typedef boost::shared_ptr< **Link** > **LinkPtr**
- typedef std::map< std::string, **LinkState** > **LinkState_M**
- typedef boost::shared_ptr < **MeshShape** > **MeshShapePtr**
- typedef std::vector< **ModelPtr** > **Model_V**
- typedef boost::shared_ptr< **Model** > **ModelPtr**
- typedef std::map< std::string, **ModelState** > **ModelState_M**
- typedef boost::shared_ptr < **MultiRayShape** > **MultiRayShapePtr**
- typedef boost::shared_ptr < **PhysicsEngine** > **PhysicsEnginePtr**
- typedef **PhysicsEnginePtr**(* **PhysicsFactoryFn**)(**WorldPtr** world)
- typedef boost::shared_ptr < **RayShape** > **RayShapePtr**
- typedef boost::shared_ptr< **Road** > **RoadPtr**
- typedef boost::shared_ptr< **Shape** > **ShapePtr**
- typedef boost::shared_ptr < **SphereShape** > **SphereShapePtr**
- typedef boost::shared_ptr < **SurfaceParams** > **SurfaceParamsPtr**
- typedef boost::shared_ptr< **World** > **WorldPtr**

Functions

- **WorldPtr create_world** (const std::string &_name="")

  Create a world given a name.
- bool **fini** ()

  *Finalize transport by calling **gazebo::transport::fini** (p. 78).*
- **WorldPtr get_world** (const std::string &_name="")

  Returns a pointer to a world by name.
- void **init_world** (**WorldPtr** _world)

Init world given a pointer to it.

- void **init_worlds** ()

  initialize multiple worlds stored in static variable gazebo::g_worlds

- bool **load** ()

  *Setup **gazebo::SystemPlugin** (p. 829)'s and call **gazebo::transport::init** (p. 79).*

- void **load_world** (**WorldPtr** _world, **sdf::ElementPtr** _sdf)

  *Load world from **sdf::Element** (p. 280) pointer.*

- void **load_worlds** (**sdf::ElementPtr** _sdf)

  *load multiple worlds from single **sdf::Element** (p. 280) pointer*

- void **pause_world** (**WorldPtr** _world, bool _pause)

  *Pause world by calling **World::SetPaused** (p. 955).*

- void **pause_worlds** (bool pause)

  pause multiple worlds stored in static variable gazebo::g_worlds

- void **remove_worlds** ()

  remove multiple worlds stored in static variable gazebo::g_worlds

- void **run_world** (**WorldPtr** _world, unsigned int _iterations=0)

  *Run world by calling **World::Run()** (p. 955) given a pointer to it.*

- void **run_worlds** (unsigned int _iterations=0)

  Run multiple worlds stored in static variable gazebo::g_worlds.

- void **stop_world** (**WorldPtr** _world)

  *Stop world by calling **World::Stop()** (p. 956) given a pointer to it.*

- void **stop_worlds** ()

  stop multiple worlds stored in static variable gazebo::g_worlds

- bool **worlds_running** ()

  Return true if any world is running.

Variables

- static std::string **EntityTypename** []

  String names for the different entity types.

9.7.1 Detailed Description

namespace for physics Physics forward declarations and type defines.

physics namespace

9.7.2 Typedef Documentation

9.7.2.1 typedef std::vector<**ActorPtr**> **gazebo::physics::Actor_V**

9.7.2.2 typedef boost::shared_ptr<**Actor**> **gazebo::physics::ActorPtr**

9.7.2.3 typedef std::vector<**BasePtr**> **gazebo::physics::Base_V**

9.7.2.4 typedef boost::shared_ptr<**Base**> **gazebo::physics::BasePtr**

9.7.2.5 typedef boost::shared_ptr<BoxShape> gazebo::physics::BoxShapePtr

9.7.2.6 typedef std::vector<CollisionPtr> gazebo::physics::Collision_V

9.7.2.7 typedef boost::shared_ptr<Collision> gazebo::physics::CollisionPtr

9.7.2.8 typedef boost::shared_ptr<Contact> gazebo::physics::ContactPtr

9.7.2.9 typedef boost::shared_ptr<CylinderShape> gazebo::physics::CylinderShapePtr

9.7.2.10 typedef boost::shared_ptr<Entity> gazebo::physics::EntityPtr

9.7.2.11 typedef boost::shared_ptr<HeightmapShape> gazebo::physics::HeightmapShapePtr

9.7.2.12 typedef boost::shared_ptr<Inertial> gazebo::physics::InertialPtr

9.7.2.13 typedef std::vector<JointPtr> gazebo::physics::Joint_V

9.7.2.14 typedef std::vector<JointControllerPtr> gazebo::physics::JointController_V

9.7.2.15 typedef boost::shared_ptr<JointController> gazebo::physics::JointControllerPtr

9.7.2.16 typedef boost::shared_ptr<Joint> gazebo::physics::JointPtr

9.7.2.17 typedef std::map<std::string, JointState> gazebo::physics::JointState_M

9.7.2.18 typedef std::vector<LinkPtr> gazebo::physics::Link_V

9.7.2.19 typedef boost::shared_ptr<Link> gazebo::physics::LinkPtr

9.7.2.20 typedef std::map<std::string, LinkState> gazebo::physics::LinkState_M

9.7.2.21 typedef boost::shared_ptr<MeshShape> gazebo::physics::MeshShapePtr

9.7.2.22 typedef std::vector<ModelPtr> gazebo::physics::Model_V

9.7.2.23 typedef boost::shared_ptr<Model> gazebo::physics::ModelPtr

9.7.2.24 typedef std::map<std::string, ModelState> gazebo::physics::ModelState_M

9.7.2.25 typedef boost::shared_ptr<MultiRayShape> gazebo::physics::MultiRayShapePtr

9.7.2.26 typedef boost::shared_ptr<PhysicsEngine> gazebo::physics::PhysicsEnginePtr

9.7.2.27 typedef boost::shared_ptr<RayShape> gazebo::physics::RayShapePtr

9.7.2.28 typedef boost::shared_ptr<Road> gazebo::physics::RoadPtr

9.7.2.29 typedef boost::shared_ptr<Shape> gazebo::physics::ShapePtr

9.7.2.30 typedef boost::shared_ptr<SphereShape> gazebo::physics::SphereShapePtr

9.7.2.31 typedef boost::shared_ptr<SurfaceParams> gazebo::physics::SurfaceParamsPtr

9.7.2.32 typedef boost::shared_ptr<World> gazebo::physics::WorldPtr

9.8 gazebo::rendering Namespace Reference

Rendering namespace.

Classes

- class **ArrowVisual**

  Basic arrow visualization.
- class **AxisVisual**

  Basic axis visualization.
- class **Camera**

  Basic camera sensor.
- class **CameraVisual**

  Basic camera visualization.
- class **COMVisual**

  Basic Center of Mass visualization.
- class **ContactVisual**

  Contact visualization.
- class **Conversions**

  Conversions *(*p. *254)* ***Conversions.hh*** *(*p. *1006)* ***rendering/Conversions.hh*** *(*p. *1006)*.
- class **DepthCamera**

  Depth camera used to render depth data into an image buffer.
- class **DynamicLines**

  Class for drawing lines that can change.
- class **DynamicRenderable**

  Abstract base class providing mechanisms for dynamically growing hardware buffers.
- class **Events**

  Base class for rendering events.
- class **FPSViewController**

  First Person Shooter style view controller.
- class **GpuLaser**

  GPU based laser distance sensor.
- class **Grid**

  Displays a grid of cells, drawn with lines.
- class **GUIOverlay**

  A class that creates a CEGUI overlay on a render window.
- class **GzTerrainMatGen**
- class **Heightmap**

  Rendering a terrain using heightmap information.
- class **JointVisual**

  Visualization for joints.
- class **LaserVisual**

  Visualization for laser data.

- class **Light**

  A light source.
- class **MovableText**

  Movable text.
- class **OrbitViewController**

  Orbit view controller.
- class **Projector**

  Projects a material onto surface, light a light projector.
- class **RenderEngine**

  Adaptor to Ogre3d.
- class **RFIDTagVisual**

  Visualization for RFID tags sensor.
- class **RFIDVisual**

  Visualization for RFID sensor.
- class **Road2d**
- class **RTShaderSystem**

  *Implements **Ogre** (p. 110)'s Run-Time Shader system.*
- class **Scene**

  Representation of an entire scene graph.
- class **SelectionObj**

  A graphical selection object.
- class **SonarVisual**

  Visualization for sonar data.
- class **UserCamera**

  A camera used for user visualization of a scene.
- class **VideoVisual**

  A visual element that displays a video as a texture.
- class **ViewController**

  Base class for view controllers.
- class **Visual**

  A renderable object.
- class **WindowManager**

  Class to mangage render windows.
- class **WireBox**

  Draws a wireframe box.
- class **WrenchVisual**

  Visualization for sonar data.

Typedefs

- typedef boost::shared_ptr < **ArrowVisual** > **ArrowVisualPtr**
- typedef boost::shared_ptr < **AxisVisual** > **AxisVisualPtr**
- typedef boost::shared_ptr< **Camera** > **CameraPtr**
- typedef boost::shared_ptr < **CameraVisual** > **CameraVisualPtr**

- typedef boost::shared_ptr < **COMVisual** > **COMVisualPtr**
- typedef boost::shared_ptr < **ContactVisual** > **ContactVisualPtr**
- typedef boost::shared_ptr < **DepthCamera** > **DepthCameraPtr**
- typedef boost::shared_ptr < **DynamicLines** > **DynamicLinesPtr**
- typedef boost::shared_ptr < **GpuLaser** > **GpuLaserPtr**
- typedef boost::shared_ptr < **JointVisual** > **JointVisualPtr**
- typedef boost::shared_ptr < **LaserVisual** > **LaserVisualPtr**
- typedef boost::shared_ptr< **Light** > **LightPtr**
- typedef boost::shared_ptr < **RFIDTagVisual** > **RFIDTagVisualPtr**
- typedef boost::shared_ptr < **RFIDVisual** > **RFIDVisualPtr**
- typedef boost::shared_ptr< **Scene** > **ScenePtr**
- typedef boost::shared_ptr < **SonarVisual** > **SonarVisualPtr**
- typedef boost::shared_ptr < **UserCamera** > **UserCameraPtr**
- typedef boost::shared_ptr< **Visual** > **VisualPtr**
- typedef boost::shared_ptr < **WindowManager** > **WindowManagerPtr**
- typedef boost::shared_ptr < **WrenchVisual** > **WrenchVisualPtr**

Enumerations

- enum **RenderOpType** { **RENDERING_POINT_LIST** = 0, **RENDERING_LINE_LIST** = 1, **RENDERING_LINE_STRIP** = 2, **RENDERING_TRIANGLE_LIST** = 3, **RENDERING_TRIANGLE_STRIP** = 4, **RENDERING_TRIANGLE_FAN** = 5, **RENDERING_MESH_RESOURCE** = 6 }

  Type of render operation for a drawable.

Functions

- **rendering::ScenePtr create_scene** (const std::string &_name, bool _enableVisualizations)

  *create **rendering::Scene*** (p. *707*) *by name.*
- bool **fini** ()

  teardown rendering engine.
- **rendering::ScenePtr get_scene** (const std::string &_name="")

  *get pointer to **rendering::Scene*** (p. *707*) *by name.*
- bool **init** ()

  init rendering engine.
- bool **load** ()

load rendering engine.

- void **remove_scene** (const std::string &_name)

  *remove a **rendering::Scene** (p. 707) by name*

9.8.1 Detailed Description

Rendering namespace.

9.8.2 Typedef Documentation

9.8.2.1 typedef boost::shared_ptr<ArrowVisual> gazebo::rendering::ArrowVisualPtr

9.8.2.2 typedef boost::shared_ptr<AxisVisual> gazebo::rendering::AxisVisualPtr

9.8.2.3 typedef boost::shared_ptr<Camera> gazebo::rendering::CameraPtr

9.8.2.4 typedef boost::shared_ptr<CameraVisual> gazebo::rendering::CameraVisualPtr

9.8.2.5 typedef boost::shared_ptr<COMVisual> gazebo::rendering::COMVisualPtr

9.8.2.6 typedef boost::shared_ptr<ContactVisual> gazebo::rendering::ContactVisualPtr

9.8.2.7 typedef boost::shared_ptr<DepthCamera> gazebo::rendering::DepthCameraPtr

9.8.2.8 typedef boost::shared_ptr<DynamicLines> gazebo::rendering::DynamicLinesPtr

9.8.2.9 typedef boost::shared_ptr<GpuLaser> gazebo::rendering::GpuLaserPtr

9.8.2.10 typedef boost::shared_ptr<JointVisual> gazebo::rendering::JointVisualPtr

9.8.2.11 typedef boost::shared_ptr<LaserVisual> gazebo::rendering::LaserVisualPtr

9.8.2.12 typedef boost::shared_ptr<Light> gazebo::rendering::LightPtr

9.8.2.13 typedef boost::shared_ptr<RFIDTagVisual> gazebo::rendering::RFIDTagVisualPtr

9.8.2.14 typedef boost::shared_ptr<RFIDVisual> gazebo::rendering::RFIDVisualPtr

9.8.2.15 typedef boost::shared_ptr<Scene> gazebo::rendering::ScenePtr

9.8.2.16 typedef boost::shared_ptr<SonarVisual> gazebo::rendering::SonarVisualPtr

9.8.2.17 typedef boost::shared_ptr<UserCamera> gazebo::rendering::UserCameraPtr

9.8.2.18 typedef boost::shared_ptr<Visual> gazebo::rendering::VisualPtr

9.8.2.19 typedef boost::shared_ptr<WindowManager> gazebo::rendering::WindowManagerPtr

9.8.2.20 typedef boost::shared_ptr<WrenchVisual> gazebo::rendering::WrenchVisualPtr

9.8.3 Enumeration Type Documentation

9.8.3.1 enum gazebo::rendering::RenderOpType

Type of render operation for a drawable.

Enumerator:

RENDERING_POINT_LIST A list of points, 1 vertex per point.

RENDERING_LINE_LIST A list of lines, 2 vertices per line.

RENDERING_LINE_STRIP A strip of connected lines, 1 vertex per line plus 1 start vertex.

RENDERING_TRIANGLE_LIST A list of triangles, 3 vertices per triangle.

RENDERING_TRIANGLE_STRIP A strip of triangles, 3 vertices for the first triangle, and 1 per triangle after that.

RENDERING_TRIANGLE_FAN A fan of triangles, 3 vertices for the first triangle, and 1 per triangle after that.

RENDERING_MESH_RESOURCE N/A.

9.9 gazebo::sensors Namespace Reference

Sensors namespace.

Classes

- class **CameraSensor**

  Basic camera sensor.
- class **ContactSensor**

  Contact sensor.
- class **DepthCameraSensor**
- class **ForceTorqueSensor**

  Sensor *(*p. *731) for measure force and torque on a joint.*
- class **GpuRaySensor**
- class **ImuSensor**

  An IMU sensor.
- class **MultiCameraSensor**

  Multiple camera sensor.
- class **RaySensor**

  Sensor *(*p. *731) with one or more rays.*
- class **RFIDSensor**

  Sensor *(*p. *731) class for RFID type of sensor.*
- class **RFIDTag**

  RFIDTag *(*p. *691) to interact with RFIDTagSensors.*
- class **Sensor**

  Base class for sensors.
- class **SensorFactory**
- class **SensorManager**

  Class to manage and update all sensors.
- class **SimTimeEvent**

- class **SimTimeEventHandler**

  Monitors simulation time, and notifies conditions when a specified time has been reached.

- class **SonarSensor**

  Sensor *(p. 731) with sonar cone.*

Typedefs

- typedef std::vector < **CameraSensorPtr** > **CameraSensor_V**
- typedef boost::shared_ptr < **CameraSensor** > **CameraSensorPtr**
- typedef std::vector < **ContactSensorPtr** > **ContactSensor_V**
- typedef boost::shared_ptr < **ContactSensor** > **ContactSensorPtr**
- typedef std::vector < **DepthCameraSensorPtr** > **DepthCameraSensor_V**
- typedef boost::shared_ptr < **DepthCameraSensor** > **DepthCameraSensorPtr**
- typedef boost::shared_ptr < **ForceTorqueSensor** > **ForceTorqueSensorPtr**
- typedef std::vector < **GpuRaySensorPtr** > **GpuRaySensor_V**
- typedef boost::shared_ptr < **GpuRaySensor** > **GpuRaySensorPtr**
- typedef std::vector< **ImuSensorPtr** > **ImuSensor_V**
- typedef boost::shared_ptr < **ImuSensor** > **ImuSensorPtr**
- typedef std::vector< **RaySensorPtr** > **RaySensor_V**
- typedef boost::shared_ptr < **RaySensor** > **RaySensorPtr**
- typedef std::vector< **RFIDSensor** > **RFIDSensor_V**
- typedef boost::shared_ptr < **RFIDSensor** > **RFIDSensorPtr**
- typedef std::vector< **RFIDTag** > **RFIDTag_V**
- typedef boost::shared_ptr < **RFIDTag** > **RFIDTagPtr**
- typedef std::vector< **SensorPtr** > **Sensor_V**
- typedef **Sensor** *(* **SensorFactoryFn**)()
- typedef boost::shared_ptr< **Sensor** > **SensorPtr**
- typedef boost::shared_ptr < **SonarSensor** > **SonarSensorPtr**

Enumerations

- enum **SensorCategory** { **IMAGE** = 0, **RAY** = 1, **OTHER** = 2, **CATEGORY_COUNT** = 3 }

  SensorClass is used to categorize sensors.

Functions

- std::string **create_sensor** (**sdf::ElementPtr** _elem, const std::string &_worldName, const std::string &_parentName)

  Create a sensor using SDF.
- bool **fini** ()

  shutdown the sensor generation loop.
- **SensorPtr get_sensor** (const std::string &_name)

  Get a sensor using by name.
- bool **init** ()

  initialize the sensor generation loop.
- bool **load** ()

  Load the sensor library.
- void **remove_sensor** (const std::string &_sensorName)

  Remove a sensor by name.
- bool **remove_sensors** ()

  Remove all sensors.
- void **run** () **GAZEBO_DEPRECATED**(1.5)

  Deprecated.
- void **run_once** (bool _force=false)

  Run the sensor generation one step.
- void **run_threads** ()

  Run sensors in a threads. This is a non-blocking call.
- void **stop** ()

  Stop the sensor generation loop.

9.9.1 Detailed Description

Sensors namespace.

9.9.2 Typedef Documentation

9.9.2.1 typedef std::vector<CameraSensorPtr> gazebo::sensors::CameraSensor_V

9.9.2.2 typedef boost::shared_ptr<CameraSensor> gazebo::sensors::CameraSensorPtr

9.9.2.3 typedef std::vector<ContactSensorPtr> gazebo::sensors::ContactSensor_V

9.9.2.4 typedef boost::shared_ptr<ContactSensor> gazebo::sensors::ContactSensorPtr

9.9.2.5 typedef std::vector<DepthCameraSensorPtr> gazebo::sensors::DepthCameraSensor_V

9.9.2.6 typedef boost::shared_ptr<DepthCameraSensor> gazebo::sensors::DepthCameraSensorPtr

9.9.2.7 typedef boost::shared_ptr<ForceTorqueSensor> gazebo::sensors::ForceTorqueSensorPtr

9.9.2.8 typedef std::vector<GpuRaySensorPtr> gazebo::sensors::GpuRaySensor_V

9.9.2.9 typedef boost::shared_ptr<GpuRaySensor> gazebo::sensors::GpuRaySensorPtr

9.9.2.10 typedef std::vector<ImuSensorPtr> gazebo::sensors::ImuSensor_V

9.9.2.11 typedef boost::shared_ptr<ImuSensor> gazebo::sensors::ImuSensorPtr

9.9.2.12 typedef std::vector<RaySensorPtr> gazebo::sensors::RaySensor_V

9.9.2.13 typedef boost::shared_ptr<RaySensor> gazebo::sensors::RaySensorPtr

9.9.2.14 typedef std::vector<RFIDSensor> gazebo::sensors::RFIDSensor_V

9.9.2.15 typedef boost::shared_ptr<RFIDSensor> gazebo::sensors::RFIDSensorPtr

9.9.2.16 typedef std::vector<RFIDTag> gazebo::sensors::RFIDTag_V

9.9.2.17 typedef boost::shared_ptr<RFIDTag> gazebo::sensors::RFIDTagPtr

9.9.2.18 typedef std::vector<SensorPtr> gazebo::sensors::Sensor_V

9.9.2.19 typedef Sensor*(* gazebo::sensors::SensorFactoryFn)()

9.9.2.20 typedef boost::shared_ptr<Sensor> gazebo::sensors::SensorPtr

9.9.2.21 typedef boost::shared_ptr<SonarSensor> gazebo::sensors::SonarSensorPtr

9.9.3 Enumeration Type Documentation

9.9.3.1 enum gazebo::sensors::SensorCategory

SensorClass is used to categorize sensors.

This is used to put sensors into different threads.

Enumerator:

IMAGE Image based sensor class. This type requires the rendering engine.

RAY Ray based sensor class.

OTHER A type of sensor is not a RAY or IMAGE sensor.

CATEGORY_COUNT Number of **Sensor** (p. 731) Categories.

9.10 gazebo::transport Namespace Reference

Classes

- class **CallbackHelper**

  A helper class to handle callbacks when messages arrive.
- class **CallbackHelperT**

  Callback helper Template.
- class **Connection**

  Single TCP/IP connection manager.

- class **ConnectionManager**

  Manager of connections.
- class **ConnectionReadTask**

- class **IOManager**

  Manages boost::asio IO.
- class **Node**

  A node can advertise and subscribe topics, publish on advertised topics and listen to subscribed topics.
- class **Publication**

  A publication for a topic.
- class **PublicationTransport**

  transport/transport.hh
- class **Publisher**

  A publisher of messages on a topic.
- class **PublishTask**

- class **RawCallbackHelper**

  Used to connect publishers to subscribers, where the subscriber wants the raw data from the publisher.
- class **SubscribeOptions**

  Options for a subscription.
- class **Subscriber**

  A subscriber to a topic.
- class **SubscriptionTransport**

  transport/transport.hh
- class **TopicManager**

  Manages topics and their subscriptions.

Typedefs

- typedef boost::shared_ptr
  < **CallbackHelper** > **CallbackHelperPtr**

  boost shared pointer to ***transport::CallbackHelper*** *(*p. *161)*
- typedef boost::shared_ptr
  < **Connection** > **ConnectionPtr**
- typedef boost::shared_ptr
  < google::protobuf::Message > **MessagePtr**
- typedef boost::shared_ptr< **Node** > **NodePtr**
- typedef boost::shared_ptr
  < **Publication** > **PublicationPtr**
- typedef boost::shared_ptr
  < **PublicationTransport** > **PublicationTransportPtr**
- typedef boost::shared_ptr
  < **Publisher** > **PublisherPtr**
- typedef boost::shared_ptr
  < **Subscriber** > **SubscriberPtr**
- typedef boost::shared_ptr
  < **SubscriptionTransport** > **SubscriptionTransportPtr**

Functions

- void **clear_buffers** ()

  Clear any remaining communication buffers.

- void **fini** ()

  Cleanup the transport component.

- bool **get_master_uri** (std::string &_master_host, unsigned int &_master_port)

  Get the hostname and port of the master from the GAZEBO_MASTER_URI environment variable.

- void **get_topic_namespaces** (std::list< std::string > &_namespaces)

  Return all the namespace (world names) on the master.

- std::map< std::string,
  std::list< std::string > > **getAdvertisedTopics** ()

  Get a list of all the topics and their message types.

- std::list< std::string > **getAdvertisedTopics** (const std::string &_msgType)

  Get a list of all the unique advertised topic names.

- bool **getMinimalComms** ()

  Get whether minimal comms has been enabled.

- std::string **getTopicMsgType** (const std::string &_topicName)

  Get the message typename that is published on the given topic.

- bool **init** (const std::string &_master_host="", unsigned int _master_port=0)

  Initialize the transport system.

- bool **is_stopped** ()

  Is the transport system stopped?

- void **pause_incoming** (bool _pause)

  Pause or unpause incoming messages.

- template<typename M >
  void **publish** (const std::string &_topic, const google::protobuf::Message &_message)

  A convenience function for a one-time publication of a message.

- boost::shared_ptr< msgs::Response > **request** (const std::string &_worldName, const std::string &_request, const std::string &_data="")

  Send a request and receive a response.

- void **requestNoReply** (const std::string &_worldName, const std::string &_request, const std::string &_data="")

  Send a request and don't wait for a response.

- void **requestNoReply** (**NodePtr** _node, const std::string &_request, const std::string &_data="")

  Send a request and don't wait for a response.

- void **run** ()

  Run the transport component.

- void **setMinimalComms** (bool _enabled)

  Set whether minimal comms should be used.

- void **stop** ()

  Stop the transport component from running.

9.10.1 Typedef Documentation

9.10.1.1 typedef boost::shared_ptr<Connection> gazebo::transport::ConnectionPtr

9.10.1.2 typedef boost::shared_ptr<google::protobuf::Message> gazebo::transport::MessagePtr

9.10.1.3 typedef boost::shared_ptr<Node> gazebo::transport::NodePtr

9.10.1.4 typedef boost::shared_ptr<Publication> gazebo::transport::PublicationPtr

9.10.1.5 typedef boost::shared_ptr<PublicationTransport> gazebo::transport::PublicationTransportPtr

9.10.1.6 typedef boost::shared_ptr<Publisher> gazebo::transport::PublisherPtr

9.10.1.7 typedef boost::shared_ptr<Subscriber> gazebo::transport::SubscriberPtr

9.10.1.8 typedef boost::shared_ptr<SubscriptionTransport> gazebo::transport::SubscriptionTransportPtr

9.11 gazebo::util Namespace Reference

Classes

- class **DiagnosticManager**

  A diagnostic manager class.
- class **DiagnosticTimer**

  A timer designed for diagnostics.
- class **LogPlay**
- class **LogRecord**

  addtogroup gazebo_util

Typedefs

- typedef boost::shared_ptr
  < **DiagnosticTimer** > **DiagnosticTimerPtr**

9.11.1 Typedef Documentation

9.11.1.1 typedef boost::shared_ptr<DiagnosticTimer> gazebo::util::DiagnosticTimerPtr

9.12 google Namespace Reference

Namespaces

- namespace **protobuf**

9.13 google::protobuf Namespace Reference

Namespaces

- namespace **compiler**

9.14 google::protobuf::compiler Namespace Reference

Namespaces

- namespace **cpp**

9.15 google::protobuf::compiler::cpp Namespace Reference

Classes

- class **GazeboGenerator**

  *Google protobuf message generator for **gazebo::msgs*** (p. *91*).

9.16 Ogre Namespace Reference

9.17 ogre Namespace Reference

9.18 sdf Namespace Reference

namespace for Simulation Description Format parser

Classes

- class **Converter**

  *Convert from one version of **SDF*** (p. *728*) *to another.*
- class **Element**

  SDF (p. *728*) ***Element*** (p. *280*) *class.*
- class **Param**

  A parameter class.
- class **ParamT**

  Templatized parameter class.
- class **Plugin**
- class **SDF**

  *Base **SDF*** (p. *728*) *class.*

Typedefs

- typedef boost::shared_ptr < **Element** > **ElementPtr**
- typedef std::vector< **ElementPtr** > **ElementPtr_V**
- typedef std::vector< **ParamPtr** > **Param_V**
- typedef boost::shared_ptr< **Param** > **ParamPtr**
- typedef boost::shared_ptr< **SDF** > **SDFPtr**

Functions

- void **addNestedModel** (**ElementPtr** _sdf, **ElementPtr** _includeSDF)
- void **addURIPath** (const std::string &_uri, const std::string &_path)

  *A function that is used in the external **SDF** (p. 728) library.*
- void **copyChildren** (**ElementPtr** _sdf, TiXmlElement *_xml)
- bool **init** (**SDFPtr** _sdf)

  Init based on the installed sdf_format.xml file.
- bool **initDoc** (TiXmlDocument *_xmlDoc, **SDFPtr** _sdf)
- bool **initDoc** (TiXmlDocument *_xmlDoc, **ElementPtr** _sdf)
- bool **initFile** (const std::string &_filename, **SDFPtr** _sdf)
- bool **initFile** (const std::string &_filename, **ElementPtr** _sdf)
- bool **initString** (const std::string &_xmlString, **SDFPtr** _sdf)
- bool **initXml** (TiXmlElement *_xml, **ElementPtr** _sdf)
- bool **readDoc** (TiXmlDocument *_xmlDoc, **SDFPtr** _sdf, const std::string &_source)

  *Populate the **SDF** (p. 728) values from a TinyXML document.*
- bool **readDoc** (TiXmlDocument *_xmlDoc, **ElementPtr** _sdf, const std::string &_source)
- bool **readFile** (const std::string &_filename, **SDFPtr** _sdf)

  *Populate the **SDF** (p. 728) values from a file.*
- bool **readString** (const std::string &_xmlString, **SDFPtr** _sdf)

  *Populate the **SDF** (p. 728) values from a string.*
- bool **readString** (const std::string &_xmlString, **ElementPtr** _sdf)
- bool **readXml** (TiXmlElement *_xml, **ElementPtr** _sdf)
- void **setFindCallback** (boost::function< std::string(const std::string &)> _cb)

  *A function that is used in the external **SDF** (p. 728) library.*

9.18.1 Detailed Description

namespace for Simulation Description Format parser

9.18.2 Typedef Documentation

9.18.2.1 typedef boost::shared_ptr<Element> sdf::ElementPtr

9.18.2.2 typedef std::vector< ElementPtr > sdf::ElementPtr_V

9.18.2.3 typedef std::vector< ParamPtr > sdf::Param_V

9.18.2.4 typedef boost::shared_ptr< Param > sdf::ParamPtr

9.18.2.5 typedef boost::shared_ptr<SDF> sdf::SDFPtr

9.18.3 Function Documentation

9.18.3.1 void sdf::addNestedModel (ElementPtr *_sdf,* ElementPtr *_includeSDF*)

9.18.3.2 void sdf::addURIPath (const std::string & *_uri,* const std::string & *_path*)

A function that is used in the external **SDF** (p. 728) library.

This is here to make the build work if the external **SDF** (p. 728) library is not present.

9.18.3.3 void sdf::copyChildren (ElementPtr *_sdf,* TiXmlElement * *_xml*)

9.18.3.4 bool sdf::init (SDFPtr *_sdf*)

Init based on the installed sdf_format.xml file.

9.18.3.5 bool sdf::initDoc (TiXmlDocument * *_xmlDoc,* SDFPtr *_sdf*)

9.18.3.6 bool sdf::initDoc (TiXmlDocument * *_xmlDoc,* ElementPtr *_sdf*)

9.18.3.7 bool sdf::initFile (const std::string & *_filename,* SDFPtr *_sdf*)

9.18.3.8 bool sdf::initFile (const std::string & *_filename,* ElementPtr *_sdf*)

9.18.3.9 bool sdf::initString (const std::string & *_xmlString,* SDFPtr *_sdf*)

9.18.3.10 bool sdf::initXml (TiXmlElement * *_xml,* ElementPtr *_sdf*)

9.18.3.11 bool sdf::readDoc (TiXmlDocument * *_xmlDoc,* SDFPtr *_sdf,* const std::string & *_source*)

Populate the **SDF** (p. 728) values from a TinyXML document.

9.18.3.12 bool sdf::readDoc (TiXmlDocument * *_xmlDoc,* ElementPtr *_sdf,* const std::string & *_source*)

9.18.3.13 bool sdf::readFile (const std::string & *_filename,* SDFPtr *_sdf*)

Populate the **SDF** (p. 728) values from a file.

9.18.3.14 bool sdf::readString (const std::string & *_xmlString,* SDFPtr *_sdf*)

Populate the **SDF** (p. 728) values from a string.

9.18.3.15 bool sdf::readString (const std::string & *_xmlString,* ElementPtr *_sdf*)

9.18.3.16 bool sdf::readXml (TiXmlElement * *_xml,* ElementPtr *_sdf*)

9.18.3.17 void sdf::setFindCallback (boost::function< std::string(const std::string &)> _cb)

A function that is used in the external **SDF** (p. 728) library.

This is here to make the build work if the external **SDF** (p. 728) library is not present.

9.19 SkyX Namespace Reference

Chapter 10

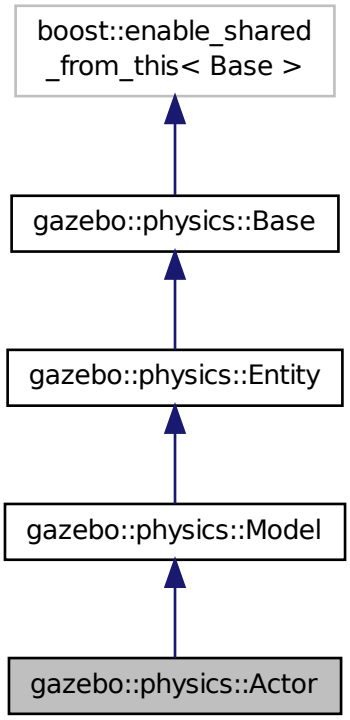
Class Documentation

10.1 gazebo::physics::Actor Class Reference

Actor (p. 115) class enables GPU based mesh model / skeleton scriptable animation.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::Actor:



Public Member Functions

- **Actor** (**BasePtr** _parent)

  Constructor.
- virtual ∼**Actor** ()

  Destructor.
- virtual void **Fini** ()

  Finalize the actor.
- virtual const **sdf::ElementPtr GetSDF** ()

  Get the SDF values for the actor.
- virtual void **Init** ()

  Initialize the actor.
- virtual bool **IsActive** ()

  Returns true when actor is playing animation.
- void **Load** (**sdf::ElementPtr** _sdf)

  Load the actor.
- virtual void **Play** ()

  Start playing the script.
- virtual void **Stop** ()

  Stop playing the script.
- void **Update** ()

  Update the actor.
- virtual void **UpdateParameters** (**sdf::ElementPtr** _sdf)

  update the parameters using new sdf values.

Protected Attributes

- bool **active**

  True if the actor is being updated.
- bool **autoStart**

  True if the actor should start running automatically.
- **transport::PublisherPtr bonePosePub**

  Where to send bone info.
- std::map< std::string, bool > **interpolateX**

  True to interpolate along x direction.
- **math::Vector3 lastPos**

  Last position of the actor.
- double **lastScriptTime**

  Time the scipt was last updated.
- unsigned int **lastTraj**

  THe last trajectory.
- bool **loop**

  True if the animation should loop.
- **LinkPtr mainLink**

  Base *(*p. *141) link.*
- const **common::Mesh** ∗ **mesh**

  Pointer to the actor's mesh.

- std::string **oldAction**

  THe old action.
- double **pathLength**

  Length of the actor's path.
- **common::Time playStartTime**

  Time when the animation was started.
- **common::Time prevFrameTime**

  Time of the previous frame.
- double **scriptLength**

  Time length of a scipt.
- std::map< std::string, **common::SkeletonAnimation** * > **skelAnimation**

  Skeleton animations.
- **common::Skeleton** * **skeleton**

  The actor's skeleton.
- std::map< std::string, std::map< std::string, std::string > > **skelNodesMap**

  Skeleton to naode map.
- std::string **skinFile**

  Filename for the skin.
- double **skinScale**

  Scaling factor to apply to the skin.
- double **startDelay**

  Amount of time to delay start by.
- std::map< unsigned int, **common::PoseAnimation** * > **trajectories**

  All the trajectories.
- std::vector< **TrajectoryInfo** > **trajInfo**

  Trajectory information.
- std::string **visualName**

  Name of the visual.

Additional Inherited Members

10.1.1 Detailed Description

Actor (p. 115) class enables GPU based mesh model / skeleton scriptable animation.

10.1.2 Constructor & Destructor Documentation

10.1.2.1 gazebo::physics::Actor::Actor (**BasePtr** *_parent*) [explicit]

Constructor.

Parameters

in	*_parent*	Parent object

10.1.2.2 virtual gazebo::physics::Actor::~Actor ( ) [virtual]

Destructor.

10.1.3 Member Function Documentation

10.1.3.1 virtual void gazebo::physics::Actor::Fini ( ) [virtual]

Finalize the actor.

Reimplemented from **gazebo::physics::Model** (p. 521).

10.1.3.2 virtual const sdf::ElementPtr gazebo::physics::Actor::GetSDF ( ) [virtual]

Get the SDF values for the actor.

Returns

Pointer to the SDF values.

Reimplemented from **gazebo::physics::Model** (p. 524).

10.1.3.3 virtual void gazebo::physics::Actor::Init ( ) [virtual]

Initialize the actor.

Reimplemented from **gazebo::physics::Model** (p. 525).

10.1.3.4 virtual bool gazebo::physics::Actor::IsActive ( ) [virtual]

Returns true when actor is playing animation.

10.1.3.5 void gazebo::physics::Actor::Load (sdf::ElementPtr *_sdf*) [virtual]

Load the actor.

Parameters

in	*_sdf*	SDF parameters

Reimplemented from **gazebo::physics::Entity** (p. 295).

10.1.3.6 virtual void gazebo::physics::Actor::Play ( ) [virtual]

Start playing the script.

10.1.3.7 virtual void gazebo::physics::Actor::Stop ( ) [virtual]

Stop playing the script.

10.1.3.8 void gazebo::physics::Actor::Update ( ) [virtual]

Update the actor.

Reimplemented from **gazebo::physics::Base** (p. 152).

10.1.3.9 virtual void gazebo::physics::Actor::UpdateParameters (sdf::ElementPtr _sdf) [virtual]

update the parameters using new sdf values.

Parameters

in	*_sdf*	SDF values to update from.

Reimplemented from **gazebo::physics::Model** (p. 529).

10.1.4 Member Data Documentation

10.1.4.1 bool gazebo::physics::Actor::active [protected]

True if the actor is being updated.

10.1.4.2 bool gazebo::physics::Actor::autoStart [protected]

True if the actor should start running automatically.

10.1.4.3 transport::PublisherPtr gazebo::physics::Actor::bonePosePub [protected]

Where to send bone info.

10.1.4.4 std::map<std::string, bool> gazebo::physics::Actor::interpolateX [protected]

True to interpolate along x direction.

10.1.4.5 math::Vector3 gazebo::physics::Actor::lastPos [protected]

Last position of the actor.

10.1.4.6 double gazebo::physics::Actor::lastScriptTime [protected]

Time the scipt was last updated.

10.1.4.7 unsigned int gazebo::physics::Actor::lastTraj [protected]

THe last trajectory.

10.1.4.8 bool gazebo::physics::Actor::loop [protected]

True if the animation should loop.

10.1.4.9 LinkPtr gazebo::physics::Actor::mainLink [protected]

Base (p. 141) link.

10.1.4.10 const common::Mesh* gazebo::physics::Actor::mesh [protected]

Pointer to the actor's mesh.

10.1.4.11 std::string gazebo::physics::Actor::oldAction [protected]

THe old action.

10.1.4.12 double gazebo::physics::Actor::pathLength [protected]

Length of the actor's path.

10.1.4.13 common::Time gazebo::physics::Actor::playStartTime [protected]

Time when the animation was started.

10.1.4.14 common::Time gazebo::physics::Actor::prevFrameTime [protected]

Time of the previous frame.

10.1.4.15 double gazebo::physics::Actor::scriptLength [protected]

Time length of a scipt.

10.1.4.16 std::map<std::string, common::SkeletonAnimation*> gazebo::physics::Actor::skelAnimation [protected]

Skeleton animations.

10.1.4.17 common::Skeleton* gazebo::physics::Actor::skeleton [protected]

The actor's skeleton.

10.1.4.18 std::map<std::string, std::map<std::string, std::string> > gazebo::physics::Actor::skelNodesMap [protected]

Skeleton to naode map.

10.1.4.19 std::string gazebo::physics::Actor::skinFile [protected]

Filename for the skin.

10.1.4.20 double gazebo::physics::Actor::skinScale [protected]

Scaling factor to apply to the skin.

10.1.4.21 double gazebo::physics::Actor::startDelay [protected]

Amount of time to delay start by.

10.1.4.22 std::map<unsigned int, common::PoseAnimation*> gazebo::physics::Actor::trajectories [protected]

All the trajectories.

10.1.4.23 std::vector<TrajectoryInfo> gazebo::physics::Actor::trajInfo [protected]

Trajectory information.

10.1.4.24 std::string gazebo::physics::Actor::visualName [protected]

Name of the visual.

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

- **Actor.hh**

10.2 gazebo::math::Angle Class Reference

An angle and related functions.

`#include <math/gzmath.hh>`

Public Member Functions

- **Angle** ()
  Constructor.
- **Angle** (double _radian)
  Copy Constructor.
- **Angle** (const **Angle** &_angle)
  Copy constructor.
- virtual ~**Angle** ()
  Destructor.
- double **Degree** () const
  Get the angle in degrees.
- void **Normalize** ()

Normalize the angle in the range -Pi to Pi.

- bool **operator!=** (const **Angle** &_angle) const

  Inequality.
- double **operator*** () const

  Dereference operator.
- **Angle operator*** (const **Angle** &_angle) const

  *Multiplication operator, result = this * _angle.*
- **Angle operator*=** (const **Angle** &_angle)

  *Multiplication set, this = this * _angle.*
- **Angle operator+** (const **Angle** &_angle) const

  Addition operator, result = this + _angle.
- **Angle operator+=** (const **Angle** &_angle)

  Addition set, this = this + _angle.
- **Angle operator-** (const **Angle** &_angle) const

  Substraction, result = this - _angle.
- **Angle operator-=** (const **Angle** &_angle)

  Subtraction set, this = this - _angle.
- **Angle operator/** (const **Angle** &_angle) const

  Division, result = this / _angle.
- **Angle operator/=** (const **Angle** &_angle)

  Division set, this = this / _angle.
- bool **operator<** (const **Angle** &_angle) const

  Less than operator.
- bool **operator<=** (const **Angle** &_angle) const

  Less or equal operator.
- bool **operator==** (const **Angle** &_angle) const

  Equality operator, result = this == _angle.
- bool **operator>** (const **Angle** &_angle) const

  Greater than operator.
- bool **operator>=** (const **Angle** &_angle) const

  Greater or equal operator.
- double **Radian** () const

  Get the angle in radians.
- void **SetFromDegree** (double _degree)

  Set the value from an angle in degrees.
- void **SetFromRadian** (double _radian)

  Set the value from an angle in radians.

Static Public Attributes

- static const **Angle HalfPi**

  math::Angle (p. *121)(M_PI * 0.5)*
- static const **Angle Pi**

  math::Angle(M_PI)
- static const **Angle TwoPi**

  *math::Angle(M_PI * 2)*
- static const **Angle Zero**

  math::Angle(0)

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::math::Angle** &_a)

  Stream insertion operator.
- std::istream & **operator**>> (std::istream &_in, **gazebo::math::Angle** &_a)

  Stream extraction operator.

10.2.1 Detailed Description

An angle and related functions.

10.2.2 Constructor & Destructor Documentation

10.2.2.1 gazebo::math::Angle::Angle ( )

Constructor.

10.2.2.2 gazebo::math::Angle::Angle (double *_radian*)

Copy Constructor.

Parameters

in	*_radian*	Radians

10.2.2.3 gazebo::math::Angle::Angle (const Angle & *_angle*)

Copy constructor.

Parameters

in	*_angle*	**Angle** (p. 121) to copy

10.2.2.4 virtual gazebo::math::Angle::~Angle ( ) [virtual]

Destructor.

10.2.3 Member Function Documentation

10.2.3.1 double gazebo::math::Angle::Degree ( ) const

Get the angle in degrees.

Returns

double containing the angle's degree value

10.2.3.2 void gazebo::math::Angle::Normalize ( )

Normalize the angle in the range -Pi to Pi.

10.2.3.3 bool gazebo::math::Angle::operator!= (const Angle & *_angle*) const

Inequality.

Parameters

in	*_angle*	**Angle** (p. 121) to check for inequality

Returns

true if this != _angle

10.2.3.4 double gazebo::math::Angle::operator* ( ) const [inline]

Dereference operator.

Returns

Double containing the angle's radian value

10.2.3.5 Angle gazebo::math::Angle::operator* (const Angle & *_angle*) const

Multiplication operator, result = this * _angle.

Parameters

in	*_angle*	**Angle** (p. 121) for multiplication

Returns

the new angle

10.2.3.6 Angle gazebo::math::Angle::operator*= (const Angle & *_angle*)

Multiplication set, this = this * _angle.

Parameters

in	*_angle*	**Angle** (p. 121) for multiplication

Returns

angle

10.2.3.7 Angle gazebo::math::Angle::operator+ (const Angle & _angle) const

Addition operator, result = this + _angle.

Parameters

in	*_angle*	**Angle** (p. 121) for addition

Returns

the new angle

10.2.3.8 Angle gazebo::math::Angle::operator+= (const Angle & _angle)

Addition set, this = this + _angle.

Parameters

in	*_angle*	**Angle** (p. 121) for addition

Returns

angle

10.2.3.9 Angle gazebo::math::Angle::operator- (const Angle & _angle) const

Substraction, result = this - _angle.

Parameters

in	*_angle*	**Angle** (p. 121) for substraction

Returns

the new angle

10.2.3.10 Angle gazebo::math::Angle::operator-= (const Angle & _angle)

Subtraction set, this = this - _angle.

Parameters

in	*_angle*	**Angle** (p. 121) for subtraction

Returns

angle

10.2.3.11 Angle gazebo::math::Angle::operator/ (const Angle & _angle) const

Division, result = this / _angle.

Parameters

in	*_angle*	**Angle** (p. 121) for division

Returns

the new angle

10.2.3.12 Angle gazebo::math::Angle::operator/= (const Angle & _angle)

Division set, this = this / _angle.

Parameters

in	*_angle*	**Angle** (p. 121) for division

Returns

angle

10.2.3.13 bool gazebo::math::Angle::operator< (const Angle & _angle) const

Less than operator.

Parameters

in	*_angle*	**Angle** (p. 121) to check

Returns

true if this < _angle

10.2.3.14 bool gazebo::math::Angle::operator<= (const Angle & _angle) const

Less or equal operator.

Parameters

in	*_angle*	**Angle** (p. 121) to check

Returns

true if this <= _angle

10.2.3.15 bool gazebo::math::Angle::operator== (const Angle & _angle) const

Equality operator, result = this == _angle.

Parameters

in	*_angle*	**Angle** (p. 121) to check for equality

Returns

true if this == _angle

10.2.3.16 bool gazebo::math::Angle::operator> (const Angle & _angle) const

Greater than operator.

Parameters

in	*_angle*	**Angle** (p. 121) to check

Returns

true if this > _angle

10.2.3.17 bool gazebo::math::Angle::operator>= (const Angle & _angle) const

Greater or equal operator.

Parameters

in	*_angle*	**Angle** (p. 121) to check

Returns

true if this >= _angle

10.2.3.18 double gazebo::math::Angle::Radian ( ) const

Get the angle in radians.

Returns

double containing the angle's radian value

10.2.3.19 void gazebo::math::Angle::SetFromDegree (double _degree)

Set the value from an angle in degrees.

Parameters

in	*_degree*	Degree value

10.2.3.20 void gazebo::math::Angle::SetFromRadian (double *_radian*)

Set the value from an angle in radians.

Parameters

in	*_radian*	Radian value

10.2.4 Friends And Related Function Documentation

10.2.4.1 std::ostream& operator<< (std::ostream & *_out,* const gazebo::math::Angle & *_a*) [friend]

Stream insertion operator.

Outputs in degrees

Parameters

in	*_out*	output stream
in	*_a*	angle to output

Returns

The output stream

10.2.4.2 std::istream& operator>> (std::istream & *_in,* gazebo::math::Angle & *_a*) [friend]

Stream extraction operator.

Assumes input is in degrees

Parameters

in	input stream
pt	angle to read value into

Returns

The input stream

10.2.5 Member Data Documentation

10.2.5.1 const Angle gazebo::math::Angle::HalfPi [static]

math::Angle (p. 121)(M_PI ∗ 0.5)

10.2.5.2 const Angle gazebo::math::Angle::Pi [static]

math::Angle(M_PI)

10.2.5.3 const Angle gazebo::math::Angle::TwoPi [static]

math::Angle(M_PI ∗ 2)

10.2.5.4 const Angle gazebo::math::Angle::Zero [static]

math::Angle(0)

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

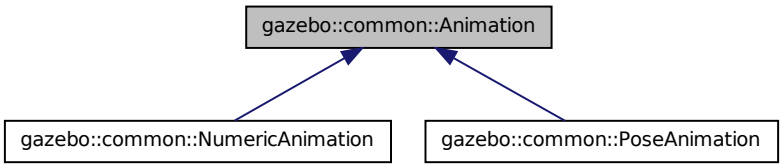
- **Angle.hh**

10.3 gazebo::common::Animation Class Reference

Manages an animation, which is a collection of keyframes and the ability to interpolate between the keyframes.

`#include <common/common.hh>`

Inheritance diagram for gazebo::common::Animation:



Public Member Functions

- **Animation** (const std::string &_name, double _length, bool _loop)

  Constructor.
- virtual ∼**Animation** ()

  Destructor.
- void **AddTime** (double _time)

  Add time to the animation.
- **KeyFrame** ∗ **GetKeyFrame** (unsigned int _index) const

  Get a key frame using an index value.
- unsigned int **GetKeyFrameCount** () const

  Return the number of key frames in the animation.
- double **GetLength** () const

  Return the duration of the animation.

- double **GetTime** () const

  Return the current time position.
- void **SetLength** (double _len)

  Set the duration of the animation.
- void **SetTime** (double _time)

  Set the current time position of the animation.

Protected Types

- typedef std::vector< **KeyFrame** * > **KeyFrame_V**

  array of keyframe type alias

Protected Member Functions

- double **GetKeyFramesAtTime** (double _time, **KeyFrame** **_kf1, **KeyFrame** **_kf2, unsigned int &_firstKeyIndex) const

  Get the two key frames that bound a time value.

Protected Attributes

- bool **build**

  determines if the interpolation splines need building
- **KeyFrame_V keyFrames**

  array of key frames
- double **length**

  animation duration
- bool **loop**

  true if animation repeats
- std::string **name**

  animation name
- double **timePos**

  current time position

10.3.1 Detailed Description

Manages an animation, which is a collection of keyframes and the ability to interpolate between the keyframes.

10.3.2 Member Typedef Documentation

10.3.2.1 typedef std::vector<KeyFrame*> gazebo::common::Animation::KeyFrame_V [protected]

array of keyframe type alias

10.3.3 Constructor & Destructor Documentation

10.3.3.1 gazebo::common::Animation::Animation (const std::string & *_name,* double *_length,* bool *_loop*)

Constructor.

Parameters

in	*_name*	Name of the animation, should be unique
in	*_length*	Duration of the animation in seconds
in	*_loop*	Set to true if the animation should repeat

10.3.3.2 virtual gazebo::common::Animation::~Animation ( ) [virtual]

Destructor.

10.3.4 Member Function Documentation

10.3.4.1 void gazebo::common::Animation::AddTime (double *_time*)

Add time to the animation.

Parameters

in	*_time*	The amount of time to add in seconds

10.3.4.2 KeyFrame* gazebo::common::Animation::GetKeyFrame (unsigned int *_index*) const

Get a key frame using an index value.

Parameters

in	*_index*	The index of the key frame

Returns

A pointer the keyframe, NULL if the _index is invalid

10.3.4.3 unsigned int gazebo::common::Animation::GetKeyFrameCount ( ) const

Return the number of key frames in the animation.

Returns

The number of keyframes

10.3.4.4 double gazebo::common::Animation::GetKeyFramesAtTime (double *_time,* KeyFrame ** *_kf1,* KeyFrame ** *_kf2,* unsigned int & *_firstKeyIndex*) const [protected]

Get the two key frames that bound a time value.

Parameters

in	*_time*	The time in seconds
out	*_kf1*	Lower bound keyframe that is returned
out	*_kf2*	Upper bound keyframe that is returned
out	*_firstKeyIndex*	Index of the lower bound key frame

Returns

The time between the two keyframe

10.3.4.5 double gazebo::common::Animation::GetLength ( ) const

Return the duration of the animation.

Returns

Duration of the animation in seconds

10.3.4.6 double gazebo::common::Animation::GetTime ( ) const

Return the current time position.

Returns

The time position in seconds

10.3.4.7 void gazebo::common::Animation::SetLength (double *_len*)

Set the duration of the animation.

Parameters

in	*_len*	The length of the animation in seconds

10.3.4.8 void gazebo::common::Animation::SetTime (double *_time*)

Set the current time position of the animation.

Parameters

in	*_time*	The time position in seconds

10.3.5 Member Data Documentation

10.3.5.1 bool gazebo::common::Animation::build [mutable], [protected]

determines if the interpolation splines need building

10.3.5.2 KeyFrame_V gazebo::common::Animation::keyFrames [protected]

array of key frames

10.3.5.3 double gazebo::common::Animation::length [protected]

animation duration

10.3.5.4 bool gazebo::common::Animation::loop [protected]

true if animation repeats

10.3.5.5 std::string gazebo::common::Animation::name [protected]

animation name

10.3.5.6 double gazebo::common::Animation::timePos [protected]

current time position

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

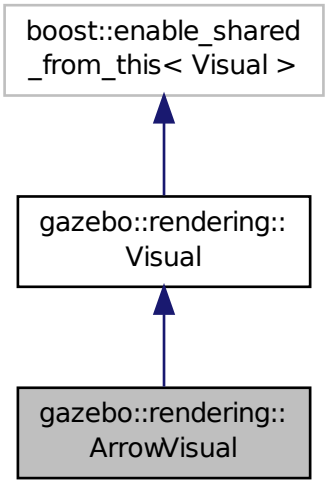
- **Animation.hh**

10.4 gazebo::rendering::ArrowVisual Class Reference

Basic arrow visualization.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::ArrowVisual:



Public Member Functions

- **ArrowVisual** (const std::string &_name, **VisualPtr** _vis)

  Constructor.
- virtual ~**ArrowVisual** ()

  Destructor.
- virtual void **Load** ()

  Load the visual with default parameters.
- void **ShowRotation** ()

  Show the rotation of the visual.

Additional Inherited Members

10.4.1 Detailed Description

Basic arrow visualization.

10.4.2 Constructor & Destructor Documentation

10.4.2.1 gazebo::rendering::ArrowVisual::ArrowVisual (const std::string & *_name,* VisualPtr *_vis*)

Constructor.

Parameters

in	*_name*	Name of the arrow visual
in	*_vis*	Pointer to the parent visual

10.4.2.2 virtual gazebo::rendering::ArrowVisual::~ArrowVisual ( ) [virtual]

Destructor.

10.4.3 Member Function Documentation

10.4.3.1 virtual void gazebo::rendering::ArrowVisual::Load ( ) [virtual]

Load the visual with default parameters.

Reimplemented from **gazebo::rendering::Visual** (p. 933).

10.4.3.2 void gazebo::rendering::ArrowVisual::ShowRotation ( )

Show the rotation of the visual.

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

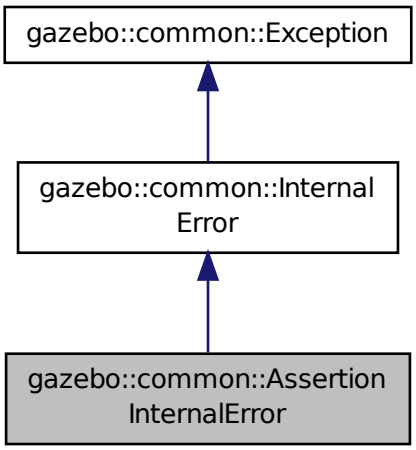
- **ArrowVisual.hh**

10.5 gazebo::common::AssertionInternalError Class Reference

Class for generating Exceptions which come from gazebo assertions.

`#include <common/common.hh>`

Inheritance diagram for gazebo::common::AssertionInternalError:



Public Member Functions

- **AssertionInternalError** (const char ∗_file, int _line, const std::string &_expr, const std::string &_function, const std::string &_msg="")

  Constructor for assertions.
- virtual ∼**AssertionInternalError** ()

  Destructor.

10.5.1 Detailed Description

Class for generating Exceptions which come from gazebo assertions.

They include information about the assertion expression violated, function where problem appeared and assertion debug message.

10.5.2 Constructor & Destructor Documentation

10.5.2.1 gazebo::common::AssertionInternalError::AssertionInternalError (const char ∗ *_file*, int *_line*, const std::string & *_expr*, const std::string & *_function*, const std::string & *_msg* = " ")

Constructor for assertions.

Parameters

in	*_file*	File name
in	*_line*	Line number where the error occurred
in	*_expr*	Assertion expression failed resulting in an internal error

in	*_function*	Function where assertion failed
in	*_msg*	Function where assertion failed

10.5.2.2 virtual gazebo::common::AssertionInternalError::~AssertionInternalError ( ) [virtual]

Destructor.

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

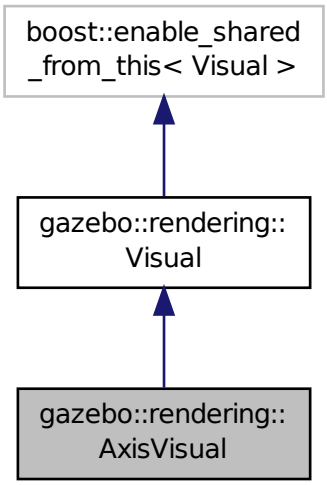
- **Exception.hh**

10.6 gazebo::rendering::AxisVisual Class Reference

Basic axis visualization.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::AxisVisual:



Public Member Functions

- **AxisVisual** (const std::string &_name, **VisualPtr** _vis)

  Constructor.
- virtual ~**AxisVisual** ()

  Destructor.
- virtual void **Load** ()

Load the axis visual.

- void **ScaleXAxis** (const **math::Vector3** &_scale)

  Scale the X axis.
- void **ScaleYAxis** (const **math::Vector3** &_scale)

  Scale the Y axis.
- void **ScaleZAxis** (const **math::Vector3** &_scale)

  Scale the Z axis.
- void **SetAxisMaterial** (unsigned int _axis, const std::string &_material)

  Set the material used to render and axis.
- void **ShowRotation** (unsigned int _axis)

  Load the rotation tube.

Additional Inherited Members

10.6.1 Detailed Description

Basic axis visualization.

10.6.2 Constructor & Destructor Documentation

10.6.2.1 gazebo::rendering::AxisVisual::AxisVisual (const std::string & *_name*, VisualPtr *_vis*)

Constructor.

Parameters

in	*_name*	Name of the **AxisVisual** (p. 137)
in	*_vis*	Parent visual

10.6.2.2 virtual gazebo::rendering::AxisVisual::~AxisVisual ( ) [virtual]

Destructor.

10.6.3 Member Function Documentation

10.6.3.1 virtual void gazebo::rendering::AxisVisual::Load ( ) [virtual]

Load the axis visual.

Reimplemented from **gazebo::rendering::Visual** (p. 933).

10.6.3.2 void gazebo::rendering::AxisVisual::ScaleXAxis (const math::Vector3 & *_scale*)

Scale the X axis.

Parameters

in	*_scale*	Scaling factor

10.6.3.3 void gazebo::rendering::AxisVisual::ScaleYAxis (const math::Vector3 & _scale)

Scale the Y axis.

Parameters

in	*_scale*	Scaling factor

10.6.3.4 void gazebo::rendering::AxisVisual::ScaleZAxis (const math::Vector3 & _scale)

Scale the Z axis.

Parameters

in	*_scale*	Scaling factor

10.6.3.5 void gazebo::rendering::AxisVisual::SetAxisMaterial (unsigned int _axis, const std::string & _material)

Set the material used to render and axis.

Parameters

in	*_axis*	The number of the axis (0, 1, 2 = x,y,z)
in	*_material*	The name of the material to apply to the axis

10.6.3.6 void gazebo::rendering::AxisVisual::ShowRotation (unsigned int _axis)

Load the rotation tube.

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

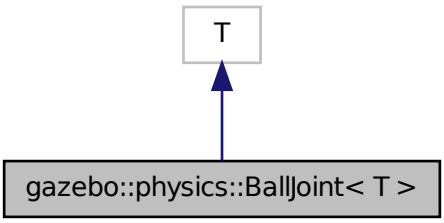
- **AxisVisual.hh**

10.7 gazebo::physics::BallJoint< T > Class Template Reference

Base (p. 141) class for a ball joint.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::BallJoint< T >:



Public Member Functions

- **BallJoint** (**BasePtr** _parent)

  Constructor.
- virtual ~**BallJoint** ()

  Destructor.
- virtual unsigned int **GetAngleCount** () const
- virtual **math::Angle GetHighStop** (int)
- virtual **math::Angle GetLowStop** (int)
- void **Load** (**sdf::ElementPtr** _sdf)

  *Template to ::Load the **BallJoint** (p. 139).*
- virtual void **SetAxis** (int, const **math::Vector3** &)
- virtual void **SetHighStop** (int, **math::Angle**)
- virtual void **SetLowStop** (int, **math::Angle**)

10.7.1 Detailed Description

template<class T>class gazebo::physics::BallJoint< T >

Base (p. 141) class for a ball joint.

Each physics engine should implement this class.

10.7.2 Constructor & Destructor Documentation

10.7.2.1 template<class T > gazebo::physics::BallJoint< T >::BallJoint (BasePtr *_parent*) [inline], [explicit]

Constructor.

Parameters

in	*_parent*	Pointer to the parent link.

References gazebo::physics::Base::BALL_JOINT.

10.7.2.2 template<class T > virtual gazebo::physics::BallJoint< T >::~BallJoint ( ) [inline], [virtual]

Destructor.

10.7.3 Member Function Documentation

10.7.3.1 template<class T > virtual unsigned int gazebo::physics::BallJoint< T >::GetAngleCount ( ) const [inline], [virtual]

10.7.3.2 template<class T > virtual math::Angle gazebo::physics::BallJoint< T >::GetHighStop (int) [inline], [virtual]

10.7.3.3 template<class T > virtual math::Angle gazebo::physics::BallJoint< T >::GetLowStop (int) [inline], [virtual]

10.7.3.4 template<class T > void gazebo::physics::BallJoint< T >::Load (sdf::ElementPtr _sdf) [inline]

Template to ::Load the **BallJoint** (p. 139).

Parameters

in	*_sdf*	SDF to load the joint from.

10.7.3.5 template<class T > virtual void gazebo::physics::BallJoint< T >::SetAxis (int , const math::Vector3 &) [inline], [virtual]

10.7.3.6 template<class T > virtual void gazebo::physics::BallJoint< T >::SetHighStop (int , math::Angle) [inline], [virtual]

10.7.3.7 template<class T > virtual void gazebo::physics::BallJoint< T >::SetLowStop (int , math::Angle) [inline], [virtual]

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

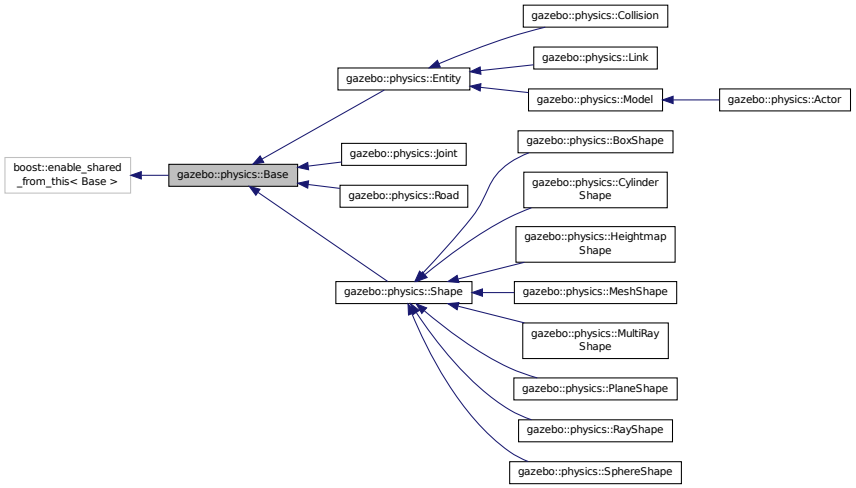
- **BallJoint.hh**

10.8 gazebo::physics::Base Class Reference

Base (p. 141) class for most physics classes.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::Base:



Public Types

- enum **EntityType** {
  BASE = 0x00000000, **ENTITY** = 0x00000001, **MODEL** = 0x00000002, **LINK** = 0x00000004,
  COLLISION = 0x00000008, **ACTOR** = 0x00000016, **LIGHT** = 0x00000010, **VISUAL** = 0x00000020,
  JOINT = 0x00000040, **BALL_JOINT** = 0x00000080, **HINGE2_JOINT** = 0x00000100, **HINGE_JOINT** = 0x00000200,
  SLIDER_JOINT = 0x00000400, **SCREW_JOINT** = 0x00000800, **UNIVERSAL_JOINT** = 0x00001000, **SHAPE** = 0x00002000,
  BOX_SHAPE = 0x00004000, **CYLINDER_SHAPE** = 0x00008000, **HEIGHTMAP_SHAPE** = 0x00010000, **MAP-_SHAPE** = 0x00020000,
  MULTIRAY_SHAPE = 0x00040000, **RAY_SHAPE** = 0x00080000, **PLANE_SHAPE** = 0x00100000, **SPHERE_-SHAPE** = 0x00200000,
  MESH_SHAPE = 0x00400000, **SENSOR_COLLISION** = 0x00800000 }

  Unique identifiers for all entity types.

Public Member Functions

- **Base** (**BasePtr** _parent)

  Constructor.
- virtual ~**Base** ()

  Destructor.
- void **AddChild** (**BasePtr** _child)

  Add a child to this entity.
- void **AddType** (**EntityType** _type)

  Add a type specifier.
- virtual void **Fini** ()

  Finialize the object.
- **BasePtr GetById** (unsigned int _id) const

This is an internal function.

- **BasePtr GetByName** (const std::string &_name)

  Get by name.
- **BasePtr GetChild** (unsigned int _i) const

  Get a child by index.
- **BasePtr GetChild** (const std::string &_name)

  Get a child by name.
- unsigned int **GetChildCount** () const

  Get the number of children.
- unsigned int **GetId** () const

  Return the ID of this entity.
- std::string **GetName** () const

  Return the name of the entity.
- **BasePtr GetParent** () const

  Get the parent.
- int **GetParentId** () const

  Return the ID of the parent.
- bool **GetSaveable** () const

  Get whether the object should be "saved", when the user selects to save the world to xml.
- std::string **GetScopedName** () const

  Return the name of this entity with the model scope world::model1::...::modelN::entityName.
- virtual const **sdf::ElementPtr GetSDF** ()

  Get the SDF values for the object.
- unsigned int **GetType** () const

  Get the full type definition.
- const **WorldPtr** & **GetWorld** () const

  *Get the **World** (p. 945) this object is in.*
- bool **HasType** (const **EntityType** &_t) const

  Returns true if this object's type definition has the given type.
- virtual void **Init** ()

  Initialize the object.
- bool **IsSelected** () const

  True if the entity is selected by the user.
- virtual void **Load** (**sdf::ElementPtr** _sdf)

  Load.
- bool **operator==** (const **Base** &_ent) const

  Returns true if the entities are the same.
- void **Print** (const std::string &_prefix)

  Print this object to screen via gzmsg.
- virtual void **RemoveChild** (unsigned int _id)

  Remove a child from this entity.
- void **RemoveChild** (const std::string &_name)

  Remove a child by name.
- void **RemoveChildren** ()

  Remove all children.
- virtual void **Reset** ()

  Reset the object.

- virtual void **Reset** (**Base::EntityType** _resetType)

  *Calls recursive Reset on one of the **Base::EntityType** (p. 144)'s.*
- virtual void **SetName** (const std::string &_name)

  Set the name of the entity.
- void **SetParent** (**BasePtr** _parent)

  Set the parent.
- void **SetSaveable** (bool _v)

  Set whether the object should be "saved", when the user selects to save the world to xml.
- virtual bool **SetSelected** (bool _show)

  Set whether this entity has been selected by the user through the gui.
- void **SetWorld** (const **WorldPtr** &_newWorld)

  Set the world this object belongs to.
- virtual void **Update** ()

  Update the object.
- virtual void **UpdateParameters** (**sdf::ElementPtr** _sdf)

  Update the parameters using new sdf values.

Protected Member Functions

- void **ComputeScopedName** ()

  Compute the scoped name of this object based on its parents.

Protected Attributes

- **Base_V children**

  Children of this entity.
- Base_V::iterator **childrenEnd**

  End of the children vector.
- **BasePtr parent**

  Parent of this entity.
- **sdf::ElementPtr sdf**

  The SDF values for this object.
- **WorldPtr world**

  Pointer to the world.

10.8.1 Detailed Description

Base (p. 141) class for most physics classes.

10.8.2 Member Enumeration Documentation

10.8.2.1 enum gazebo::physics::Base::EntityType

Unique identifiers for all entity types.

Enumerator:

BASE **Base** (p. 141) type.

ENTITY **Entity** (p. 288) type.

MODEL **Model** (p. 516) type.

LINK **Link** (p. 439) type.

COLLISION **Collision** (p. 199) type.

ACTOR **Actor** (p. 115) type.

LIGHT Light type.

VISUAL Visual type.

JOINT **Joint** (p. 401) type.

BALL_JOINT **BallJoint** (p. 139) type.

HINGE2_JOINT Hing2Joint type.

HINGE_JOINT **HingeJoint** (p. 377) type.

SLIDER_JOINT **SliderJoint** (p. 780) type.

SCREW_JOINT **ScrewJoint** (p. 724) type.

UNIVERSAL_JOINT **UniversalJoint** (p. 862) type.

SHAPE **Shape** (p. 754) type.

BOX_SHAPE **BoxShape** (p. 157) type.

CYLINDER_SHAPE **CylinderShape** (p. 257) type.

HEIGHTMAP_SHAPE **HeightmapShape** (p. 371) type.

MAP_SHAPE MapShape type.

MULTIRAY_SHAPE **MultiRayShape** (p. 556) type.

RAY_SHAPE **RayShape** (p. 679) type.

PLANE_SHAPE **PlaneShape** (p. 619) type.

SPHERE_SHAPE **SphereShape** (p. 790) type.

MESH_SHAPE **MeshShape** (p. 513) type.

SENSOR_COLLISION Indicates a collision shape used for sensing.

10.8.3 Constructor & Destructor Documentation

10.8.3.1 gazebo::physics::Base::Base (BasePtr *_parent*) [explicit]

Constructor.

Parameters

in	*_parent*	Parent of this object

10.8.3.2 virtual gazebo::physics::Base::~Base ( ) [virtual]

Destructor.

10.8.4 Member Function Documentation

10.8.4.1 void gazebo::physics::Base::AddChild (BasePtr _child)

Add a child to this entity.

Parameters

in	*_child*	Child entity.

10.8.4.2 void gazebo::physics::Base::AddType (EntityType _type)

Add a type specifier.

Parameters

in	*_type*	New type to append to this objects type definition.

10.8.4.3 void gazebo::physics::Base::ComputeScopedName ( ) [protected]

Compute the scoped name of this object based on its parents.

See Also

Base::GetScopedName (p. 148)

10.8.4.4 virtual void gazebo::physics::Base::Fini ( ) [virtual]

Finialize the object.

Reimplemented in **gazebo::physics::Actor** (p. 118), **gazebo::physics::Model** (p. 521), **gazebo::physics::Link** (p. 447), **gazebo::physics::Entity** (p. 291), and **gazebo::physics::Collision** (p. 203).

10.8.4.5 BasePtr gazebo::physics::Base::GetById (unsigned int _id) const

This is an internal function.

Get a child or self by id.

Parameters

in	*_id*	ID of the object to retreive.

Returns

A pointer to the object, NULL if not found

10.8.4.6 BasePtr gazebo::physics::Base::GetByName (const std::string & _name)

Get by name.

Parameters

in	*_name*	Get a child (or self) object by name

Returns

A pointer to the object, NULL if not found

10.8.4.7 BasePtr gazebo::physics::Base::GetChild (unsigned int *_i*) const

Get a child by index.

Parameters

in	*_i*	Index of the child to retreive.

Returns

A pointer to the object, NULL if the index is invalid.

10.8.4.8 BasePtr gazebo::physics::Base::GetChild (const std::string & *_name*)

Get a child by name.

Parameters

in	*_name*	Name of the child.

Returns

A pointer to the object, NULL if not found

10.8.4.9 unsigned int gazebo::physics::Base::GetChildCount ( ) const

Get the number of children.

Returns

The number of children.

10.8.4.10 unsigned int gazebo::physics::Base::GetId ( ) const

Return the ID of this entity.

This id is unique.

Returns

Integer ID.

10.8.4.11 std::string gazebo::physics::Base::GetName ( ) const

Return the name of the entity.

Returns

Name of the entity.

10.8.4.12 BasePtr gazebo::physics::Base::GetParent ( ) const

Get the parent.

Returns

Pointer to the parent entity.

10.8.4.13 int gazebo::physics::Base::GetParentId ( ) const

Return the ID of the parent.

Returns

Integer ID.

10.8.4.14 bool gazebo::physics::Base::GetSaveable ( ) const

Get whether the object should be "saved", when the user selects to save the world to xml.

Returns

True if the object is saveable.

10.8.4.15 std::string gazebo::physics::Base::GetScopedName ( ) const

Return the name of this entity with the model scope world::model1::...::modelN::entityName.

Returns

The scoped name.

10.8.4.16 virtual const sdf::ElementPtr gazebo::physics::Base::GetSDF ( ) [virtual]

Get the SDF values for the object.

Returns

The SDF values for the object.

Reimplemented in **gazebo::physics::Actor** (p. 118), and **gazebo::physics::Model** (p. 524).

10.8.4.17 unsigned int gazebo::physics::Base::GetType ( ) const

Get the full type definition.

Returns

The full type definition.

10.8.4.18 const WorldPtr& gazebo::physics::Base::GetWorld ( ) const

Get the **World** (p. 945) this object is in.

Returns

The **World** (p. 945) this object is part of.

10.8.4.19 bool gazebo::physics::Base::HasType (const EntityType & _t) const

Returns true if this object's type definition has the given type.

Parameters

in	*_t*	Type to check.

Returns

True if this object's type definition has the.

10.8.4.20 virtual void gazebo::physics::Base::Init ( ) [inline], [virtual]

Initialize the object.

Reimplemented in **gazebo::physics::Joint** (p. 413), **gazebo::physics::RayShape** (p. 682), **gazebo::physics::Actor** (p. 118), **gazebo::physics::Link** (p. 453), **gazebo::physics::Model** (p. 525), **gazebo::physics::Collision** (p. 206), **gazebo::physics::HeightmapShape** (p. 374), **gazebo::physics::MeshShape** (p. 515), **gazebo::physics::MultiRayShape** (p. 563), **gazebo::physics::PlaneShape** (p. 622), **gazebo::physics::Road** (p. 698), **gazebo::physics::Shape** (p. 756), **gazebo::physics::SphereShape** (p. 792), **gazebo::physics::BoxShape** (p. 159), and **gazebo::physics::CylinderShape** (p. 260).

10.8.4.21 bool gazebo::physics::Base::IsSelected ( ) const

True if the entity is selected by the user.

Returns

True if the entity is selected.

10.8.4.22 virtual void gazebo::physics::Base::Load (sdf::ElementPtr _sdf) [virtual]

Load.

Parameters

in	*node*	Pointer to an SDF parameters

Reimplemented in **gazebo::physics::Joint** (p. 414), **gazebo::physics::Actor** (p. 118), **gazebo::physics::Entity** (p. 295), **gazebo::physics::Link** (p. 453), **gazebo::physics::Model** (p. 525), **gazebo::physics::Collision** (p. 207), **gazebo::physics::HeightmapShape** (p. 375), and **gazebo::physics::Road** (p. 698).

10.8.4.23 bool gazebo::physics::Base::operator== (const Base & *_ent*) const

Returns true if the entities are the same.

Checks only the name.

Parameters

in	*_ent*	**Base** (p. 141) object to compare with.

Returns

True if the entities are the same.

10.8.4.24 void gazebo::physics::Base::Print (const std::string & *_prefix*)

Print this object to screen via gzmsg.

Parameters

in	*_prefix*	Usually a set of spaces.

10.8.4.25 virtual void gazebo::physics::Base::RemoveChild (unsigned int *_id*) [virtual]

Remove a child from this entity.

Parameters

in	*_id*	ID of the child to remove.

10.8.4.26 void gazebo::physics::Base::RemoveChild (const std::string & *_name*)

Remove a child by name.

Parameters

in	*_name*	Name of the child.

10.8.4.27 void gazebo::physics::Base::RemoveChildren ( )

Remove all children.

10.8.4.28 virtual void gazebo::physics::Base::Reset ( ) [virtual]

Reset the object.

Reimplemented in **gazebo::physics::Joint** (p. 414), **gazebo::physics::Model** (p. 526), **gazebo::physics::Link** (p. 454), and **gazebo::physics::Entity** (p. 296).

10.8.4.29 virtual void gazebo::physics::Base::Reset (Base::EntityType *_resetType*) [virtual]

Calls recursive Reset on one of the **Base::EntityType** (p. 144)'s.

Parameters

in	*_resetType*	The type of reset operation

10.8.4.30 virtual void gazebo::physics::Base::SetName (const std::string & *_name*) [virtual]

Set the name of the entity.

Parameters

in	*_name*	New name.

Reimplemented in **gazebo::physics::Entity** (p. 297).

10.8.4.31 void gazebo::physics::Base::SetParent (BasePtr *_parent*)

Set the parent.

Parameters

in	*_parent*	Parent object.

10.8.4.32 void gazebo::physics::Base::SetSaveable (bool *_v*)

Set whether the object should be "saved", when the user selects to save the world to xml.

Parameters

in	*_v*	Set to True if the object should be saved.

10.8.4.33 virtual bool gazebo::physics::Base::SetSelected (bool *_show*) [virtual]

Set whether this entity has been selected by the user through the gui.

Parameters

in	*_show*	True to set this entity as selected.

Reimplemented in **gazebo::physics::Link** (p. 457).

10.8.4.34 void gazebo::physics::Base::SetWorld (const WorldPtr & *_newWorld*)

Set the world this object belongs to.

This will also set the world for all children.

Parameters

in	*_newWorld*	The new **World** (p. 945) this object is part of.

10.8.4.35 virtual void gazebo::physics::Base::Update ( ) [inline], [virtual]

Update the object.

Reimplemented in **gazebo::physics::Joint** (p. 416), **gazebo::physics::MultiRayShape** (p. 563), **gazebo::physics::Link** (p. 458), **gazebo::physics::Actor** (p. 119), **gazebo::physics::RayShape** (p. 683), **gazebo::physics::Model** (p. 529), and **gazebo::physics::MeshShape** (p. 516).

10.8.4.36 virtual void gazebo::physics::Base::UpdateParameters (sdf::ElementPtr *_sdf*) [virtual]

Update the parameters using new sdf values.

Parameters

in	*_sdf*	Update the object's parameters based on SDF values.

Reimplemented in **gazebo::physics::Joint** (p. 417), **gazebo::physics::Actor** (p. 119), **gazebo::physics::Link** (p. 458), **gazebo::physics::Model** (p. 529), **gazebo::physics::Entity** (p. 298), and **gazebo::physics::Collision** (p. 209).

10.8.5 Member Data Documentation

10.8.5.1 Base_V gazebo::physics::Base::children [protected]

Children of this entity.

10.8.5.2 Base_V::iterator gazebo::physics::Base::childrenEnd [protected]

End of the children vector.

10.8.5.3 BasePtr gazebo::physics::Base::parent [protected]

Parent of this entity.

10.8.5.4 sdf::ElementPtr gazebo::physics::Base::sdf [protected]

The SDF values for this object.

10.8.5.5 WorldPtr gazebo::physics::Base::world [protected]

Pointer to the world.

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

- **Base.hh**

10.9 gazebo::math::Box Class Reference

Mathematical representation of a box and related functions.

`#include <math/gzmath.hh>`

Public Member Functions

- **Box** ()

  Default constructor.
- **Box** (const **Vector3** &_min, const **Vector3** &_max)

  Constructor.
- **Box** (const **Box** &_b)

  Copy Constructor.
- virtual ~**Box** ()

  Destructor.
- **math::Vector3 GetCenter** () const

  Get the box center.
- **math::Vector3 GetSize** () const

  Get the size of the box.
- double **GetXLength** () const

  Get the length along the x dimension.
- double **GetYLength** () const

  Get the length along the y dimension.
- double **GetZLength** () const

  Get the length along the z dimension.
- void **Merge** (const **Box** &_box)

  Merge a box with this box.
- **Box operator+** (const **Box** &_b) const

  Addition operator.
- const **Box** & **operator+=** (const **Box** &_b)

  Addition set operator.
- **Box operator-** (const **Vector3** &_v)

  Subtract a vector from the min and max values.
- **Box** & **operator=** (const **Box** &_b)

  Assignment operator.
- bool **operator==** (const **Box** &_b)

  Equality test operatoer.

Public Attributes

- **Vector3 max**

  Maximum corner of the box.
- **Vector3 min**

  Minimum corner of the box.

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::math::Box** &_b)

  Output operator.

10.9.1 Detailed Description

Mathematical representation of a box and related functions.

10.9.2 Constructor & Destructor Documentation

10.9.2.1 gazebo::math::Box::Box ( )

Default constructor.

10.9.2.2 gazebo::math::Box::Box (const Vector3 & *_min*, const Vector3 & *_max*)

Constructor.

Parameters

in	*_min*	Minimum corner of the box
in	*_max*	Maximum corner of the box

10.9.2.3 gazebo::math::Box::Box (const Box & *_b*)

Copy Constructor.

Parameters

in	*_b*	**Box** (p. 153) to copy

10.9.2.4 virtual gazebo::math::Box::~Box ( ) [virtual]

Destructor.

10.9.3 Member Function Documentation

10.9.3.1 math::Vector3 gazebo::math::Box::GetCenter ( ) const

Get the box center.

Returns

The center position of the box

10.9.3.2 math::Vector3 gazebo::math::Box::GetSize ( ) const

Get the size of the box.

Returns

Size of the box

10.9.3.3 double gazebo::math::Box::GetXLength ( ) const

Get the length along the x dimension.

Returns

Double value of the length in the x dimension

10.9.3.4 double gazebo::math::Box::GetYLength ( ) const

Get the length along the y dimension.

Returns

Double value of the length in the y dimension

10.9.3.5 double gazebo::math::Box::GetZLength ( ) const

Get the length along the z dimension.

Returns

Double value of the length in the z dimension

10.9.3.6 void gazebo::math::Box::Merge (const Box & *_box*)

Merge a box with this box.

Parameters

in	*_box*	**Box** (p. 153) to add to this box

10.9.3.7 Box gazebo::math::Box::operator+ (const Box & _b) const

Addition operator.

result = this + _b

Parameters

in	*_b*	**Box** (p. 153) to add

Returns

The new box

10.9.3.8 const Box& gazebo::math::Box::operator+= (const Box & _b)

Addition set operator.

this = this + _b

Parameters

in	*_b*	**Box** (p. 153) to add

Returns

This new box

10.9.3.9 Box gazebo::math::Box::operator- (const Vector3 & _v)

Subtract a vector from the min and max values.

Parameters

_v	The vector to use during subtraction

Returns

The new box

10.9.3.10 Box& gazebo::math::Box::operator= (const Box & _b)

Assignment operator.

Set this box to the parameter

Parameters

in	*_b*	**Box** (p. 153) to copy

Returns

The new box.

10.9.3.11 bool gazebo::math::Box::operator== (const Box & _b)

Equality test operatoer.

Parameters

in	*_b*	**Box** (p. 153) to test

Returns

True if equal

10.9.4 Friends And Related Function Documentation

10.9.4.1 std::ostream& operator<< (std::ostream & _out, const gazebo::math::Box & _b) [friend]

Output operator.

Parameters

in	*_out*	Output stream
in	*_b*	**Box** (p. 153) to output to the stream

Returns

The stream

10.9.5 Member Data Documentation

10.9.5.1 Vector3 gazebo::math::Box::max

Maximum corner of the box.

10.9.5.2 Vector3 gazebo::math::Box::min

Minimum corner of the box.

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

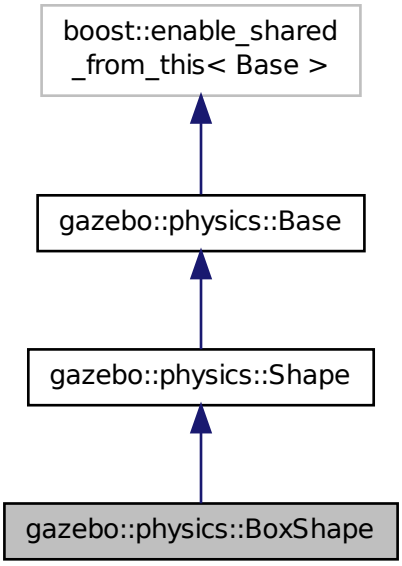
- **Box.hh**

10.10 gazebo::physics::BoxShape Class Reference

Box geometry primitive.

`#include <physics/physcs.hh>`

Inheritance diagram for gazebo::physics::BoxShape:



Public Member Functions

- **BoxShape** (**CollisionPtr** _parent)

  Constructor.
- virtual ~**BoxShape** ()

  Destructor.
- void **FillMsg** (msgs::Geometry &_msg)

  Fill in the values for a geomertry message.
- **math::Vector3 GetSize** () const

  Get the size of the box.
- virtual void **Init** ()

  Initialize the box.
- virtual void **ProcessMsg** (const msgs::Geometry &_msg)

  Process a geometry message.
- virtual void **SetSize** (const **math::Vector3** &_size)

  Set the size of the box.

Additional Inherited Members

10.10.1 Detailed Description

Box geometry primitive.

10.10.2 Constructor & Destructor Documentation

10.10.2.1 gazebo::physics::BoxShape::BoxShape (CollisionPtr _parent) [explicit]

Constructor.

Parameters

in	*_parent*	Parent **Collision** (p. 199).

10.10.2.2 virtual gazebo::physics::BoxShape::~BoxShape ( ) [virtual]

Destructor.

10.10.3 Member Function Documentation

10.10.3.1 void gazebo::physics::BoxShape::FillMsg (msgs::Geometry & _msg) [virtual]

Fill in the values for a geomertry message.

Parameters

out	*_msg*	The geometry message to fill.

Implements **gazebo::physics::Shape** (p. 756).

10.10.3.2 math::Vector3 gazebo::physics::BoxShape::GetSize ( ) const

Get the size of the box.

Returns

The size of each side of the box.

10.10.3.3 virtual void gazebo::physics::BoxShape::Init ( ) [virtual]

Initialize the box.

Implements **gazebo::physics::Shape** (p. 756).

10.10.3.4 virtual void gazebo::physics::BoxShape::ProcessMsg (const msgs::Geometry & _msg) [virtual]

Process a geometry message.

Parameters

in	*_msg*	The message to set values from.

Implements **gazebo::physics::Shape** (p. 756).

10.10.3.5 virtual void gazebo::physics::BoxShape::SetSize (const math::Vector3 & _size) [virtual]

Set the size of the box.

Parameters

in	*_size*	Size of each side of the box.

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

- **BoxShape.hh**

10.11 gazebo::common::BVHLoader Class Reference

Handles loading BVH animation files.

`#include <common/common.hh>`

Public Member Functions

- **BVHLoader** ()
  Constructor.
- ~**BVHLoader** ()
  Desutrctor.
- **Skeleton** * **Load** (const std::string &_filename, double _scale)
  Load a BVH file.

10.11.1 Detailed Description

Handles loading BVH animation files.

10.11.2 Constructor & Destructor Documentation

10.11.2.1 gazebo::common::BVHLoader::BVHLoader ( )

Constructor.

10.11.2.2 gazebo::common::BVHLoader::~BVHLoader ( )

Desutrctor.

10.11.3 Member Function Documentation

10.11.3.1 Skeleton* gazebo::common::BVHLoader::Load (const std::string & _filename, double _scale)

Load a BVH file.

Parameters

in	*_filename*	BVH file to load
in	*_scale*	Scaling factor to apply to the skeleton

Returns

A pointer to a new **Skeleton** (p. 760)

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

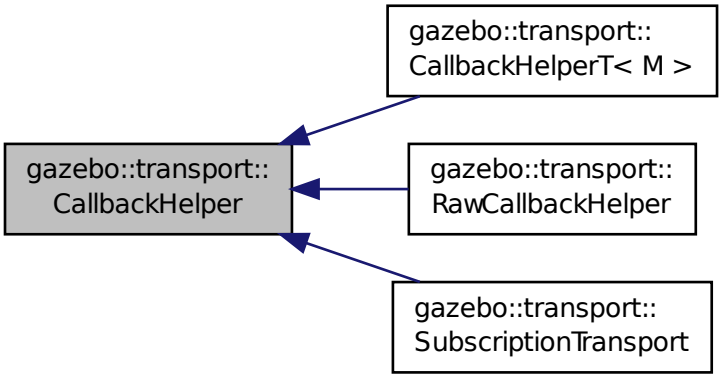
- **BVHLoader.hh**

10.12 gazebo::transport::CallbackHelper Class Reference

A helper class to handle callbacks when messages arrive.

`#include <transport/transport.hh>`

Inheritance diagram for gazebo::transport::CallbackHelper:



Public Member Functions

- **CallbackHelper** (bool _latching=false)

  Constructor.
- virtual ~**CallbackHelper** ()

  Destructor.
- unsigned int **GetId** () const

  Get the unique ID of this callback.
- bool **GetLatching** () const

  Is the callback latching?

- virtual std::string **GetMsgType** () const

  Get the typename of the message that is handled.
- virtual bool **HandleData** (const std::string &_newdata, boost::function< void(uint32_t)> _cb, uint32_t _id)=0

  Process new incoming data.
- virtual bool **HandleMessage** (**MessagePtr** _newMsg)=0

  Process new incoming message.
- virtual bool **IsLocal** () const =0

  Is the callback local?

Protected Attributes

- bool **latching**

  True means that the callback helper will get the last published message on the topic.

10.12.1 Detailed Description

A helper class to handle callbacks when messages arrive.

10.12.2 Constructor & Destructor Documentation

10.12.2.1 gazebo::transport::CallbackHelper::CallbackHelper (bool *_latching* = `false`)

Constructor.

Parameters

in	*_latching*	Set to true to make the callback helper latching.

10.12.2.2 virtual gazebo::transport::CallbackHelper::~CallbackHelper ( ) [virtual]

Destructor.

10.12.3 Member Function Documentation

10.12.3.1 unsigned int gazebo::transport::CallbackHelper::GetId ( ) const

Get the unique ID of this callback.

Returns

The unique ID of this callback.

10.12.3.2 bool gazebo::transport::CallbackHelper::GetLatching ( ) const

Is the callback latching?

Returns

true if the callback is latching, false otherwise

10.12.3.3 virtual std::string gazebo::transport::CallbackHelper::GetMsgType ( ) const [virtual]

Get the typename of the message that is handled.

Returns

String representation of the message type

Reimplemented in **gazebo::transport::RawCallbackHelper** (p. 671), and **gazebo::transport::CallbackHelperT< M >** (p. 165).

10.12.3.4 virtual bool gazebo::transport::CallbackHelper::HandleData (const std::string & *_newdata,* boost::function< void(uint32_t)> *_cb,* uint32_t *_id*) [pure virtual]

Process new incoming data.

Parameters

in	*_newdata*	Incoming data to be processed

Returns

true if successfully processed; false otherwise

Parameters

in	*_cb*	If non-null, callback to be invoked which signals that transmission is complete.
in	*_id*	ID associated with the message data.

Implemented in **gazebo::transport::RawCallbackHelper** (p. 671), **gazebo::transport::CallbackHelperT< M >** (p. 165), and **gazebo::transport::SubscriptionTransport** (p. 819).

10.12.3.5 virtual bool gazebo::transport::CallbackHelper::HandleMessage (MessagePtr *_newMsg*) [pure virtual]

Process new incoming message.

Parameters

in	*_newMsg*	Incoming message to be processed

Returns

true if successfully processed; false otherwise

Implemented in **gazebo::transport::RawCallbackHelper** (p. 671), **gazebo::transport::CallbackHelperT< M >** (p. 166), and **gazebo::transport::SubscriptionTransport** (p. 819).

10.12.3.6 virtual bool gazebo::transport::CallbackHelper::IsLocal ( ) const [pure virtual]

Is the callback local?

Returns

true if the callback is local, false if the callback is tied to a remote connection

Implemented in **gazebo::transport::RawCallbackHelper** (p. 671), **gazebo::transport::CallbackHelperT< M >** (p. 166), and **gazebo::transport::SubscriptionTransport** (p. 820).

10.12.4 Member Data Documentation

10.12.4.1 bool gazebo::transport::CallbackHelper::latching [protected]

True means that the callback helper will get the last published message on the topic.

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

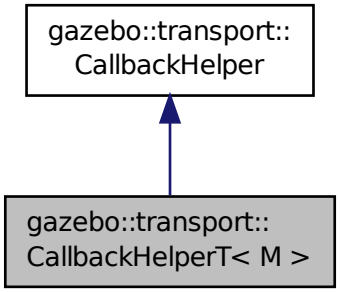
- **CallbackHelper.hh**

10.13 gazebo::transport::CallbackHelperT< M > Class Template Reference

Callback helper Template.

`#include <transport/transport.hh>`

Inheritance diagram for gazebo::transport::CallbackHelperT< M >:



Public Member Functions

- **CallbackHelperT** (const boost::function< void(const boost::shared_ptr< M const > &)> &_cb, bool _latching=false)

  Constructor.

- std::string **GetMsgType** () const

  Get the typename of the message that is handled.
- virtual bool **HandleData** (const std::string &_newdata, boost::function< void(uint32_t)> _cb, uint32_t _id)

  Process new incoming data.
- virtual bool **HandleMessage** (**MessagePtr** _newMsg)

  Process new incoming message.
- virtual bool **IsLocal** () const

  Is the callback local?

Additional Inherited Members

10.13.1 Detailed Description

template<class M>class gazebo::transport::CallbackHelperT< M >

Callback helper Template.

10.13.2 Constructor & Destructor Documentation

10.13.2.1 template<class M > gazebo::transport::CallbackHelperT< M >::CallbackHelperT (const boost::function< void(const boost::shared_ptr< M const > &)> & *_cb,* bool *_latching* = `false`) `[inline]`

Constructor.

Parameters

in	*_cb*	boost function to call on incoming messages
in	*_latching*	Set to true to make the callback helper latching.

10.13.3 Member Function Documentation

10.13.3.1 template<class M > std::string gazebo::transport::CallbackHelperT< M >::GetMsgType ( ) const `[inline]`, `[virtual]`

Get the typename of the message that is handled.

Returns

String representation of the message type

Reimplemented from **gazebo::transport::CallbackHelper** (p. 163).

References gzthrow, and NULL.

10.13.3.2 template<class M > virtual bool gazebo::transport::CallbackHelperT< M >::HandleData (const std::string & *_newdata,* boost::function< void(uint32_t)> *_cb,* uint32_t *_id*) `[inline]`, `[virtual]`

Process new incoming data.

Parameters

in	*_newdata*	Incoming data to be processed

Returns

true if successfully processed; false otherwise

Parameters

in	*_cb*	If non-null, callback to be invoked which signals that transmission is complete.
in	*_id*	ID associated with the message data.

Implements **gazebo::transport::CallbackHelper** (p. 163).

10.13.3.3 template<class M > virtual bool gazebo::transport::CallbackHelperT< M >::HandleMessage (MessagePtr *_newMsg*) [inline], [virtual]

Process new incoming message.

Parameters

in	*_newMsg*	Incoming message to be processed

Returns

true if successfully processed; false otherwise

Implements **gazebo::transport::CallbackHelper** (p. 163).

10.13.3.4 template<class M > virtual bool gazebo::transport::CallbackHelperT< M >::IsLocal ( ) const [inline], [virtual]

Is the callback local?

Returns

true if the callback is local, false if the callback is tied to a remote connection

Implements **gazebo::transport::CallbackHelper** (p. 164).

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

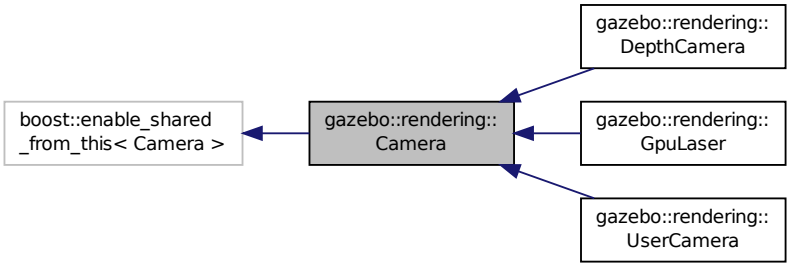
- **CallbackHelper.hh**

10.14 gazebo::rendering::Camera Class Reference

Basic camera sensor.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::Camera:



Public Member Functions

- **Camera** (const std::string &_namePrefix, **ScenePtr** _scene, bool _autoRender=true)

  Constructor.
- virtual ~**Camera** ()

  Destructor.
- void **AttachToVisual** (const std::string &_visualName, bool _inheritOrientation, double _minDist=0.0, double _maxDist=0.0)

  Attach the camera to a scene node.
- template<typename T >
  event::ConnectionPtr ConnectNewImageFrame (T _subscriber)

  Connect to the new image signal.
- void **CreateRenderTexture** (const std::string &_textureName)

  Set the render target.
- void **DisconnectNewImageFrame** (**event::ConnectionPtr** &_c)

  Disconnect from an image frame.
- void **EnableSaveFrame** (bool _enable)

  Enable or disable saving.
- virtual void **Fini** ()

  Finalize the camera.
- float **GetAspectRatio** () const

  Get the apect ratio.
- virtual float **GetAvgFPS** ()

  Get the average FPS.
- void **GetCameraToViewportRay** (int _screenx, int _screeny, **math::Vector3** &_origin, **math::Vector3** &_dir)

  Get a world space ray as cast from the camera through the viewport.
- **math::Vector3 GetDirection** () const

  Get the camera's direction vector.
- double **GetFarClip** ()

  Get the far clip distance.
- **math::Angle GetHFOV** () const

  Get the camera FOV (horizontal)

- size_t **GetImageByteSize** () const

  Get the image size in bytes.
- virtual const unsigned char * **GetImageData** (unsigned int i=0)

  Get a pointer to the image data.
- unsigned int **GetImageDepth** () const

  Get the depth of the image.
- std::string **GetImageFormat** () const

  Get the string representation of the image format.
- virtual unsigned int **GetImageHeight** () const

  Get the height of the image.
- virtual unsigned int **GetImageWidth** () const

  Get the width of the image.
- bool **GetInitialized** () const

  Return true if the camera has been initialized.
- **common::Time GetLastRenderWallTime** ()

  Get the last time the camera was rendered.
- std::string **GetName** () const

  Get the camera's name.
- double **GetNearClip** ()

  Get the near clip distance.
- Ogre::Camera * **GetOgreCamera** () const

  Get a pointer to the ogre camera.
- Ogre::SceneNode * **GetPitchNode** () const

  Get the camera's pitch scene node.
- double **GetRenderRate** () const

  Get the render Hz rate.
- Ogre::Texture * **GetRenderTexture** () const

  Get the render texture.
- **math::Vector3 GetRight** ()

  Get the viewport right vector.
- **ScenePtr GetScene** () const

  Get the scene this camera is in.
- Ogre::SceneNode * **GetSceneNode** () const

  Get the camera's scene node.
- std::string **GetScreenshotPath** () const

  Get the path to saved screenshots.
- unsigned int **GetTextureHeight** () const

  Get the height of the off-screen render texture.
- unsigned int **GetTextureWidth** () const

  Get the width of the off-screen render texture.
- virtual unsigned int **GetTriangleCount** ()

  Get the triangle count.
- **math::Vector3 GetUp** ()

  Get the viewport up vector.
- **math::Angle GetVFOV** () const

  Get the camera FOV (vertical)
- Ogre::Viewport * **GetViewport** () const

Get a pointer to the Ogre::Viewport.

- unsigned int **GetViewportHeight** () const

  Get the viewport height in pixels.
- unsigned int **GetViewportWidth** () const

  Get the viewport width in pixels.
- unsigned int **GetWindowId** () const

  Get the ID of the window this camera is rendering into.
- bool **GetWorldPointOnPlane** (int _x, int _y, const **math::Plane** &_plane, **math::Vector3** &_result)

  Get point on a plane.
- **math::Pose GetWorldPose** ()

  Get the global pose of the camera.
- **math::Vector3 GetWorldPosition** () const

  Get the camera position in the world.
- **math::Quaternion GetWorldRotation** () const

  Get the camera's orientation in the world.
- double **GetZValue** (int _x, int _y)

  Get the Z-buffer value at the given image coordinate.
- virtual void **Init** ()

  Initialize the camera.
- bool **IsAnimating** () const

  Return true if the camera is moving due to an animation.
- bool **IsInitialized** () const **GAZEBO_DEPRECATED**(1.5)

  Deprecated.
- bool **IsVisible** (**VisualPtr** _visual)

  Return true if the visual is within the camera's view frustum.
- bool **IsVisible** (const std::string &_visualName)

  Return true if the visual is within the camera's view frustum.
- virtual void **Load** (**sdf::ElementPtr** _sdf)

  Load the camera with a set of parmeters.
- virtual void **Load** ()

  Load the camera with default parmeters.
- virtual bool **MoveToPosition** (const **math::Pose** &_pose, double _time)

  Move the camera to a position (this is an animated motion).
- bool **MoveToPositions** (const std::vector< **math::Pose** > &_pts, double _time, boost::function< void()> _onComplete=**NULL**)

  Move the camera to a series of poses (this is an animated motion).
- virtual void **PostRender** ()

  Post render.
- void **Render** ()

  Render the camera.
- void **RotatePitch** (**math::Angle** _angle)

  Rotate the camera around the pitch axis.
- void **RotateYaw** (**math::Angle** _angle)

  Rotate the camera around the yaw axis.
- bool **SaveFrame** (const std::string &_filename)

  Save the last frame to disk.
- void **SetAspectRatio** (float _ratio)

Set the aspect ratio.

- void **SetCaptureData** (bool _value)

  Set whether to capture data.

- void **SetCaptureDataOnce** ()

  Capture data once and save to disk.

- void **SetClipDist** (float _near, float _far)

  Set the clip distances.

- void **SetHFOV** (**math::Angle** _angle)

  Set the camera FOV (horizontal)

- void **SetImageHeight** (unsigned int _h)

  Set the image height.

- void **SetImageSize** (unsigned int _w, unsigned int _h)

  Set the image size.

- void **SetImageWidth** (unsigned int _w)

  Set the image height.

- void **SetName** (const std::string &_name)

  Set the camera's name.

- void **SetRenderRate** (double _hz)

  Set the render Hz rate.

- virtual void **SetRenderTarget** (Ogre::RenderTarget *_target)

  Set the camera's render target.

- void **SetSaveFramePathname** (const std::string &_pathname)

  Set the save frame pathname.

- void **SetScene** (**ScenePtr** _scene)

  Set the scene this camera is viewing.

- void **SetSceneNode** (Ogre::SceneNode *_node)

  Set the camera's scene node.

- void **SetWindowId** (unsigned int _windowId)

- virtual void **SetWorldPose** (const **math::Pose** &_pose)

  Set the global pose of the camera.

- void **SetWorldPosition** (const **math::Vector3** &_pos)

  Set the world position.

- void **SetWorldRotation** (const **math::Quaternion** &_quat)

  Set the world orientation.

- void **ShowWireframe** (bool _s)

  Set whether to view the world in wireframe.

- void **ToggleShowWireframe** ()

  Toggle whether to view the world in wireframe.

- void **TrackVisual** (const std::string &_visualName)

  Set the camera to track a scene node.

- void **Translate** (const **math::Vector3** &_direction)

  Translate the camera.

- virtual void **Update** ()

Static Public Member Functions

- static size_t **GetImageByteSize** (unsigned int _width, unsigned int _height, const std::string &_format)

  Calculate image byte size base on a few parameters.
- static bool **SaveFrame** (const unsigned char *_image, unsigned int _width, unsigned int _height, int _depth, const std::string &_format, const std::string &_filename)

  Save a frame using an image buffer.

Protected Member Functions

- virtual void **AnimationComplete** ()

  Internal function used to indicate that an animation has completed.
- virtual bool **AttachToVisualImpl** (const std::string &_name, bool _inheritOrientation, double _minDist=0, double _maxDist=0)

  Attach the camera to a scene node.
- virtual bool **AttachToVisualImpl** (**VisualPtr** _visual, bool _inheritOrientation, double _minDist=0, double _maxDist=0)

  Attach the camera to a visual.
- std::string **GetFrameFilename** ()

  Get the next frame filename based on SDF parameters.
- void **ReadPixelBuffer** ()

  Read image data from pixel buffer.
- virtual void **RenderImpl** ()

  Implementation of the render call.
- bool **TrackVisualImpl** (const std::string &_visualName)

  *Implementation of the **Camera::TrackVisual** (p. 189) call.*
- virtual bool **TrackVisualImpl** (**VisualPtr** _visual)

  Set the camera to track a scene node.

Protected Attributes

- Ogre::AnimationState * **animState**

  Animation state, used to animate the camera.
- unsigned char * **bayerFrameBuffer**

  Buffer for a bayer image frame.
- Ogre::Camera * **camera**

  The OGRE camera.
- bool **captureData**

  True to capture frames into an image buffer.
- bool **captureDataOnce**

  True to capture a frame once and save to disk.
- std::vector< **event::ConnectionPtr** > **connections**

  The camera's event connections.
- int **imageFormat**

  Format for saving images.
- int **imageHeight**

  Save image height.

- int **imageWidth**

  Save image width.
- bool **initialized**

  True if initialized.
- **common::Time lastRenderWallTime**

  Time the last frame was rendered.
- std::string **name**

  Name of the camera.
- bool **newData**

  True if new data is available.
- **event::EventT**< void(const unsigned char *, unsigned int, unsigned int, unsigned int, const std::string &)> **newImageFrame**

  Event triggered when a new frame is generated.
- boost::function< void()> **onAnimationComplete**

  User callback for when an animation completes.
- Ogre::SceneNode * **pitchNode**

  Scene *(p. 707) nod that controls camera pitch.*
- **common::Time prevAnimTime**

  Previous time the camera animation was updated.
- Ogre::RenderTarget * **renderTarget**

  Target that renders frames.
- Ogre::Texture * **renderTexture**

  Texture that receives results from rendering.
- std::list< msgs::Request > **requests**

  List of requests.
- unsigned int **saveCount**

  Number of saved frames.
- unsigned char * **saveFrameBuffer**
- **ScenePtr scene**

  Pointer to the scene.
- Ogre::SceneNode * **sceneNode**

  Scene *(p. 707) node that controls camera position.*
- std::string **screenshotPath**

  Path to saved screenshots.
- **sdf::ElementPtr sdf**

  Camera *(p. 166)'s SDF values.*
- unsigned int **textureHeight**

  Height of the render texture.
- unsigned int **textureWidth**

  Width of the render texture.
- Ogre::Viewport * **viewport**

  Viewport the ogre camera uses.
- unsigned int **windowId**

  ID of the window that the camera is attached to.

10.14.1 Detailed Description

Basic camera sensor.

This is the base class for all cameras.

10.14.2 Constructor & Destructor Documentation

10.14.2.1 gazebo::rendering::Camera::Camera (const std::string & *_namePrefix,* ScenePtr *_scene,* bool *_autoRender* = `true`)

Constructor.

Parameters

in	*_namePrefix*	Unique prefix name for the camera.
in	*_scene*	**Scene** (p. 707) that will contain the camera
in	*_autoRender*	Almost everyone should leave this as true.

10.14.2.2 virtual gazebo::rendering::Camera::~Camera ( ) [virtual]

Destructor.

10.14.3 Member Function Documentation

10.14.3.1 virtual void gazebo::rendering::Camera::AnimationComplete ( ) [protected], [virtual]

Internal function used to indicate that an animation has completed.

Reimplemented in **gazebo::rendering::UserCamera** (p. 867).

10.14.3.2 void gazebo::rendering::Camera::AttachToVisual (const std::string & *_visualName,* bool *_inheritOrientation,* double *_minDist* = `0.0`, double *_maxDist* = `0.0`)

Attach the camera to a scene node.

Parameters

in	*_visualName*	Name of the visual to attach the camera to
in	*_inherit-Orientation*	True means camera acquires the visual's orientation
in	*_minDist*	Minimum distance the camera is allowed to get to the visual
in	*_maxDist*	Maximum distance the camera is allowd to get from the visual

10.14.3.3 virtual bool gazebo::rendering::Camera::AttachToVisualImpl (const std::string & *_name,* bool *_inheritOrientation,* double *_minDist* = `0`, double *_maxDist* = `0`) [protected], [virtual]

Attach the camera to a scene node.

Parameters

in	*_visualName*	Name of the visual to attach the camera to
in	*_inherit-Orientation*	True means camera acquires the visual's orientation
in	*_minDist*	Minimum distance the camera is allowed to get to the visual
in	*_maxDist*	Maximum distance the camera is allowd to get from the visual

Returns

True on success

10.14.3.4 virtual bool gazebo::rendering::Camera::AttachToVisualImpl (VisualPtr *_visual,* bool *_inheritOrientation,* double *_minDist* = 0, double *_maxDist* = 0) [protected], [virtual]

Attach the camera to a visual.

Parameters

in	*_visual*	The visual to attach the camera to
in	*_inherit-Orientation*	True means camera acquires the visual's orientation
in	*_minDist*	Minimum distance the camera is allowed to get to the visual
in	*_maxDist*	Maximum distance the camera is allowd to get from the visual

Returns

True on success

Reimplemented in **gazebo::rendering::UserCamera** (p. 867).

10.14.3.5 template<typename T > event::ConnectionPtr gazebo::rendering::Camera::ConnectNewImageFrame (T *_subscriber*) [inline]

Connect to the new image signal.

Parameters

in	*_subscriber*	Callback that is called when a new image is generated

Returns

A pointer to the connection. This must be kept in scope.

References gazebo::event::EventT< T >::Connect(), and newImageFrame.

10.14.3.6 void gazebo::rendering::Camera::CreateRenderTexture (const std::string & *_textureName*)

Set the render target.

Parameters

in	*_textureName*	Name of the new render texture

10.14.3.7 void gazebo::rendering::Camera::DisconnectNewImageFrame (event::ConnectionPtr & *_c*) [inline]

Disconnect from an image frame.

Parameters

in	*_c*	The connection to disconnect

References gazebo::event::EventT< T >::Disconnect(), and newImageFrame.

10.14.3.8 void gazebo::rendering::Camera::EnableSaveFrame (bool *_enable*)

Enable or disable saving.

Parameters

in	*_enable*	Set to True to enable saving of frames

10.14.3.9 virtual void gazebo::rendering::Camera::Fini ( ) [virtual]

Finalize the camera.

This function is called before the camera is destructed

Reimplemented in **gazebo::rendering::GpuLaser** (p. 339), **gazebo::rendering::DepthCamera** (p. 264), and **gazebo::rendering::UserCamera** (p. 868).

10.14.3.10 float gazebo::rendering::Camera::GetAspectRatio ( ) const

Get the apect ratio.

Returns

The aspect ratio (width / height) in pixels

10.14.3.11 virtual float gazebo::rendering::Camera::GetAvgFPS ( ) [inline], [virtual]

Get the average FPS.

Returns

The average frames per second

10.14.3.12 void gazebo::rendering::Camera::GetCameraToViewportRay (int *_screenx*, int *_screeny*, math::Vector3 & *_origin*, math::Vector3 & *_dir*)

Get a world space ray as cast from the camera through the viewport.

Parameters

in	*_screenx*	X coordinate in the camera's viewport, in pixels.
in	*_screeny*	Y coordinate in the camera's viewport, in pixels.
out	*_origin*	Origin in the world coordinate frame of the resulting ray
out	*_dir*	Direction of the resulting ray

10.14.3.13 math::Vector3 gazebo::rendering::Camera::GetDirection ( ) const

Get the camera's direction vector.

Returns

Direction the camera is facing

10.14.3.14 double gazebo::rendering::Camera::GetFarClip ( )

Get the far clip distance.

Returns

Far clip distance

10.14.3.15 std::string gazebo::rendering::Camera::GetFrameFilename ( ) [protected]

Get the next frame filename based on SDF parameters.

Returns

The frame's filename

10.14.3.16 math::Angle gazebo::rendering::Camera::GetHFOV ( ) const

Get the camera FOV (horizontal)

Returns

The horizontal field of view

10.14.3.17 size_t gazebo::rendering::Camera::GetImageByteSize ( ) const

Get the image size in bytes.

Returns

Size in bytes

10.14.3.18 static size_t gazebo::rendering::Camera::GetImageByteSize (unsigned int *_width*, unsigned int *_height*, const std::string & *_format*) [static]

Calculate image byte size base on a few parameters.

Parameters

in	*_width*	Width of an image
in	*_height*	Height of an image
in	*_format*	Image format

Returns

Size of an image based on the parameters

10.14.3.19 virtual const unsigned char* gazebo::rendering::Camera::GetImageData (unsigned int *i* = 0) [virtual]

Get a pointer to the image data.

Get the raw image data from a camera's buffer.

Parameters

in	*_i*	Index of the camera's texture (0 = RGB, 1 = depth).

Returns

Pointer to the raw data, null if data is not available.

10.14.3.20 unsigned int gazebo::rendering::Camera::GetImageDepth ( ) const

Get the depth of the image.

Returns

Depth of the image

10.14.3.21 std::string gazebo::rendering::Camera::GetImageFormat ( ) const

Get the string representation of the image format.

Returns

String representation of the image format.

10.14.3.22 virtual unsigned int gazebo::rendering::Camera::GetImageHeight ( ) const [virtual]

Get the height of the image.

Returns

Image height

Reimplemented in **gazebo::rendering::UserCamera** (p. 868).

10.14.3.23 virtual unsigned int gazebo::rendering::Camera::GetImageWidth ( ) const [virtual]

Get the width of the image.

Returns

Image width

Reimplemented in **gazebo::rendering::UserCamera** (p. 868).

10.14.3.24 bool gazebo::rendering::Camera::GetInitialized ( ) const

Return true if the camera has been initialized.

Returns

True if initialized was successful

10.14.3.25 common::Time gazebo::rendering::Camera::GetLastRenderWallTime ( )

Get the last time the camera was rendered.

Returns

Time the camera was last rendered

10.14.3.26 std::string gazebo::rendering::Camera::GetName ( ) const

Get the camera's name.

Returns

The name of the camera

10.14.3.27 double gazebo::rendering::Camera::GetNearClip ( )

Get the near clip distance.

Returns

Near clip distance

10.14.3.28 Ogre::Camera* gazebo::rendering::Camera::GetOgreCamera ( ) const

Get a pointer to the ogre camera.

Returns

Pointer to the OGRE camera

10.14.3.29 Ogre::SceneNode* gazebo::rendering::Camera::GetPitchNode ( ) const

Get the camera's pitch scene node.

Returns

The pitch node the camera is attached to

10.14.3.30 double gazebo::rendering::Camera::GetRenderRate ( ) const

Get the render Hz rate.

Returns

The Hz rate

10.14.3.31 Ogre::Texture* gazebo::rendering::Camera::GetRenderTexture ( ) const

Get the render texture.

Returns

Pointer to the render texture

10.14.3.32 math::Vector3 gazebo::rendering::Camera::GetRight ( )

Get the viewport right vector.

Returns

The viewport right vector

10.14.3.33 ScenePtr gazebo::rendering::Camera::GetScene ( ) const

Get the scene this camera is in.

Returns

Pointer to scene containing this camera

10.14.3.34 Ogre::SceneNode* gazebo::rendering::Camera::GetSceneNode ( ) const

Get the camera's scene node.

Returns

The scene node the camera is attached to

10.14.3.35 std::string gazebo::rendering::Camera::GetScreenshotPath ( ) const

Get the path to saved screenshots.

Returns

Path to saved screenshots.

10.14.3.36 unsigned int gazebo::rendering::Camera::GetTextureHeight ( ) const

Get the height of the off-screen render texture.

Returns

Render texture height

10.14.3.37 unsigned int gazebo::rendering::Camera::GetTextureWidth ( ) const

Get the width of the off-screen render texture.

Returns

Render texture width

10.14.3.38 virtual unsigned int gazebo::rendering::Camera::GetTriangleCount ( ) [inline], [virtual]

Get the triangle count.

Returns

The current triangle count

10.14.3.39 math::Vector3 gazebo::rendering::Camera::GetUp ( )

Get the viewport up vector.

Returns

The viewport up vector

10.14.3.40 math::Angle gazebo::rendering::Camera::GetVFOV ( ) const

Get the camera FOV (vertical)

Returns

The vertical field of view

10.14.3.41 Ogre::Viewport* gazebo::rendering::Camera::GetViewport ( ) const

Get a pointer to the Ogre::Viewport.

Returns

Pointer to the Ogre::Viewport

10.14.3.42 unsigned int gazebo::rendering::Camera::GetViewportHeight ( ) const

Get the viewport height in pixels.

Returns

The viewport height

10.14.3.43 unsigned int gazebo::rendering::Camera::GetViewportWidth ( ) const

Get the viewport width in pixels.

Returns

The viewport width

10.14.3.44 unsigned int gazebo::rendering::Camera::GetWindowId ( ) const

Get the ID of the window this camera is rendering into.

Returns

The ID of the window.

10.14.3.45 bool gazebo::rendering::Camera::GetWorldPointOnPlane (int *_x*, int *_y*, const math::Plane & *_plane*, math::Vector3 & *_result*)

Get point on a plane.

Parameters

in	*_x*	X cooridnate in camera's viewport, in pixels
in	*_y*	Y cooridnate in camera's viewport, in pixels
in	*_plane*	Plane on which to find the intersecting point
out	*_result*	Point on the plane

Returns

True if a valid point was found

10.14.3.46 math::Pose gazebo::rendering::Camera::GetWorldPose ( )

Get the global pose of the camera.

Returns

Pose of the camera in the world coordinate frame

10.14.3.47 math::Vector3 gazebo::rendering::Camera::GetWorldPosition ( ) const

Get the camera position in the world.

Returns

The world position of the camera

10.14.3.48 math::Quaternion gazebo::rendering::Camera::GetWorldRotation ( ) const

Get the camera's orientation in the world.

Returns

The camera's orientation as a **math::Quaternion** (p. 654)

10.14.3.49 double gazebo::rendering::Camera::GetZValue (int _x, int _y)

Get the Z-buffer value at the given image coordinate.

Parameters

in	_x	Image coordinate; (0, 0) specifies the top-left corner.
in	_y	Image coordinate; (0, 0) specifies the top-left corner.

Returns

Image z value; note that this is abitrarily scaled and is *not* the same as the depth value.

10.14.3.50 virtual void gazebo::rendering::Camera::Init ( ) [virtual]

Initialize the camera.

Reimplemented in **gazebo::rendering::GpuLaser** (p. 341), **gazebo::rendering::DepthCamera** (p. 264), and **gazebo::rendering::UserCamera** (p. 870).

10.14.3.51 bool gazebo::rendering::Camera::IsAnimating ( ) const

Return true if the camera is moving due to an animation.

10.14.3.52 bool gazebo::rendering::Camera::IsInitialized ( ) const [inline]

Deprecated.

See Also

GetInitialized (p. 178)

10.14.3.53 bool gazebo::rendering::Camera::IsVisible (VisualPtr *_visual*)

Return true if the visual is within the camera's view frustum.

Parameters

in	*_visual*	The visual to check for visibility

Returns

True if the _visual is in the camera's frustum

10.14.3.54 bool gazebo::rendering::Camera::IsVisible (const std::string & *_visualName*)

Return true if the visual is within the camera's view frustum.

Parameters

in	*_visualName*	Name of the visual to check for visibility

Returns

True if the _visual is in the camera's frustum

10.14.3.55 virtual void gazebo::rendering::Camera::Load (sdf::ElementPtr *_sdf*) [virtual]

Load the camera with a set of parmeters.

Parameters

in	*_sdf*	The SDF camera info

Reimplemented in **gazebo::rendering::UserCamera** (p. 870).

10.14.3.56 virtual void gazebo::rendering::Camera::Load ( ) [virtual]

Load the camera with default parmeters.

Reimplemented in **gazebo::rendering::GpuLaser** (p. 342), **gazebo::rendering::DepthCamera** (p. 264), and **gazebo::rendering::UserCamera** (p. 870).

10.14.3.57 virtual bool gazebo::rendering::Camera::MoveToPosition (const math::Pose & *_pose,* double *_time*) [virtual]

Move the camera to a position (this is an animated motion).

See Also

Camera::MoveToPositions (p. 184)

Parameters

in	*_pose*	End position of the camera
in	*_time*	Duration of the camera's movement

Reimplemented in **gazebo::rendering::UserCamera** (p. 870).

10.14.3.58 bool gazebo::rendering::Camera::MoveToPositions (const std::vector< math::Pose > & *_pts,* double *_time,* boost::function< void() > *_onComplete* = NULL)

Move the camera to a series of poses (this is an animated motion).

See Also

Camera::MoveToPosition (p. 184)

Parameters

in	*_pts*	Vector of poses to move to
in	*_time*	Duration of the entire move
in	*_onComplete*	Callback that is called when the move is complete

10.14.3.59 virtual void gazebo::rendering::Camera::PostRender ( ) [virtual]

Post render.

Called afer the render signal.

Reimplemented in **gazebo::rendering::GpuLaser** (p. 342), **gazebo::rendering::DepthCamera** (p. 264), and **gazebo::rendering::UserCamera** (p. 871).

10.14.3.60 void gazebo::rendering::Camera::ReadPixelBuffer ( ) [protected]

Read image data from pixel buffer.

10.14.3.61 void gazebo::rendering::Camera::Render ( )

Render the camera.

Called after the pre-render signal. This function will generate camera images

10.14.3.62 virtual void gazebo::rendering::Camera::RenderImpl ( ) [protected], [virtual]

Implementation of the render call.

10.14.3.63 void gazebo::rendering::Camera::RotatePitch (math::Angle _angle)

Rotate the camera around the pitch axis.

Parameters

in	*_angle*	Pitch amount

10.14.3.64 void gazebo::rendering::Camera::RotateYaw (math::Angle _angle)

Rotate the camera around the yaw axis.

Parameters

in	*_angle*	Rotation amount

10.14.3.65 bool gazebo::rendering::Camera::SaveFrame (const std::string & _filename)

Save the last frame to disk.

Parameters

in	*_filename*	File in which to save a single frame

Returns

True if saving was successful

10.14.3.66 static bool gazebo::rendering::Camera::SaveFrame (const unsigned char * _image, unsigned int _width, unsigned int _height, int _depth, const std::string & _format, const std::string & _filename) [static]

Save a frame using an image buffer.

Parameters

in	*_image*	The raw image buffer
in	*_width*	Width of the image
in	*_height*	Height of the image
in	*_depth*	Depth of the image data
in	*_format*	Format the image data is in
in	*_filename*	Name of the file in which to write the frame

Returns

True if saving was successful

10.14.3.67 void gazebo::rendering::Camera::SetAspectRatio (float _ratio)

Set the aspect ratio.

Parameters

in	*_ratio*	The aspect ratio (width / height) in pixels

10.14.3.68 void gazebo::rendering::Camera::SetCaptureData (bool _value)

Set whether to capture data.

Parameters

in	*_value*	Set to true to capture data into a memory buffer.

10.14.3.69 void gazebo::rendering::Camera::SetCaptureDataOnce ( )

Capture data once and save to disk.

10.14.3.70 void gazebo::rendering::Camera::SetClipDist (float _near, float _far)

Set the clip distances.

Parameters

in	*_near*	Near clip distance in meters
in	*_far*	Far clip distance in meters

10.14.3.71 void gazebo::rendering::Camera::SetHFOV (math::Angle _angle)

Set the camera FOV (horizontal)

Parameters

in	*_radians*	Horizontal field of view

10.14.3.72 void gazebo::rendering::Camera::SetImageHeight (unsigned int _h)

Set the image height.

Parameters

in	*_h*	Image height

10.14.3.73 void gazebo::rendering::Camera::SetImageSize (unsigned int _w, unsigned int _h)

Set the image size.

Parameters

in	*_w*	Image width
in	*_h*	Image height

10.14.3.74 void gazebo::rendering::Camera::SetImageWidth (unsigned int *_w*)

Set the image height.

Parameters

in	*_w*	Image width

10.14.3.75 void gazebo::rendering::Camera::SetName (const std::string & *_name*)

Set the camera's name.

Parameters

in	*_name*	New name for the camera

10.14.3.76 void gazebo::rendering::Camera::SetRenderRate (double *_hz*)

Set the render Hz rate.

Parameters

in	*_hz*	The Hz rate

10.14.3.77 virtual void gazebo::rendering::Camera::SetRenderTarget (Ogre::RenderTarget * *_target*) [virtual]

Set the camera's render target.

Parameters

in	*_target*	Pointer to the render target

Reimplemented in **gazebo::rendering::UserCamera** (p. 871).

10.14.3.78 void gazebo::rendering::Camera::SetSaveFramePathname (const std::string & *_pathname*)

Set the save frame pathname.

Parameters

in	*_pathname*	Directory in which to store saved image frames

10.14.3.79 void gazebo::rendering::Camera::SetScene (ScenePtr _scene)

Set the scene this camera is viewing.

Parameters

in	*_scene*	Pointer to the scene

10.14.3.80 void gazebo::rendering::Camera::SetSceneNode (Ogre::SceneNode * _node)

Set the camera's scene node.

Parameters

in	*_node*	The scene nodes to attach the camera to

10.14.3.81 void gazebo::rendering::Camera::SetWindowId (unsigned int _windowId)

10.14.3.82 virtual void gazebo::rendering::Camera::SetWorldPose (const math::Pose & _pose) [virtual]

Set the global pose of the camera.

Parameters

in	*_pose*	The new **math::Pose** (p. 626) of the camera

Reimplemented in **gazebo::rendering::UserCamera** (p. 872).

10.14.3.83 void gazebo::rendering::Camera::SetWorldPosition (const math::Vector3 & _pos)

Set the world position.

Parameters

in	*_pos*	The new position of the camera

10.14.3.84 void gazebo::rendering::Camera::SetWorldRotation (const math::Quaternion & _quat)

Set the world orientation.

Parameters

in	*_quat*	The new orientation of the camera

10.14.3.85 void gazebo::rendering::Camera::ShowWireframe (bool _s)

Set whether to view the world in wireframe.

Parameters

in	*_s*	Set to True to render objects as wireframe

10.14.3.86 void gazebo::rendering::Camera::ToggleShowWireframe ( )

Toggle whether to view the world in wireframe.

10.14.3.87 void gazebo::rendering::Camera::TrackVisual (const std::string & *_visualName*)

Set the camera to track a scene node.

Parameters

in	*_visualName*	Name of the visual to track

10.14.3.88 bool gazebo::rendering::Camera::TrackVisualImpl (const std::string & *_visualName*) [protected]

Implementation of the **Camera::TrackVisual** (p. 189) call.

Parameters

in	*_visualName*	Name of the visual to track

Returns

True if able to track the visual

10.14.3.89 virtual bool gazebo::rendering::Camera::TrackVisualImpl (VisualPtr *_visual*) [protected], [virtual]

Set the camera to track a scene node.

Parameters

in	*_visual*	The visual to track

Returns

True if able to track the visual

Reimplemented in **gazebo::rendering::UserCamera** (p. 872).

10.14.3.90 void gazebo::rendering::Camera::Translate (const math::Vector3 & *_direction*)

Translate the camera.

Parameters

in	*_direction*	The translation vector

10.14.3.91 virtual void gazebo::rendering::Camera::Update ( ) [virtual]

Reimplemented in **gazebo::rendering::UserCamera** (p. 873).

10.14.4 Member Data Documentation

10.14.4.1 Ogre::AnimationState* gazebo::rendering::Camera::animState [protected]

Animation state, used to animate the camera.

10.14.4.2 unsigned char* gazebo::rendering::Camera::bayerFrameBuffer [protected]

Buffer for a bayer image frame.

10.14.4.3 Ogre::Camera* gazebo::rendering::Camera::camera [protected]

The OGRE camera.

10.14.4.4 bool gazebo::rendering::Camera::captureData [protected]

True to capture frames into an image buffer.

10.14.4.5 bool gazebo::rendering::Camera::captureDataOnce [protected]

True to capture a frame once and save to disk.

10.14.4.6 std::vector<event::ConnectionPtr> gazebo::rendering::Camera::connections [protected]

The camera's event connections.

10.14.4.7 int gazebo::rendering::Camera::imageFormat [protected]

Format for saving images.

10.14.4.8 int gazebo::rendering::Camera::imageHeight [protected]

Save image height.

10.14.4.9 int gazebo::rendering::Camera::imageWidth [protected]

Save image width.

10.14.4.10 bool gazebo::rendering::Camera::initialized [protected]

True if initialized.

10.14.4.11 common::Time gazebo::rendering::Camera::lastRenderWallTime [protected]

Time the last frame was rendered.

10.14.4.12 std::string gazebo::rendering::Camera::name [protected]

Name of the camera.

10.14.4.13 bool gazebo::rendering::Camera::newData [protected]

True if new data is available.

10.14.4.14 event::EventT<void(const unsigned char *, unsigned int, unsigned int, unsigned int, const std::string &)> gazebo::rendering::Camera::newImageFrame [protected]

Event triggered when a new frame is generated.

Referenced by ConnectNewImageFrame(), and DisconnectNewImageFrame().

10.14.4.15 boost::function<void()> gazebo::rendering::Camera::onAnimationComplete [protected]

User callback for when an animation completes.

10.14.4.16 Ogre::SceneNode* gazebo::rendering::Camera::pitchNode [protected]

Scene (p. 707) nod that controls camera pitch.

10.14.4.17 common::Time gazebo::rendering::Camera::prevAnimTime [protected]

Previous time the camera animation was updated.

10.14.4.18 Ogre::RenderTarget* gazebo::rendering::Camera::renderTarget [protected]

Target that renders frames.

10.14.4.19 Ogre::Texture* gazebo::rendering::Camera::renderTexture [protected]

Texture that receives results from rendering.

10.14.4.20 std::list<msgs::Request> gazebo::rendering::Camera::requests [protected]

List of requests.

10.14.4.21 unsigned int gazebo::rendering::Camera::saveCount [protected]

Number of saved frames.

10.14.4.22 unsigned char* gazebo::rendering::Camera::saveFrameBuffer [protected]

10.14.4.23 ScenePtr gazebo::rendering::Camera::scene [protected]

Pointer to the scene.

10.14.4.24 Ogre::SceneNode* gazebo::rendering::Camera::sceneNode [protected]

Scene (p. 707) node that controls camera position.

10.14.4.25 std::string gazebo::rendering::Camera::screenshotPath [protected]

Path to saved screenshots.

10.14.4.26 sdf::ElementPtr gazebo::rendering::Camera::sdf [protected]

Camera (p. 166)'s SDF values.

10.14.4.27 unsigned int gazebo::rendering::Camera::textureHeight [protected]

Height of the render texture.

10.14.4.28 unsigned int gazebo::rendering::Camera::textureWidth [protected]

Width of the render texture.

10.14.4.29 Ogre::Viewport* gazebo::rendering::Camera::viewport [protected]

Viewport the ogre camera uses.

10.14.4.30 unsigned int gazebo::rendering::Camera::windowId [protected]

ID of the window that the camera is attached to.

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

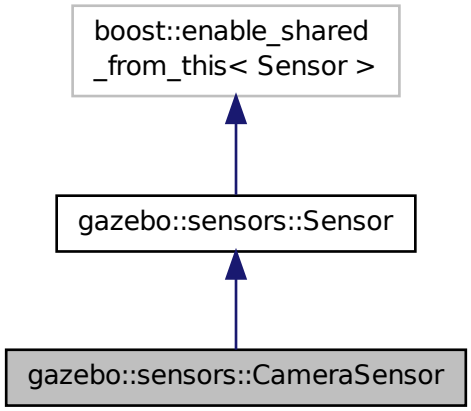
- **Camera.hh**

10.15 gazebo::sensors::CameraSensor Class Reference

Basic camera sensor.

`#include <sensors/sensors.hh>`

Inheritance diagram for gazebo::sensors::CameraSensor:



Public Member Functions

- **CameraSensor** ()
  Constructor.
- virtual ~**CameraSensor** ()
  Destructor.
- **rendering::CameraPtr GetCamera** () const
  *Returns a pointer to the **rendering::Camera*** (p. *166*).
- const unsigned char * **GetImageData** ()
  Gets the raw image data from the sensor.
- unsigned int **GetImageHeight** () const
  Gets the height of the image in pixels.
- unsigned int **GetImageWidth** () const
  Gets the width of the image in pixels.
- virtual std::string **GetTopic** () const
  Gets the topic name of the sensor.
- virtual void **Init** ()
  Initialize the camera.
- virtual bool **IsActive** ()
  Returns true if sensor generation is active.
- virtual void **Load** (const std::string &_worldName, **sdf::ElementPtr** _sdf)
  Load the sensor with SDF parameters.
- virtual void **Load** (const std::string &_worldName)
  Load the sensor with default parameters.
- bool **SaveFrame** (const std::string &_filename)
  Saves the image to the disk.
- virtual void **SetParent** (const std::string &_name)
  Set the parent of the sensor.

Protected Member Functions

- virtual void **Fini** ()

  Finalize the camera.
- virtual void **UpdateImpl** (bool _force)

  Update the sensor information.

Additional Inherited Members

10.15.1 Detailed Description

Basic camera sensor.

This sensor is used for simulating standard monocular cameras

10.15.2 Constructor & Destructor Documentation

10.15.2.1 gazebo::sensors::CameraSensor::CameraSensor ( )

Constructor.

10.15.2.2 virtual gazebo::sensors::CameraSensor::~CameraSensor ( ) [virtual]

Destructor.

10.15.3 Member Function Documentation

10.15.3.1 virtual void gazebo::sensors::CameraSensor::Fini ( ) [protected], [virtual]

Finalize the camera.

Reimplemented from **gazebo::sensors::Sensor** (p. 735).

10.15.3.2 rendering::CameraPtr gazebo::sensors::CameraSensor::GetCamera ( ) const [inline]

Returns a pointer to the **rendering::Camera** (p. 166).

Returns

The Pointer to the camera sensor.

10.15.3.3 const unsigned char* gazebo::sensors::CameraSensor::GetImageData ( )

Gets the raw image data from the sensor.

Returns

The pointer to the image data array.

10.15.3.4 unsigned int gazebo::sensors::CameraSensor::GetImageHeight ( ) const

Gets the height of the image in pixels.

Returns

The image height in pixels.

10.15.3.5 unsigned int gazebo::sensors::CameraSensor::GetImageWidth ( ) const

Gets the width of the image in pixels.

Returns

The image width in pixels.

10.15.3.6 virtual std::string gazebo::sensors::CameraSensor::GetTopic ( ) const [virtual]

Gets the topic name of the sensor.

Returns

Topic name

Todo to be implemented

Reimplemented from **gazebo::sensors::Sensor** (p. 737).

10.15.3.7 virtual void gazebo::sensors::CameraSensor::Init ( ) [virtual]

Initialize the camera.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.15.3.8 virtual bool gazebo::sensors::CameraSensor::IsActive ( ) [virtual]

Returns true if sensor generation is active.

Returns

True if active, false if not.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.15.3.9 virtual void gazebo::sensors::CameraSensor::Load (const std::string & *_worldName,* **sdf::ElementPtr** *_sdf*) [virtual]

Load the sensor with SDF parameters.

Parameters

in	*_sdf*	SDF **Sensor** (p. 731) parameters
in	*_worldName*	Name of world to load from

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.15.3.10 virtual void gazebo::sensors::CameraSensor::Load (const std::string & *_worldName*) [virtual]

Load the sensor with default parameters.

Parameters

in	*_worldName*	Name of world to load from

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.15.3.11 bool gazebo::sensors::CameraSensor::SaveFrame (const std::string & *_filename*)

Saves the image to the disk.

Parameters

in	*_filename*	The name of the file to be saved.

Returns

True if successful, false if unsuccessful.

10.15.3.12 virtual void gazebo::sensors::CameraSensor::SetParent (const std::string & *_name*) [virtual]

Set the parent of the sensor.

Parameters

_name	The name of the parent

Reimplemented from **gazebo::sensors::Sensor** (p. 739).

10.15.3.13 virtual void gazebo::sensors::CameraSensor::UpdateImpl (bool *_force*) [protected], [virtual]

Update the sensor information.

Parameters

in	*_force*	True if update is forced, false if not

Reimplemented from **gazebo::sensors::Sensor** (p. 739).

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

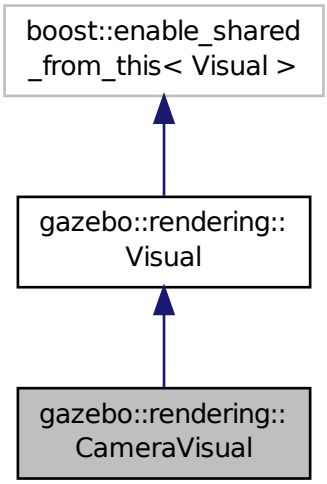
- **CameraSensor.hh**

10.16 gazebo::rendering::CameraVisual Class Reference

Basic camera visualization.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::CameraVisual:



Public Member Functions

- **CameraVisual** (const std::string &_name, **VisualPtr** _vis)

  Constructor.
- virtual ~**CameraVisual** ()

  Destructor.
- void **Load** (unsigned int _width, unsigned int _height)

  *Load the **Visual** (*p. *920).*

Additional Inherited Members

10.16.1 Detailed Description

Basic camera visualization.

This class is used to visualize a camera image generated from a CameraSensor. The sensor's image is drawn on a billboard in the 3D environment.

10.16.2 Constructor & Destructor Documentation

10.16.2.1 gazebo::rendering::CameraVisual::CameraVisual (const std::string & *_name*, VisualPtr *_vis*)

Constructor.

Parameters

in	*_name*	Name of the **Visual** (p. 920)
in	*_vis*	Pointer to the parent **Visual** (p. 920)

10.16.2.2 virtual gazebo::rendering::CameraVisual::~CameraVisual ( ) [virtual]

Destructor.

10.16.3 Member Function Documentation

10.16.3.1 void gazebo::rendering::CameraVisual::Load (unsigned int *_width*, unsigned int *_height*)

Load the **Visual** (p. 920).

Parameters

in	*_width*	Width of the **Camera** (p. 166) image
in	*_height*	Height of the **Camera** (p. 166) image

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

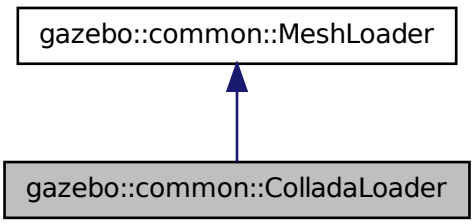
- **CameraVisual.hh**

10.17 gazebo::common::ColladaLoader Class Reference

Class used to load Collada mesh files.

`#include <common/common.hh>`

Inheritance diagram for gazebo::common::ColladaLoader:



Public Member Functions

- **ColladaLoader** ()

  Constructor.
- virtual ∼**ColladaLoader** ()

  Destructor.
- virtual **Mesh** ∗ **Load** (const std::string &_filename)

  Load a mesh.

10.17.1 Detailed Description

Class used to load Collada mesh files.

10.17.2 Constructor & Destructor Documentation

10.17.2.1 gazebo::common::ColladaLoader::ColladaLoader ( )

Constructor.

10.17.2.2 virtual gazebo::common::ColladaLoader::∼ColladaLoader ( ) [virtual]

Destructor.

10.17.3 Member Function Documentation

10.17.3.1 virtual Mesh∗ gazebo::common::ColladaLoader::Load (const std::string & *_filename*) [virtual]

Load a mesh.

Parameters

in	*_filename*	Collada file to load

Returns

Pointer to a new **Mesh** (p. 498)

Implements **gazebo::common::MeshLoader** (p. 507).

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

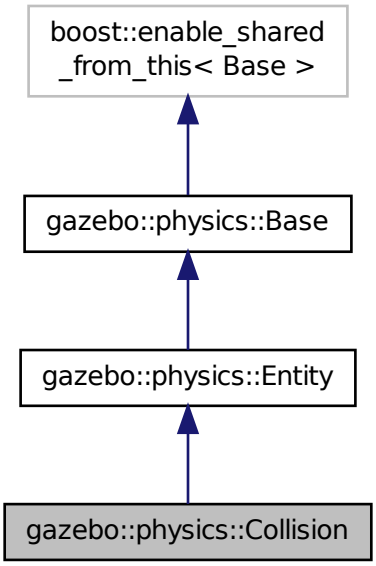
- **ColladaLoader.hh**

10.18 gazebo::physics::Collision Class Reference

Base (p. 141) class for all collision entities.

`#include <Collision.hh>`

Inheritance diagram for gazebo::physics::Collision:



Public Member Functions

- **Collision** (**LinkPtr** _link)

  Constructor.
- virtual ~**Collision** ()

  Destructor.
- void **AddContact** (const **Contact** &_contact)

  Add an occurance of a contact to this collision.
- template<typename T >
  event::ConnectionPtr ConnectContact (T _subscriber)

  Deprecated.
- void **DisconnectContact** (**event::ConnectionPtr** &_conn)

  Deprecated.
- void **FillMsg** (msgs::Collision &_msg)

  Fill a collision message.
- virtual void **Fini** ()

  Finalize the collision.
- virtual **math::Box GetBoundingBox** () const =0

  Get the bounding box for this collision.
- bool **GetContactsEnabled** () const

  Return true of contacts are on.
- float **GetLaserRetro** () const

Get the laser retro reflectiveness.

- **LinkPtr GetLink** () const

  Get the link this collision belongs to.
- virtual int **GetMaxContacts** ()

  returns number of contacts allowed for this collision.
- **ModelPtr GetModel** () const

  Get the model this collision belongs to.
- virtual **math::Vector3 GetRelativeAngularAccel** () const

  Get the angular acceleration of the collision.
- virtual **math::Vector3 GetRelativeAngularVel** () const

  Get the angular velocity of the collision.
- virtual **math::Vector3 GetRelativeLinearAccel** () const

  Get the linear acceleration of the collision.
- virtual **math::Vector3 GetRelativeLinearVel** () const

  Get the linear velocity of the collision.
- **ShapePtr GetShape** () const

  Get the collision shape.
- unsigned int **GetShapeType** ()

  Get the shape type.
- **CollisionState GetState** ()

  Get the collision state.
- **SurfaceParamsPtr GetSurface** () const

  Get the surface parameters.
- virtual **math::Vector3 GetWorldAngularAccel** () const

  Get the angular acceleration of the collision in the world frame.
- virtual **math::Vector3 GetWorldAngularVel** () const

  Get the angular velocity of the collision in the world frame.
- virtual **math::Vector3 GetWorldLinearAccel** () const

  Get the linear acceleration of the collision in the world frame.
- virtual **math::Vector3 GetWorldLinearVel** () const

  Get the linear velocity of the collision in the world frame.
- virtual void **Init** ()

  Initialize the collision.
- bool **IsPlaceable** () const

  Return whether this collision is movable.
- virtual void **Load** (**sdf::ElementPtr** _sdf)

  Load the collision.
- void **ProcessMsg** (const msgs::Collision &_msg)

  Update parameters from a message.
- virtual void **SetCategoryBits** (unsigned int _bits)=0

  Set the category bits, used during collision detection.
- virtual void **SetCollideBits** (unsigned int _bits)=0

  Set the collide bits, used during collision detection.
- void **SetCollision** (bool _placeable)

  Set the encapsulated collsion object.
- void **SetContactsEnabled** (bool _enable)

  Turn contact recording on or off.

- void **SetLaserRetro** (float _retro)

  Set the laser retro reflectiveness.
- virtual void **SetMaxContacts** (double _maxContacts)

  Number of contacts allowed for this collision.
- void **SetShape** (**ShapePtr** _shape)

  Set the shape for this collision.
- void **SetState** (const **CollisionState** &_state)

  Set the current collision state.
- virtual void **UpdateParameters** (**sdf::ElementPtr** _sdf)

  Update the parameters using new sdf values.

Protected Attributes

- **LinkPtr link**

  The link this collision belongs to.
- bool **placeable**

  Flag for placeable.
- **ShapePtr shape**

  *Pointer to **physics::Shape** (p. 754).*

Additional Inherited Members

10.18.1 Detailed Description

Base (p. 141) class for all collision entities.

10.18.2 Constructor & Destructor Documentation

10.18.2.1 gazebo::physics::Collision::Collision (LinkPtr _link) [explicit]

Constructor.

Parameters

in	*_link*	**Link** (p. 439) that contains this collision object.

10.18.2.2 virtual gazebo::physics::Collision::~Collision ( ) [virtual]

Destructor.

10.18.3 Member Function Documentation

10.18.3.1 void gazebo::physics::Collision::AddContact (const Contact & _contact)

Add an occurance of a contact to this collision.

Parameters

in	*_contact*	The contact which was detected by a collision engine.

10.18.3.2 template<typename T > event::ConnectionPtr gazebo::physics::Collision::ConnectContact (T *_subscriber*) [inline]

Deprecated.

References gazebo::event::EventT< T >::Connect().

10.18.3.3 void gazebo::physics::Collision::DisconnectContact (event::ConnectionPtr & *_conn*) [inline]

Deprecated.

References gazebo::event::EventT< T >::Disconnect().

10.18.3.4 void gazebo::physics::Collision::FillMsg (msgs::Collision & *_msg*)

Fill a collision message.

Parameters

out	*_msg*	The message to fill with this collision's data.

10.18.3.5 virtual void gazebo::physics::Collision::Fini ( ) [virtual]

Finalize the collision.

Reimplemented from **gazebo::physics::Entity** (p. 291).

10.18.3.6 virtual math::Box gazebo::physics::Collision::GetBoundingBox ( ) const [pure virtual]

Get the bounding box for this collision.

Returns

The bounding box.

Reimplemented from **gazebo::physics::Entity** (p. 291).

10.18.3.7 bool gazebo::physics::Collision::GetContactsEnabled ( ) const

Return true of contacts are on.

Returns

True of contact are on.

10.18.3.8 float gazebo::physics::Collision::GetLaserRetro ( ) const

Get the laser retro reflectiveness.

Returns

The laser retro value.

10.18.3.9 LinkPtr gazebo::physics::Collision::GetLink ( ) const

Get the link this collision belongs to.

Returns

The parent **Link** (p. 439).

10.18.3.10 virtual int gazebo::physics::Collision::GetMaxContacts ( ) [virtual]

returns number of contacts allowed for this collision.

This overrides global value (in **PhysicsEngine** (p. 597)) if specified.

Returns

max num contacts allowed for this collision.

10.18.3.11 ModelPtr gazebo::physics::Collision::GetModel ( ) const

Get the model this collision belongs to.

Returns

The parent model.

10.18.3.12 virtual math::Vector3 gazebo::physics::Collision::GetRelativeAngularAccel ( ) const [virtual]

Get the angular acceleration of the collision.

Returns

The angular acceleration of the collision.

Reimplemented from **gazebo::physics::Entity** (p. 293).

10.18.3.13 virtual math::Vector3 gazebo::physics::Collision::GetRelativeAngularVel ( ) const [virtual]

Get the angular velocity of the collision.

Returns

The angular velocity of the collision.

Reimplemented from **gazebo::physics::Entity** (p. 293).

10.18.3.14 virtual math::Vector3 gazebo::physics::Collision::GetRelativeLinearAccel ( ) const [virtual]

Get the linear acceleration of the collision.

Returns

The linear acceleration of the collision.

Reimplemented from **gazebo::physics::Entity** (p. 293).

10.18.3.15 virtual math::Vector3 gazebo::physics::Collision::GetRelativeLinearVel ( ) const [virtual]

Get the linear velocity of the collision.

Returns

The linear velocity relative to the parent model.

Reimplemented from **gazebo::physics::Entity** (p. 294).

10.18.3.16 ShapePtr gazebo::physics::Collision::GetShape ( ) const

Get the collision shape.

Returns

The collision shape.

10.18.3.17 unsigned int gazebo::physics::Collision::GetShapeType ( )

Get the shape type.

Returns

The shape type.

See Also

EntityType (p. 144)

10.18.3.18 CollisionState gazebo::physics::Collision::GetState ( )

Get the collision state.

Returns

The collision state.

10.18.3.19 SurfaceParamsPtr gazebo::physics::Collision::GetSurface ( ) const [inline]

Get the surface parameters.

Returns

The surface parameters.

10.18.3.20 virtual math::Vector3 gazebo::physics::Collision::GetWorldAngularAccel ( ) const [virtual]

Get the angular acceleration of the collision in the world frame.

Returns

The angular acceleration of the collision in the world frame.

Reimplemented from **gazebo::physics::Entity** (p. 294).

10.18.3.21 virtual math::Vector3 gazebo::physics::Collision::GetWorldAngularVel ( ) const [virtual]

Get the angular velocity of the collision in the world frame.

Returns

The angular velocity of the collision in the world frame.

Reimplemented from **gazebo::physics::Entity** (p. 294).

10.18.3.22 virtual math::Vector3 gazebo::physics::Collision::GetWorldLinearAccel ( ) const [virtual]

Get the linear acceleration of the collision in the world frame.

Returns

The linear acceleration of the collision in the world frame.

Reimplemented from **gazebo::physics::Entity** (p. 294).

10.18.3.23 virtual math::Vector3 gazebo::physics::Collision::GetWorldLinearVel ( ) const [virtual]

Get the linear velocity of the collision in the world frame.

Returns

The linear velocity of the collision in the world frame.

Reimplemented from **gazebo::physics::Entity** (p. 295).

10.18.3.24 virtual void gazebo::physics::Collision::Init ( ) [virtual]

Initialize the collision.

Reimplemented from **gazebo::physics::Base** (p. 149).

10.18.3.25 bool gazebo::physics::Collision::IsPlaceable ( ) const

Return whether this collision is movable.

Example on an immovable object is a ray.

Returns

True if the object is immovable.

10.18.3.26 virtual void gazebo::physics::Collision::Load (sdf::ElementPtr _sdf) [virtual]

Load the collision.

Parameters

in	*_sdf*	SDF to load from.

Reimplemented from **gazebo::physics::Entity** (p. 295).

10.18.3.27 void gazebo::physics::Collision::ProcessMsg (const msgs::Collision & _msg)

Update parameters from a message.

Parameters

in	*_msg*	Message to update from.

10.18.3.28 virtual void gazebo::physics::Collision::SetCategoryBits (unsigned int _bits) [pure virtual]

Set the category bits, used during collision detection.

Parameters

in	*_bits*	The bits to set.

10.18.3.29 virtual void gazebo::physics::Collision::SetCollideBits (unsigned int _bits) [pure virtual]

Set the collide bits, used during collision detection.

Parameters

in	*_bits*	The bits to set.

10.18.3.30 void gazebo::physics::Collision::SetCollision (bool _placeable)

Set the encapsulated collsion object.

Parameters

in	*_placeable*	True to make the object movable.

10.18.3.31 void gazebo::physics::Collision::SetContactsEnabled (bool *_enable*)

Turn contact recording on or off.

Parameters

in	*_enable*	True to enable collision contacts.

10.18.3.32 void gazebo::physics::Collision::SetLaserRetro (float *_retro*)

Set the laser retro reflectiveness.

Parameters

in	*_retro*	The laser retro value.

10.18.3.33 virtual void gazebo::physics::Collision::SetMaxContacts (double *_maxContacts*) [virtual]

Number of contacts allowed for this collision.

This overrides global value (in **PhysicsEngine** (p. 597)) if specified.

Parameters

in	*_maxContacts*	max num contacts allowed for this collision.

10.18.3.34 void gazebo::physics::Collision::SetShape (ShapePtr *_shape*)

Set the shape for this collision.

Parameters

in	*_shape*	The shape for this collision object.

10.18.3.35 void gazebo::physics::Collision::SetState (const CollisionState & *_state*)

Set the current collision state.

Parameters

in	*The*	collision state.

10.18.3.36 virtual void gazebo::physics::Collision::UpdateParameters (sdf::ElementPtr _sdf) [virtual]

Update the parameters using new sdf values.

Parameters

in	*_sdf*	SDF values to update from.

Reimplemented from **gazebo::physics::Entity** (p. 298).

10.18.4 Member Data Documentation

10.18.4.1 LinkPtr gazebo::physics::Collision::link [protected]

The link this collision belongs to.

10.18.4.2 bool gazebo::physics::Collision::placeable [protected]

Flag for placeable.

10.18.4.3 ShapePtr gazebo::physics::Collision::shape [protected]

Pointer to **physics::Shape** (p. 754).

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

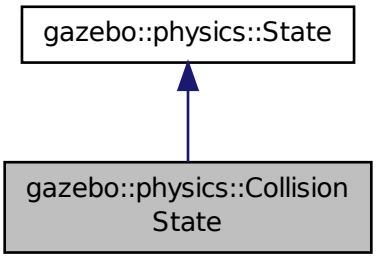
- **Collision.hh**

10.19 gazebo::physics::CollisionState Class Reference

Store state information of a **physics::Collision** (p. 199) object.

`#include <physics/phyiscs.hh>`

Inheritance diagram for gazebo::physics::CollisionState:



Public Member Functions

- **CollisionState** ()

  Default constructor.
- **CollisionState** (const **CollisionPtr** _collision)

  Constructor.
- **CollisionState** (const **sdf::ElementPtr** _sdf)

  Constructor.
- virtual ∼**CollisionState** ()

  Destructor.
- void **FillSDF** (**sdf::ElementPtr** _sdf)

  Populate a state SDF element with data from the object.
- const **math::Pose** & **GetPose** () const

  *Get the **Collision** (p. 199) pose.*
- bool **IsZero** () const

  Return true if the values in the state are zero.
- virtual void **Load** (const **sdf::ElementPtr** _elem)

  Load state from SDF element.
- **CollisionState operator+** (const **CollisionState** &_state) const

  Addition operator.
- **CollisionState operator-** (const **CollisionState** &_state) const

  Subtraction operator.
- **CollisionState** & **operator=** (const **CollisionState** &_state)

  Assignment operator.

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::physics::CollisionState** &_state)

  Stream insertion operator.

Additional Inherited Members

10.19.1 Detailed Description

Store state information of a **physics::Collision** (p. 199) object.

This class captures the entire state of a **Collision** (p. 199) at one specific time during a simulation run.

State (p. 797) of a **Collision** (p. 199) is its Pose.

10.19.2 Constructor & Destructor Documentation

10.19.2.1 gazebo::physics::CollisionState::CollisionState ( )

Default constructor.

10.19.2.2 gazebo::physics::CollisionState::CollisionState (const CollisionPtr _collision) [explicit]

Constructor.

Build a **CollisionState** (p. 209) from an existing **Collision** (p. 199).

Parameters

in	*_model*	Pointer to the **Link** (p. 439) from which to gather state info.

10.19.2.3 gazebo::physics::CollisionState::CollisionState (const sdf::ElementPtr _sdf) [explicit]

Constructor.

Build a **CollisionState** (p. 209) from SDF data

Parameters

in	*_sdf*	SDF data to load a collision state from.

10.19.2.4 virtual gazebo::physics::CollisionState::~CollisionState ( ) [virtual]

Destructor.

10.19.3 Member Function Documentation

10.19.3.1 void gazebo::physics::CollisionState::FillSDF (sdf::ElementPtr _sdf)

Populate a state SDF element with data from the object.

Parameters

out	*_sdf*	SDF element to populate.

10.19.3.2 const math::Pose& gazebo::physics::CollisionState::GetPose ( ) const

Get the **Collision** (p. 199) pose.

Returns

The pose of the **CollisionState** (p. 209)

10.19.3.3 bool gazebo::physics::CollisionState::IsZero ( ) const

Return true if the values in the state are zero.

Returns

True if the values in the state are zero.

10.19.3.4 virtual void gazebo::physics::CollisionState::Load (const sdf::ElementPtr _elem) [virtual]

Load state from SDF element.

Load **CollisionState** (p. 209) information from stored data in and SDF::Element

Parameters

in	*_elem*	Pointer to the SDF::Element containing state info.

Reimplemented from **gazebo::physics::State** (p. 800).

10.19.3.5 CollisionState gazebo::physics::CollisionState::operator+ (const CollisionState & _state) const

Addition operator.

Parameters

in	*_pt*	A state to add.

Returns

The resulting state.

10.19.3.6 CollisionState gazebo::physics::CollisionState::operator- (const CollisionState & _state) const

Subtraction operator.

Parameters

in	*_pt*	A state to substract.

Returns

The resulting state.

10.19.3.7 CollisionState& gazebo::physics::CollisionState::operator= (const CollisionState & _state)

Assignment operator.

Parameters

in	*_state*	**State** (p. 797) value

Returns

Reference to this

10.19.4 Friends And Related Function Documentation

10.19.4.1 std::ostream& operator<< (std::ostream & *_out,* const gazebo::physics::CollisionState & *_state*) [friend]

Stream insertion operator.

Parameters

in	*_out*	output stream
in	*_state*	**Collision** (p. 199) state to output

Returns

the stream

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

- **CollisionState.hh**

10.20 gazebo::common::Color Class Reference

Defines a color.

`#include <common/common.hh>`

Public Types

- typedef unsigned int **ABGR**
- typedef unsigned int **ARGB**
- typedef unsigned int **BGRA**
- typedef unsigned int **RGBA**

Public Member Functions

- **Color** ()

  Constructor.
- **Color** (float _r, float _g, float _b, float _a=1.0)

  Constructor.
- **Color** (const **Color** &_clr)

  Copy Constructor.
- virtual ~**Color** ()

  Destructor.
- **ABGR GetAsABGR** () const

  Get as uint32 ABGR packed value.
- **ARGB GetAsARGB** () const

  Get as uint32 ARGB packed value.
- **BGRA GetAsBGRA** () const

  Get as uint32 BGRA packed value.
- **math::Vector3 GetAsHSV** () const

  Get the color in HSV colorspace.

- **RGBA GetAsRGBA** () const

  Get as uint32 RGBA packed value.
- **math::Vector3 GetAsYUV** () const

  Get the color in YUV colorspace.
- bool **operator!=** (const **Color** &_pt) const

  Inequality operator.
- const **Color operator**∗ (const **Color** &_pt) const

  Multiplication operator.
- const **Color operator**∗ (const float &_v) const

  Multiply all color components by _v.
- const **Color** & **operator**∗**=** (const **Color** &_pt)

  Multiplication equal operator.
- **Color operator+** (const **Color** &_pt) const

  Addition operator (this + _pt)
- **Color operator+** (const float &_v) const

  Add _v to all color components.
- const **Color** & **operator+=** (const **Color** &_pt)

  Addition equal operator.
- **Color operator-** (const **Color** &_pt) const

  Subtraction operator.
- **Color operator-** (const float &_v) const

  Subtract _v from all color components.
- const **Color** & **operator-=** (const **Color** &_pt)

  Subtraction equal operator.
- const **Color operator**/ (const **Color** &_pt) const

  Division operator.
- const **Color operator**/ (const float &_v) const

  Divide all color component by _v.
- const **Color** & **operator/=** (const **Color** &_pt)

  Division equal operator.
- **Color** & **operator=** (const **Color** &_pt)

  Equal operator.
- bool **operator==** (const **Color** &_pt) const

  Equality operator.
- float **operator[ ]** (unsigned int _index)

  Array index operator.
- void **Reset** ()

  Reset the color to default values.
- void **Set** (float _r=1, float _g=1, float _b=1, float _a=1)

  Set the contents of the vector.
- void **SetFromABGR** (const **ABGR** _v)

  Set from uint32 ABGR packed value.
- void **SetFromARGB** (const **ARGB** _v)

  Set from uint32 ARGB packed value.
- void **SetFromBGRA** (const **BGRA** _v)

  Set from uint32 BGRA packed value.
- void **SetFromHSV** (float _h, float _s, float _v)

Set a color based on HSV values.

- void **SetFromRGBA** (const **RGBA** _v)

  Set from uint32 RGBA packed value.

- void **SetFromYUV** (float _y, float _u, float _v)

  Set from yuv.

Public Attributes

- float **a**
- float **b**
- float **g**
- float **r**

Static Public Attributes

- static const **Color Black**

  (0, 0, 0)

- static const **Color Blue**

  (0, 0, 1)

- static const **Color Green**

  (0, 1, 0)

- static const **Color Purple**

  (1, 0, 1)

- static const **Color Red**

  (1, 0, 0)

- static const **Color White**

  (1, 1, 1)

- static const **Color Yellow**

  (1, 1, 0)

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **Color** &_pt)

  Stream insertion operator.

- std::istream & **operator**>> (std::istream &_in, **Color** &_pt)

  Stream insertion operator.

10.20.1 Detailed Description

Defines a color.

10.20.2 Member Typedef Documentation

10.20.2.1 typedef unsigned int gazebo::common::Color::ABGR

10.20.2.2 typedef unsigned int gazebo::common::Color::ARGB

10.20.2.3 typedef unsigned int gazebo::common::Color::BGRA

10.20.2.4 typedef unsigned int gazebo::common::Color::RGBA

10.20.3 Constructor & Destructor Documentation

10.20.3.1 gazebo::common::Color::Color ( )

Constructor.

10.20.3.2 gazebo::common::Color::Color (float *_r*, float *_g*, float *_b*, float *_a* = `1.0`)

Constructor.

Parameters

in	*_r*	Red value (range 0 to 1)
in	*_g*	Green value (range 0 to 1
in	*_b*	Blue value (range 0 to 1
in	*_a*	Alpha value (0=transparent, 1=opaque)

10.20.3.3 gazebo::common::Color::Color (const **Color** & *_clr*)

Copy Constructor.

Parameters

in	*_clr*	**Color** (p. 213) to copy

10.20.3.4 virtual gazebo::common::Color::~Color ( ) [virtual]

Destructor.

10.20.4 Member Function Documentation

10.20.4.1 ABGR gazebo::common::Color::GetAsABGR ( ) const

Get as uint32 ABGR packed value.

Returns

the color

10.20.4.2 ARGB gazebo::common::Color::GetAsARGB ( ) const

Get as uint32 ARGB packed value.

Returns

the color

10.20.4.3 BGRA gazebo::common::Color::GetAsBGRA ( ) const

Get as uint32 BGRA packed value.

Returns

the color

10.20.4.4 math::Vector3 gazebo::common::Color::GetAsHSV ( ) const

Get the color in HSV colorspace.

Returns

HSV values in a **math::Vector3** (p. 890) format

10.20.4.5 RGBA gazebo::common::Color::GetAsRGBA ( ) const

Get as uint32 RGBA packed value.

Returns

the color

10.20.4.6 math::Vector3 gazebo::common::Color::GetAsYUV ( ) const

Get the color in YUV colorspace.

Returns

the YUV color

10.20.4.7 bool gazebo::common::Color::operator!= (const Color & *_pt*) const

Inequality operator.

Parameters

in	*_pt*	The color to check for inequality

Returns

True if the this color does not equal _pt

10.20.4.8 const Color gazebo::common::Color::operator∗ (const Color & _pt) const

Multiplication operator.

Parameters

in	*_pt*	The color to muliply by

Returns

The resulting color

10.20.4.9 const Color gazebo::common::Color::operator∗ (const float & _v) const

Multiply all color components by _v.

Parameters

in	*_v*	The value to multiply by

Returns

The resulting color

10.20.4.10 const Color& gazebo::common::Color::operator∗= (const Color & _pt)

Multiplication equal operator.

Parameters

in	*_pt*	The color to muliply by

Returns

The resulting color

10.20.4.11 Color gazebo::common::Color::operator+ (const Color & _pt) const

Addition operator (this + _pt)

Parameters

in	*_pt*	**Color** (p. 213) to add

Returns

The resulting color

10.20.4.12 Color gazebo::common::Color::operator+ (const float & _v) const

Add _v to all color components.

Parameters

in	*_v*	Value to add to each color component

Returns

The resulting color

10.20.4.13 const Color& gazebo::common::Color::operator+= (const Color & _pt)

Addition equal operator.

Parameters

in	*_pt*	**Color** (p. 213) to add

Returns

The resulting color

10.20.4.14 Color gazebo::common::Color::operator- (const Color & _pt) const

Subtraction operator.

Parameters

in	*_pt*	The color to substract

Returns

The resulting color

10.20.4.15 Color gazebo::common::Color::operator- (const float & _v) const

Subtract _v from all color components.

Parameters

in	*_v*	Value to subtract

Returns

The resulting color

10.20.4.16 const Color& gazebo::common::Color::operator-= (const Color & _pt)

Subtraction equal operator.

Parameters

in	*_pt*	**Color** (p. 213) to subtract

Returns

The resulting color

10.20.4.17 const Color gazebo::common::Color::operator/ (const Color & _pt) const

Division operator.

Parameters

in	*_pt*	**Color** (p. 213) to divide by

Returns

The resulting color

10.20.4.18 const Color gazebo::common::Color::operator/ (const float & _v) const

Divide all color component by _v.

Parameters

in	*_v*	The value to divide by

Returns

The resulting color

10.20.4.19 const Color& gazebo::common::Color::operator/= (const Color & _pt)

Division equal operator.

Parameters

in	*_pt*	**Color** (p. 213) to divide by

Returns

The resulting color

10.20.4.20 Color& gazebo::common::Color::operator= (const Color & _pt)

Equal operator.

Parameters

in	*_pt*	**Color** (p. 213) to copy

Returns

Reference to this color

10.20.4.21 bool gazebo::common::Color::operator== (const Color & _pt) const

Equality operator.

Parameters

in	*_pt*	The color to check for equality

Returns

True if the this color equals _pt

10.20.4.22 float gazebo::common::Color::operator[] (unsigned int _index)

Array index operator.

Parameters

in	*_index*	**Color** (p. 213) component index(0=red, 1=green, 2=blue)

Returns

r, g, b, or a when _index is 0, 1, 2 or 3

10.20.4.23 void gazebo::common::Color::Reset ( )

Reset the color to default values.

10.20.4.24 void gazebo::common::Color::Set (float _r = 1, float _g = 1, float _b = 1, float _a = 1)

Set the contents of the vector.

Parameters

in	_r	Red value (range 0 to 1)
in	_g	Green value (range 0 to 1)
in	_b	Blue value (range 0 to 1)
in	_a	Alpha value (0=transparent, 1=opaque)

10.20.4.25 void gazebo::common::Color::SetFromABGR (const ABGR _v)

Set from uint32 ABGR packed value.

Parameters

in	_v	the new color

10.20.4.26 void gazebo::common::Color::SetFromARGB (const ARGB _v)

Set from uint32 ARGB packed value.

Parameters

in	_v	the new color

10.20.4.27 void gazebo::common::Color::SetFromBGRA (const BGRA _v)

Set from uint32 BGRA packed value.

Parameters

in	_v	the new color

10.20.4.28 void gazebo::common::Color::SetFromHSV (float _h, float _s, float _v)

Set a color based on HSV values.

Parameters

in	_h	Hue(0..360)
in	_s	Saturation(0..1)
in	_v	Value(0..1)

10.20.4.29 void gazebo::common::Color::SetFromRGBA (const RGBA _v)

Set from uint32 RGBA packed value.

Parameters

in	_v	the new color

10.20.4.30 void gazebo::common::Color::SetFromYUV (float _y, float _u, float _v)

Set from yuv.

Parameters

in	_y	value
in	_u	value
in	_v	value

10.20.5 Friends And Related Function Documentation

10.20.5.1 std::ostream& operator<< (std::ostream & _out, const Color & _pt) [friend]

Stream insertion operator.

Parameters

in	_out	the output stream
in	_pt	the color

Returns

the output stream

10.20.5.2 std::istream& operator>> (std::istream & _in, Color & _pt) [friend]

Stream insertion operator.

Parameters

in	_in	the input stream
in	_pt	

10.20.6 Member Data Documentation

10.20.6.1 float gazebo::common::Color::a

10.20.6.2 float gazebo::common::Color::b

10.20.6.3 const Color gazebo::common::Color::Black [static]

(0, 0, 0)

10.20.6.4 const Color gazebo::common::Color::Blue [static]

(0, 0, 1)

10.20.6.5 float gazebo::common::Color::g

10.20.6.6 const Color gazebo::common::Color::Green [static]

(0, 1, 0)

10.20.6.7 const Color gazebo::common::Color::Purple [static]

(1, 0, 1)

10.20.6.8 float gazebo::common::Color::r

10.20.6.9 const Color gazebo::common::Color::Red [static]

(1, 0, 0)

10.20.6.10 const Color gazebo::common::Color::White [static]

(1, 1, 1)

10.20.6.11 const Color gazebo::common::Color::Yellow [static]

(1, 1, 0)

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

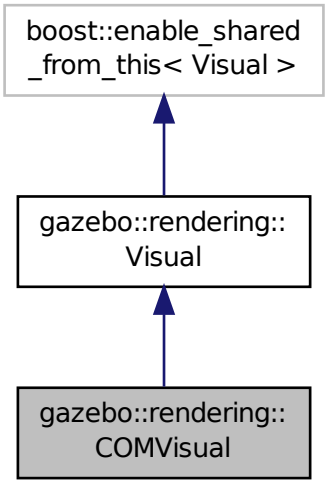
- **Color.hh**

10.21 gazebo::rendering::COMVisual Class Reference

Basic Center of Mass visualization.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::COMVisual:



Public Member Functions

- **COMVisual** (const std::string &_name, **VisualPtr** _vis)

  Constructor.
- virtual ~**COMVisual** ()

  Destructor.
- virtual void **Load** (**sdf::ElementPtr** _elem)

  *Load the **Visual** (p. 920) from an SDF pointer.*
- virtual void **Load** (ConstLinkPtr &_msg)

  Load from a message.

Additional Inherited Members

10.21.1 Detailed Description

Basic Center of Mass visualization.

10.21.2 Constructor & Destructor Documentation

10.21.2.1 gazebo::rendering::COMVisual::COMVisual (const std::string & *_name*, VisualPtr *_vis*)

Constructor.

Parameters

in	*_name*	Name of the **Visual** (p. 920)
in	*_vis*	Parent **Visual** (p. 920)

10.21.2.2 virtual gazebo::rendering::COMVisual::~COMVisual ( ) [virtual]

Destructor.

10.21.3 Member Function Documentation

10.21.3.1 virtual void gazebo::rendering::COMVisual::Load (sdf::ElementPtr *_elem*) [virtual]

Load the **Visual** (p. 920) from an SDF pointer.

Parameters

in	*_elem*	SDF Element pointer

10.21.3.2 virtual void gazebo::rendering::COMVisual::Load (ConstLinkPtr & *_msg*) [virtual]

Load from a message.

Parameters

in	*_msg*	Pointer to the message

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

- **COMVisual.hh**

10.22 gazebo::event::Connection Class Reference

A class that encapsulates a connection.

`#include <Event.hh>`

Public Member Functions

- **Connection** ()

  Constructor.
- **Connection** (**Event** *_e, int _i)

  Constructor.
- ~**Connection** ()

  Destructor.
- int **GetId** () const

  Get the id of this connection.

10.22.1 Detailed Description

A class that encapsulates a connection.

10.22.2 Constructor & Destructor Documentation

10.22.2.1 gazebo::event::Connection::Connection ( ) [inline]

Constructor.

10.22.2.2 gazebo::event::Connection::Connection (Event * _e, int _i)

Constructor.

Parameters

in	*_e*	**Event** (p. 299) pointer to connect with
in	*_i*	Unique id

10.22.2.3 gazebo::event::Connection::~Connection ( )

Destructor.

10.22.3 Member Function Documentation

10.22.3.1 int gazebo::event::Connection::GetId ( ) const

Get the id of this connection.

Returns

The id of this connection

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

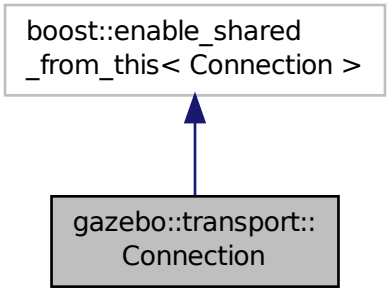
- **Event.hh**

10.23 gazebo::transport::Connection Class Reference

Single TCP/IP connection manager.

`#include <transport/transport.hh>`

Inheritance diagram for gazebo::transport::Connection:



Public Types

- typedef boost::function< void(const **ConnectionPtr** &)> **AcceptCallback**

  The signature of a connection accept callback.
- typedef boost::function< void(const std::string &_data)> **ReadCallback**

  The signature of a connection read callback.

Public Member Functions

- **Connection** ()

  Constructor.
- virtual ~**Connection** ()

  Destructor.
- template<typename Handler >
  void **AsyncRead** (Handler _handler)

  Peform an asyncronous read param[in] _handler Callback to invoke on received data.
- void **Cancel** ()

  Cancel all async operations on an open socket.
- bool **Connect** (const std::string &_host, unsigned int _port)

  Connect to a remote host.
- **event::ConnectionPtr ConnectToShutdown** (boost::function< void()> _subscriber)

  Register a function to be called when the connection is shut down.
- void **DisconnectShutdown** (**event::ConnectionPtr** _subscriber)

  Unregister a function to be called when the connection is shut down.
- void **EnqueueMsg** (const std::string &_buffer, boost::function< void(uint32_t)> _cb, uint32_t _id, bool _force=false)

  Write data to the socket.
- void **EnqueueMsg** (const std::string &_buffer, bool _force=false)

Write data to the socket.

- unsigned int **GetId** () const

  Get the ID of the connection.

- std::string **GetIPWhiteList** () const

  Get the IP white list, from GAZEBO_IP_WHITE_LIST environment variable.

- std::string **GetLocalAddress** () const

  Get the local address of this connection.

- unsigned int **GetLocalPort** () const

  Get the port of this connection.

- std::string **GetLocalURI** () const

  Get the local URI.

- std::string **GetRemoteAddress** () const

  Get the remote address.

- std::string **GetRemoteHostname** () const

  Get the remote hostname.

- unsigned int **GetRemotePort** () const

  Get the remote port number.

- std::string **GetRemoteURI** () const

  Get the remote URI.

- bool **IsOpen** () const

  Is the connection open?

- void **Listen** (unsigned int _port, const **AcceptCallback** &_acceptCB)

  Start a server that listens on a port.

- void **ProcessWriteQueue** (bool _blocking=false)

  Handle on-write callbacks.

- bool **Read** (std::string &_data)

  Read data from the socket.

- void **Shutdown** ()

  Shutdown the socket.

- void **StartRead** (const **ReadCallback** &_cb)

  Start a thread that reads from the connection and passes new message to the ReadCallback.

- void **StopRead** ()

  Stop the read loop.

Static Public Member Functions

- static std::string **GetLocalHostname** ()

  Get the local hostname.

- static bool **ValidateIP** (const std::string &_ip)

  Return true if the _ip is a valid.

10.23.1 Detailed Description

Single TCP/IP connection manager.

10.23.2 Member Typedef Documentation

10.23.2.1 typedef boost::function<void(const ConnectionPtr&)> gazebo::transport::Connection::AcceptCallback

The signature of a connection accept callback.

10.23.2.2 typedef boost::function<void(const std::string &_data)> gazebo::transport::Connection::ReadCallback

The signature of a connection read callback.

10.23.3 Constructor & Destructor Documentation

10.23.3.1 gazebo::transport::Connection::Connection ( )

Constructor.

10.23.3.2 virtual gazebo::transport::Connection::~Connection ( ) [virtual]

Destructor.

10.23.4 Member Function Documentation

10.23.4.1 template<typename Handler > void gazebo::transport::Connection::AsyncRead (Handler *_handler*) [inline]

Peform an asyncronous read param[in] _handler Callback to invoke on received data.

References gzerr, HEADER_LENGTH, and IsOpen().

10.23.4.2 void gazebo::transport::Connection::Cancel ( )

Cancel all async operations on an open socket.

10.23.4.3 bool gazebo::transport::Connection::Connect (const std::string & *_host*, unsigned int *_port*)

Connect to a remote host.

Parameters

in	*_host*	The host to connect to
in	*_port*	The port to connect to

Returns

true if connection succeeded, false otherwise

10.23.4.4 event::ConnectionPtr gazebo::transport::Connection::ConnectToShutdown (boost::function< void()> *_subscriber*) [inline]

Register a function to be called when the connection is shut down.

Parameters

in	*_subscriber*	Function to be called

Returns

Handle that can be used to unregister the function

References gazebo::event::EventT< T >::Connect().

10.23.4.5 void gazebo::transport::Connection::DisconnectShutdown (event::ConnectionPtr *_subscriber*) [inline]

Unregister a function to be called when the connection is shut down.

Parameters

in	*_subscriber*	Handle previously returned by **ConnectToShutdown()** (p. 230)

References gazebo::event::EventT< T >::Disconnect().

10.23.4.6 void gazebo::transport::Connection::EnqueueMsg (const std::string & *_buffer,* boost::function< void(uint32_t)> *_cb,* uint32_t *_id,* bool *_force* = false)

Write data to the socket.

Parameters

in	*_buffer*	Data to write
in	*_force*	If true, block until the data has been written to the socket, otherwise just enqueue the data for asynchronous write
in	*_cb*	If non-null, callback to be invoked after transmission is complete.
in	*_id*	ID associated with the message data.

10.23.4.7 void gazebo::transport::Connection::EnqueueMsg (const std::string & *_buffer,* bool *_force* = false)

Write data to the socket.

Parameters

in	*_buffer*	Data to write
in	*_force*	If true, block until the data has been written to the socket, otherwise just enqueue the data for asynchronous write

10.23.4.8 unsigned int gazebo::transport::Connection::GetId ( ) const

Get the ID of the connection.

Returns

The connection's unique ID.

10.23.4.9 std::string gazebo::transport::Connection::GetIPWhiteList ( ) const

Get the IP white list, from GAZEBO_IP_WHITE_LIST environment variable.

Returns

GAZEBO_IP_WHITE_LIST

10.23.4.10 std::string gazebo::transport::Connection::GetLocalAddress ( ) const

Get the local address of this connection.

Returns

The local address

10.23.4.11 static std::string gazebo::transport::Connection::GetLocalHostname ( ) [static]

Get the local hostname.

Returns

The local hostname

10.23.4.12 unsigned int gazebo::transport::Connection::GetLocalPort ( ) const

Get the port of this connection.

Returns

The local port

10.23.4.13 std::string gazebo::transport::Connection::GetLocalURI ( ) const

Get the local URI.

Returns

The local URI

10.23.4.14 std::string gazebo::transport::Connection::GetRemoteAddress ( ) const

Get the remote address.

Returns

The remote address

10.23.4.15 std::string gazebo::transport::Connection::GetRemoteHostname ( ) const

Get the remote hostname.

Returns

The remote hostname

10.23.4.16 unsigned int gazebo::transport::Connection::GetRemotePort ( ) const

Get the remote port number.

Returns

The remote port

10.23.4.17 std::string gazebo::transport::Connection::GetRemoteURI ( ) const

Get the remote URI.

Returns

The remote URI

10.23.4.18 bool gazebo::transport::Connection::IsOpen ( ) const

Is the connection open?

Returns

true if the connection is open; false otherwise

Referenced by AsyncRead().

10.23.4.19 void gazebo::transport::Connection::Listen (unsigned int *_port,* const **AcceptCallback** & *_acceptCB*)

Start a server that listens on a port.

Parameters

in	*_port*	The port to listen on
in	*_acceptCB*	The callback to invoke when a new connection has been accepted

10.23.4.20 void gazebo::transport::Connection::ProcessWriteQueue (bool *_blocking* = `false`)

Handle on-write callbacks.

10.23.4.21 bool gazebo::transport::Connection::Read (std::string & *_data*)

Read data from the socket.

Parameters

out	*_data*	Destination for data that is read

Returns

true if data was successfully read, false otherwise

10.23.4.22 void gazebo::transport::Connection::Shutdown ( )

Shutdown the socket.

10.23.4.23 void gazebo::transport::Connection::StartRead (const **ReadCallback** & *_cb*)

Start a thread that reads from the connection and passes new message to the ReadCallback.

Parameters

in	*_cb*	The callback to invoke when a new message is received

10.23.4.24 void gazebo::transport::Connection::StopRead ( )

Stop the read loop.

10.23.4.25 static bool gazebo::transport::Connection::ValidateIP (const std::string & *_ip*) [static]

Return true if the _ip is a valid.

Parameters

in	*_ip*	Dotted quad to validate.

Returns

True if the _ip is a valid.

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

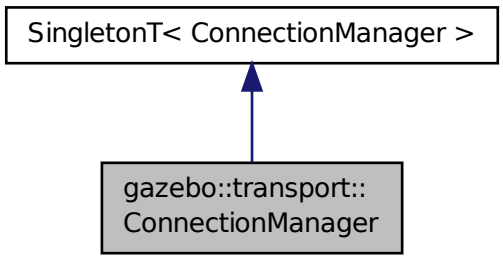
- **Connection.hh**

10.24 gazebo::transport::ConnectionManager Class Reference

Manager of connections.

`#include <transport/transport.hh>`

Inheritance diagram for gazebo::transport::ConnectionManager:



Public Member Functions

- void **Advertise** (const std::string &_topic, const std::string &_msgType)

  Advertise a topic.
- **ConnectionPtr ConnectToRemoteHost** (const std::string &_host, unsigned int _port)

  Connect to a remote server.
- void **Fini** ()

  Finalize the connection manager.
- void **GetAllPublishers** (std::list< msgs::Publish > &_publishers)

  Explicitly update the publisher list.
- void **GetTopicNamespaces** (std::list< std::string > &_namespaces)

  Get all the topic namespaces.
- bool **Init** (const std::string &_masterHost, unsigned int _masterPort)

  Initialize the connection manager.
- bool **IsRunning** () const

  Is the manager running?
- void **RegisterTopicNamespace** (const std::string &_name)

  Register a new topic namespace.
- void **RemoveConnection** (**ConnectionPtr** &_conn)

  Remove a connection from the manager.
- void **Run** ()

  Run the connection manager loop.
- void **Stop** ()

  Stop the conneciton manager.
- void **Subscribe** (const std::string &_topic, const std::string &_msgType, bool _latching)

  Subscribe to a topic.
- void **TriggerUpdate** ()

  Inform the connection manager that it needs an update.
- void **Unadvertise** (const std::string &_topic)

  Unadvertise a topic.

- void **Unsubscribe** (const msgs::Subscribe &_sub)

  Unsubscribe from a topic.
- void **Unsubscribe** (const std::string &_topic, const std::string &_msgType)

  Unsubscribe from a topic.

Protected Attributes

- std::vector< **event::ConnectionPtr** > **eventConnections**

Additional Inherited Members

10.24.1 Detailed Description

Manager of connections.

10.24.2 Member Function Documentation

10.24.2.1 void gazebo::transport::ConnectionManager::Advertise (const std::string & *_topic,* const std::string & *_msgType*)

Advertise a topic.

Parameters

in	*_topic*	The topic to advertise
in	*_msgType*	The type of the topic

10.24.2.2 ConnectionPtr gazebo::transport::ConnectionManager::ConnectToRemoteHost (const std::string & *_host,* unsigned int *_port*)

Connect to a remote server.

Parameters

in	*_host*	Host to connect to
in	*_port*	Port to connect to

Returns

Pointer to the connection; can be null (if connection failed)

10.24.2.3 void gazebo::transport::ConnectionManager::Fini ( )

Finalize the connection manager.

10.24.2.4 void gazebo::transport::ConnectionManager::GetAllPublishers (std::list< msgs::Publish > & *_publishers*)

Explicitly update the publisher list.

Parameters

out	*_publishers*	The updated list of publishers is written here

10.24.2.5 void gazebo::transport::ConnectionManager::GetTopicNamespaces (std::list< std::string > & *_namespaces*)

Get all the topic namespaces.

Parameters

out	*_namespaces*	The list of namespace is written here

10.24.2.6 bool gazebo::transport::ConnectionManager::Init (const std::string & *_masterHost,* unsigned int *_masterPort*)

Initialize the connection manager.

Parameters

in	*_masterHost*	Host where the master is running
in	*_masterPort*	Port where the master is running

Returns

true if initialization succeeded, false otherwise

10.24.2.7 bool gazebo::transport::ConnectionManager::IsRunning ( ) const

Is the manager running?

Returns

true if running, false otherwise

10.24.2.8 void gazebo::transport::ConnectionManager::RegisterTopicNamespace (const std::string & *_name*)

Register a new topic namespace.

Parameters

in	*_name*	The name of the topic namespace to be registered

10.24.2.9 void gazebo::transport::ConnectionManager::RemoveConnection (ConnectionPtr & *_conn*)

Remove a connection from the manager.

Parameters

in	*_conn*	The connection to be removed

10.24.2.10 void gazebo::transport::ConnectionManager::Run ( )

Run the connection manager loop.

Does not return until stopped.

10.24.2.11 void gazebo::transport::ConnectionManager::Stop ( )

Stop the conneciton manager.

10.24.2.12 void gazebo::transport::ConnectionManager::Subscribe (const std::string & *_topic,* const std::string & *_msgType,* bool *_latching*)

Subscribe to a topic.

Parameters

in	*_topic*	The topic to subscribe to
in	*_msgType*	The type of the topic
in	*_latching*	If true, latch the latest incoming message; otherwise don't

10.24.2.13 void gazebo::transport::ConnectionManager::TriggerUpdate ( )

Inform the connection manager that it needs an update.

10.24.2.14 void gazebo::transport::ConnectionManager::Unadvertise (const std::string & *_topic*)

Unadvertise a topic.

Parameters

in	*_topic*	The topic to unadvertise

10.24.2.15 void gazebo::transport::ConnectionManager::Unsubscribe (const msgs::Subscribe & *_sub*)

Unsubscribe from a topic.

Parameters

in	*_sub*	A subscription object

10.24.2.16 void gazebo::transport::ConnectionManager::Unsubscribe (const std::string & *_topic,* const std::string & *_msgType*)

Unsubscribe from a topic.

Parameters

in	*_topic*	The topic to unsubscribe from
in	*_msgType*	The type of the topic

10.24.3 Member Data Documentation

10.24.3.1 std::vector<event::ConnectionPtr> gazebo::transport::ConnectionManager::eventConnections [protected]

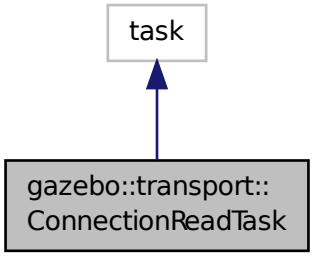
The documentation for this class was generated from the following file:

- **ConnectionManager.hh**

10.25 gazebo::transport::ConnectionReadTask Class Reference

`#include <Connection.hh>`

Inheritance diagram for gazebo::transport::ConnectionReadTask:



Public Member Functions

- **ConnectionReadTask** (boost::function< void(const std::string &)> _func, const std::string &_data)

  Constructor.
- tbb::task * **execute** ()

  Overridden function from tbb::task that exectues the data callback.

10.25.1 Detailed Description

A task instance that is created when data is read from a socket and used by TBB

10.25.2 Constructor & Destructor Documentation

10.25.2.1 gazebo::transport::ConnectionReadTask::ConnectionReadTask (boost::function< void(const std::string &)> *_func*, const std::string & *_data*) [inline]

Constructor.

Parameters

	_in]	_func Boost function pointer, which is the function that receives the data.
in	*_data*	Data to send to the boost function pointer.

10.25.3 Member Function Documentation

10.25.3.1 tbb::task* gazebo::transport::ConnectionReadTask::execute ( ) [inline]

Overridden function from tbb::task that exectues the data callback.

References NULL.

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

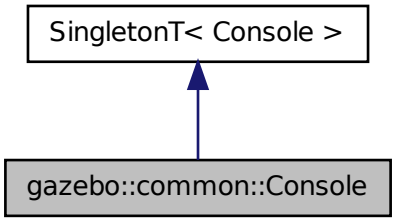
- **Connection.hh**

10.26 gazebo::common::Console Class Reference

Message, error, warning functionality.

`#include <common/commom.hh>`

Inheritance diagram for gazebo::common::Console:



Public Member Functions

- std::ostream & **ColorErr** (const std::string &_lbl, const std::string &_file, unsigned int _line, int _color)

  Use this to output an error to the terminal.
- std::ostream & **ColorMsg** (const std::string &_lbl, int _color)

  Use this to output a colored message to the terminal.
- bool **GetQuiet** () const

  Get whether quiet output is set.
- void **Init** (const std::string &_logFilename)

  Load the message parameters.
- bool **IsInitialized** () const

  Return true if Init has been called.

- std::ofstream & **Log** ()

  Use this to output a colored message to the terminal.
- void **SetQuiet** (bool _q)

  Set quiet output.

Additional Inherited Members

10.26.1 Detailed Description

Message, error, warning functionality.

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

- **Console.hh**

10.27 gazebo::physics::Contact Class Reference

A contact between two collisions.

`#include <physics/physics.hh>`

Public Member Functions

- **Contact** ()

  Constructor.
- **Contact** (const **Contact** &_contact)

  Copy constructor.
- virtual ~**Contact** ()

  Destructor.
- std::string **DebugString** () const

  Produce a debug string.
- void **FillMsg** (msgs::Contact &_msg) const

  Populate a msgs::Contact with data from this.
- **Contact** & **operator=** (const **Contact** &_contact)

  Operator =.
- **Contact** & **operator=** (const msgs::Contact &_contact)

  Operator =.
- void **Reset** ()

  Reset to default values.

Public Attributes

- **Collision** * **collision1**

  Pointer to the first collision object.
- **Collision** * **collision2**

  Pointer to the second collision object.
- int **count**

Length of all the arrays.

- double **depths** [32]

  Array of contact depths.

- **math::Vector3 normals** [32]

  Array of force normals.

- **math::Vector3 positions** [32]

  Array of force positions.

- **common::Time time**

  Time at which the contact occurred.

- **WorldPtr world**

  World (p. *945) in which the contact occurred.*

- **JointWrench wrench** [32]

  Array of forces for the contact.

10.27.1 Detailed Description

A contact between two collisions.

Each contact can consist of a number of contact points

10.27.2 Constructor & Destructor Documentation

10.27.2.1 gazebo::physics::Contact::Contact ( )

Constructor.

10.27.2.2 gazebo::physics::Contact::Contact (const Contact & *_contact*)

Copy constructor.

Parameters

in	*_contact*	**Contact** (p. 241) to copy.

10.27.2.3 virtual gazebo::physics::Contact::~Contact ( ) [virtual]

Destructor.

10.27.3 Member Function Documentation

10.27.3.1 std::string gazebo::physics::Contact::DebugString ( ) const

Produce a debug string.

Returns

A string that contains the values of the contact.

10.27.3.2 void gazebo::physics::Contact::FillMsg (msgs::Contact & *_msg*) const

Populate a msgs::Contact with data from this.

Parameters

out	*_msg*	**Contact** (p. 241) message the will hold the data.

10.27.3.3 Contact& gazebo::physics::Contact::operator= (const Contact & *_contact*)

Operator =.

Parameters

in	*_contact*	**Contact** (p. 241) to copy.

Returns

Reference to this contact

10.27.3.4 Contact& gazebo::physics::Contact::operator= (const msgs::Contact & *_contact*)

Operator =.

Parameters

in	*_contact*	msgs::Contact to copy.

Returns

Reference to this contact

10.27.3.5 void gazebo::physics::Contact::Reset ( )

Reset to default values.

10.27.4 Member Data Documentation

10.27.4.1 Collision* gazebo::physics::Contact::collision1

Pointer to the first collision object.

10.27.4.2 Collision* gazebo::physics::Contact::collision2

Pointer to the second collision object.

10.27.4.3 int gazebo::physics::Contact::count

Length of all the arrays.

10.27.4.4 double gazebo::physics::Contact::depths[32]

Array of contact depths.

10.27.4.5 math::Vector3 gazebo::physics::Contact::normals[32]

Array of force normals.

10.27.4.6 math::Vector3 gazebo::physics::Contact::positions[32]

Array of force positions.

10.27.4.7 common::Time gazebo::physics::Contact::time

Time at which the contact occurred.

10.27.4.8 WorldPtr gazebo::physics::Contact::world

World (p. 945) in which the contact occurred.

10.27.4.9 JointWrench gazebo::physics::Contact::wrench[32]

Array of forces for the contact.

All forces and torques are relative to the center of mass of the respective links that the collision elments are attached to.

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

- **Contact.hh**

10.28 gazebo::physics::ContactManager Class Reference

Aggregates all the contact information generated by the collision detection engine.

`#include <physics/physics.hh>`

Public Member Functions

- **ContactManager** ()
  Constructor.
- virtual ~**ContactManager** ()
  Destructor.
- void **Clear** ()
  Clear all stored contacts.
- std::string **CreateFilter** (const std::string &_topic, const std::vector< std::string > &_collisions)
  Create a filter for contacts.
- std::string **CreateFilter** (const std::string &_topic, const std::string &_collision)
  Create a filter for contacts.

- **Contact** ∗ **GetContact** (unsigned int _index) const

  Get a single contact by index.
- unsigned int **GetContactCount** () const

  Return the number of valid contacts.
- const std::vector< **Contact** ∗ > & **GetContacts** () const

  Get all the contacts.
- void **Init** (**WorldPtr** _world)

  *Initialize the **ContactManager** (p. 244).*
- **Contact** ∗ **NewContact** (**Collision** ∗_collision1, **Collision** ∗_collision2, const **common::Time** &_time)

  Add a new contact.
- void **PublishContacts** ()

  Publish all contacts in a msgs::Contacts message.
- void **ResetCount** ()

  Set the contact count to zero.

10.28.1 Detailed Description

Aggregates all the contact information generated by the collision detection engine.

10.28.2 Constructor & Destructor Documentation

10.28.2.1 gazebo::physics::ContactManager::ContactManager ( )

Constructor.

10.28.2.2 virtual gazebo::physics::ContactManager::~ContactManager ( ) [virtual]

Destructor.

10.28.3 Member Function Documentation

10.28.3.1 void gazebo::physics::ContactManager::Clear ( )

Clear all stored contacts.

10.28.3.2 std::string gazebo::physics::ContactManager::CreateFilter (const std::string & *_topic,* const std::vector< std::string > & *_collisions*)

Create a filter for contacts.

A new publisher will be created that publishes contacts associated to the input collisions. param[in] _name Filter name. param[in] _collisions A list of collision names used for filtering.

Returns

New topic where filtered messages will be published to.

10.28.3.3 std::string gazebo::physics::ContactManager::CreateFilter (const std::string & *_topic,* const std::string & *_collision*)

Create a filter for contacts.

A new publisher will be created that publishes contacts associated to the input collision. param[in] _name Filter name. param[in] _collision A collision name used for filtering.

Returns

New topic where filtered messages will be published to.

10.28.3.4 Contact* gazebo::physics::ContactManager::GetContact (unsigned int *_index*) const

Get a single contact by index.

The index must be between 0 and **ContactManager::GetContactCount** (p. 246).

Parameters

in	*_index*	Index of the **Contact** (p. 241) to return.

Returns

Pointer to a contact, NULL If index is invalid.

10.28.3.5 unsigned int gazebo::physics::ContactManager::GetContactCount ( ) const

Return the number of valid contacts.

10.28.3.6 const std::vector<Contact *>& gazebo::physics::ContactManager::GetContacts ( ) const

Get all the contacts.

The return vector may have invalid contacts. Only use contents of the vector between 0 and **ContactManager::GetContactCount** (p. 246)

Returns

Vector of contact pointers.

10.28.3.7 void gazebo::physics::ContactManager::Init (WorldPtr *_world*)

Initialize the **ContactManager** (p. 244).

This is required in order to publish contact messages via the **ContactManager::PublishContacts** (p. 247) method.

Parameters

in	*_world*	Pointer to the world that is initializing the contact manager.

10.28.3.8 Contact* gazebo::physics::ContactManager::NewContact (Collision * _collision1, Collision * _collision2, const common::Time & _time)

Add a new contact.

Noramlly this is only used by a Physics/Collision engine when a new contact is generated. All other users should just make use of the accessor functions.

If no one is listening, then the return value will be NULL. This is a signal to the Physics engine that it can skip the extra processing necessary to get back contact information.

Returns

The new contact. The physics engine should populate the contact's parameters. NULL will be returned if there are no subscribers to the contact topic.

10.28.3.9 void gazebo::physics::ContactManager::PublishContacts ( )

Publish all contacts in a msgs::Contacts message.

10.28.3.10 void gazebo::physics::ContactManager::ResetCount ( )

Set the contact count to zero.

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

- **ContactManager.hh**

10.29 gazebo::physics::ContactPublisher Class Reference

A custom contact publisher created for each contact filter in the **Contact** (p. 241) Manager.

`#include <ContactManager.hh>`

Public Attributes

- std::vector< std::string > **collisionNames**
- boost::unordered_set< **Collision** * > **collisions**

  Pointers of collisions monitored by contact manager for contacts.
- std::vector< **Contact** * > **contacts**

  A list of contacts associated to the collisions.
- **transport::PublisherPtr publisher**

  Contact *(p. 241) message publisher.*

10.29.1 Detailed Description

A custom contact publisher created for each contact filter in the **Contact** (p. 241) Manager.

10.29.2 Member Data Documentation

10.29.2.1 std::vector<std::string> gazebo::physics::ContactPublisher::collisionNames

10.29.2.2 boost::unordered_set<Collision *> gazebo::physics::ContactPublisher::collisions

Pointers of collisions monitored by contact manager for contacts.

10.29.2.3 std::vector<Contact *> gazebo::physics::ContactPublisher::contacts

A list of contacts associated to the collisions.

10.29.2.4 transport::PublisherPtr gazebo::physics::ContactPublisher::publisher

Contact (p. 241) message publisher.

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

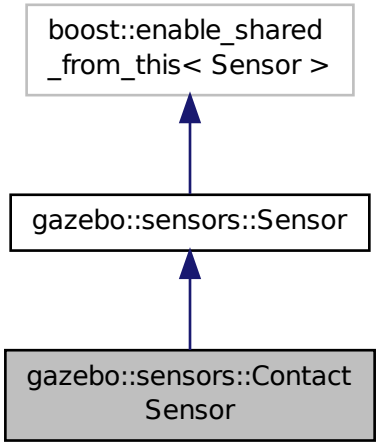
- **ContactManager.hh**

10.30 gazebo::sensors::ContactSensor Class Reference

Contact sensor.

`#include <sensors/sensors.hh>`

Inheritance diagram for gazebo::sensors::ContactSensor:



Public Member Functions

- **ContactSensor** ()

  Constructor.
- virtual ~**ContactSensor** ()

  Destructor.
- unsigned int **GetCollisionContactCount** (const std::string &_collisionName) const

  Return the number of contacts for an observed collision.
- unsigned int **GetCollisionCount** () const

  Get the number of collisions that the sensor is observing.
- std::string **GetCollisionName** (unsigned int _index) const

  Get a collision name at index _index.
- msgs::Contacts **GetContacts** () const

  *Get all the contacts for the **ContactSensor** (p. 248).*
- std::map< std::string, **physics::Contact** > **GetContacts** (const std::string &_collisionName)

  Gets contacts of a collision.
- virtual void **Init** ()

  Initialize the sensor.
- virtual bool **IsActive** ()

  Returns true if sensor generation is active.
- virtual void **Load** (const std::string &_worldName, **sdf::ElementPtr** _sdf)

  Load the sensor with SDF parameters.
- virtual void **Load** (const std::string &_worldName)

  Load the sensor with default parameters.

Protected Member Functions

- virtual void **Fini** ()

  Finalize the sensor.
- virtual void **UpdateImpl** (bool _force)

  Update the sensor information.

Additional Inherited Members

10.30.1 Detailed Description

Contact sensor.

This sensor detects and reports contacts between objects

10.30.2 Constructor & Destructor Documentation

10.30.2.1 gazebo::sensors::ContactSensor::ContactSensor ( )

Constructor.

10.30.2.2 virtual gazebo::sensors::ContactSensor::~ContactSensor ( ) [virtual]

Destructor.

10.30.3 Member Function Documentation

10.30.3.1 virtual void gazebo::sensors::ContactSensor::Fini ( ) [protected], [virtual]

Finalize the sensor.

Reimplemented from **gazebo::sensors::Sensor** (p. 735).

10.30.3.2 unsigned int gazebo::sensors::ContactSensor::GetCollisionContactCount (const std::string & *_collisionName*) const

Return the number of contacts for an observed collision.

Parameters

in	*_collisionName*	The name of the observed collision.

Returns

The collision contact count.

10.30.3.3 unsigned int gazebo::sensors::ContactSensor::GetCollisionCount ( ) const

Get the number of collisions that the sensor is observing.

Returns

Number of collisions.

10.30.3.4 std::string gazebo::sensors::ContactSensor::GetCollisionName (unsigned int *_index*) const

Get a collision name at index _index.

Parameters

in	*_index*	Index of collision in collection of collisions.

Returns

name of collision.

10.30.3.5 msgs::Contacts gazebo::sensors::ContactSensor::GetContacts ( ) const

Get all the contacts for the **ContactSensor** (p. 248).

Returns

Message that contains contact information between collision pairs.

During ODEPhysics::UpdateCollisions, all collision pairs in the world are pushed into a buffer within ContactManager. Subsequently, World::Update invokes ContactManager::PublishContacts to publish all contacts generated within a timestep onto Gazebo topic ~/physics/contacts.

Each **ContactSensor** (p. 248) subscribes to the Gazebo ~/physics/contacts topic, retrieves all contact pairs in a time step and filters them wthin ContactSensor::OnContacts against <collision> body name specified by the **ContactSensor** (p. 248) SDF. All collision pairs between **ContactSensor** (p. 248) <collision> body and other bodies in the world are stored in an array inside contacts.proto.

Within each element of the contact.proto array inside contacts.proto, list of collisions between collision bodies (collision1 and collision 2) are stored in an array of elements, (position, normal, depth, wrench). A timestamp has also been added (time). Details are described below:

- string collision1 name of the first collision object.
- string collision2 name of the second collision object.
- Vector3d position position of the contact joint in inertial frame.
- Vector3d normal normal of the contact joint in inertial frame.
- double depth intersection (penetration) depth of two collision bodies.
- JointWrench wrench Forces and torques acting on both collision bodies. See joint_wrench.proto for details. The forces and torques are applied at the CG of perspective links for each collision body, specified in the inertial frame.
- Time time time at which this contact happened.

10.30.3.6 std::map<std::string, physics::Contact> gazebo::sensors::ContactSensor::GetContacts (const std::string & _collisionName)

Gets contacts of a collision.

Parameters

in	*_collisionName*	Name of collision

Returns

Container of contacts

10.30.3.7 virtual void gazebo::sensors::ContactSensor::Init ( ) [virtual]

Initialize the sensor.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.30.3.8 virtual bool gazebo::sensors::ContactSensor::IsActive ( ) [virtual]

Returns true if sensor generation is active.

Returns

True if active, false if not.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.30.3.9 virtual void gazebo::sensors::ContactSensor::Load (const std::string & *_worldName,* sdf::ElementPtr *_sdf*) [virtual]

Load the sensor with SDF parameters.

Parameters

in	*_sdf*	SDF **Sensor** (p. 731) parameters
in	*_worldName*	Name of world to load from

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.30.3.10 virtual void gazebo::sensors::ContactSensor::Load (const std::string & *_worldName*) [virtual]

Load the sensor with default parameters.

Parameters

in	*_worldName*	Name of world to load from.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.30.3.11 virtual void gazebo::sensors::ContactSensor::UpdateImpl (bool *_force*) [protected], [virtual]

Update the sensor information.

Parameters

in	*_force*	True if update is forced, false if not.

Reimplemented from **gazebo::sensors::Sensor** (p. 739).

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

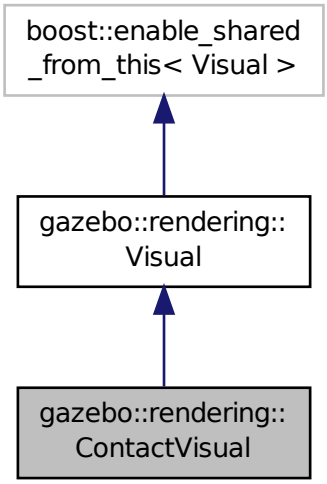
- **ContactSensor.hh**

10.31 gazebo::rendering::ContactVisual Class Reference

Contact visualization.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::ContactVisual:



Public Member Functions

- **ContactVisual** (const std::string &_name, **VisualPtr** _vis, const std::string &_topicName)

  Constructor.
- virtual ~**ContactVisual** ()

  Destructor.
- void **SetEnabled** (bool _enabled)

  Set to true to enable contact visualization.

Additional Inherited Members

10.31.1 Detailed Description

Contact visualization.

This class visualizes contact points by drawing arrows in the 3D environment.

10.31.2 Constructor & Destructor Documentation

10.31.2.1 gazebo::rendering::ContactVisual::ContactVisual (const std::string & *_name*, VisualPtr *_vis*, const std::string & *_topicName*)

Constructor.

Parameters

in	*_name*	Name of the **ContactVisual** (p. 252)
in	*_vis*	Pointer the parent **Visual** (p. 920)
in	*_topicName*	Name of the topic which publishes the contact information.

10.31.2.2 virtual gazebo::rendering::ContactVisual::~ContactVisual ( ) [virtual]

Destructor.

10.31.3 Member Function Documentation

10.31.3.1 void gazebo::rendering::ContactVisual::SetEnabled (bool *_enabled*)

Set to true to enable contact visualization.

Parameters

in	*_enabled*	True to show contacts, false to hide.

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

- **ContactVisual.hh**

10.32 gazebo::rendering::Conversions Class Reference

Conversions (p. 254) **Conversions.hh** (p. 1006) **rendering/Conversions.hh** (p. 1006).

```
#include <Conversions.hh>
```

Static Public Member Functions

- static Ogre::ColourValue **Convert** (const **common::Color** &_clr)
  Return the equivalent ogre color.
- static **common::Color Convert** (const Ogre::ColourValue &_clr)
  Return the equivalent gazebo color.
- static Ogre::Vector3 **Convert** (const **math::Vector3** &_v)
  *return **Ogre** (p. 110) Vector from Gazebo Vector3*
- static **math::Vector3 Convert** (const Ogre::Vector3 &_v)
  return gazebo Vector from ogre Vector3
- static Ogre::Quaternion **Convert** (const **math::Quaternion** &_v)
  *Gazebo quaternion to **Ogre** (p. 110) quaternion.*
- static **math::Quaternion Convert** (const Ogre::Quaternion &_v)
  ***Ogre** (p. 110) quaternion to Gazebo quaternion.*

10.32.1 Detailed Description

Conversions (p. 254) **Conversions.hh** (p. 1006) **rendering/Conversions.hh** (p. 1006).

A set of utility function to convert between Gazebo and **Ogre** (p. 110) data types

10.32.2 Member Function Documentation

10.32.2.1 static Ogre::ColourValue gazebo::rendering::Conversions::Convert (const common::Color & _clr) [static]

Return the equivalent ogre color.

Parameters

in	*_clr*	Gazebo color to convert

Returns

Ogre (p. 110) color value

10.32.2.2 static common::Color gazebo::rendering::Conversions::Convert (const Ogre::ColourValue & _clr) [static]

Return the equivalent gazebo color.

Parameters

in	*_clr*	**Ogre** (p. 110) color to convert

Returns

Gazebo color value

10.32.2.3 static Ogre::Vector3 gazebo::rendering::Conversions::Convert (const math::Vector3 & _v) [static]

return **Ogre** (p. 110) Vector from Gazebo Vector3

Parameters

in	*_v*	Gazebo vector

Returns

Ogre (p. 110) vector

10.32.2.4 static math::Vector3 gazebo::rendering::Conversions::Convert (const Ogre::Vector3 & _v) [static]

return gazebo Vector from ogre Vector3

Parameters

in	*_v*	**Ogre** (p. 110) vector

Returns

Gazebo vector

10.32.2.5 static Ogre::Quaternion gazebo::rendering::Conversions::Convert (const math::Quaternion & _v) [static]

Gazebo quaternion to **Ogre** (p. 110) quaternion.

Parameters

in	_v	Gazebo quaternion

Returns

Ogre (p. 110) quaternion

10.32.2.6 static math::Quaternion gazebo::rendering::Conversions::Convert (const Ogre::Quaternion & _v) [static]

Ogre (p. 110) quaternion to Gazebo quaternion.

Parameters

in	_v	**Ogre** (p. 110) quaternion return Gazebo quaternion

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

- **Conversions.hh**

10.33 sdf::Converter Class Reference

Convert from one version of **SDF** (p. 728) to another.

`#include <Converter.hh>`

Static Public Member Functions

- static bool **Convert** (TiXmlDocument *_doc, const std::string &_toVersion, bool _quiet=false)
  *Convert **SDF** (p. 728) to the specified version.*
- static void **Convert** (TiXmlDocument *_doc, TiXmlDocument *_convertDoc)
  This is an internal function.

10.33.1 Detailed Description

Convert from one version of **SDF** (p. 728) to another.

10.33.2 Member Function Documentation

10.33.2.1 static bool sdf::Converter::Convert (TiXmlDocument * _doc, const std::string & _toVersion, bool _quiet = false) [static]

Convert **SDF** (p. 728) to the specified version.

Parameters

in	*_doc*	**SDF** (p. 728) xml doc
in	*_toVersion*	Version number in string format
in	*_quiet*	False to be more verbose

10.33.2.2 static void sdf::Converter::Convert (TiXmlDocument * *_doc*, TiXmlDocument * *_convertDoc*) [static]

This is an internal function.

Generic convert function that converts the **SDF** (p. 728) based on the given Convert file.

Parameters

in	*_doc*	**SDF** (p. 728) xml doc
in	*_convertDoc*	Convert xml doc

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

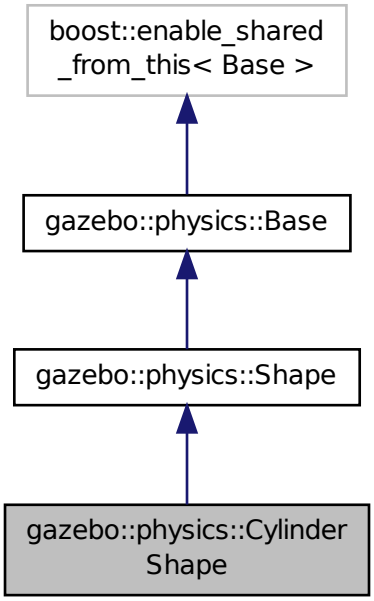
- **Converter.hh**

10.34 gazebo::physics::CylinderShape Class Reference

Cylinder collision.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::CylinderShape:



Public Member Functions

- **CylinderShape** (**CollisionPtr** _parent)
  Constructor.
- virtual ~**CylinderShape** ()
  Destructor.
- void **FillMsg** (msgs::Geometry &_msg)
  Fill in the values for a geomertry message.
- double **GetLength** () const
  Get length.
- double **GetRadius** () const
  Get radius.
- void **Init** ()
  Initialize the cylinder.
- virtual void **ProcessMsg** (const msgs::Geometry &_msg)
  Update values based on a message.
- void **SetLength** (double _length)
  Set length.
- void **SetRadius** (double _radius)
  Set radius.
- virtual void **SetSize** (double _radius, double _length)
  Set the size of the cylinder.

Additional Inherited Members

10.34.1 Detailed Description

Cylinder collision.

10.34.2 Constructor & Destructor Documentation

10.34.2.1 gazebo::physics::CylinderShape::CylinderShape (CollisionPtr *_parent*) [explicit]

Constructor.

Parameters

in	*_parent*	Parent of the shape.

10.34.2.2 virtual gazebo::physics::CylinderShape::~CylinderShape ( ) [virtual]

Destructor.

10.34.3 Member Function Documentation

10.34.3.1 void gazebo::physics::CylinderShape::FillMsg (msgs::Geometry & *_msg*) [virtual]

Fill in the values for a geomertry message.

Parameters

out	*_msg*	The geometry message to fill.

Implements **gazebo::physics::Shape** (p. 756).

10.34.3.2 double gazebo::physics::CylinderShape::GetLength ( ) const

Get length.

Returns

The cylinder length.

10.34.3.3 double gazebo::physics::CylinderShape::GetRadius ( ) const

Get radius.

Returns

The cylinder radius.

10.34.3.4 void gazebo::physics::CylinderShape::Init ( ) [virtual]

Initialize the cylinder.

Implements **gazebo::physics::Shape** (p. 756).

10.34.3.5 virtual void gazebo::physics::CylinderShape::ProcessMsg (const msgs::Geometry & *_msg*) [virtual]

Update values based on a message.

Parameters

in	*_msg*	Message to update from.

Implements **gazebo::physics::Shape** (p. 756).

10.34.3.6 void gazebo::physics::CylinderShape::SetLength (double *_length*)

Set length.

Parameters

in	*_length*	New length of the cylinder.

10.34.3.7 void gazebo::physics::CylinderShape::SetRadius (double *_radius*)

Set radius.

Parameters

in}	_radius New radius of the cylinder.

10.34.3.8 virtual void gazebo::physics::CylinderShape::SetSize (double *_radius,* double *_length*) [virtual]

Set the size of the cylinder.

Parameters

in	*_radius*	New radius.
in	*_lenght*	New length.

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

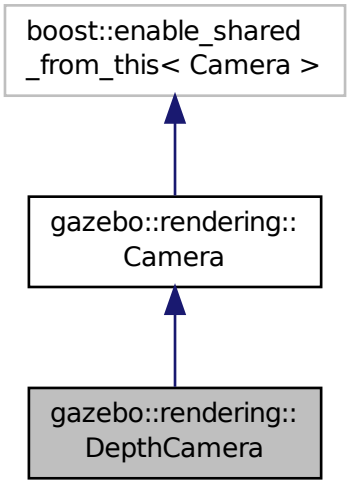
- **CylinderShape.hh**

10.35 gazebo::rendering::DepthCamera Class Reference

Depth camera used to render depth data into an image buffer.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::DepthCamera:



Public Member Functions

- **DepthCamera** (const std::string &_namePrefix, **ScenePtr** _scene, bool _autoRender=true)

  Constructor.
- virtual ~**DepthCamera** ()

  Destructor.
- template<typename T >
  event::ConnectionPtr ConnectNewDepthFrame (T _subscriber)

  Connect a to the new depth image signal.
- template<typename T >
  event::ConnectionPtr ConnectNewRGBPointCloud (T _subscriber)

  Connect a to the new rgb point cloud signal.
- void **CreateDepthTexture** (const std::string &_textureName)

  Create a texture which will hold the depth data.
- void **DisconnectNewDepthFrame** (**event::ConnectionPtr** &_c)

  Disconnect from an depth image singal.
- void **DisconnectNewRGBPointCloud** (**event::ConnectionPtr** &c)

  Disconnect from an rgb point cloud singal.
- void **Fini** ()

  Finalize the camera.
- virtual const float * **GetDepthData** ()

  All things needed to get back z buffer for depth data.
- void **Init** ()

  Initialize the camera.
- void **Load** (**sdf::ElementPtr** &_sdf)

Load the camera with a set of parmeters.

- void **Load** ()

  Load the camera with default parmeters.

- virtual void **PostRender** ()

  Render the camera.

- virtual void **SetDepthTarget** (Ogre::RenderTarget ∗_target)

  Set the render target, which renders the depth data.

Protected Attributes

- Ogre::RenderTarget ∗ **depthTarget**

  Pointer to the depth target.

- Ogre::Texture ∗ **depthTexture**

  Pointer to the depth texture.

- Ogre::Viewport ∗ **depthViewport**

  Pointer to the depth viewport.

Additional Inherited Members

10.35.1 Detailed Description

Depth camera used to render depth data into an image buffer.

10.35.2 Constructor & Destructor Documentation

10.35.2.1 gazebo::rendering::DepthCamera::DepthCamera (const std::string & *_namePrefix,* ScenePtr *_scene,* bool *_autoRender* = `true`)

Constructor.

Parameters

in	*_namePrefix*	Unique prefix name for the camera.
in	*_scene*	**Scene** (p. 707) that will contain the camera
in	*_autoRender*	Almost everyone should leave this as true.

10.35.2.2 virtual gazebo::rendering::DepthCamera::∼DepthCamera ( ) `[virtual]`

Destructor.

10.35.3 Member Function Documentation

10.35.3.1 template<typename T > event::ConnectionPtr gazebo::rendering::DepthCamera::ConnectNewDepthFrame (T *_subscriber*) `[inline]`

Connect a to the new depth image signal.

Parameters

in	*_subscriber*	Subscriber callback function

Returns

Pointer to the new Connection. This must be kept in scope

References gazebo::event::EventT< T >::Connect().

10.35.3.2 template<typename T > event::ConnectionPtr gazebo::rendering::DepthCamera::ConnectNewRGBPointCloud (T *_subscriber*) [inline]

Connect a to the new rgb point cloud signal.

Parameters

in	*_subscriber*	Subscriber callback function

Returns

Pointer to the new Connection. This must be kept in scope

References gazebo::event::EventT< T >::Connect().

10.35.3.3 void gazebo::rendering::DepthCamera::CreateDepthTexture (const std::string & *_textureName*)

Create a texture which will hold the depth data.

Parameters

in	*_textureName*	Name of the texture to create

10.35.3.4 void gazebo::rendering::DepthCamera::DisconnectNewDepthFrame (event::ConnectionPtr & *_c*) [inline]

Disconnect from an depth image singal.

Parameters

in	*_c*	The connection to disconnect

References gazebo::event::EventT< T >::Disconnect().

10.35.3.5 void gazebo::rendering::DepthCamera::DisconnectNewRGBPointCloud (event::ConnectionPtr & *c*) [inline]

Disconnect from an rgb point cloud singal.

Parameters

in	*_c*	The connection to disconnect

References gazebo::event::EventT< T >::Disconnect().

10.35.3.6 void gazebo::rendering::DepthCamera::Fini ( ) [virtual]

Finalize the camera.

Reimplemented from **gazebo::rendering::Camera** (p. 175).

10.35.3.7 virtual const float* gazebo::rendering::DepthCamera::GetDepthData ( ) [virtual]

All things needed to get back z buffer for depth data.

Returns

The z-buffer as a float array

10.35.3.8 void gazebo::rendering::DepthCamera::Init ( ) [virtual]

Initialize the camera.

Reimplemented from **gazebo::rendering::Camera** (p. 182).

10.35.3.9 void gazebo::rendering::DepthCamera::Load (sdf::ElementPtr & *_sdf*)

Load the camera with a set of parmeters.

Parameters

in	*_sdf*	The SDF camera info

10.35.3.10 void gazebo::rendering::DepthCamera::Load ( ) [virtual]

Load the camera with default parmeters.

Reimplemented from **gazebo::rendering::Camera** (p. 183).

10.35.3.11 virtual void gazebo::rendering::DepthCamera::PostRender ( ) [virtual]

Render the camera.

Reimplemented from **gazebo::rendering::Camera** (p. 184).

10.35.3.12 virtual void gazebo::rendering::DepthCamera::SetDepthTarget (Ogre::RenderTarget * *_target*) [virtual]

Set the render target, which renders the depth data.

Parameters

in	*_target*	Pointer to the render target

10.35.4 Member Data Documentation

10.35.4.1 Ogre::RenderTarget* gazebo::rendering::DepthCamera::depthTarget [protected]

Pointer to the depth target.

10.35.4.2 Ogre::Texture* gazebo::rendering::DepthCamera::depthTexture [protected]

Pointer to the depth texture.

10.35.4.3 Ogre::Viewport* gazebo::rendering::DepthCamera::depthViewport [protected]

Pointer to the depth viewport.

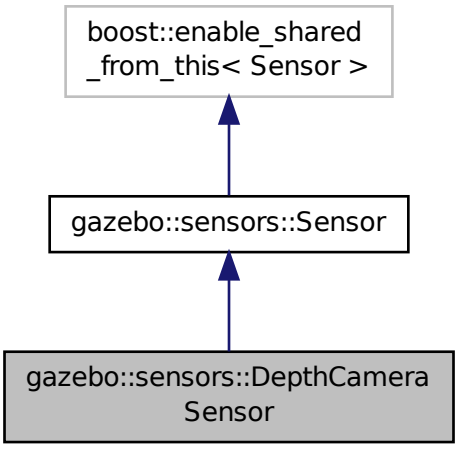
The documentation for this class was generated from the following file:

- **DepthCamera.hh**

10.36 gazebo::sensors::DepthCameraSensor Class Reference

`#include <sensors/sensors.hh>`

Inheritance diagram for gazebo::sensors::DepthCameraSensor:



Public Member Functions

- **DepthCameraSensor** ()

  Constructor.

- virtual ~**DepthCameraSensor** ()

  Destructor.
- **rendering::DepthCameraPtr GetDepthCamera** () const

  *Returns a pointer to the **rendering::DepthCamera** (p. 260).*
- bool **SaveFrame** (const std::string &_filename)

  Saves an image frame of depth camera sensor to file.
- virtual void **SetActive** (bool _value)

  Set whether the sensor is active or not.
- virtual void **SetParent** (const std::string &_name)

  Set the parent of the sensor.

Protected Member Functions

- virtual void **Fini** ()

  Finalize the camera.
- virtual void **Init** ()

  Initialize the camera.
- virtual void **Load** (const std::string &_worldName, **sdf::ElementPtr** &_sdf)

  Load the sensor with SDF parameters.
- virtual void **Load** (const std::string &_worldName)

  Load the sensor with default parameters.
- virtual void **UpdateImpl** (bool _force)

  Update the sensor information.

Additional Inherited Members

10.36.1 Constructor & Destructor Documentation

10.36.1.1 gazebo::sensors::DepthCameraSensor::DepthCameraSensor ( )

Constructor.

10.36.1.2 virtual gazebo::sensors::DepthCameraSensor::~DepthCameraSensor ( ) [virtual]

Destructor.

10.36.2 Member Function Documentation

10.36.2.1 virtual void gazebo::sensors::DepthCameraSensor::Fini ( ) [protected], [virtual]

Finalize the camera.

Reimplemented from **gazebo::sensors::Sensor** (p. 735).

10.36.2.2 rendering::DepthCameraPtr gazebo::sensors::DepthCameraSensor::GetDepthCamera ( ) const [inline]

Returns a pointer to the **rendering::DepthCamera** (p. 260).

Returns

Depth Camera pointer

10.36.2.3 virtual void gazebo::sensors::DepthCameraSensor::Init ( ) [protected], [virtual]

Initialize the camera.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.36.2.4 virtual void gazebo::sensors::DepthCameraSensor::Load (const std::string & *_worldName,* sdf::ElementPtr & *_sdf*) [protected], [virtual]

Load the sensor with SDF parameters.

Parameters

in	*_sdf*	SDF **Sensor** (p. 731) parameters
in	*_worldName*	Name of world to load from

10.36.2.5 virtual void gazebo::sensors::DepthCameraSensor::Load (const std::string & *_worldName*) [protected], [virtual]

Load the sensor with default parameters.

Parameters

in	*_worldName*	Name of world to load from

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.36.2.6 bool gazebo::sensors::DepthCameraSensor::SaveFrame (const std::string & *_filename*)

Saves an image frame of depth camera sensor to file.

Parameters

in	*Name*	of file to save as

Returns

True if saved, false if not

10.36.2.7 virtual void gazebo::sensors::DepthCameraSensor::SetActive (bool *_value*) [virtual]

Set whether the sensor is active or not.

Parameters

in	*_value*	True if active, false if not

Reimplemented from **gazebo::sensors::Sensor** (p. 739).

10.36.2.8 virtual void gazebo::sensors::DepthCameraSensor::SetParent (const std::string & *_name*) [virtual]

Set the parent of the sensor.

Parameters

in	*_name*	Name of parent

Reimplemented from **gazebo::sensors::Sensor** (p. 739).

10.36.2.9 virtual void gazebo::sensors::DepthCameraSensor::UpdateImpl (bool *_force*) [protected], [virtual]

Update the sensor information.

Parameters

in	*_force*	True if update is forced, false if not

Reimplemented from **gazebo::sensors::Sensor** (p. 739).

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

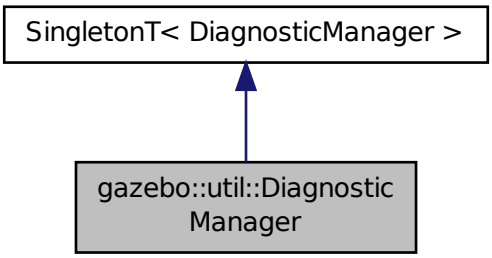
- **DepthCameraSensor.hh**

10.37 gazebo::util::DiagnosticManager Class Reference

A diagnostic manager class.

`#include <util/util.hh>`

Inheritance diagram for gazebo::util::DiagnosticManager:



Public Member Functions

- std::string **GetLabel** (int _index) const

  Get a label for a timer.
- boost::filesystem::path **GetLogPath** () const

  Get the path in which logs are stored.
- **common::Time GetTime** (int _index) const

  Get the time of a timer instance.
- **common::Time GetTime** (const std::string &_label) const

  Get a time based on a label.
- int **GetTimerCount** () const

  Get the number of timers.
- void **Init** (const std::string &_worldName)

  Initialize to report diagnostics about a world.
- void **Lap** (const std::string &_name, const std::string &_prefix)

  Output the current elapsed time of an active timer with a prefix string.
- void **StartTimer** (const std::string &_name)

  Start a new timer instance.
- void **StopTimer** (const std::string &_name)

  Stop a currently running timer.

Additional Inherited Members

10.37.1 Detailed Description

A diagnostic manager class.

10.37.2 Member Function Documentation

10.37.2.1 std::string gazebo::util::DiagnosticManager::GetLabel (int *_index*) const

Get a label for a timer.

Parameters

in	*_index*	Index of a timer instance

Returns

Label of the specified timer

10.37.2.2 boost::filesystem::path gazebo::util::DiagnosticManager::GetLogPath ( ) const

Get the path in which logs are stored.

Returns

The path in which logs are stored.

10.37.2.3 common::Time gazebo::util::DiagnosticManager::GetTime (int _index) const

Get the time of a timer instance.

Parameters

in	*_index*	The index of a timer instance

Returns

Time of the specified timer

10.37.2.4 common::Time gazebo::util::DiagnosticManager::GetTime (const std::string & _label) const

Get a time based on a label.

Parameters

in	*_label*	Name of the timer instance

Returns

Time of the specified timer

10.37.2.5 int gazebo::util::DiagnosticManager::GetTimerCount ( ) const

Get the number of timers.

Returns

The number of timers

10.37.2.6 void gazebo::util::DiagnosticManager::Init (const std::string & _worldName)

Initialize to report diagnostics about a world.

Parameters

in	*_worldName*	Name of the world.

10.37.2.7 void gazebo::util::DiagnosticManager::Lap (const std::string & _name, const std::string & _prefix)

Output the current elapsed time of an active timer with a prefix string.

This also resets the timer and keeps it running.

Parameters

in	*_name*	Name of the timer to access.
in	*_prefix*	Informational string that is output with the elapsed time.

10.37.2.8 void gazebo::util::DiagnosticManager::StartTimer (const std::string & _name)

Start a new timer instance.

Parameters

in	*_name*	Name of the timer.

Returns

A pointer to the new diagnostic timer

10.37.2.9 void gazebo::util::DiagnosticManager::StopTimer (const std::string & _name)

Stop a currently running timer.

Parameters

in	*_name*	Name of the timer to stop.

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

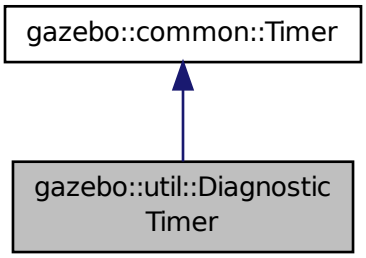
- **Diagnostics.hh**

10.38 gazebo::util::DiagnosticTimer Class Reference

A timer designed for diagnostics.

`#include <util/util.hh>`

Inheritance diagram for gazebo::util::DiagnosticTimer:



Public Member Functions

- **DiagnosticTimer** (const std::string &_name)

Constructor.

- virtual ~**DiagnosticTimer** ()

  Destructor.

- const std::string **GetName** () const

  Get the name of the timer.

- void **Lap** (const std::string &_prefix)

  Output a lap time.

- virtual void **Start** ()

  Start the timer.

- virtual void **Stop** ()

  Stop the timer.

10.38.1 Detailed Description

A timer designed for diagnostics.

10.38.2 Constructor & Destructor Documentation

10.38.2.1 gazebo::util::DiagnosticTimer::DiagnosticTimer (const std::string & *_name*)

Constructor.

Parameters

in	*_name*	Name of the timer

10.38.2.2 virtual gazebo::util::DiagnosticTimer::~DiagnosticTimer ( ) [virtual]

Destructor.

10.38.3 Member Function Documentation

10.38.3.1 const std::string gazebo::util::DiagnosticTimer::GetName ( ) const [inline]

Get the name of the timer.

Returns

The name of timer

10.38.3.2 void gazebo::util::DiagnosticTimer::Lap (const std::string & *_prefix*)

Output a lap time.

Parameters

in	*_prefix*	Annotation to output with the elapsed time.

10.38.3.3 virtual void gazebo::util::DiagnosticTimer::Start ( ) [virtual]

Start the timer.

Reimplemented from **gazebo::common::Timer** (p. 854).

10.38.3.4 virtual void gazebo::util::DiagnosticTimer::Stop ( ) [virtual]

Stop the timer.

Reimplemented from **gazebo::common::Timer** (p. 854).

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

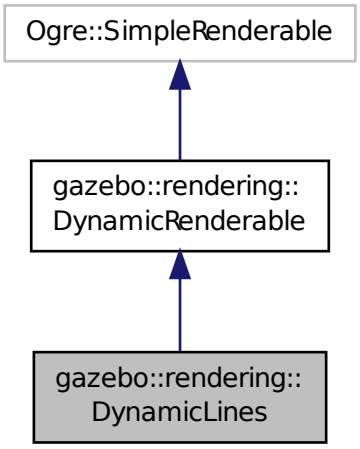
- **Diagnostics.hh**

10.39 gazebo::rendering::DynamicLines Class Reference

Class for drawing lines that can change.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::DynamicLines:



Public Member Functions

- **DynamicLines** (**RenderOpType** _opType=**RENDERING_LINE_STRIP**)

  Constructor.
- virtual ~**DynamicLines** ()

  Destructor.

- void **AddPoint** (const **math::Vector3** &_pt)

  Add a point to the point list.
- void **AddPoint** (double _x, double _y, double _z)

  Add a point to the point list.
- void **Clear** ()

  Remove all points from the point list.
- virtual const Ogre::String & **getMovableType** () const

  *Overridden function from **Ogre** (p. 110)'s base class.*
- const **math::Vector3** & **GetPoint** (unsigned int _index) const

  Return the location of an existing point in the point list.
- unsigned int **GetPointCount** () const

  Return the total number of points in the point list.
- void **SetPoint** (unsigned int _index, const **math::Vector3** &_value)

  Change the location of an existing point in the point list.
- void **Update** ()

  Call this to update the hardware buffer after making changes.

Static Public Member Functions

- static std::string **GetMovableType** ()

  Get type of movable.

Protected Member Functions

- virtual void **CreateVertexDeclaration** ()

  *Implementation **DynamicRenderable** (p. 276), creates a simple vertex-only decl.*
- virtual void **FillHardwareBuffers** ()

  *Implementation **DynamicRenderable** (p. 276), pushes point list out to hardware memory.*

Additional Inherited Members

10.39.1 Detailed Description

Class for drawing lines that can change.

10.39.2 Constructor & Destructor Documentation

10.39.2.1 gazebo::rendering::DynamicLines::DynamicLines (RenderOpType *_opType* = RENDERING_LINE_STRIP)

Constructor.

Parameters

in	*_opType*	The type of Line

10.39.2.2 virtual gazebo::rendering::DynamicLines::~DynamicLines ( ) [virtual]

Destructor.

10.39.3 Member Function Documentation

10.39.3.1 void gazebo::rendering::DynamicLines::AddPoint (const math::Vector3 & _pt)

Add a point to the point list.

Parameters

in	*pt*	**math::Vector3** (p. 890) point

10.39.3.2 void gazebo::rendering::DynamicLines::AddPoint (double _x, double _y, double _z)

Add a point to the point list.

Parameters

in	*_x*	X position.
in	*_y*	Y position.
in	*_z*	Z position.

10.39.3.3 void gazebo::rendering::DynamicLines::Clear ( )

Remove all points from the point list.

10.39.3.4 virtual void gazebo::rendering::DynamicLines::CreateVertexDeclaration ( ) [protected], [virtual]

Implementation **DynamicRenderable** (p. 276), creates a simple vertex-only decl.

Implements **gazebo::rendering::DynamicRenderable** (p. 278).

10.39.3.5 virtual void gazebo::rendering::DynamicLines::FillHardwareBuffers ( ) [protected], [virtual]

Implementation **DynamicRenderable** (p. 276), pushes point list out to hardware memory.

Implements **gazebo::rendering::DynamicRenderable** (p. 278).

10.39.3.6 static std::string gazebo::rendering::DynamicLines::GetMovableType ( ) [static]

Get type of movable.

Returns

This returns "gazebo::dynamiclines"

10.39.3.7 virtual const Ogre::String& gazebo::rendering::DynamicLines::getMovableType ( ) const [virtual]

Overridden function from **Ogre** (p. 110)'s base class.

Returns

Returns "gazebo::ogredynamicslines"

10.39.3.8 const math::Vector3& gazebo::rendering::DynamicLines::GetPoint (unsigned int _index) const

Return the location of an existing point in the point list.

Parameters

in	*_index*	Number of the point to return

Returns

math::Vector3 (p. 890) value of the point

10.39.3.9 unsigned int gazebo::rendering::DynamicLines::GetPointCount ( ) const

Return the total number of points in the point list.

Returns

Number of points

10.39.3.10 void gazebo::rendering::DynamicLines::SetPoint (unsigned int _index, const math::Vector3 & _value)

Change the location of an existing point in the point list.

Parameters

in	*_index*	Index of the point to set
in	*_value*	**math::Vector3** (p. 890) value to set the point to

10.39.3.11 void gazebo::rendering::DynamicLines::Update ( )

Call this to update the hardware buffer after making changes.

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

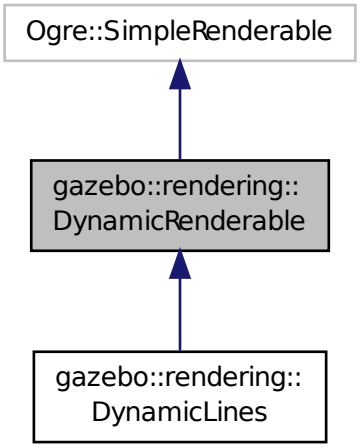
- **DynamicLines.hh**

10.40 gazebo::rendering::DynamicRenderable Class Reference

Abstract base class providing mechanisms for dynamically growing hardware buffers.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::DynamicRenderable:



Public Member Functions

- **DynamicRenderable** ()
  Constructor.
- virtual ~**DynamicRenderable** ()
  Virtual destructor.
- virtual Ogre::Real **getBoundingRadius** () const
  Implementation of Ogre::SimpleRenderable.
- **RenderOpType GetOperationType** () const
  Get the render operation type.
- virtual Ogre::Real **getSquaredViewDepth** (const Ogre::Camera *_cam) const
  Implementation of Ogre::SimpleRenderable.
- void **Init** (**RenderOpType** _opType, bool _useIndices=false)
  Initializes the dynamic renderable.
- void **SetOperationType** (**RenderOpType** _opType)
  Set the render operation type.

Protected Member Functions

- virtual void **CreateVertexDeclaration** ()=0
  Creates the vertex declaration.
- virtual void **FillHardwareBuffers** ()=0
  Fills the hardware vertex and index buffers with data.
- void **PrepareHardwareBuffers** (size_t _vertexCount, size_t _indexCount)
  Prepares the hardware buffers for the requested vertex and index counts.

Protected Attributes

- size_t **indexBufferCapacity**

  Maximum capacity of the currently allocated index buffer.
- size_t **vertexBufferCapacity**

  Maximum capacity of the currently allocated vertex buffer.

10.40.1 Detailed Description

Abstract base class providing mechanisms for dynamically growing hardware buffers.

10.40.2 Constructor & Destructor Documentation

10.40.2.1 gazebo::rendering::DynamicRenderable::DynamicRenderable ( )

Constructor.

10.40.2.2 virtual gazebo::rendering::DynamicRenderable::~DynamicRenderable ( ) [virtual]

Virtual destructor.

10.40.3 Member Function Documentation

10.40.3.1 virtual void gazebo::rendering::DynamicRenderable::CreateVertexDeclaration ( ) [protected], [pure virtual]

Creates the vertex declaration.

Remarks

Override and set mRenderOp.vertexData->vertexDeclaration here. mRenderOp.vertexData will be created for you before this method is called.

Implemented in **gazebo::rendering::DynamicLines** (p. 275).

10.40.3.2 virtual void gazebo::rendering::DynamicRenderable::FillHardwareBuffers ( ) [protected], [pure virtual]

Fills the hardware vertex and index buffers with data.

Remarks

This function must call prepareHardwareBuffers() before locking the buffers to ensure the they are large enough for the data to be written. Afterwards the vertex and index buffers (if using indices) can be locked, and data can be written to them.

Implemented in **gazebo::rendering::DynamicLines** (p. 275).

10.40.3.3 virtual Ogre::Real gazebo::rendering::DynamicRenderable::getBoundingRadius ( ) const [virtual]

Implementation of Ogre::SimpleRenderable.

Returns

The bounding radius

10.40.3.4 RenderOpType gazebo::rendering::DynamicRenderable::GetOperationType ( ) const

Get the render operation type.

Returns

The render operation type.

10.40.3.5 virtual Ogre::Real gazebo::rendering::DynamicRenderable::getSquaredViewDepth (const Ogre::Camera * _cam) const [virtual]

Implementation of Ogre::SimpleRenderable.

Parameters

in	*_cam*	Pointer to the **Ogre** (p. 110) camera that views the renderable.

Returns

The squared depth in the **Camera** (p. 166)'s view

10.40.3.6 void gazebo::rendering::DynamicRenderable::Init (RenderOpType _opType, bool _useIndices = false)

Initializes the dynamic renderable.

Remarks

This function should only be called once. It initializes the render operation, and calls the abstract function **CreateVertexDeclaration()** (p. 278).

Parameters

in	*_opType*	The type of render operation to perform.
in	*_useIndices*	Specifies whether to use indices to determine the vertices to use as input.

10.40.3.7 void gazebo::rendering::DynamicRenderable::PrepareHardwareBuffers (size_t _vertexCount, size_t _indexCount) [protected]

Prepares the hardware buffers for the requested vertex and index counts.

Remarks

This function must be called before locking the buffers in fillHardwareBuffers(). It guarantees that the hardware buffers are large enough to hold at least the requested number of vertices and indices (if using indices). The buffers are possibly reallocated to achieve this.

The vertex and index count in the render operation are set to

the values of vertexCount and indexCount respectively.

Parameters

in	*_vertexCount*	The number of vertices the buffer must hold.
in	*_indexCount*	The number of indices the buffer must hold. This parameter is ignored if not using indices.

10.40.3.8 void gazebo::rendering::DynamicRenderable::SetOperationType (RenderOpType *_opType*)

Set the render operation type.

Parameters

in	*_opType*	The type of render operation to perform.

10.40.4 Member Data Documentation

10.40.4.1 size_t gazebo::rendering::DynamicRenderable::indexBufferCapacity [protected]

Maximum capacity of the currently allocated index buffer.

10.40.4.2 size_t gazebo::rendering::DynamicRenderable::vertexBufferCapacity [protected]

Maximum capacity of the currently allocated vertex buffer.

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

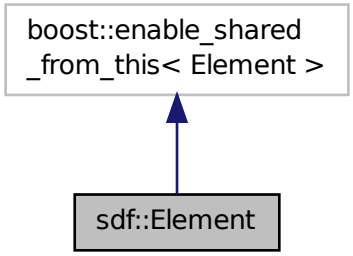
- **DynamicRenderable.hh**

10.41 sdf::Element Class Reference

SDF (p. 728) **Element** (p. 280) class.

`#include <SDF.hh>`

Inheritance diagram for sdf::Element:



Public Member Functions

- **Element** ()
- virtual ~**Element** () **GAZEBO_DEPRECATED**(1.6)
- void **AddAttribute** (const std::string &_key, const std::string &_type, const std::string &_defaultvalue, bool _required, const std::string &_description="") **GAZEBO_DEPRECATED**(1.6)
- **ElementPtr AddElement** (const std::string &_name) **GAZEBO_DEPRECATED**(1.6)
- void **AddElementDescription** (**ElementPtr** _elem) **GAZEBO_DEPRECATED**(1.6)

  Add a new element description.
- void **AddValue** (const std::string &_type, const std::string &_defaultValue, bool _required, const std::string &_description="") **GAZEBO_DEPRECATED**(1.6)
- void **ClearElements** () **GAZEBO_DEPRECATED**(1.6)

  Remove all child elements.
- boost::shared_ptr< **Element** > **Clone** () const **GAZEBO_DEPRECATED**(1.6)
- void **Copy** (const **ElementPtr** _elem) **GAZEBO_DEPRECATED**(1.6)

  *Copy values from an **Element** (p. 280).*
- template<typename T >
  T **Get** (const std::string &_key="")
- **ParamPtr GetAttribute** (const std::string &_key) **GAZEBO_DEPRECATED**(1.6)

  Get the param of an attribute.
- **ParamPtr GetAttribute** (unsigned int _index) const **GAZEBO_DEPRECATED**(1.6)

  Get an attribute using an index.
- unsigned int **GetAttributeCount** () const **GAZEBO_DEPRECATED**(1.6)

  Get the number of attributes.
- bool **GetAttributeSet** (const std::string &_key) **GAZEBO_DEPRECATED**(1.6)

  Return true if the attribute was set (i.e. not default value)
- bool **GetCopyChildren** () const **GAZEBO_DEPRECATED**(1.6)
- std::string **GetDescription** () const **GAZEBO_DEPRECATED**(1.6)

  Get a text description of the element.
- **ElementPtr GetElement** (const std::string &_name) const **GAZEBO_DEPRECATED**(1.6)
- **ElementPtr GetElement** (const std::string &_name) **GAZEBO_DEPRECATED**(1.6)
- **ElementPtr GetElementDescription** (unsigned int _index) const **GAZEBO_DEPRECATED**(1.6)

Get an element description using an index.

- **ElementPtr GetElementDescription** (const std::string &_key) const **GAZEBO_DEPRECATED**(1.6)

  Get an element descriptio using a key.

- unsigned int **GetElementDescriptionCount** () const **GAZEBO_DEPRECATED**(1.6)

  Get the number of element descriptions.

- **ElementPtr GetElementImpl** (const std::string &_name) const **GAZEBO_DEPRECATED**(1.6)
- **ElementPtr GetFirstElement** () const **GAZEBO_DEPRECATED**(1.6)
- std::string **GetInclude** () const **GAZEBO_DEPRECATED**(1.6)
- const std::string & **GetName** () const **GAZEBO_DEPRECATED**(1.6)
- **ElementPtr GetNextElement** (const std::string &_name="") const **GAZEBO_DEPRECATED**(1.6)
- **ElementPtr GetParent** () const **GAZEBO_DEPRECATED**(1.6)
- const std::string & **GetRequired** () const **GAZEBO_DEPRECATED**(1.6)
- **ParamPtr GetValue** () **GAZEBO_DEPRECATED**(1.6)

  Get the param of the elements value.

- bool **GetValueBool** (const std::string &_key="") **GAZEBO_DEPRECATED**(1.6)
- char **GetValueChar** (const std::string &_key="") **GAZEBO_DEPRECATED**(1.6)
- **gazebo::common::Color GetValueColor** (const std::string &_key="") **GAZEBO_DEPRECATED**(1.6)
- double **GetValueDouble** (const std::string &_key="") **GAZEBO_DEPRECATED**(1.6)
- float **GetValueFloat** (const std::string &_key="") **GAZEBO_DEPRECATED**(1.6)
- int **GetValueInt** (const std::string &_key="") **GAZEBO_DEPRECATED**(1.6)
- **gazebo::math::Pose GetValuePose** (const std::string &_key="") **GAZEBO_DEPRECATED**(1.6)
- **gazebo::math::Quaternion GetValueQuaternion** (const std::string &_key="") **GAZEBO_DEPRECATED**(1.6)
- std::string **GetValueString** (const std::string &_key="") **GAZEBO_DEPRECATED**(1.6)
- **gazebo::common::Time GetValueTime** (const std::string &_key="") **GAZEBO_DEPRECATED**(1.6)
- unsigned int **GetValueUInt** (const std::string &_key="") **GAZEBO_DEPRECATED**(1.6)
- **gazebo::math::Vector2d GetValueVector2d** (const std::string &_key="") **GAZEBO_DEPRECATED**(1.6)
- **gazebo::math::Vector3 GetValueVector3** (const std::string &_key="") **GAZEBO_DEPRECATED**(1.6)
- bool **HasAttribute** (const std::string &_key) **GAZEBO_DEPRECATED**(1.6)
- bool **HasElement** (const std::string &_name) const **GAZEBO_DEPRECATED**(1.6)
- bool **HasElementDescription** (const std::string &_name) **GAZEBO_DEPRECATED**(1.6)

  Return true if an element description exists.

- void **InsertElement** (**ElementPtr** _elem) **GAZEBO_DEPRECATED**(1.6)
- void **PrintDescription** (std::string _prefix) **GAZEBO_DEPRECATED**(1.6)
- void **PrintDocLeftPane** (std::string &_html, int _spacing, int &_index) **GAZEBO_DEPRECATED**(1.6)

  *Helper function for **SDF::PrintDoc** (p. 728).*

- void **PrintDocRightPane** (std::string &_html, int _spacing, int &_index) **GAZEBO_DEPRECATED**(1.6)

  *Helper function for **SDF::PrintDoc** (p. 728).*

- void **PrintValues** (std::string _prefix) **GAZEBO_DEPRECATED**(1.6)
- void **PrintWiki** (std::string _prefix) **GAZEBO_DEPRECATED**(1.6)
- void **RemoveChild** (**ElementPtr** _child) **GAZEBO_DEPRECATED**(1.6)

  Remove a child element.

- void **RemoveFromParent** () **GAZEBO_DEPRECATED**(1.6)

  Remove this element from its parent.

- void **Reset** () **GAZEBO_DEPRECATED**(1.6)
- bool **Set** (const bool &_value) **GAZEBO_DEPRECATED**(1.6)
- bool **Set** (const int &_value) **GAZEBO_DEPRECATED**(1.6)
- bool **Set** (const unsigned int &_value) **GAZEBO_DEPRECATED**(1.6)
- bool **Set** (const float &_value) **GAZEBO_DEPRECATED**(1.6)
- bool **Set** (const double &_value) **GAZEBO_DEPRECATED**(1.6)

- bool **Set** (const char &_value) **GAZEBO_DEPRECATED**(1.6)
- bool **Set** (const std::string &_value) **GAZEBO_DEPRECATED**(1.6)
- bool **Set** (const char *_value) **GAZEBO_DEPRECATED**(1.6)
- bool **Set** (const **gazebo::math::Vector3** &_value) **GAZEBO_DEPRECATED**(1.6)
- bool **Set** (const **gazebo::math::Vector2i** &_value) **GAZEBO_DEPRECATED**(1.6)
- bool **Set** (const **gazebo::math::Vector2d** &_value) **GAZEBO_DEPRECATED**(1.6)
- bool **Set** (const **gazebo::math::Quaternion** &_value) **GAZEBO_DEPRECATED**(1.6)
- bool **Set** (const **gazebo::math::Pose** &_value) **GAZEBO_DEPRECATED**(1.6)
- bool **Set** (const **gazebo::common::Color** &_value) **GAZEBO_DEPRECATED**(1.6)
- bool **Set** (const **gazebo::common::Time** &_value) **GAZEBO_DEPRECATED**(1.6)
- void **SetCopyChildren** (bool _value) **GAZEBO_DEPRECATED**(1.6)
- void **SetDescription** (const std::string &_desc) **GAZEBO_DEPRECATED**(1.6)

  Set a text description for the element.
- void **SetInclude** (const std::string &_filename) **GAZEBO_DEPRECATED**(1.6)
- void **SetName** (const std::string &_name) **GAZEBO_DEPRECATED**(1.6)
- void **SetParent** (const **ElementPtr** _parent) **GAZEBO_DEPRECATED**(1.6)
- void **SetRequired** (const std::string &_req) **GAZEBO_DEPRECATED**(1.6)
- std::string **ToString** (const std::string &_prefix) const **GAZEBO_DEPRECATED**(1.6)
- void **Update** () **GAZEBO_DEPRECATED**(1.6)

10.41.1 Detailed Description

SDF (p. 728) **Element** (p. 280) class.

10.41.2 Constructor & Destructor Documentation

10.41.2.1 sdf::Element::Element ( )

10.41.2.2 virtual sdf::Element::~Element ( ) [virtual]

10.41.3 Member Function Documentation

10.41.3.1 void sdf::Element::AddAttribute (const std::string & *_key,* const std::string & *_type,* const std::string & *_defaultvalue,* bool *_required,* const std::string & *_description* = " ")

10.41.3.2 ElementPtr sdf::Element::AddElement (const std::string & *_name*)

10.41.3.3 void sdf::Element::AddElementDescription (ElementPtr *_elem*)

Add a new element description.

10.41.3.4 void sdf::Element::AddValue (const std::string & *_type,* const std::string & *_defaultValue,* bool *_required,* const std::string & *_description* = " ")

10.41.3.5 void sdf::Element::ClearElements ( )

Remove all child elements.

10.41.3.6 boost::shared_ptr<Element> sdf::Element::Clone ( ) const

10.41.3.7 void sdf::Element::Copy (const ElementPtr _elem)

Copy values from an **Element** (p. 280).

10.41.3.8 template<typename T > T sdf::Element::Get (const std::string & _key = " ") [inline]

References GetAttribute(), GetElementDescription(), GetElementImpl(), gzerr, HasElement(), and HasElement-Description().

10.41.3.9 ParamPtr sdf::Element::GetAttribute (const std::string & _key)

Get the param of an attribute.

Parameters

_key	the name of the attribute

Referenced by Get().

10.41.3.10 ParamPtr sdf::Element::GetAttribute (unsigned int _index) const

Get an attribute using an index.

10.41.3.11 unsigned int sdf::Element::GetAttributeCount ( ) const

Get the number of attributes.

10.41.3.12 bool sdf::Element::GetAttributeSet (const std::string & _key)

Return true if the attribute was set (i.e. not default value)

10.41.3.13 bool sdf::Element::GetCopyChildren ( ) const

10.41.3.14 std::string sdf::Element::GetDescription ( ) const

Get a text description of the element.

10.41.3.15 ElementPtr sdf::Element::GetElement (const std::string & _name) const

10.41.3.16 ElementPtr sdf::Element::GetElement (const std::string & _name)

10.41.3.17 ElementPtr sdf::Element::GetElementDescription (unsigned int _index) const

Get an element description using an index.

Referenced by Get().

10.41.3.18 ElementPtr sdf::Element::GetElementDescription (const std::string & _key) const

Get an element descriptio using a key.

10.41.3.19 unsigned int sdf::Element::GetElementDescriptionCount ( ) const

Get the number of element descriptions.

10.41.3.20 ElementPtr sdf::Element::GetElementImpl (const std::string & _name) const

Referenced by Get().

10.41.3.21 ElementPtr sdf::Element::GetFirstElement ( ) const

10.41.3.22 std::string sdf::Element::GetInclude ( ) const

10.41.3.23 const std::string& sdf::Element::GetName ( ) const

10.41.3.24 ElementPtr sdf::Element::GetNextElement (const std::string & _name = " ") const

10.41.3.25 ElementPtr sdf::Element::GetParent ( ) const

10.41.3.26 const std::string& sdf::Element::GetRequired ( ) const

10.41.3.27 ParamPtr sdf::Element::GetValue ( )

Get the param of the elements value.

10.41.3.28 bool sdf::Element::GetValueBool (const std::string & _key = " ")

10.41.3.29 char sdf::Element::GetValueChar (const std::string & _key = " ")

10.41.3.30 gazebo::common::Color sdf::Element::GetValueColor (const std::string & _key = " ")

10.41.3.31 double sdf::Element::GetValueDouble (const std::string & _key = " ")

10.41.3.32 float sdf::Element::GetValueFloat (const std::string & _key = " ")

10.41.3.33 int sdf::Element::GetValueInt (const std::string & _key = " ")

10.41.3.34 gazebo::math::Pose sdf::Element::GetValuePose (const std::string & _key = " ")

10.41.3.35 gazebo::math::Quaternion sdf::Element::GetValueQuaternion (const std::string & _key = " ")

10.41.3.36 std::string sdf::Element::GetValueString (const std::string & _key = " ")

10.41.3.37 gazebo::common::Time sdf::Element::GetValueTime (const std::string & _key = " ")

10.41.3.38 unsigned int sdf::Element::GetValueUInt (const std::string & _key = " ")

10.41.3.39 gazebo::math::Vector2d sdf::Element::GetValueVector2d (const std::string & _key = " ")

10.41.3.40 gazebo::math::Vector3 sdf::Element::GetValueVector3 (const std::string & _key = " ")

10.41.3.41 bool sdf::Element::HasAttribute (const std::string & _key)

10.41.3.42 bool sdf::Element::HasElement (const std::string & _name) const

Referenced by Get().

10.41.3.43 bool sdf::Element::HasElementDescription (const std::string & _name)

Return true if an element description exists.

Referenced by Get().

10.41.3.44 void sdf::Element::InsertElement (ElementPtr _elem)

10.41.3.45 void sdf::Element::PrintDescription (std::string _prefix)

10.41.3.46 void sdf::Element::PrintDocLeftPane (std::string & _html, int _spacing, int & _index)

Helper function for **SDF::PrintDoc** (p. 728).

This generates the **SDF** (p. 728) html documentation.

Parameters

out	*_html*	Accumulated HTML for output.
in	*_spacing*	Amount of spacing for this element.
in	*_index*	Unique index for this element.

10.41.3.47 void sdf::Element::PrintDocRightPane (std::string & _html, int _spacing, int & _index)

Helper function for **SDF::PrintDoc** (p. 728).

This generates the **SDF** (p. 728) html documentation.

Parameters

out	*_html*	Accumulated HTML for output
in	*_spacing*	Amount of spacing for this element.

10.41.3.48 void sdf::Element::PrintValues (std::string _prefix)

10.41.3.49 void sdf::Element::PrintWiki (std::string _prefix)

10.41.3.50 void sdf::Element::RemoveChild (ElementPtr _child)

Remove a child element.

Parameters

in	*_child*	Pointer to the child to remove.

10.41.3.51 void sdf::Element::RemoveFromParent ( )

Remove this element from its parent.

10.41.3.52 void sdf::Element::Reset ( )

10.41.3.53 bool sdf::Element::Set (const bool & *_value*)

10.41.3.54 bool sdf::Element::Set (const int & *_value*)

10.41.3.55 bool sdf::Element::Set (const unsigned int & *_value*)

10.41.3.56 bool sdf::Element::Set (const float & *_value*)

10.41.3.57 bool sdf::Element::Set (const double & *_value*)

10.41.3.58 bool sdf::Element::Set (const char & *_value*)

10.41.3.59 bool sdf::Element::Set (const std::string & *_value*)

10.41.3.60 bool sdf::Element::Set (const char * *_value*)

10.41.3.61 bool sdf::Element::Set (const gazebo::math::Vector3 & *_value*)

10.41.3.62 bool sdf::Element::Set (const gazebo::math::Vector2i & *_value*)

10.41.3.63 bool sdf::Element::Set (const gazebo::math::Vector2d & *_value*)

10.41.3.64 bool sdf::Element::Set (const gazebo::math::Quaternion & *_value*)

10.41.3.65 bool sdf::Element::Set (const gazebo::math::Pose & *_value*)

10.41.3.66 bool sdf::Element::Set (const gazebo::common::Color & *_value*)

10.41.3.67 bool sdf::Element::Set (const gazebo::common::Time & *_value*)

10.41.3.68 void sdf::Element::SetCopyChildren (bool *_value*)

10.41.3.69 void sdf::Element::SetDescription (const std::string & *_desc*)

Set a text description for the element.

10.41.3.70 void sdf::Element::SetInclude (const std::string & *_filename*)

10.41.3.71 void sdf::Element::SetName (const std::string & *_name*)

10.41.3.72 void sdf::Element::SetParent (const ElementPtr _parent)

10.41.3.73 void sdf::Element::SetRequired (const std::string & _req)

10.41.3.74 std::string sdf::Element::ToString (const std::string & _prefix) const

10.41.3.75 void sdf::Element::Update ( )

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

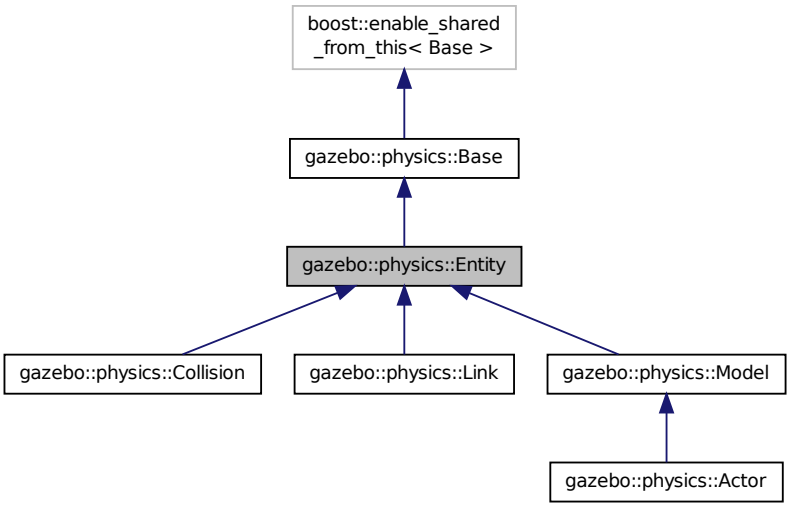
- **SDF.hh**

10.42 gazebo::physics::Entity Class Reference

Base (p. 141) class for all physics objects in Gazebo.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::Entity:



Public Member Functions

- **Entity** (**BasePtr** _parent)

  Constructor.
- virtual ~**Entity** ()

  Destructor.
- virtual void **Fini** ()

  Finalize the entity.
- virtual **math::Box GetBoundingBox** () const

Return the bounding box for the entity.

- **CollisionPtr GetChildCollision** (const std::string &_name)

  Get a child collision entity, if one exists.
- **LinkPtr GetChildLink** (const std::string &_name)

  Get a child linke entity, if one exists.
- **math::Box GetCollisionBoundingBox** () const

  Returns collision bounding box.
- const **math::Pose** & **GetDirtyPose** () const

  *Returns **Entity::dirtyPose** (*p. *299**).*
- **math::Pose GetInitialRelativePose** () const

  Get the initial relative pose.
- void **GetNearestEntityBelow** (double &_distBelow, std::string &_entityName)

  Get the distance to the nearest entity below (along the Z-axis) this entity.
- **ModelPtr GetParentModel** ()

  Get the parent model, if one exists.
- virtual **math::Vector3 GetRelativeAngularAccel** () const

  Get the angular acceleration of the entity.
- virtual **math::Vector3 GetRelativeAngularVel** () const

  Get the angular velocity of the entity.
- virtual **math::Vector3 GetRelativeLinearAccel** () const

  Get the linear acceleration of the entity.
- virtual **math::Vector3 GetRelativeLinearVel** () const

  Get the linear velocity of the entity.
- **math::Pose GetRelativePose** () const

  Get the pose of the entity relative to its parent.
- virtual **math::Vector3 GetWorldAngularAccel** () const

  Get the angular acceleration of the entity in the world frame.
- virtual **math::Vector3 GetWorldAngularVel** () const

  Get the angular velocity of the entity in the world frame.
- virtual **math::Vector3 GetWorldLinearAccel** () const

  Get the linear acceleration of the entity in the world frame.
- virtual **math::Vector3 GetWorldLinearVel** () const

  Get the linear velocity of the entity in the world frame.
- const **math::Pose** & **GetWorldPose** () const

  Get the absolute pose of the entity.
- bool **IsCanonicalLink** () const

  A helper function that checks if this is a canonical body.
- bool **IsStatic** () const

  Return whether this entity is static.
- virtual void **Load** (**sdf::ElementPtr** _sdf)

  Load the entity.
- void **PlaceOnEntity** (const std::string &_entityName)

  Move this entity to be ontop of another entity by name.
- void **PlaceOnNearestEntityBelow** ()

  Move this entity to be ontop of the nearest entity below.
- virtual void **Reset** ()

  Reset the entity.

- void **SetAnimation** (const **common::PoseAnimationPtr** &_anim, boost::function< void()> _onComplete)

  Set an animation for this entity.
- void **SetAnimation** (**common::PoseAnimationPtr** _anim)

  Set an animation for this entity.
- void **SetCanonicalLink** (bool _value)

  Set to true if this entity is a canonical link for a model.
- void **SetInitialRelativePose** (const **math::Pose** &_pose)

  Set the initial pose.
- virtual void **SetName** (const std::string &_name)

  Set the name of the entity.
- void **SetRelativePose** (const **math::Pose** &_pose, bool _notify=true, bool _publish=true)

  Set the pose of the entity relative to its parent.
- void **SetStatic** (const bool &_static)

  Set whether this entity is static: immovable.
- void **SetWorldPose** (const **math::Pose** &_pose, bool _notify=true, bool _publish=true)

  Set the world pose of the entity.
- void **SetWorldTwist** (const **math::Vector3** &_linear, const **math::Vector3** &_angular, bool _updateChildren=true)

  *Set angular and linear rates of an **physics::Entity*** (p. 288).
- virtual void **StopAnimation** ()

  Stop the current animation, if any.
- virtual void **UpdateParameters** (**sdf::ElementPtr** _sdf)

  Update the parameters using new sdf values.

Protected Member Functions

- virtual void **OnPoseChange** ()=0

  This function is called when the entity's (or one of its parents) pose of the parent has changed.

Protected Attributes

- **common::PoseAnimationPtr animation**

  Current pose animation.
- **event::ConnectionPtr animationConnection**

  Connection used to update an animation.
- **math::Pose animationStartPose**

  Start pose of an animation.
- std::vector< **event::ConnectionPtr** > **connections**

  All our event connections.
- **math::Pose dirtyPose**

  The pose set by a physics engine.
- **transport::NodePtr node**

  Communication node.
- **EntityPtr parentEntity**

  A helper that prevents numerous dynamic_casts.
- msgs::Pose * **poseMsg**

  Pose message container.

- **common::Time prevAnimationTime**

  Previous time an animation was updated.
- **transport::PublisherPtr requestPub**

  Request publisher.
- **transport::PublisherPtr visPub**

  Visual publisher.
- msgs::Visual ∗ **visualMsg**

  Visual message container.

Additional Inherited Members

10.42.1 Detailed Description

Base (p. 141) class for all physics objects in Gazebo.

10.42.2 Constructor & Destructor Documentation

10.42.2.1 gazebo::physics::Entity::Entity (BasePtr *_parent*) [explicit]

Constructor.

Parameters

in	*_parent*	Parent of the entity.

10.42.2.2 virtual gazebo::physics::Entity::∼Entity ( ) [virtual]

Destructor.

10.42.3 Member Function Documentation

10.42.3.1 virtual void gazebo::physics::Entity::Fini ( ) [virtual]

Finalize the entity.

Reimplemented from **gazebo::physics::Base** (p. 146).

Reimplemented in **gazebo::physics::Actor** (p. 118), **gazebo::physics::Model** (p. 521), **gazebo::physics::Link** (p. 447), and **gazebo::physics::Collision** (p. 203).

10.42.3.2 virtual math::Box gazebo::physics::Entity::GetBoundingBox ( ) const [virtual]

Return the bounding box for the entity.

Returns

The bounding box.

Reimplemented in **gazebo::physics::Link** (p. 447), **gazebo::physics::Model** (p. 521), and **gazebo::physics::-Collision** (p. 203).

10.42.3.3 CollisionPtr gazebo::physics::Entity::GetChildCollision (const std::string & _name)

Get a child collision entity, if one exists.

Parameters

in	*_name*	Name of the child collision object.

Returns

Pointer to the **Collision** (p. 199) object, or NULL if not found.

10.42.3.4 LinkPtr gazebo::physics::Entity::GetChildLink (const std::string & _name)

Get a child linke entity, if one exists.

Parameters

in	*_name*	Name of the child **Link** (p. 439) object.

Returns

Pointer to the **Link** (p. 439) object, or NULL if not found.

10.42.3.5 math::Box gazebo::physics::Entity::GetCollisionBoundingBox ( ) const

Returns collision bounding box.

Returns

Collsiion boundin box.

10.42.3.6 const math::Pose& gazebo::physics::Entity::GetDirtyPose ( ) const

Returns **Entity::dirtyPose** (p. 299).

The dirty pose is the pose set by the physics engine before it's value is propagated to the rest of the simulator.

Returns

The dirty pose of the entity.

10.42.3.7 math::Pose gazebo::physics::Entity::GetInitialRelativePose ( ) const

Get the initial relative pose.

Returns

The initial relative pose.

10.42.3.8 void gazebo::physics::Entity::GetNearestEntityBelow (double & *_distBelow,* std::string & *_entityName*)

Get the distance to the nearest entity below (along the Z-axis) this entity.

Parameters

out	*_distBelow*	The distance to the nearest entity below.
out	*_entityName*	The name of the nearest entity below.

10.42.3.9 ModelPtr gazebo::physics::Entity::GetParentModel ( )

Get the parent model, if one exists.

Returns

Pointer to a model, or NULL if no parent model exists.

10.42.3.10 virtual math::Vector3 gazebo::physics::Entity::GetRelativeAngularAccel ( ) const [inline], [virtual]

Get the angular acceleration of the entity.

Returns

A **math::Vector3** (p. 890) for the acceleration.

Reimplemented in **gazebo::physics::Link** (p. 450), **gazebo::physics::Collision** (p. 204), and **gazebo::physics::Model** (p. 523).

10.42.3.11 virtual math::Vector3 gazebo::physics::Entity::GetRelativeAngularVel ( ) const [inline], [virtual]

Get the angular velocity of the entity.

Returns

A **math::Vector3** (p. 890) for the velocity.

Reimplemented in **gazebo::physics::Link** (p. 450), **gazebo::physics::Collision** (p. 204), and **gazebo::physics::Model** (p. 523).

10.42.3.12 virtual math::Vector3 gazebo::physics::Entity::GetRelativeLinearAccel ( ) const [inline], [virtual]

Get the linear acceleration of the entity.

Returns

A **math::Vector3** (p. 890) for the acceleration.

Reimplemented in **gazebo::physics::Link** (p. 450), **gazebo::physics::Collision** (p. 205), and **gazebo::physics::Model** (p. 523).

10.42.3.13 virtual math::Vector3 gazebo::physics::Entity::GetRelativeLinearVel ( ) const [inline], [virtual]

Get the linear velocity of the entity.

Returns

A **math::Vector3** (p. 890) for the linear velocity.

Reimplemented in **gazebo::physics::Link** (p. 450), **gazebo::physics::Collision** (p. 205), and **gazebo::physics::Model** (p. 524).

10.42.3.14 math::Pose gazebo::physics::Entity::GetRelativePose ( ) const

Get the pose of the entity relative to its parent.

Returns

The pose of the entity relative to its parent.

10.42.3.15 virtual math::Vector3 gazebo::physics::Entity::GetWorldAngularAccel ( ) const [inline], [virtual]

Get the angular acceleration of the entity in the world frame.

Returns

A **math::Vector3** (p. 890) for the acceleration.

Reimplemented in **gazebo::physics::Link** (p. 451), **gazebo::physics::Collision** (p. 206), and **gazebo::physics::Model** (p. 524).

10.42.3.16 virtual math::Vector3 gazebo::physics::Entity::GetWorldAngularVel ( ) const [inline], [virtual]

Get the angular velocity of the entity in the world frame.

Returns

A **math::Vector3** (p. 890) for the velocity.

Reimplemented in **gazebo::physics::Collision** (p. 206), and **gazebo::physics::Model** (p. 524).

10.42.3.17 virtual math::Vector3 gazebo::physics::Entity::GetWorldLinearAccel ( ) const [inline], [virtual]

Get the linear acceleration of the entity in the world frame.

Returns

A **math::Vector3** (p. 890) for the acceleration.

Reimplemented in **gazebo::physics::Link** (p. 452), **gazebo::physics::Collision** (p. 206), and **gazebo::physics::Model** (p. 525).

10.42.3.18 virtual math::Vector3 gazebo::physics::Entity::GetWorldLinearVel ( ) const [inline], [virtual]

Get the linear velocity of the entity in the world frame.

Returns

A **math::Vector3** (p. 890) for the linear velocity.

Reimplemented in **gazebo::physics::Collision** (p. 206), and **gazebo::physics::Model** (p. 525).

10.42.3.19 const math::Pose& gazebo::physics::Entity::GetWorldPose ( ) const [inline]

Get the absolute pose of the entity.

Returns

The absolute pose of the entity.

10.42.3.20 bool gazebo::physics::Entity::IsCanonicalLink ( ) const [inline]

A helper function that checks if this is a canonical body.

Returns

True if the link is canonical.

10.42.3.21 bool gazebo::physics::Entity::IsStatic ( ) const

Return whether this entity is static.

Returns

True if static.

10.42.3.22 virtual void gazebo::physics::Entity::Load (sdf::ElementPtr _sdf) [virtual]

Load the entity.

Parameters

in	*_sdf*	Pointer to an SDF element.

Reimplemented from **gazebo::physics::Base** (p. 149).

Reimplemented in **gazebo::physics::Actor** (p. 118), **gazebo::physics::Link** (p. 453), **gazebo::physics::Model** (p. 525), and **gazebo::physics::Collision** (p. 207).

10.42.3.23 virtual void gazebo::physics::Entity::OnPoseChange ( ) [protected], [pure virtual]

This function is called when the entity's (or one of its parents) pose of the parent has changed.

Implemented in **gazebo::physics::Link** (p. 453), and **gazebo::physics::Model** (p. 525).

10.42.3.24 void gazebo::physics::Entity::PlaceOnEntity (const std::string & *_entityName*)

Move this entity to be ontop of another entity by name.

Parameters

in	*_entityName*	Name of the **Entity** (p. 288) this **Entity** (p. 288) should be ontop of.

10.42.3.25 void gazebo::physics::Entity::PlaceOnNearestEntityBelow ( )

Move this entity to be ontop of the nearest entity below.

10.42.3.26 virtual void gazebo::physics::Entity::Reset ( ) [virtual]

Reset the entity.

Reimplemented from **gazebo::physics::Base** (p. 151).

Reimplemented in **gazebo::physics::Model** (p. 526), and **gazebo::physics::Link** (p. 454).

10.42.3.27 void gazebo::physics::Entity::SetAnimation (const common::PoseAnimationPtr & *_anim*, boost::function< void() > *_onComplete*)

Set an animation for this entity.

Parameters

in	*_anim*	Pose animation.
in	*_onComplete*	Callback for when the animation completes.

10.42.3.28 void gazebo::physics::Entity::SetAnimation (common::PoseAnimationPtr *_anim*)

Set an animation for this entity.

Parameters

in	*_anim*	Pose animation.

10.42.3.29 void gazebo::physics::Entity::SetCanonicalLink (bool *_value*)

Set to true if this entity is a canonical link for a model.

Parameters

in	*_value*	True if the link is canonical.

10.42.3.30 void gazebo::physics::Entity::SetInitialRelativePose (const math::Pose & _pose)

Set the initial pose.

Parameters

in	*_pose*	The initial pose.

10.42.3.31 virtual void gazebo::physics::Entity::SetName (const std::string & _name) [virtual]

Set the name of the entity.

Parameters

in	*_name*	The new name.

Reimplemented from **gazebo::physics::Base** (p. 151).

10.42.3.32 void gazebo::physics::Entity::SetRelativePose (const math::Pose & _pose, bool _notify = true, bool _publish = true)

Set the pose of the entity relative to its parent.

Parameters

in	*_pose*	The new pose.
in	*_notify*	True = tell children of the pose change.
in	*_publish*	True to publish the pose.

10.42.3.33 void gazebo::physics::Entity::SetStatic (const bool & _static)

Set whether this entity is static: immovable.

Parameters

in	*_static*	True = static.

10.42.3.34 void gazebo::physics::Entity::SetWorldPose (const math::Pose & _pose, bool _notify = true, bool _publish = true)

Set the world pose of the entity.

Parameters

in	*_pose*	The new world pose.
in	*_notify*	True = tell children of the pose change.
in	*_publish*	True to publish the pose.

10.42.3.35 void gazebo::physics::Entity::SetWorldTwist (const math::Vector3 & _linear, const math::Vector3 & _angular, bool _updateChildren = true)

Set angular and linear rates of an **physics::Entity** (p. 288).

Parameters

in	*_linear*	Linear twist.
in	*_angular*	Angular twist.
in	*_updateChildren*	True to pass this update to child entities.

10.42.3.36 virtual void gazebo::physics::Entity::StopAnimation ( ) [virtual]

Stop the current animation, if any.

Reimplemented in **gazebo::physics::Model** (p. 529).

10.42.3.37 virtual void gazebo::physics::Entity::UpdateParameters (sdf::ElementPtr _sdf) [virtual]

Update the parameters using new sdf values.

Parameters

in	*_sdf*	SDF to update from.

Reimplemented from **gazebo::physics::Base** (p. 152).

Reimplemented in **gazebo::physics::Actor** (p. 119), **gazebo::physics::Link** (p. 458), **gazebo::physics::Model** (p. 529), and **gazebo::physics::Collision** (p. 209).

10.42.4 Member Data Documentation

10.42.4.1 common::PoseAnimationPtr gazebo::physics::Entity::animation [protected]

Current pose animation.

10.42.4.2 event::ConnectionPtr gazebo::physics::Entity::animationConnection [protected]

Connection used to update an animation.

10.42.4.3 math::Pose gazebo::physics::Entity::animationStartPose [protected]

Start pose of an animation.

10.42.4.4 std::vector<event::ConnectionPtr> gazebo::physics::Entity::connections [protected]

All our event connections.

10.42.4.5 math::Pose gazebo::physics::Entity::dirtyPose [protected]

The pose set by a physics engine.

10.42.4.6 transport::NodePtr gazebo::physics::Entity::node [protected]

Communication node.

10.42.4.7 EntityPtr gazebo::physics::Entity::parentEntity [protected]

A helper that prevents numerous dynamic_casts.

10.42.4.8 msgs::Pose* gazebo::physics::Entity::poseMsg [protected]

Pose message container.

10.42.4.9 common::Time gazebo::physics::Entity::prevAnimationTime [protected]

Previous time an animation was updated.

10.42.4.10 transport::PublisherPtr gazebo::physics::Entity::requestPub [protected]

Request publisher.

10.42.4.11 transport::PublisherPtr gazebo::physics::Entity::visPub [protected]

Visual publisher.

10.42.4.12 msgs::Visual* gazebo::physics::Entity::visualMsg [protected]

Visual message container.

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

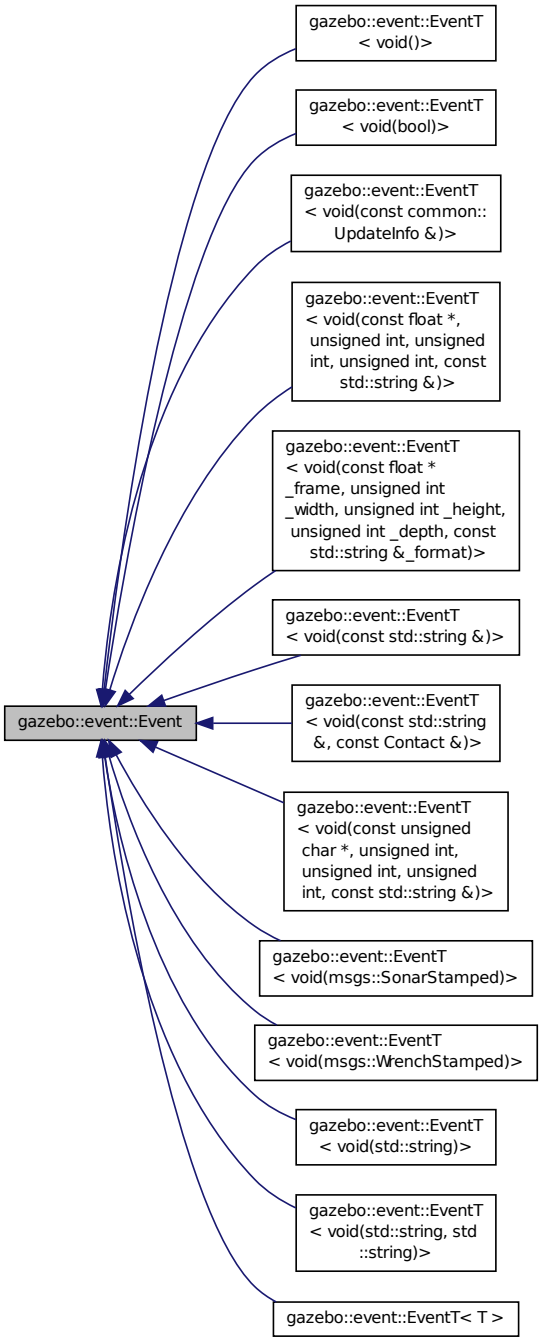
- **Entity.hh**

10.43 gazebo::event::Event Class Reference

Base class for all events.

`#include <common/common.hh>`

Inheritance diagram for gazebo::event::Event:



Public Member Functions

- virtual ~**Event** ()

Constructor.

- virtual void **Disconnect** (**ConnectionPtr** _c)=0

  Disconnect.

- virtual void **Disconnect** (int _id)=0

  Disconnect.

10.43.1 Detailed Description

Base class for all events.

10.43.2 Constructor & Destructor Documentation

10.43.2.1 virtual gazebo::event::Event::~Event ( ) [inline], [virtual]

Constructor.

10.43.3 Member Function Documentation

10.43.3.1 virtual void gazebo::event::Event::Disconnect (ConnectionPtr _c) [pure virtual]

Disconnect.

Parameters

in	*_c*	A pointer to a connection

Implemented in **gazebo::event::EventT< T >** (p. 40), **gazebo::event::EventT< void(msgs::SonarStamped)>** (p. 40), **gazebo::event::EventT< void(std::string)>** (p. 40), **gazebo::event::EventT< void(const unsigned char *, unsigned int, unsigned int, unsigned int, const std::string &)>** (p. 40), **gazebo::event::EventT< void(msgs::WrenchStamped)>** (p. 40), **gazebo::event::EventT< void(const float *_frame, unsigned int _width, unsigned int _height, unsigned int _depth, const std::string &_format)>** (p. 40), **gazebo::event::EventT< void(const std::string &)>** (p. 40), **gazebo::event::EventT< void()>** (p. 40), **gazebo::event::EventT< void(const common::UpdateInfo &)>** (p. 40), **gazebo::event::EventT< void(const float *, unsigned int, unsigned int, unsigned int, const std::string &)>** (p. 40), **gazebo::event::EventT< void(const std::string &, const Contact &)>** (p. 40), **gazebo::event::EventT< void(std::string, std::string)>** (p. 40), and **gazebo::event::EventT< void(bool)>** (p. 40).

10.43.3.2 virtual void gazebo::event::Event::Disconnect (int _id) [pure virtual]

Disconnect.

Parameters

in	*_id*	Integer ID of a connection

Implemented in **gazebo::event::EventT< T >** (p. 41), **gazebo::event::EventT< void(msgs::SonarStamped)>** (p. 41), **gazebo::event::EventT< void(std::string)>** (p. 41), **gazebo::event::EventT< void(const unsigned char *, unsigned int, unsigned int, unsigned int, const std::string &)>** (p. 41), **gazebo::event::EventT< void(msgs::WrenchStamped)>** (p. 41), **gazebo::event::EventT< void(const float *_frame, unsigned int _width, unsigned int _height, unsigned int _depth, const std::string &_format)>** (p. 41), **gazebo::event::EventT< void(const std::string &)>** (p. 41), **gazebo::event::EventT< void()>** (p. 41), **gazebo::event::EventT< void(const common::-**

UpdateInfo &)> (p. 41), **gazebo::event::EventT< void(const float *, unsigned int, unsigned int, unsigned int, const std::string &)>** (p. 41), **gazebo::event::EventT< void(const std::string &, const Contact &)>** (p. 41), **gazebo::event::EventT< void(std::string, std::string)>** (p. 41), and **gazebo::event::EventT< void(bool)>** (p. 41).

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

- **Event.hh**

10.44 gazebo::rendering::Events Class Reference

Base class for rendering events.

`#include <rendering/rendering.hh>`

Static Public Member Functions

- template<typename T >
  static **event::ConnectionPtr ConnectCreateScene** (T _subscriber)

  Connect to a scene created event.
- template<typename T >
  static **event::ConnectionPtr ConnectRemoveScene** (T _subscriber)

  Connect to a scene removed event.
- static void **DisconnectCreateScene** (**event::ConnectionPtr** _connection)

  Disconnect from a scene created event.
- static void **DisconnectRemoveScene** (**event::ConnectionPtr** _connection)

  Disconnect from a scene removed event.

Static Public Attributes

- static **event::EventT**< void(const std::string &)> **createScene**

  The event used to trigger a create scene event.
- static **event::EventT**< void(const std::string &)> **removeScene**

  The event used to trigger a remve scene event.

10.44.1 Detailed Description

Base class for rendering events.

10.44.2 Member Function Documentation

10.44.2.1 template<typename T > static event::ConnectionPtr gazebo::rendering::Events::ConnectCreateScene (T *_subscriber*) [inline], [static]

Connect to a scene created event.

Parameters

in	*_subscriber*	Callback to trigger when event occurs.

Returns

Pointer the connection. This must stay in scope.

References gazebo::event::EventT< T >::Connect(), and createScene.

10.44.2.2 template<typename T > static event::ConnectionPtr gazebo::rendering::Events::ConnectRemoveScene (T *_subscriber*) [inline], [static]

Connect to a scene removed event.

Parameters

in	*_subscriber*	Callback to trigger when event occurs.

Returns

Pointer the connection. This must stay in scope.

References gazebo::event::EventT< T >::Connect(), and removeScene.

10.44.2.3 static void gazebo::rendering::Events::DisconnectCreateScene (event::ConnectionPtr *_connection*) [inline], [static]

Disconnect from a scene created event.

Parameters

in	*_connection*	The connection to disconnect.

References createScene, and gazebo::event::EventT< T >::Disconnect().

10.44.2.4 static void gazebo::rendering::Events::DisconnectRemoveScene (event::ConnectionPtr *_connection*) [inline], [static]

Disconnect from a scene removed event.

Parameters

in	*_connection*	The connection to disconnect.

References gazebo::event::EventT< T >::Disconnect(), and removeScene.

10.44.3 Member Data Documentation

10.44.3.1 event::EventT<void (const std::string &)> gazebo::rendering::Events::createScene [static]

The event used to trigger a create scene event.

Referenced by ConnectCreateScene(), and DisconnectCreateScene().

10.44.3.2 event::EventT<void (const std::string &)> gazebo::rendering::Events::removeScene [static]

The event used to trigger a remve scene event.

Referenced by ConnectRemoveScene(), and DisconnectRemoveScene().

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

- **RenderEvents.hh**

10.45 gazebo::event::Events Class Reference

An **Event** (p. 299) class to get notifications for simulator events.

`#include <common/common.hh>`

Static Public Member Functions

- template<typename T >
  static **ConnectionPtr ConnectAddEntity** (T _subscriber)

  Connect a boost::slot the the add entity signal.
- template<typename T >
  static **ConnectionPtr ConnectCreateEntity** (T _subscriber)

  Connect a boost::slot the the add entity signal.
- template<typename T >
  static **ConnectionPtr ConnectDeleteEntity** (T _subscriber)

  Connect a boost::slot the delete entity.
- template<typename T >
  static **ConnectionPtr ConnectDiagTimerStart** (T _subscriber)

  Connect a boost::slot the diagnostic timer start signal.
- template<typename T >
  static **ConnectionPtr ConnectDiagTimerStop** (T _subscriber)

  Connect a boost::slot the diagnostic timer stop signal.
- template<typename T >
  static **ConnectionPtr ConnectPause** (T _subscriber)

  Connect a boost::slot the the pause signal.
- template<typename T >
  static **ConnectionPtr ConnectPostRender** (T _subscriber)

  Connect a boost::slot the post render update signal.
- template<typename T >
  static **ConnectionPtr ConnectPreRender** (T _subscriber)

  Render start signal.
- template<typename T >
  static **ConnectionPtr ConnectRender** (T _subscriber)

  Connect a boost::slot the render update signal.
- template<typename T >
  static **ConnectionPtr ConnectSetSelectedEntity** (T _subscriber)

Connect a boost::slot the set selected entity.

- template<typename T >
  static **ConnectionPtr ConnectSigInt** (T _subscriber)

  Connect a boost::slot to the sigint event.
- template<typename T >
  static **ConnectionPtr ConnectStep** (T _subscriber)

  Connect a boost::slot the the step signal.
- template<typename T >
  static **ConnectionPtr ConnectStop** (T _subscriber)

  Connect a boost::slot the the stop signal.
- template<typename T >
  static **ConnectionPtr ConnectWorldCreated** (T _subscriber)

  Connect a boost::slot the the world created signal.
- template<typename T >
  static **ConnectionPtr ConnectWorldUpdateBegin** (T _subscriber)

  Connect a boost::slot the the world update start signal.
- template<typename T >
  static **ConnectionPtr ConnectWorldUpdateEnd** (T _subscriber)

  Connect a boost::slot the the world update end signal.
- template<typename T >
  static **ConnectionPtr ConnectWorldUpdateStart** (T _subscriber)

  Connect a boost::slot the the world update start signal.
- static void **DisconnectAddEntity** (**ConnectionPtr** _subscriber)

  Disconnect a boost::slot the the add entity signal.
- static void **DisconnectCreateEntity** (**ConnectionPtr** _subscriber)

  Disconnect a boost::slot the the add entity signal.
- static void **DisconnectDeleteEntity** (**ConnectionPtr** _subscriber)

  Disconnect a boost::slot the delete entity.
- static void **DisconnectDiagTimerStart** (**ConnectionPtr** _subscriber)

  Disconnect a boost::slot the diagnostic timer start signal.
- static void **DisconnectDiagTimerStop** (**ConnectionPtr** _subscriber)

  Disconnect a boost::slot the diagnostic timer stop signal.
- static void **DisconnectPause** (**ConnectionPtr** _subscriber)

  Disconnect a boost::slot the the pause signal.
- static void **DisconnectPostRender** (**ConnectionPtr** _subscriber)

  Disconnect a boost::slot the post render update signal.
- static void **DisconnectPreRender** (**ConnectionPtr** _subscriber)

  Disconnect a render start signal.
- static void **DisconnectRender** (**ConnectionPtr** _subscriber)

  Disconnect a boost::slot the render update signal.
- static void **DisconnectSetSelectedEntity** (**ConnectionPtr** _subscriber)

  Disconnect a boost::slot the set selected entity.
- static void **DisconnectSigInt** (**ConnectionPtr** _subscriber)

  Disconnect a boost::slot to the sigint event.
- static void **DisconnectStep** (**ConnectionPtr** _subscriber)

  Disconnect a boost::slot the the step signal.
- static void **DisconnectStop** (**ConnectionPtr** _subscriber)

  Disconnect a boost::slot the the stop signal.

- static void **DisconnectWorldCreated** (**ConnectionPtr** _subscriber)

  Disconnect a boost::slot the the world created signal.
- static void **DisconnectWorldUpdateBegin** (**ConnectionPtr** _subscriber)

  Disconnect a boost::slot the the world update start signal.
- static void **DisconnectWorldUpdateEnd** (**ConnectionPtr** _subscriber)

  Disconnect a boost::slot the the world update end signal.
- static void **DisconnectWorldUpdateStart** (**ConnectionPtr** _subscriber) **GAZEBO_DEPRECATED**(1.5)

  Disconnect a boost::slot the the world update start signal.

Static Public Attributes

- static **EventT**< void(std::string)> **addEntity**

  An entity has been added.
- static **EventT**< void(std::string)> **deleteEntity**

  An entity has been deleted.
- static **EventT**< void(std::string)> **diagTimerStart**

  Diagnostic timer start.
- static **EventT**< void(std::string)> **diagTimerStop**

  Diagnostic timer stop.
- static **EventT**< void(std::string)> **entityCreated**

  An entity has been created.
- static **EventT**< void(bool)> **pause**

  Pause signal.
- static **EventT**< void()> **postRender**

  Post-Render.
- static **EventT**< void()> **preRender**

  Pre-render.
- static **EventT**< void()> **render**

  Render.
- static **EventT**< void(std::string, std::string)> **setSelectedEntity**

  An entity has been selected.
- static **EventT**< void()> **sigInt**

  Simulation stop signal.
- static **EventT**< void()> **step**

  Step the simulation once signal.
- static **EventT**< void()> **stop**

  Simulation stop signal.
- static **EventT**< void(std::string)> **worldCreated**

  A world has been created.
- static **EventT**< void(const **common::UpdateInfo** &)> **worldUpdateBegin**

  World update has started.
- static **EventT**< void()> **worldUpdateEnd**

  World update has ended.
- static **EventT**< void()> **worldUpdateStart**

  World update has started.

10.45.1 Detailed Description

An **Event** (p. 299) class to get notifications for simulator events.

10.45.2 Member Function Documentation

10.45.2.1 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectAddEntity (T _subscriber) [inline], [static]

Connect a boost::slot the the add entity signal.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References addEntity, and gazebo::event::EventT< T >::Connect().

10.45.2.2 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectCreateEntity (T _subscriber) [inline], [static]

Connect a boost::slot the the add entity signal.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), and entityCreated.

10.45.2.3 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectDeleteEntity (T _subscriber) [inline], [static]

Connect a boost::slot the delete entity.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), and deleteEntity.

10.45.2.4 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectDiagTimerStart (T _subscriber) [inline], [static]

Connect a boost::slot the diagnostic timer start signal.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), and diagTimerStart.

10.45.2.5 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectDiagTimerStop (T _subscriber) [inline], [static]

Connect a boost::slot the diagnostic timer stop signal.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), and diagTimerStop.

10.45.2.6 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectPause (T _subscriber) [inline], [static]

Connect a boost::slot the the pause signal.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), and pause.

10.45.2.7 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectPostRender (T _subscriber) [inline], [static]

Connect a boost::slot the post render update signal.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), and postRender.

10.45.2.8 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectPreRender (T *_subscriber*) [inline], [static]

Render start signal.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), and preRender.

10.45.2.9 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectRender (T *_subscriber*) [inline], [static]

Connect a boost::slot the render update signal.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), and render.

10.45.2.10 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectSetSelectedEntity (T *_subscriber*) [inline], [static]

Connect a boost::slot the set selected entity.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), and setSelectedEntity.

10.45.2.11 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectSigInt (T _subscriber) [inline], [static]

Connect a boost::slot to the sigint event.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), and sigInt.

10.45.2.12 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectStep (T _subscriber) [inline], [static]

Connect a boost::slot the the step signal.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), and step.

10.45.2.13 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectStop (T _subscriber) [inline], [static]

Connect a boost::slot the the stop signal.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), and stop.

10.45.2.14 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectWorldCreated (T _subscriber) [inline], [static]

Connect a boost::slot the the world created signal.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), and worldCreated.

10.45.2.15 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectWorldUpdateBegin (T *_subscriber*) [inline], [static]

Connect a boost::slot the the world update start signal.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), and worldUpdateBegin.

10.45.2.16 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectWorldUpdateEnd (T *_subscriber*) [inline], [static]

Connect a boost::slot the the world update end signal.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), and worldUpdateEnd.

10.45.2.17 template<typename T > static ConnectionPtr gazebo::event::Events::ConnectWorldUpdateStart (T *_subscriber*) [inline], [static]

Connect a boost::slot the the world update start signal.

Parameters

in	*_subscriber*	the subscriber to this event

Returns

a connection

References gazebo::event::EventT< T >::Connect(), gzerr, and worldUpdateStart.

10.45.2.18 static void gazebo::event::Events::DisconnectAddEntity (ConnectionPtr *_subscriber*) [inline], [static]

Disconnect a boost::slot the the add entity signal.

Parameters

in	*_subscriber*	the subscriber to this event

References addEntity, and gazebo::event::EventT< T >::Disconnect().

10.45.2.19 static void gazebo::event::Events::DisconnectCreateEntity (ConnectionPtr *_subscriber*) [inline], [static]

Disconnect a boost::slot the the add entity signal.

Parameters

in	*_subscriber*	the subscriber to this event

References gazebo::event::EventT< T >::Disconnect(), and entityCreated.

10.45.2.20 static void gazebo::event::Events::DisconnectDeleteEntity (ConnectionPtr *_subscriber*) [inline], [static]

Disconnect a boost::slot the delete entity.

Parameters

in	*_subscriber*	the subscriber to this event

References deleteEntity, and gazebo::event::EventT< T >::Disconnect().

10.45.2.21 static void gazebo::event::Events::DisconnectDiagTimerStart (ConnectionPtr *_subscriber*) [inline], [static]

Disconnect a boost::slot the diagnostic timer start signal.

Parameters

in	*_subscriber*	the subscriber to this event

References diagTimerStart, and gazebo::event::EventT< T >::Disconnect().

10.45.2.22 static void gazebo::event::Events::DisconnectDiagTimerStop (ConnectionPtr *_subscriber*) [inline], [static]

Disconnect a boost::slot the diagnostic timer stop signal.

Parameters

in	*_subscriber*	the subscriber to this event

References diagTimerStop, and gazebo::event::EventT< T >::Disconnect().

10.45.2.23 static void gazebo::event::Events::DisconnectPause (ConnectionPtr *_subscriber*) [inline], [static]

Disconnect a boost::slot the the pause signal.

Parameters

in	*_subscriber*	the subscriber to this event

References gazebo::event::EventT< T >::Disconnect(), and pause.

10.45.2.24 static void gazebo::event::Events::DisconnectPostRender (ConnectionPtr *_subscriber*) [inline], [static]

Disconnect a boost::slot the post render update signal.

Parameters

in	*_subscriber*	the subscriber to this event

References gazebo::event::EventT< T >::Disconnect(), and postRender.

10.45.2.25 static void gazebo::event::Events::DisconnectPreRender (ConnectionPtr *_subscriber*) [inline], [static]

Disconnect a render start signal.

Parameters

in	*_subscriber*	the subscriber to this event

References gazebo::event::EventT< T >::Disconnect(), and preRender.

10.45.2.26 static void gazebo::event::Events::DisconnectRender (ConnectionPtr *_subscriber*) [inline], [static]

Disconnect a boost::slot the render update signal.

Parameters

in	*_subscriber*	the subscriber to this event

References gazebo::event::EventT< T >::Disconnect(), and render.

10.45.2.27 static void gazebo::event::Events::DisconnectSetSelectedEntity (ConnectionPtr *_subscriber*) [inline], [static]

Disconnect a boost::slot the set selected entity.

Parameters

in	*_subscriber*	the subscriber to this event

References gazebo::event::EventT< T >::Disconnect(), and setSelectedEntity.

10.45.2.28 static void gazebo::event::Events::DisconnectSigInt (ConnectionPtr *_subscriber*) [inline], [static]

Disconnect a boost::slot to the sigint event.

Parameters

in	*_subscriber*	the subscriber to this event

References gazebo::event::EventT< T >::Disconnect(), and sigInt.

10.45.2.29 static void gazebo::event::Events::DisconnectStep (ConnectionPtr *_subscriber*) [inline], [static]

Disconnect a boost::slot the the step signal.

Parameters

in	*_subscriber*	the subscriber to this event

References gazebo::event::EventT< T >::Disconnect(), and step.

10.45.2.30 static void gazebo::event::Events::DisconnectStop (ConnectionPtr *_subscriber*) [inline], [static]

Disconnect a boost::slot the the stop signal.

Parameters

in	*_subscriber*	the subscriber to this event

References gazebo::event::EventT< T >::Disconnect(), and stop.

10.45.2.31 static void gazebo::event::Events::DisconnectWorldCreated (ConnectionPtr *_subscriber*) [inline], [static]

Disconnect a boost::slot the the world created signal.

References gazebo::event::EventT< T >::Disconnect(), and worldCreated.

10.45.2.32 static void gazebo::event::Events::DisconnectWorldUpdateBegin (ConnectionPtr *_subscriber*) [static]

Disconnect a boost::slot the the world update start signal.

Parameters

in	*_subscriber*	the subscriber to this event

10.45.2.33 static void gazebo::event::Events::DisconnectWorldUpdateEnd (ConnectionPtr _subscriber) [inline], [static]

Disconnect a boost::slot the the world update end signal.

Parameters

in	*_subscriber*	the subscriber to this event

References gazebo::event::EventT< T >::Disconnect(), and worldUpdateEnd.

10.45.2.34 static void gazebo::event::Events::DisconnectWorldUpdateStart (ConnectionPtr _subscriber) [static]

Disconnect a boost::slot the the world update start signal.

Parameters

in	*_subscriber*	the subscriber to this event

10.45.3 Member Data Documentation

10.45.3.1 EventT<void (std::string)> gazebo::event::Events::addEntity [static]

An entity has been added.

Referenced by ConnectAddEntity(), and DisconnectAddEntity().

10.45.3.2 EventT<void (std::string)> gazebo::event::Events::deleteEntity [static]

An entity has been deleted.

Referenced by ConnectDeleteEntity(), and DisconnectDeleteEntity().

10.45.3.3 EventT<void (std::string)> gazebo::event::Events::diagTimerStart [static]

Diagnostic timer start.

Referenced by ConnectDiagTimerStart(), and DisconnectDiagTimerStart().

10.45.3.4 EventT<void (std::string)> gazebo::event::Events::diagTimerStop [static]

Diagnostic timer stop.

Referenced by ConnectDiagTimerStop(), and DisconnectDiagTimerStop().

10.45.3.5 EventT<void (std::string)> gazebo::event::Events::entityCreated [static]

An entity has been created.

Referenced by ConnectCreateEntity(), and DisconnectCreateEntity().

10.45.3.6 EventT<void (bool)> gazebo::event::Events::pause [static]

Pause signal.

Referenced by ConnectPause(), and DisconnectPause().

10.45.3.7 EventT<void ()> gazebo::event::Events::postRender [static]

Post-Render.

Referenced by ConnectPostRender(), and DisconnectPostRender().

10.45.3.8 EventT<void ()> gazebo::event::Events::preRender [static]

Pre-render.

Referenced by ConnectPreRender(), and DisconnectPreRender().

10.45.3.9 EventT<void ()> gazebo::event::Events::render [static]

Render.

Referenced by ConnectRender(), and DisconnectRender().

10.45.3.10 EventT<void (std::string, std::string)> gazebo::event::Events::setSelectedEntity [static]

An entity has been selected.

Referenced by ConnectSetSelectedEntity(), and DisconnectSetSelectedEntity().

10.45.3.11 EventT<void ()> gazebo::event::Events::sigInt [static]

Simulation stop signal.

Referenced by ConnectSigInt(), and DisconnectSigInt().

10.45.3.12 EventT<void ()> gazebo::event::Events::step [static]

Step the simulation once signal.

Referenced by ConnectStep(), and DisconnectStep().

10.45.3.13 EventT<void ()> gazebo::event::Events::stop [static]

Simulation stop signal.

Referenced by ConnectStop(), and DisconnectStop().

10.45.3.14 EventT<void (std::string)> gazebo::event::Events::worldCreated [static]

A world has been created.

Referenced by ConnectWorldCreated(), and DisconnectWorldCreated().

10.45.3.15 EventT<void (const common::UpdateInfo &)> gazebo::event::Events::worldUpdateBegin [static]

World update has started.

Referenced by ConnectWorldUpdateBegin().

10.45.3.16 EventT<void ()> gazebo::event::Events::worldUpdateEnd [static]

World update has ended.

Referenced by ConnectWorldUpdateEnd(), and DisconnectWorldUpdateEnd().

10.45.3.17 EventT<void ()> gazebo::event::Events::worldUpdateStart [static]

World update has started.

Referenced by ConnectWorldUpdateStart().

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

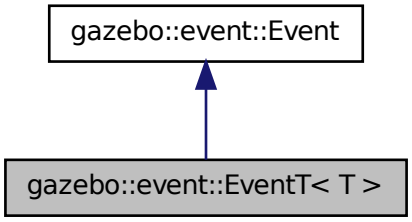
- **Events.hh**

10.46 gazebo::event::EventT< T > Class Template Reference

A class for event processing.

`#include <common/common.hh>`

Inheritance diagram for gazebo::event::EventT< T >:



Public Member Functions

- virtual ~**EventT** ()

  Destructor.
- **ConnectionPtr Connect** (const boost::function< T > &_subscriber)

  Connect a callback to this event.
- unsigned int **ConnectionCount** () const

Get the number of connections.

- virtual void **Disconnect** (**ConnectionPtr** _c)

  Disconnect a callback to this event.

- virtual void **Disconnect** (int _id)

  Disconnect a callback to this event.

- void **operator()** ()

  Access the signal.

- template<typename P >
  void **operator()** (const P &_p)

  Signal the event with one parameter.

- template<typename P1 , typename P2 >
  void **operator()** (const P1 &_p1, const P2 &_p2)

  Signal the event with two parameters.

- template<typename P1 , typename P2 , typename P3 >
  void **operator()** (const P1 &_p1, const P2 &_p2, const P3 &_p3)

  Signal the event with three parameters.

- template<typename P1 , typename P2 , typename P3 , typename P4 >
  void **operator()** (const P1 &_p1, const P2 &_p2, const P3 &_p3, const P4 &_p4)

  Signal the event with four parameters.

- template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 >
  void **operator()** (const P1 &_p1, const P2 &_p2, const P3 &_p3, const P4 &_p4, const P5 &_p5)

  Signal the event with five parameters.

- template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 >
  void **operator()** (const P1 &_p1, const P2 &_p2, const P3 &_p3, const P4 &_p4, const P5 &_p5, const P6 &_p6)

  Signal the event with six parameters.

- template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 >
  void **operator()** (const P1 &_p1, const P2 &_p2, const P3 &_p3, const P4 &_p4, const P5 &_p5, const P6 &_p6, const P7 &_p7)

  Signal the event with seven parameters.

- template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 , typename P8 >
  void **operator()** (const P1 &_p1, const P2 &_p2, const P3 &_p3, const P4 &_p4, const P5 &_p5, const P6 &_p6, const P7 &_p7, const P8 &_p8)

  Signal the event with eight parameters.

- template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 , typename P8 , typename P9 >
  void **operator()** (const P1 &_p1, const P2 &_p2, const P3 &_p3, const P4 &_p4, const P5 &_p5, const P6 &_p6, const P7 &_p7, const P8 &_p8, const P9 &_p9)

  Signal the event with nine parameters.

- template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 , typename P8 , typename P9 , typename P10 >
  void **operator()** (const P1 &_p1, const P2 &_p2, const P3 &_p3, const P4 &_p4, const P5 &_p5, const P6 &_p6, const P7 &_p7, const P8 &_p8, const P9 &_p9, const P10 &_p10)

  Signal the event with ten parameters.

- void **Signal** ()

  Signal the event for all subscribers.

- template<typename P >
  void **Signal** (const P &_p)

  Signal the event with one parameter.

- template<typename P1 , typename P2 >
  void **Signal** (const P1 &_p1, const P2 &_p2)

Signal the event with two parameter.

- template<typename P1 , typename P2 , typename P3 >
  void **Signal** (const P1 &_p1, const P2 &_p2, const P3 &_p3)

  Signal the event with three parameter.
- template<typename P1 , typename P2 , typename P3 , typename P4 >
  void **Signal** (const P1 &_p1, const P2 &_p2, const P3 &_p3, const P4 &_p4)

  Signal the event with four parameter.
- template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 >
  void **Signal** (const P1 &_p1, const P2 &_p2, const P3 &_p3, const P4 &_p4, const P5 &_p5)

  Signal the event with five parameter.
- template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 >
  void **Signal** (const P1 &_p1, const P2 &_p2, const P3 &_p3, const P4 &_p4, const P5 &_p5, const P6 &_p6)

  Signal the event with six parameter.
- template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 >
  void **Signal** (const P1 &_p1, const P2 &_p2, const P3 &_p3, const P4 &_p4, const P5 &_p5, const P6 &_p6, const P7 &_p7)

  Signal the event with seven parameter.
- template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 , typename P8 >
  void **Signal** (const P1 &_p1, const P2 &_p2, const P3 &_p3, const P4 &_p4, const P5 &_p5, const P6 &_p6, const P7 &_p7, const P8 &_p8)

  Signal the event with eight parameter.
- template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 , typename P8 , typename P9 >
  void **Signal** (const P1 &_p1, const P2 &_p2, const P3 &_p3, const P4 &_p4, const P5 &_p5, const P6 &_p6, const P7 &_p7, const P8 &_p8, const P9 &_p9)

  Signal the event with nine parameter.
- template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 , typename P8 , typename P9 , typename P10 >
  void **Signal** (const P1 &_p1, const P2 &_p2, const P3 &_p3, const P4 &_p4, const P5 &_p5, const P6 &_p6, const P7 &_p7, const P8 &_p8, const P9 &_p9, const P10 &_p10)

  Signal the event with ten parameter.

10.46.1 Detailed Description

template<typename T>class gazebo::event::EventT< T >

A class for event processing.

10.46.2 Member Function Documentation

10.46.2.1 template<typename T> void gazebo::event::EventT< T >::operator() ( ) [inline]

Access the signal.

10.46.2.2 template<typename T> template<typename P > void gazebo::event::EventT< T >::operator() (const P & _p) [inline]

Signal the event with one parameter.

Parameters

in	*_p*	the parameter

10.46.2.3 template<typename T> template<typename P1 , typename P2 > void gazebo::event::EventT< T >::operator() (const P1 & *_p1,* const P2 & *_p2*) [inline]

Signal the event with two parameters.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter

10.46.2.4 template<typename T> template<typename P1 , typename P2 , typename P3 > void gazebo::event::EventT< T >::operator() (const P1 & *_p1,* const P2 & *_p2,* const P3 & *_p3*) [inline]

Signal the event with three parameters.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter
in	*_p3*	the second parameter

10.46.2.5 template<typename T> template<typename P1 , typename P2 , typename P3 , typename P4 > void gazebo::event::EventT< T >::operator() (const P1 & *_p1,* const P2 & *_p2,* const P3 & *_p3,* const P4 & *_p4*) [inline]

Signal the event with four parameters.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter
in	*_p3*	the second parameter
in	*_p4*	the first parameter

10.46.2.6 template<typename T> template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 > void gazebo::event::EventT< T >::operator() (const P1 & *_p1,* const P2 & *_p2,* const P3 & *_p3,* const P4 & *_p4,* const P5 & *_p5*) [inline]

Signal the event with five parameters.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter
in	*_p3*	the second parameter
in	*_p4*	the first parameter
in	*_p5*	the fift parameter

10.46.2.7 template<typename T> template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 > void gazebo::event::EventT< T >::operator() (const P1 & *_p1,* const P2 & *_p2,* const P3 & *_p3,* const P4 & *_p4,* const P5 & *_p5,* const P6 & *_p6*) [inline]

Signal the event with six parameters.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter
in	*_p3*	the second parameter
in	*_p4*	the first parameter
in	*_p5*	the fift parameter
in	*_p6*	the sixt parameter

10.46.2.8 template<typename T> template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 > void gazebo::event::EventT< T >::operator() (const P1 & *_p1,* const P2 & *_p2,* const P3 & *_p3,* const P4 & *_p4,* const P5 & *_p5,* const P6 & *_p6,* const P7 & *_p7*) [inline]

Signal the event with seven parameters.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter
in	*_p3*	the second parameter
in	*_p4*	the first parameter
in	*_p5*	the fifth parameter
in	*_p6*	the sixth parameter
in	*_p7*	the seventh parameter

10.46.2.9 template<typename T> template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 , typename P8 > void gazebo::event::EventT< T >::operator() (const P1 & *_p1,* const P2 & *_p2,* const P3 & *_p3,* const P4 & *_p4,* const P5 & *_p5,* const P6 & *_p6,* const P7 & *_p7,* const P8 & *_p8*) [inline]

Signal the event with eight parameters.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter
in	*_p3*	the second parameter
in	*_p4*	the first parameter
in	*_p5*	the fifth parameter
in	*_p6*	the sixth parameter
in	*_p7*	the seventh parameter
in	*_p8*	the eighth parameter

10.46.2.10 template<typename T> template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 , typename P8 , typename P9 > void gazebo::event::EventT< T >::operator() (const P1 & _p1, const P2 & _p2, const P3 & _p3, const P4 & _p4, const P5 & _p5, const P6 & _p6, const P7 & _p7, const P8 & _p8, const P9 & _p9) [inline]

Signal the event with nine parameters.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter
in	*_p3*	the second parameter
in	*_p4*	the first parameter
in	*_p5*	the fifth parameter
in	*_p6*	the sixth parameter
in	*_p7*	the seventh parameter
in	*_p8*	the eighth parameter
in	*_p9*	the ninth parameter

10.46.2.11 template<typename T> template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 , typename P8 , typename P9 , typename P10 > void gazebo::event::EventT< T >::operator() (const P1 & _p1, const P2 & _p2, const P3 & _p3, const P4 & _p4, const P5 & _p5, const P6 & _p6, const P7 & _p7, const P8 & _p8, const P9 & _p9, const P10 & _p10) [inline]

Signal the event with ten parameters.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter
in	*_p3*	the second parameter
in	*_p4*	the first parameter
in	*_p5*	the fifth parameter
in	*_p6*	the sixth parameter
in	*_p7*	the seventh parameter
in	*_p8*	the eighth parameter
in	*_p9*	the ninth parameter
in	*_p10*	the tenth parameter

10.46.2.12 template<typename T> void gazebo::event::EventT< T >::Signal ( ) [inline]

Signal the event for all subscribers.

Referenced by gazebo::event::EventT< void(bool)>::operator()().

10.46.2.13 template<typename T> template<typename P > void gazebo::event::EventT< T >::Signal (const P & _p) [inline]

Signal the event with one parameter.

Parameters

in	*_p*	parameter

10.46.2.14 template<typename T> template<typename P1 , typename P2 > void gazebo::event::EventT< T >::Signal (const P1 & *_p1*, const P2 & *_p2*) [inline]

Signal the event with two parameter.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter

10.46.2.15 template<typename T> template<typename P1 , typename P2 , typename P3 > void gazebo::event::EventT< T >::Signal (const P1 & *_p1*, const P2 & *_p2*, const P3 & *_p3*) [inline]

Signal the event with three parameter.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter
in	*_p3*	the second parameter

10.46.2.16 template<typename T> template<typename P1 , typename P2 , typename P3 , typename P4 > void gazebo::event::EventT< T >::Signal (const P1 & *_p1*, const P2 & *_p2*, const P3 & *_p3*, const P4 & *_p4*) [inline]

Signal the event with four parameter.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter
in	*_p3*	the second parameter
in	*_p4*	the first parameter

10.46.2.17 template<typename T> template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 > void gazebo::event::EventT< T >::Signal (const P1 & *_p1*, const P2 & *_p2*, const P3 & *_p3*, const P4 & *_p4*, const P5 & *_p5*) [inline]

Signal the event with five parameter.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter
in	*_p3*	the second parameter
in	*_p4*	the first parameter
in	*_p5*	the fifth parameter

10.46.2.18 template<typename T> template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 > void gazebo::event::EventT< T >::Signal (const P1 & _p1, const P2 & _p2, const P3 & _p3, const P4 & _p4, const P5 & _p5, const P6 & _p6) [inline]

Signal the event with six parameter.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter
in	*_p3*	the second parameter
in	*_p4*	the first parameter
in	*_p5*	the fifth parameter
in	*_p6*	the sixth parameter

10.46.2.19 template<typename T> template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 > void gazebo::event::EventT< T >::Signal (const P1 & _p1, const P2 & _p2, const P3 & _p3, const P4 & _p4, const P5 & _p5, const P6 & _p6, const P7 & _p7) [inline]

Signal the event with seven parameter.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter
in	*_p3*	the second parameter
in	*_p4*	the first parameter
in	*_p5*	the fifth parameter
in	*_p6*	the sixth parameter
in	*_p7*	the seventh parameter

10.46.2.20 template<typename T> template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 , typename P8 > void gazebo::event::EventT< T >::Signal (const P1 & _p1, const P2 & _p2, const P3 & _p3, const P4 & _p4, const P5 & _p5, const P6 & _p6, const P7 & _p7, const P8 & _p8) [inline]

Signal the event with eight parameter.

Parameters

in	*_p1*	the first parameter
in	*_p2*	the second parameter
in	*_p3*	the second parameter
in	*_p4*	the first parameter
in	*_p5*	the fifth parameter
in	*_p6*	the sixth parameter
in	*_p7*	the seventh parameter
in	*_p8*	the eighth parameter

10.46.2.21 template<typename T> template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 , typename P8 , typename P9 > void gazebo::event::EventT< T >::Signal (const P1 & _p1, const P2 & _p2, const P3 & _p3, const P4 & _p4, const P5 & _p5, const P6 & _p6, const P7 & _p7, const P8 & _p8, const P9 & _p9) [inline]

Signal the event with nine parameter.

Parameters

in	_p1	the first parameter
in	_p2	the second parameter
in	_p3	the second parameter
in	_p4	the first parameter
in	_p5	the fifth parameter
in	_p6	the sixth parameter
in	_p7	the seventh parameter
in	_p8	the eighth parameter
in	_p9	the ninth parameter

10.46.2.22 template<typename T> template<typename P1 , typename P2 , typename P3 , typename P4 , typename P5 , typename P6 , typename P7 , typename P8 , typename P9 , typename P10 > void gazebo::event::EventT< T >::Signal (const P1 & _p1, const P2 & _p2, const P3 & _p3, const P4 & _p4, const P5 & _p5, const P6 & _p6, const P7 & _p7, const P8 & _p8, const P9 & _p9, const P10 & _p10) [inline]

Signal the event with ten parameter.

Parameters

in	_p1	the first parameter
in	_p2	the second parameter
in	_p3	the second parameter
in	_p4	the first parameter
in	_p5	the fifth parameter
in	_p6	the sixth parameter
in	_p7	the seventh parameter
in	_p8	the eighth parameter
in	_p9	the ninth parameter
in	_p10	the tenth parameter

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

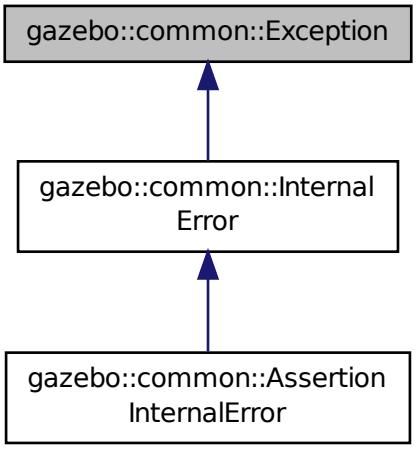
- **Event.hh**

10.47 gazebo::common::Exception Class Reference

Class for generating exceptions.

`#include <common/common.hh>`

Inheritance diagram for gazebo::common::Exception:



Public Member Functions

- **Exception** ()

  Constructor.
- **Exception** (const char ∗_file, int _line, std::string _msg)

  Default constructor.
- virtual ∼**Exception** ()

  Destructor.
- std::string **GetErrorFile** () const

  Return the error function.
- std::string **GetErrorStr** () const

  Return the error string.
- void **Print** () const

  Print the exception to std out.

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::common::Exception** &_err)

  stream insertion operator for Gazebo Error

10.47.1 Detailed Description

Class for generating exceptions.

10.47.2 Constructor & Destructor Documentation

10.47.2.1 gazebo::common::Exception::Exception ( )

Constructor.

10.47.2.2 gazebo::common::Exception::Exception (const char * *_file*, int *_line*, std::string *_msg*)

Default constructor.

Parameters

in	*_file*	File name
in	*_line*	Line number where the error occurred
in	*_msg*	Error message

10.47.2.3 virtual gazebo::common::Exception::~Exception ( ) [virtual]

Destructor.

10.47.3 Member Function Documentation

10.47.3.1 std::string gazebo::common::Exception::GetErrorFile ( ) const

Return the error function.

Returns

The error function name

10.47.3.2 std::string gazebo::common::Exception::GetErrorStr ( ) const

Return the error string.

Returns

The error string

10.47.3.3 void gazebo::common::Exception::Print ( ) const

Print the exception to std out.

10.47.4 Friends And Related Function Documentation

10.47.4.1 std::ostream& operator<< (std::ostream & *_out*, const gazebo::common::Exception & *_err*) [friend]

stream insertion operator for Gazebo Error

Parameters

in	*_out*	the output stream
in	*_err*	the exception

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

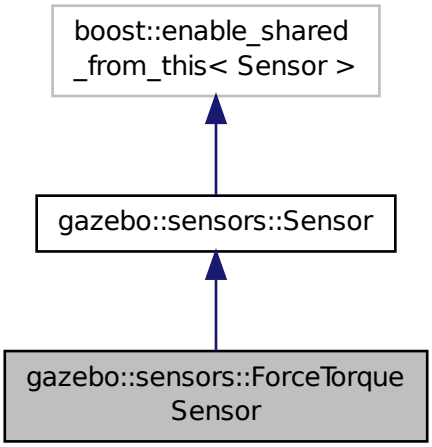
- **Exception.hh**

10.48 gazebo::sensors::ForceTorqueSensor Class Reference

Sensor (p. 731) for measure force and torque on a joint.

`#include <sensors/sensors.hh>`

Inheritance diagram for gazebo::sensors::ForceTorqueSensor:



Public Member Functions

- **ForceTorqueSensor** ()

  Constructor.
- virtual ~**ForceTorqueSensor** ()

  Destructor.
- template<typename T >
  event::ConnectionPtr ConnectUpdate (T _subscriber)

  Connect a to the update signal.
- void **DisconnectUpdate** (**event::ConnectionPtr** &_conn)

  Disconnect from the update signal.
- **math::Vector3 GetForce** () const

Get the current joint force.

- virtual std::string **GetTopic** () const

  Returns the topic name as set in SDF.

- **math::Vector3 GetTorque** () const

  Get the current joint torque.

- virtual void **Init** ()

  Initialize the sensor.

- virtual bool **IsActive** ()

  Returns true if sensor generation is active.

- virtual void **Load** (const std::string &_worldName)

  Load the sensor with default parameters.

Protected Member Functions

- virtual void **Fini** ()

  Finalize the sensor.

- virtual void **UpdateImpl** (bool _force)

  This gets overwritten by derived sensor types.

Protected Attributes

- **event::EventT**< void(msgs::WrenchStamped)> **update**

  Update event.

10.48.1 Detailed Description

Sensor (p. 731) for measure force and torque on a joint.

10.48.2 Constructor & Destructor Documentation

10.48.2.1 gazebo::sensors::ForceTorqueSensor::ForceTorqueSensor ( )

Constructor.

10.48.2.2 virtual gazebo::sensors::ForceTorqueSensor::~ForceTorqueSensor ( ) [virtual]

Destructor.

10.48.3 Member Function Documentation

10.48.3.1 template<typename T > event::ConnectionPtr gazebo::sensors::ForceTorqueSensor::ConnectUpdate (T *_subscriber*) [inline]

Connect a to the update signal.

Parameters

in	*_subscriber*	Callback function.

Returns

The connection, which must be kept in scope.

References gazebo::event::EventT< T >::Connect(), and update.

10.48.3.2 void gazebo::sensors::ForceTorqueSensor::DisconnectUpdate (event::ConnectionPtr & *_conn*) [inline]

Disconnect from the update signal.

Parameters

in	*_conn*	Connection to remove.

References gazebo::event::EventT< T >::Disconnect(), and update.

10.48.3.3 virtual void gazebo::sensors::ForceTorqueSensor::Fini ( ) [protected], [virtual]

Finalize the sensor.

Reimplemented from **gazebo::sensors::Sensor** (p. 735).

10.48.3.4 math::Vector3 gazebo::sensors::ForceTorqueSensor::GetForce ( ) const

Get the current joint force.

Returns

The latested measured force.

10.48.3.5 virtual std::string gazebo::sensors::ForceTorqueSensor::GetTopic ( ) const [virtual]

Returns the topic name as set in SDF.

Returns

Topic name.

Reimplemented from **gazebo::sensors::Sensor** (p. 737).

10.48.3.6 math::Vector3 gazebo::sensors::ForceTorqueSensor::GetTorque ( ) const

Get the current joint torque.

Returns

The latested measured torque.

10.48.3.7 virtual void gazebo::sensors::ForceTorqueSensor::Init ( ) [virtual]

Initialize the sensor.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.48.3.8 virtual bool gazebo::sensors::ForceTorqueSensor::IsActive ( ) [virtual]

Returns true if sensor generation is active.

Returns

True if active, false if not.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.48.3.9 virtual void gazebo::sensors::ForceTorqueSensor::Load (const std::string & *_worldName*) [virtual]

Load the sensor with default parameters.

Parameters

in	*_worldName*	Name of world to load from.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.48.3.10 virtual void gazebo::sensors::ForceTorqueSensor::UpdateImpl (bool) [protected], [virtual]

This gets overwritten by derived sensor types.

```
This function is called during Sensor::Update.
And in turn, Sensor::Update is called by
SensorManager::Update
```

Parameters

in	*_force*	True if update is forced, false if not

Reimplemented from **gazebo::sensors::Sensor** (p. 739).

10.48.4 Member Data Documentation

10.48.4.1 event::EventT<void(msgs::WrenchStamped)> gazebo::sensors::ForceTorqueSensor::update [protected]

Update event.

Referenced by ConnectUpdate(), and DisconnectUpdate().

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

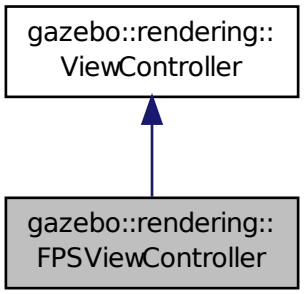
- **ForceTorqueSensor.hh**

10.49 gazebo::rendering::FPSViewController Class Reference

First Person Shooter style view controller.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::FPSViewController:



Public Member Functions

- **FPSViewController** (**UserCameraPtr** _camera)

  Constructor.
- virtual ~**FPSViewController** ()

  Destructor.
- void **HandleKeyPressEvent** (const std::string &_key)

  Handle a key press event.
- void **HandleKeyReleaseEvent** (const std::string &_key)

  Handle a key release event.
- virtual void **HandleMouseEvent** (const **common::MouseEvent** &_event)

  Handle a mouse event.
- virtual void **Init** ()

  Initialize the controller.
- virtual void **Update** ()

  Update the camera position.

Static Public Member Functions

- static std::string **GetTypeString** ()

  Get the type name of this view controller.

Additional Inherited Members

10.49.1 Detailed Description

First Person Shooter style view controller.

10.49.2 Constructor & Destructor Documentation

10.49.2.1 gazebo::rendering::FPSViewController::FPSViewController (UserCameraPtr _camera)

Constructor.

Parameters

in	***Camera*** *(p. 166)*	to controll

10.49.2.2 virtual gazebo::rendering::FPSViewController::~FPSViewController ( ) [virtual]

Destructor.

10.49.3 Member Function Documentation

10.49.3.1 static std::string gazebo::rendering::FPSViewController::GetTypeString ( ) [static]

Get the type name of this view controller.

Returns

The name of the controller type: "fps"

10.49.3.2 void gazebo::rendering::FPSViewController::HandleKeyPressEvent (const std::string & _key) [virtual]

Handle a key press event.

Parameters

in	*_key*	The key that was pressed.

Implements **gazebo::rendering::ViewController** (p. 918).

10.49.3.3 void gazebo::rendering::FPSViewController::HandleKeyReleaseEvent (const std::string & _key) [virtual]

Handle a key release event.

Parameters

in	*_key*	The key that was released.

Implements **gazebo::rendering::ViewController** (p. 918).

10.49.3.4 virtual void gazebo::rendering::FPSViewController::HandleMouseEvent (const common::MouseEvent & _event) [virtual]

Handle a mouse event.

Parameters

in	_event	The mouse position.

Implements **gazebo::rendering::ViewController** (p. 918).

10.49.3.5 virtual void gazebo::rendering::FPSViewController::Init ( ) [virtual]

Initialize the controller.

Implements **gazebo::rendering::ViewController** (p. 919).

10.49.3.6 virtual void gazebo::rendering::FPSViewController::Update ( ) [virtual]

Update the camera position.

Implements **gazebo::rendering::ViewController** (p. 919).

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

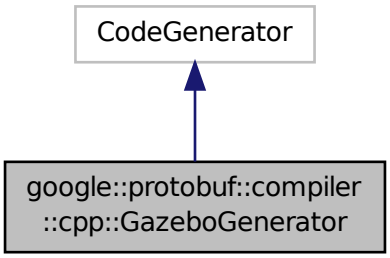
- **FPSViewController.hh**

10.50 google::protobuf::compiler::cpp::GazeboGenerator Class Reference

Google protobuf message generator for **gazebo::msgs** (p. 91).

`#include <GazeboGenerator.hh>`

Inheritance diagram for google::protobuf::compiler::cpp::GazeboGenerator:



Public Member Functions

- **GazeboGenerator** (const std::string &_name)
- virtual ∼**GazeboGenerator** ()
- virtual bool **Generate** (const FileDescriptor ∗file, const string &parameter, OutputDirectory ∗directory, string ∗error) const

10.50.1 Detailed Description

Google protobuf message generator for **gazebo::msgs** (p. 91).

10.50.2 Constructor & Destructor Documentation

10.50.2.1 google::protobuf::compiler::cpp::GazeboGenerator::GazeboGenerator (const std::string & *_name*)

10.50.2.2 virtual google::protobuf::compiler::cpp::GazeboGenerator::∼GazeboGenerator ( ) [virtual]

10.50.3 Member Function Documentation

10.50.3.1 virtual bool google::protobuf::compiler::cpp::GazeboGenerator::Generate (const FileDescriptor ∗ *file,* const string & *parameter,* OutputDirectory ∗ *directory,* string ∗ *error*) const [virtual]

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

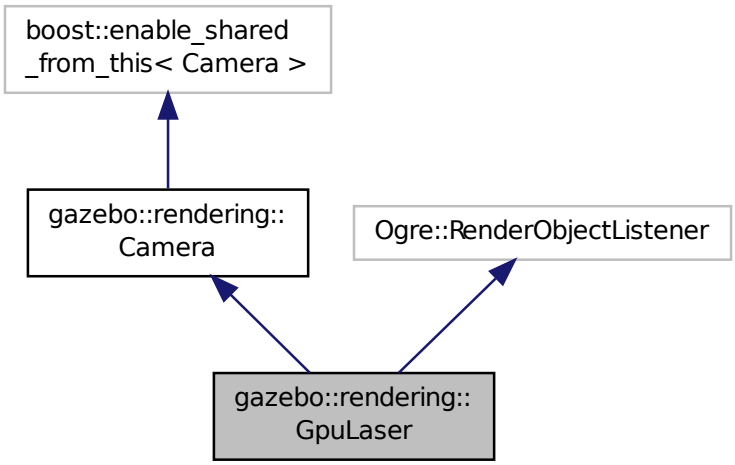
- **GazeboGenerator.hh**

10.51 gazebo::rendering::GpuLaser Class Reference

GPU based laser distance sensor.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::GpuLaser:



Public Member Functions

- **GpuLaser** (const std::string &_namePrefix, **ScenePtr** _scene, bool _autoRender=true)

  Constructor.
- virtual ~**GpuLaser** ()

  Destructor.
- template<typename T >
  event::ConnectionPtr ConnectNewLaserFrame (T _subscriber)

  Connect to a laser frame signal.
- void **CreateLaserTexture** (const std::string &_textureName)

  Create the texture which is used to render laser data.
- void **DisconnectNewLaserFrame** (**event::ConnectionPtr** &_c)

  Disconnect from a laser frame signal.
- virtual void **Fini** ()

  Finalize the camera.
- double **GetCameraCount** () const

  Get the number of cameras required.
- double **GetCosHorzFOV** () const

  Get Cos Horz field-of-view.
- double **GetCosVertFOV** () const

  Get Cos Vert field-of-view.
- double **GetFarClip** () const

  Get far clip.
- double **GetHorzFOV** () const

  Get the horizontal field of view of the laser sensor.

- double **GetHorzHalfAngle** () const

  Get (horizontal_max_angle + horizontal_min_angle) ∗ 0.5.
- const float ∗ **GetLaserData** ()

  All things needed to get back z buffer for laser data.
- double **GetNearClip** () const

  Get near clip.
- double **GetRayCountRatio** () const

  Get the ray count ratio (equivalent to aspect ratio)
- double **GetVertFOV** () const

  Get the vertical field-of-view.
- double **GetVertHalfAngle** () const

  Get (vertical_max_angle + vertical_min_angle) ∗ 0.5.
- virtual void **Init** ()

  Initialize the camera.
- bool **IsHorizontal** () const

  Gets if sensor is horizontal.
- virtual void **Load** (**sdf::ElementPtr** &_sdf)
- virtual void **Load** ()

  Load the camera with default parmeters.
- virtual void **notifyRenderSingleObject** (Ogre::Renderable ∗_rend, const Ogre::Pass ∗_p, const Ogre::AutoParamDataSource ∗_s, const Ogre::LightList ∗_ll, bool _supp)
- virtual void **PostRender** ()

  Post render.
- void **SetCameraCount** (double _cameraCount)

  Set the number of cameras required.
- void **SetCosHorzFOV** (double _chfov)

  Set the Cos Horz FOV.
- void **SetCosVertFOV** (double _cvfov)

  Set the Cos Horz FOV.
- void **SetFarClip** (double _far)

  Set the far clip distance.
- void **SetHorzFOV** (double _hfov)

  Set the horizontal fov.
- void **SetHorzHalfAngle** (double _angle)

  Set the horizontal half angle.
- void **SetIsHorizontal** (bool _horizontal)

  Set sensor horizontal or vertical.
- void **SetNearClip** (double _near)

  Set the near clip distance.
- void **SetRangeCount** (unsigned int _w, unsigned int _h=1)

  Set the number of laser samples in the width and height.
- void **SetRayCountRatio** (double _rayCountRatio)

  Sets the ray count ratio (equivalen to aspect ratio)
- void **SetVertFOV** (double _vfov)

  Set the vertical fov.
- void **SetVertHalfAngle** (double _angle)

  Set the vertical half angle.

Protected Attributes

- unsigned int **cameraCount**

  Number of cameras needed to generate the rays.
- double **chfov**

  Cos horizontal field-of-view.
- double **cvfov**

  Cos vertical field-of-view.
- double **far**

  Far clip plane.
- double **hfov**

  Horizontal field-of-view.
- double **horzHalfAngle**

  Horizontal half angle.
- bool **isHorizontal**

  True if the sensor is horizontal only.
- double **near**

  Near clip plane.
- double **rayCountRatio**

  Ray count ratio.
- double **vertHalfAngle**

  Vertical half angle.
- double **vfov**

  Vertical field-of-view.

Additional Inherited Members

10.51.1 Detailed Description

GPU based laser distance sensor.

10.51.2 Constructor & Destructor Documentation

10.51.2.1 gazebo::rendering::GpuLaser::GpuLaser (const std::string & *_namePrefix,* ScenePtr *_scene,* bool *_autoRender =* `true`)

Constructor.

Parameters

in	*_namePrefix*	Unique prefix name for the camera.
in	*_scene*	**Scene** (p. 707) that will contain the camera
in	*_autoRender*	Almost everyone should leave this as true.

10.51.2.2 virtual gazebo::rendering::GpuLaser::~GpuLaser ( ) [virtual]

Destructor.

10.51.3 Member Function Documentation

10.51.3.1 template<typename T > event::ConnectionPtr gazebo::rendering::GpuLaser::ConnectNewLaserFrame (T _subscriber) [inline]

Connect to a laser frame signal.

Parameters

in	*_subscriber*	Callback that is called when a new image is generated

Returns

A pointer to the connection. This must be kept in scope.

References gazebo::event::EventT< T >::Connect().

10.51.3.2 void gazebo::rendering::GpuLaser::CreateLaserTexture (const std::string & _textureName)

Create the texture which is used to render laser data.

Parameters

in	*_textureName*	Name of the new texture.

10.51.3.3 void gazebo::rendering::GpuLaser::DisconnectNewLaserFrame (event::ConnectionPtr & _c) [inline]

Disconnect from a laser frame signal.

Parameters

in	*_c*	The connection to disconnect

References gazebo::event::EventT< T >::Disconnect().

10.51.3.4 virtual void gazebo::rendering::GpuLaser::Fini ( ) [virtual]

Finalize the camera.

This function is called before the camera is destructed

Reimplemented from **gazebo::rendering::Camera** (p. 175).

10.51.3.5 double gazebo::rendering::GpuLaser::GetCameraCount ( ) const

Get the number of cameras required.

Returns

Number of cameras needed to generate the rays

10.51.3.6 double gazebo::rendering::GpuLaser::GetCosHorzFOV ( ) const

Get Cos Horz field-of-view.

Returns

2 ∗ atan(tan(this->hfov/2) / cos(this->vfov/2))

10.51.3.7 double gazebo::rendering::GpuLaser::GetCosVertFOV ( ) const

Get Cos Vert field-of-view.

Returns

2 ∗ atan(tan(this->vfov/2) / cos(this->hfov/2))

10.51.3.8 double gazebo::rendering::GpuLaser::GetFarClip ( ) const

Get far clip.

Returns

far clip distance

10.51.3.9 double gazebo::rendering::GpuLaser::GetHorzFOV ( ) const

Get the horizontal field of view of the laser sensor.

Returns

The horizontal field of view of the laser sensor.

10.51.3.10 double gazebo::rendering::GpuLaser::GetHorzHalfAngle ( ) const

Get (horizontal_max_angle + horizontal_min_angle) ∗ 0.5.

Returns

(horizontal_max_angle + horizontal_min_angle) ∗ 0.5

10.51.3.11 const float∗ gazebo::rendering::GpuLaser::GetLaserData ( )

All things needed to get back z buffer for laser data.

Returns

Array of laser data.

10.51.3.12 double gazebo::rendering::GpuLaser::GetNearClip ( ) const

Get near clip.

Returns

near clip distance

10.51.3.13 double gazebo::rendering::GpuLaser::GetRayCountRatio ( ) const

Get the ray count ratio (equivalent to aspect ratio)

Returns

The ray count ratio (equivalent to aspect ratio)

10.51.3.14 double gazebo::rendering::GpuLaser::GetVertFOV ( ) const

Get the vertical field-of-view.

Returns

The vertical field of view of the laser sensor.

10.51.3.15 double gazebo::rendering::GpuLaser::GetVertHalfAngle ( ) const

Get (vertical_max_angle + vertical_min_angle) $*$ 0.5.

Returns

(vertical_max_angle + vertical_min_angle) $*$ 0.5

10.51.3.16 virtual void gazebo::rendering::GpuLaser::Init ( ) [virtual]

Initialize the camera.

Reimplemented from **gazebo::rendering::Camera** (p. 182).

10.51.3.17 bool gazebo::rendering::GpuLaser::IsHorizontal ( ) const

Gets if sensor is horizontal.

Returns

True if horizontal, false if not

10.51.3.18 virtual void gazebo::rendering::GpuLaser::Load (sdf::ElementPtr & _sdf) [virtual]

10.51.3.19 virtual void gazebo::rendering::GpuLaser::Load ( ) [virtual]

Load the camera with default parmeters.

Reimplemented from **gazebo::rendering::Camera** (p. 183).

10.51.3.20 virtual void gazebo::rendering::GpuLaser::notifyRenderSingleObject (Ogre::Renderable * _rend, const Ogre::Pass * _p, const Ogre::AutoParamDataSource * _s, const Ogre::LightList * _ll, bool _supp) [virtual]

10.51.3.21 virtual void gazebo::rendering::GpuLaser::PostRender ( ) [virtual]

Post render.

Called afer the render signal.

Reimplemented from **gazebo::rendering::Camera** (p. 184).

10.51.3.22 void gazebo::rendering::GpuLaser::SetCameraCount (double _cameraCount)

Set the number of cameras required.

Parameters

in	*_cameraCount*	The number of cameras required to generate the rays

10.51.3.23 void gazebo::rendering::GpuLaser::SetCosHorzFOV (double _chfov)

Set the Cos Horz FOV.

Parameters

in	*_chfov*	Cos Horz FOV

10.51.3.24 void gazebo::rendering::GpuLaser::SetCosVertFOV (double _cvfov)

Set the Cos Horz FOV.

Parameters

in	*_cvfov*	Cos Horz FOV

10.51.3.25 void gazebo::rendering::GpuLaser::SetFarClip (double _far)

Set the far clip distance.

Parameters

in	*_far*	far clip distance

10.51.3.26 void gazebo::rendering::GpuLaser::SetHorzFOV (double _hfov)

Set the horizontal fov.

Parameters

in	*_hfov*	horizontal fov

10.51.3.27 void gazebo::rendering::GpuLaser::SetHorzHalfAngle (double _angle)

Set the horizontal half angle.

Parameters

in	*_angle*	horizontal half angle

10.51.3.28 void gazebo::rendering::GpuLaser::SetIsHorizontal (bool _horizontal)

Set sensor horizontal or vertical.

Parameters

in	*_horizontal*	True if horizontal, false if not

10.51.3.29 void gazebo::rendering::GpuLaser::SetNearClip (double _near)

Set the near clip distance.

Parameters

in	*_near*	near clip distance

10.51.3.30 void gazebo::rendering::GpuLaser::SetRangeCount (unsigned int _w, unsigned int _h = 1)

Set the number of laser samples in the width and height.

Parameters

in	*_w*	Number of samples in the horizontal sweep
in	*_h*	Number of samples in the vertical sweep

10.51.3.31 void gazebo::rendering::GpuLaser::SetRayCountRatio (double _rayCountRatio)

Sets the ray count ratio (equivalen to aspect ratio)

Parameters

in	*_rayCountRatio*	ray count ratio (equivalent to aspect ratio)

10.51.3.32 void gazebo::rendering::GpuLaser::SetVertFOV (double *_vfov*)

Set the vertical fov.

Parameters

in	*_vfov*	vertical fov

10.51.3.33 void gazebo::rendering::GpuLaser::SetVertHalfAngle (double *_angle*)

Set the vertical half angle.

Parameters

in	*_angle*	vertical half angle

10.51.4 Member Data Documentation

10.51.4.1 unsigned int gazebo::rendering::GpuLaser::cameraCount [protected]

Number of cameras needed to generate the rays.

10.51.4.2 double gazebo::rendering::GpuLaser::chfov [protected]

Cos horizontal field-of-view.

10.51.4.3 double gazebo::rendering::GpuLaser::cvfov [protected]

Cos vertical field-of-view.

10.51.4.4 double gazebo::rendering::GpuLaser::far [protected]

Far clip plane.

10.51.4.5 double gazebo::rendering::GpuLaser::hfov [protected]

Horizontal field-of-view.

10.51.4.6 double gazebo::rendering::GpuLaser::horzHalfAngle [protected]

Horizontal half angle.

10.51.4.7 bool gazebo::rendering::GpuLaser::isHorizontal [protected]

True if the sensor is horizontal only.

10.51.4.8 double gazebo::rendering::GpuLaser::near [protected]

Near clip plane.

10.51.4.9 double gazebo::rendering::GpuLaser::rayCountRatio [protected]

Ray count ratio.

10.51.4.10 double gazebo::rendering::GpuLaser::vertHalfAngle [protected]

Vertical half angle.

10.51.4.11 double gazebo::rendering::GpuLaser::vfov [protected]

Vertical field-of-view.

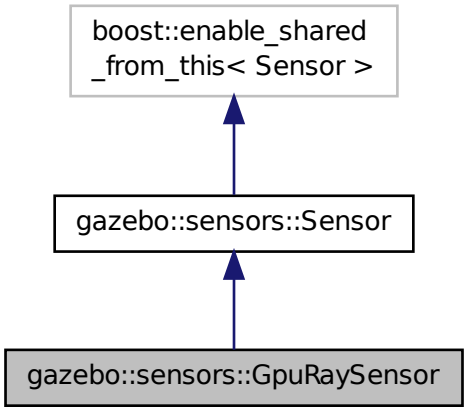
The documentation for this class was generated from the following file:

- **GpuLaser.hh**

10.52 gazebo::sensors::GpuRaySensor Class Reference

`#include <sensors/sensors.hh>`

Inheritance diagram for gazebo::sensors::GpuRaySensor:



Public Member Functions

- **GpuRaySensor** ()

Constructor.

- virtual ~**GpuRaySensor** ()

  Destructor.

- **event::ConnectionPtr ConnectNewLaserFrame** (boost::function< void(const float *, unsigned int, unsigned int, unsigned int, const std::string &)> _subscriber)

  Connect to the new laser frame event.

- void **DisconnectNewLaserFrame** (**event::ConnectionPtr** &_conn)

  Disconnect Laser Frame.

- **math::Angle GetAngleMax** () const

  Get the maximum angle.

- **math::Angle GetAngleMin** () const

  Get the minimum angle.

- double **GetAngleResolution** () const

  Get radians between each range.

- unsigned int **GetCameraCount** () const

  Gets the camera count.

- double **GetCosHorzFOV** () const

  Get Cos Horz field-of-view.

- double **GetCosVertFOV** () const

  Get Cos Vert field-of-view.

- int **GetFiducial** (int _index) const

  Get detected fiducial value for a ray.

- double **GetHorzFOV** () const

  Get the horizontal field of view of the laser sensor.

- double **GetHorzHalfAngle** () const

  *Get (horizontal_max_angle + horizontal_min_angle) * 0.5.*

- **rendering::GpuLaserPtr GetLaserCamera** () const

  *Returns a pointer to the internally kept **rendering::GpuLaser** (p. 335).*

- double **GetRange** (int _index)

  Get detected range for a ray.

- int **GetRangeCount** () const

  Get the range count.

- double **GetRangeCountRatio** () const

  Return the ratio of horizontal range count to vertical range count.

- double **GetRangeMax** () const

  Get the maximum range.

- double **GetRangeMin** () const

  Get the minimum range.

- double **GetRangeResolution** () const

  Get the range resolution If RangeResolution is 1, the number of simulated rays is equal to the number of returned range readings.

- void **GetRanges** (std::vector< double > &_ranges)

  Get all the ranges.

- int **GetRayCount** () const

  Get the ray count.

- double **GetRayCountRatio** () const

  Return the ratio of horizontal ray count to vertical ray count.

- double **GetRetro** (int _index) const

  Get detected retro (intensity) value for a ray.
- virtual std::string **GetTopic** () const

  Returns the topic name as set in SDF.
- double **GetVertFOV** () const

  Get the vertical field-of-view.
- double **GetVertHalfAngle** () const

  Get (vertical_max_angle + vertical_min_angle) ∗ 0.5.
- **math::Angle GetVerticalAngleMax** () const

  Get the vertical scan line top angle.
- **math::Angle GetVerticalAngleMin** () const

  Get the vertical scan bottom angle.
- int **GetVerticalRangeCount** () const

  Get the vertical scan line count.
- int **GetVerticalRayCount** () const

  Get the vertical scan line count.
- virtual void **Init** ()

  Initialize the ray.
- virtual bool **IsActive** ()

  Returns true if sensor generation is active.
- bool **IsHorizontal** () const

  Gets if sensor is horizontal.
- virtual void **Load** (const std::string &_worldName, **sdf::ElementPtr** &_sdf)

  Load the sensor with SDF parameters.
- virtual void **Load** (const std::string &_worldName)

  Load the sensor with default parameters.
- void **SetAngleMax** (double _angle)

  Set the scan maximum angle.
- void **SetAngleMin** (double _angle)

  Set the scan minimum angle.
- void **SetVerticalAngleMax** (double _angle)

  Set the vertical scan line top angle.
- void **SetVerticalAngleMin** (double _angle)

  Set the vertical scan bottom angle.

Protected Member Functions

- virtual void **Fini** ()

  Finalize the ray.
- virtual void **UpdateImpl** (bool _force)

  Update the sensor information.

Protected Attributes

- **sdf::ElementPtr cameraElem**

  Camera SDF element.
- **sdf::ElementPtr horzElem**

  Horizontal SDF element.
- unsigned int **horzRangeCount**

  Horizontal range count.
- unsigned int **horzRayCount**

  Horizontal ray count.
- double **rangeCountRatio**

  Range count ratio.
- **sdf::ElementPtr rangeElem**

  Range SDF element.
- **sdf::ElementPtr scanElem**

  Scan SDF elementz.
- **sdf::ElementPtr vertElem**

  Vertical SDF element.
- unsigned int **vertRangeCount**

  Vertical range count.
- unsigned int **vertRayCount**

  Vertical ray count.

10.52.1 Constructor & Destructor Documentation

10.52.1.1 gazebo::sensors::GpuRaySensor::GpuRaySensor ( )

Constructor.

10.52.1.2 virtual gazebo::sensors::GpuRaySensor::~GpuRaySensor ( ) [virtual]

Destructor.

10.52.2 Member Function Documentation

10.52.2.1 event::ConnectionPtr gazebo::sensors::GpuRaySensor::ConnectNewLaserFrame (boost::function< void(const float *, unsigned int, unsigned int, unsigned int, const std::string &)> *_subscriber*)

Connect to the new laser frame event.

Parameters

in	*_subscriber*	Event callback.

10.52.2.2 void gazebo::sensors::GpuRaySensor::DisconnectNewLaserFrame (event::ConnectionPtr & *_conn*)

Disconnect Laser Frame.

Parameters

in,out	*_conn*	Connection pointer to disconnect.

10.52.2.3 virtual void gazebo::sensors::GpuRaySensor::Fini ( ) [protected],[virtual]

Finalize the ray.

Reimplemented from **gazebo::sensors::Sensor** (p. 735).

10.52.2.4 math::Angle gazebo::sensors::GpuRaySensor::GetAngleMax ( ) const

Get the maximum angle.

Returns

the maximum angle

10.52.2.5 math::Angle gazebo::sensors::GpuRaySensor::GetAngleMin ( ) const

Get the minimum angle.

Returns

The minimum angle

10.52.2.6 double gazebo::sensors::GpuRaySensor::GetAngleResolution ( ) const

Get radians between each range.

10.52.2.7 unsigned int gazebo::sensors::GpuRaySensor::GetCameraCount ( ) const

Gets the camera count.

Returns

Number of cameras

10.52.2.8 double gazebo::sensors::GpuRaySensor::GetCosHorzFOV ( ) const

Get Cos Horz field-of-view.

Returns

2 ∗ atan(tan(this->hfov/2) / cos(this->vfov/2))

10.52.2.9 double gazebo::sensors::GpuRaySensor::GetCosVertFOV ( ) const

Get Cos Vert field-of-view.

Returns

2 ∗ atan(tan(this->vfov/2) / cos(this->hfov/2))

10.52.2.10 int gazebo::sensors::GpuRaySensor::GetFiducial (int *_index*) const

Get detected fiducial value for a ray.

```
Warning: If you are accessing all the ray data in a loop
it's possible that the Ray will update in the middle of
your access loop. This means some data will come from one
scan, and some from another scan. You can solve this
problem by using SetActive(false) <your accessor loop>
SetActive(true).
```

Parameters

in	*_index*	Index of specific ray

Returns

Fiducial value of ray

10.52.2.11 double gazebo::sensors::GpuRaySensor::GetHorzFOV ( ) const

Get the horizontal field of view of the laser sensor.

Returns

The horizontal field of view of the laser sensor.

10.52.2.12 double gazebo::sensors::GpuRaySensor::GetHorzHalfAngle ( ) const

Get (horizontal_max_angle + horizontal_min_angle) ∗ 0.5.

Returns

(horizontal_max_angle + horizontal_min_angle) ∗ 0.5

10.52.2.13 rendering::GpuLaserPtr gazebo::sensors::GpuRaySensor::GetLaserCamera ( ) const [inline]

Returns a pointer to the internally kept **rendering::GpuLaser** (p. 335).

Returns

Pointer to GpuLaser

10.52.2.14 double gazebo::sensors::GpuRaySensor::GetRange (int *_index*)

Get detected range for a ray.

```
Warning: If you are accessing all the ray data in a loop
it's possible that the Ray will update in the middle of
your access loop. This means some data will come from one
scan, and some from another scan. You can solve this
problem by using SetActive(false) <your accessor loop>
SetActive(true).
```

Parameters

in	*_index*	Index of specific ray

Returns

Returns DBL_MAX for no detection.

10.52.2.15 int gazebo::sensors::GpuRaySensor::GetRangeCount ( ) const

Get the range count.

Returns

The number of ranges

10.52.2.16 double gazebo::sensors::GpuRaySensor::GetRangeCountRatio ( ) const

Return the ratio of horizontal range count to vertical range count.

A ray count is the number of simulated rays. Whereas a range count is the total number of data points returned. When range count != ray count, then values are interpolated between rays.

10.52.2.17 double gazebo::sensors::GpuRaySensor::GetRangeMax ( ) const

Get the maximum range.

Returns

The maximum range

10.52.2.18 double gazebo::sensors::GpuRaySensor::GetRangeMin ( ) const

Get the minimum range.

Returns

The minimum range

10.52.2.19 double gazebo::sensors::GpuRaySensor::GetRangeResolution ( ) const

Get the range resolution If RangeResolution is 1, the number of simulated rays is equal to the number of returned range readings.

If it's less than 1, fewer simulated rays than actual returned range readings are used, the results are interpolated from two nearest neighbors, and vice versa.

Returns

The Range Resolution

10.52.2.20 void gazebo::sensors::GpuRaySensor::GetRanges (std::vector< double > & *_ranges*)

Get all the ranges.

Parameters

out	*_range*	A vector that will contain all the range data

10.52.2.21 int gazebo::sensors::GpuRaySensor::GetRayCount ( ) const

Get the ray count.

Returns

The number of rays

10.52.2.22 double gazebo::sensors::GpuRaySensor::GetRayCountRatio ( ) const

Return the ratio of horizontal ray count to vertical ray count.

A ray count is the number of simulated rays. Whereas a range count is the total number of data points returned. When range count != ray count, then values are interpolated between rays.

10.52.2.23 double gazebo::sensors::GpuRaySensor::GetRetro (int *_index*) const

Get detected retro (intensity) value for a ray.

```
Warning: If you are accessing all the ray data in a loop
it's possible that the Ray will update in the middle of
your access loop. This means some data will come from one
scan, and some from another scan. You can solve this
problem by using SetActive(false) <your accessor loop>
SetActive(true).
```

Parameters

in	*_index*	Index of specific ray

Returns

Intensity value of ray

10.52.2.24 virtual std::string gazebo::sensors::GpuRaySensor::GetTopic ( ) const [virtual]

Returns the topic name as set in SDF.

Returns

Topic name.

Reimplemented from **gazebo::sensors::Sensor** (p. 737).

10.52.2.25 double gazebo::sensors::GpuRaySensor::GetVertFOV ( ) const

Get the vertical field-of-view.

10.52.2.26 double gazebo::sensors::GpuRaySensor::GetVertHalfAngle ( ) const

Get (vertical_max_angle + vertical_min_angle) $*$ 0.5.

Returns

(vertical_max_angle + vertical_min_angle) $*$ 0.5

10.52.2.27 math::Angle gazebo::sensors::GpuRaySensor::GetVerticalAngleMax ( ) const

Get the vertical scan line top angle.

Returns

The Maximum angle of the scan block

10.52.2.28 math::Angle gazebo::sensors::GpuRaySensor::GetVerticalAngleMin ( ) const

Get the vertical scan bottom angle.

Returns

The minimum angle of the scan block

10.52.2.29 int gazebo::sensors::GpuRaySensor::GetVerticalRangeCount ( ) const

Get the vertical scan line count.

Returns

The number of scan lines vertically

10.52.2.30 int gazebo::sensors::GpuRaySensor::GetVerticalRayCount ( ) const

Get the vertical scan line count.

Returns

The number of scan lines vertically

10.52.2.31 virtual void gazebo::sensors::GpuRaySensor::Init ( ) [virtual]

Initialize the ray.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.52.2.32 virtual bool gazebo::sensors::GpuRaySensor::IsActive ( ) [virtual]

Returns true if sensor generation is active.

Returns

True if active, false if not.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.52.2.33 bool gazebo::sensors::GpuRaySensor::IsHorizontal ( ) const

Gets if sensor is horizontal.

Returns

True if horizontal, false if not

10.52.2.34 virtual void gazebo::sensors::GpuRaySensor::Load (const std::string & *_worldName*, sdf::ElementPtr & *_sdf*) [virtual]

Load the sensor with SDF parameters.

Parameters

in	*_sdf*	SDF **Sensor** (p. 731) parameters
in	*_worldName*	Name of world to load from

10.52.2.35 virtual void gazebo::sensors::GpuRaySensor::Load (const std::string & *_worldName*) [virtual]

Load the sensor with default parameters.

Parameters

in	*_worldName*	Name of world to load from

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.52.2.36 void gazebo::sensors::GpuRaySensor::SetAngleMax (double *_angle*)

Set the scan maximum angle.

Parameters

in	*_angle*	The maximum angle

10.52.2.37 void gazebo::sensors::GpuRaySensor::SetAngleMin (double *_angle*)

Set the scan minimum angle.

Parameters

in	*_angle*	The minimum angle

10.52.2.38 void gazebo::sensors::GpuRaySensor::SetVerticalAngleMax (double *_angle*)

Set the vertical scan line top angle.

Parameters

in	*_angle*	The Maximum angle of the scan block

10.52.2.39 void gazebo::sensors::GpuRaySensor::SetVerticalAngleMin (double *_angle*)

Set the vertical scan bottom angle.

Parameters

in	*_angle*	The minimum angle of the scan block

10.52.2.40 virtual void gazebo::sensors::GpuRaySensor::UpdateImpl (bool *_force*) [protected], [virtual]

Update the sensor information.

Parameters

in	*_force*	True if update is forced, false if not

Reimplemented from **gazebo::sensors::Sensor** (p. 739).

10.52.3 Member Data Documentation

10.52.3.1 sdf::ElementPtr gazebo::sensors::GpuRaySensor::cameraElem [protected]

Camera SDF element.

10.52.3.2 sdf::ElementPtr gazebo::sensors::GpuRaySensor::horzElem [protected]

Horizontal SDF element.

10.52.3.3 unsigned int gazebo::sensors::GpuRaySensor::horzRangeCount [protected]

Horizontal range count.

10.52.3.4 unsigned int gazebo::sensors::GpuRaySensor::horzRayCount [protected]

Horizontal ray count.

10.52.3.5 double gazebo::sensors::GpuRaySensor::rangeCountRatio [protected]

Range count ratio.

10.52.3.6 sdf::ElementPtr gazebo::sensors::GpuRaySensor::rangeElem [protected]

Range SDF element.

10.52.3.7 sdf::ElementPtr gazebo::sensors::GpuRaySensor::scanElem [protected]

Scan SDF elementz.

10.52.3.8 sdf::ElementPtr gazebo::sensors::GpuRaySensor::vertElem [protected]

Vertical SDF element.

10.52.3.9 unsigned int gazebo::sensors::GpuRaySensor::vertRangeCount [protected]

Vertical range count.

10.52.3.10 unsigned int gazebo::sensors::GpuRaySensor::vertRayCount [protected]

Vertical ray count.

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

- **GpuRaySensor.hh**

10.53 gazebo::rendering::Grid Class Reference

Displays a grid of cells, drawn with lines.

`#include <rendering/rendering.hh>`

Public Member Functions

- **Grid** (**Scene** *_scene, uint32_t _cellCount, float _cellLength, float _lineWidth, const **common::Color** &_color)

  Constructor.
- ~**Grid** ()

  Destructor.
- void **Enable** (bool _enable)

  Enable or disable the grid.
- uint32_t **GetCellCount** () const

  Get the number of cells.
- float **GetCellLength** () const

  Get the cell length.
- **common::Color GetColor** () const

  Return the grid color.
- uint32_t **GetHeight** () const

  Get the height of the grid.
- float **GetLineWidth** () const

  Get the width of the grid line.
- Ogre::SceneNode * **GetSceneNode** ()

  *Get the **Ogre** (p. 110) scene node associated with this grid.*
- void **Init** ()

  Initialize the grid.
- void **SetCellCount** (uint32_t _count)

  Set the number of cells.
- void **SetCellLength** (float _len)

  Set the cell length.
- void **SetColor** (const **common::Color** &_color)

  Sets the color of the grid.
- void **SetHeight** (uint32_t _count)

  Set the height of the grid.
- void **SetLineWidth** (float _width)

  Set the line width.
- void **SetUserData** (const Ogre::Any &_data)

  Sets user data on all ogre objects we own.

10.53.1 Detailed Description

Displays a grid of cells, drawn with lines.

Displays a grid of cells, drawn with lines. A grid with an identity orientation is drawn along the XY plane.

10.53.2 Constructor & Destructor Documentation

10.53.2.1 gazebo::rendering::Grid::Grid (Scene * _scene, uint32_t _cellCount, float _cellLength, float _lineWidth, const common::Color & _color)

Constructor.

Parameters

in	*_scene*	The scene this object is part of
in	*_cellCount*	The number of cells to draw
in	*_cellLength*	The size of each cell
in	*_lineWidth*	The width of the lines to use
in	*_color*	The color of the grid

10.53.2.2 gazebo::rendering::Grid::~Grid ( )

Destructor.

10.53.3 Member Function Documentation

10.53.3.1 void gazebo::rendering::Grid::Enable (bool _enable)

Enable or disable the grid.

Parameters

in	*_enable*	Set to true to view the grid, false to make invisible.

10.53.3.2 uint32_t gazebo::rendering::Grid::GetCellCount ( ) const [inline]

Get the number of cells.

10.53.3.3 float gazebo::rendering::Grid::GetCellLength ( ) const [inline]

Get the cell length.

Returns

The cell length

10.53.3.4 common::Color gazebo::rendering::Grid::GetColor ( ) const [inline]

Return the grid color.

Returns

The grid color

10.53.3.5 uint32_t gazebo::rendering::Grid::GetHeight ( ) const [inline]

Get the height of the grid.

Returns

The height

10.53.3.6 float gazebo::rendering::Grid::GetLineWidth ( ) const [inline]

Get the width of the grid line.

Returns

The line width

10.53.3.7 Ogre::SceneNode* gazebo::rendering::Grid::GetSceneNode ( ) [inline]

Get the **Ogre** (p. 110) scene node associated with this grid.

Returns

The **Ogre** (p. 110) scene node associated with this grid

10.53.3.8 void gazebo::rendering::Grid::Init ( )

Initialize the grid.

10.53.3.9 void gazebo::rendering::Grid::SetCellCount (uint32_t *_count*)

Set the number of cells.

Parameters

in	*_count*	The number of cells

10.53.3.10 void gazebo::rendering::Grid::SetCellLength (float *_len*)

Set the cell length.

Parameters

in	*_len*	The cell length

10.53.3.11 void gazebo::rendering::Grid::SetColor (const common::Color & *_color*)

Sets the color of the grid.

Parameters

in	*_color*	The grid color

10.53.3.12 void gazebo::rendering::Grid::SetHeight (uint32_t *_count*)

Set the height of the grid.

Parameters

in	*_count*	**Grid** (p. 357) height

10.53.3.13 void gazebo::rendering::Grid::SetLineWidth (float *_width*)

Set the line width.

Parameters

in	*_width*	The width of the grid

10.53.3.14 void gazebo::rendering::Grid::SetUserData (const Ogre::Any & *_data*)

Sets user data on all ogre objects we own.

Parameters

in	*_data*	The user data

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

- **Grid.hh**

10.54 gazebo::physics::Gripper Class Reference

A gripper abstraction.

`#include <physics/physics.hh>`

Public Member Functions

- **Gripper** (**ModelPtr** _model)
  Constructor.
- virtual ~**Gripper** ()
  Destructor.
- virtual void **Init** ()
  Initialize.
- virtual void **Load** (**sdf::ElementPtr** _sdf)
  Load the gripper.

10.54.1 Detailed Description

A gripper abstraction.

A gripper is a collection of links that act as a gripper. This class will intelligently generate fixed joints between the gripper and an object within the gripper. This allows the object to be manipulated without falling or behaving poorly.

10.54.2 Constructor & Destructor Documentation

10.54.2.1 gazebo::physics::Gripper::Gripper (ModelPtr _model) [explicit]

Constructor.

Parameters

in	*_model*	The model which contains the **Gripper** (p. 360).

10.54.2.2 virtual gazebo::physics::Gripper::~Gripper ( ) [virtual]

Destructor.

10.54.3 Member Function Documentation

10.54.3.1 virtual void gazebo::physics::Gripper::Init ( ) [virtual]

Initialize.

10.54.3.2 virtual void gazebo::physics::Gripper::Load (sdf::ElementPtr _sdf) [virtual]

Load the gripper.

Parameters

in	*_sdf*	Shared point to an sdf element that contains the list of links in the gripper.

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

- **Gripper.hh**

10.55 gazebo::rendering::GUIOverlay Class Reference

A class that creates a CEGUI overlay on a render window.

`#include <rendering/rendering.hh>`

Public Member Functions

- **GUIOverlay** ()

  Constructor.

- virtual ~**GUIOverlay** ()

  Destructor.
- bool **AttachCameraToImage** (**CameraPtr** &_camera, const std::string &_windowName)

  *Use this function to draw the output from a **rendering::Camera** (p. 166) to and overlay window.*
- bool **AttachCameraToImage** (**DepthCameraPtr** &_camera, const std::string &_windowName)

  *Use this function to draw the output from a **rendering::DepthCamera** (p. 260) to and overlay window.*
- template<typename T >
  void **ButtonCallback** (const std::string &_buttonName, void(T::*_fp)(), T *_obj)

  Register a CEGUI button callback.
- void **CreateWindow** (const std::string &_type, const std::string &_name, const std::string &_parent, const **math::Vector2d** &_position, const **math::Vector2d** &_size, const std::string &_text)

  Create a new window on the overlay.
- bool **HandleKeyPressEvent** (const std::string &_key)

  Handle a key press event.
- bool **HandleKeyReleaseEvent** (const std::string &_key)

  Handle a key release event.
- bool **HandleMouseEvent** (const **common::MouseEvent** &_evt)

  Handle a mouse event.
- void **Hide** ()

  Make the overlay invisible.
- void **Init** (Ogre::RenderTarget *_renderTarget)

  Initialize the overlay.
- bool **IsInitialized** ()

  Return true if the overlay has been initialized.
- void **LoadLayout** (const std::string &_filename)

  Load a CEGUI layout file.
- void **Resize** (unsigned int _width, unsigned int _height)

  Resize the window.
- void **Show** ()

  Make the overlay visible.
- void **Update** ()

  Update the overlay's objects.

10.55.1 Detailed Description

A class that creates a CEGUI overlay on a render window.

10.55.2 Constructor & Destructor Documentation

10.55.2.1 gazebo::rendering::GUIOverlay::GUIOverlay ( )

Constructor.

10.55.2.2 virtual gazebo::rendering::GUIOverlay::~GUIOverlay ( ) [virtual]

Destructor.

10.55.3 Member Function Documentation

10.55.3.1 bool gazebo::rendering::GUIOverlay::AttachCameraToImage (CameraPtr & _camera, const std::string & _windowName)

Use this function to draw the output from a **rendering::Camera** (p. 166) to and overlay window.

Parameters

in	*_camera*	Pointer to the camera.
in	*_windowName*	Name of the window to receive the camera image

Returns

True if successful

10.55.3.2 bool gazebo::rendering::GUIOverlay::AttachCameraToImage (DepthCameraPtr & _camera, const std::string & _windowName)

Use this function to draw the output from a **rendering::DepthCamera** (p. 260) to and overlay window.

Parameters

in	*_camera*	Pointer to the camera.
in	*_windowName*	Name of the window to receive the camera image

Returns

True if successful

10.55.3.3 template<typename T > void gazebo::rendering::GUIOverlay::ButtonCallback (const std::string & _buttonName, void(T::*)() _fp, T * _obj) [inline]

Register a CEGUI button callback.

Assign a callback to a name button.

Parameters

in	*_buttonName*	Name of the button.
in	*_fp*	Function pointer to the callback.
in	*_obj*	Class pointer that contains _fp.

10.55.3.4 void gazebo::rendering::GUIOverlay::CreateWindow (const std::string & _type, const std::string & _name, const std::string & _parent, const math::Vector2d & _position, const math::Vector2d & _size, const std::string & _text)

Create a new window on the overlay.

Parameters

in	*_type*	The window type. This should match a CEGUI window type. See CEGUI::Window-Manager::getSingleton().createWindow().
in	*_name*	Unique name for the window.
in	*_parent*	Name of the parent window.
in	*_position*	Position of the window within the parent.
in	*_size*	Size of the window.
in	*_text*	Display title of the window.

10.55.3.5 bool gazebo::rendering::GUIOverlay::HandleKeyPressEvent (const std::string & *_key*)

Handle a key press event.

Parameters

in	*_key*	The key pressed.

Returns

True if the key press event was handled.

10.55.3.6 bool gazebo::rendering::GUIOverlay::HandleKeyReleaseEvent (const std::string & *_key*)

Handle a key release event.

Parameters

in	*_key*	The key released.

Returns

True if the key release event was handled.

10.55.3.7 bool gazebo::rendering::GUIOverlay::HandleMouseEvent (const common::MouseEvent & *_evt*)

Handle a mouse event.

Parameters

in	*_evt*	The mouse event.

Returns

True if the mouse event was handled.

10.55.3.8 void gazebo::rendering::GUIOverlay::Hide ( )

Make the overlay invisible.

10.55.3.9 void gazebo::rendering::GUIOverlay::Init (Ogre::RenderTarget * _renderTarget)

Initialize the overlay.

Parameters

in	*_renderTarget*	The render target which will have the overlay.

10.55.3.10 bool gazebo::rendering::GUIOverlay::IsInitialized ( )

Return true if the overlay has been initialized.

Returns

True if initialized

10.55.3.11 void gazebo::rendering::GUIOverlay::LoadLayout (const std::string & _filename)

Load a CEGUI layout file.

Parameters

in	*_filename*	Name of the layout file.

10.55.3.12 void gazebo::rendering::GUIOverlay::Resize (unsigned int _width, unsigned int _height)

Resize the window.

10.55.3.13 void gazebo::rendering::GUIOverlay::Show ( )

Make the overlay visible.

10.55.3.14 void gazebo::rendering::GUIOverlay::Update ( )

Update the overlay's objects.

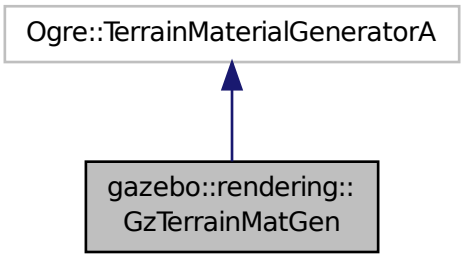
The documentation for this class was generated from the following file:

- **GUIOverlay.hh**

10.56 gazebo::rendering::GzTerrainMatGen Class Reference

```
#include <Heightmap.hh>
```

Inheritance diagram for gazebo::rendering::GzTerrainMatGen:



Classes

- class **SM2Profile**
  Shader model 2 profile target.

Public Member Functions

- **GzTerrainMatGen** ()
  Constructor.
- virtual ~**GzTerrainMatGen** ()
  Destructor.

10.56.1 Constructor & Destructor Documentation

10.56.1.1 gazebo::rendering::GzTerrainMatGen::GzTerrainMatGen ( )

Constructor.

10.56.1.2 virtual gazebo::rendering::GzTerrainMatGen::~GzTerrainMatGen ( ) [virtual]

Destructor.

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

- **Heightmap.hh**

10.57 gazebo::rendering::Heightmap Class Reference

Rendering a terrain using heightmap information.

`#include <rendering/rendering.hh>`

Public Member Functions

- **Heightmap** (**ScenePtr** _scene)

  Constructor.
- virtual ∼**Heightmap** ()

  Destructor.
- bool **Flatten** (**CameraPtr** _camera, **math::Vector2i** _mousePos, double _outsideRadius, double _insideRadius, double _weight=0.1)

  Flatten the terrain based on a mouse press.
- double **GetAvgHeight** (Ogre::Vector3 _pos, double _brushSize)

  Get the average height around a point.
- double **GetHeight** (double _x, double _y, double _z=1000)

  Get the height at a location.
- **common::Image GetImage** () const

  Get the heightmap as an image.
- Ogre::TerrainGroup::RayResult **GetMouseHit** (**CameraPtr** _camera, **math::Vector2i** _mousePos)

  Calculate a mouse ray hit on the terrain.
- Ogre::TerrainGroup ∗ **GetOgreTerrain** () const

  Get a pointer to the OGRE terrain group object.
- void **Load** ()

  Load the heightmap.
- void **LoadFromMsg** (ConstVisualPtr &_msg)

  Load the heightmap from a visual message.
- bool **Lower** (**CameraPtr** _camera, **math::Vector2i** _mousePos, double _outsideRadius, double _insideRadius, double _weight=0.1)

  Lower the terrain based on a mouse press.
- bool **Raise** (**CameraPtr** _camera, **math::Vector2i** _mousePos, double _outsideRadius, double _insideRadius, double _weight=0.1)

  Raise the terrain based on a mouse press.
- void **SetWireframe** (bool _show)

  Set the heightmap to render in wireframe mode.
- bool **Smooth** (**CameraPtr** _camera, **math::Vector2i** _mousePos, double _outsideRadius, double _insideRadius, double _weight=0.1)

  Smooth the terrain based on a mouse press.

10.57.1 Detailed Description

Rendering a terrain using heightmap information.

10.57.2 Constructor & Destructor Documentation

10.57.2.1 gazebo::rendering::Heightmap::Heightmap (ScenePtr _scene)

Constructor.

Parameters

in	*_scene*	Pointer to the scene that will contain the heightmap

10.57.2.2 virtual gazebo::rendering::Heightmap::~Heightmap ( ) [virtual]

Destructor.

10.57.3 Member Function Documentation

10.57.3.1 bool gazebo::rendering::Heightmap::Flatten (CameraPtr *_camera*, math::Vector2i *_mousePos*, double *_outsideRadius*, double *_insideRadius*, double *_weight* = 0.1)

Flatten the terrain based on a mouse press.

Parameters

in	*_camera*	**Camera** (p. 166) associated with the mouse press.
in	*_mousePos*	Position of the mouse in viewport coordinates.
in	*_outsideRadius*	Controls the radius of effect.
in	*_insideRadius*	Controls the size of the radius with the maximum effect (value between 0 and 1).
in	*_weight*	Controls modification magnitude.

Returns

True if the terrain was modified

10.57.3.2 double gazebo::rendering::Heightmap::GetAvgHeight (Ogre::Vector3 *_pos*, double *_brushSize*)

Get the average height around a point.

Parameters

in	*_pos*	Position in world coordinates.
in	*_brushSize*	Controls the radius of effect.

10.57.3.3 double gazebo::rendering::Heightmap::GetHeight (double *_x*, double *_y*, double *_z* = 1000)

Get the height at a location.

Parameters

in	*_x*	X location
in	*_y*	Y location
in	*_z*	Z location

Returns

The height at the specified location

10.57.3.4 common::Image gazebo::rendering::Heightmap::GetImage ( ) const

Get the heightmap as an image.

Returns

An image that contains the terrain data.

10.57.3.5 Ogre::TerrainGroup::RayResult gazebo::rendering::Heightmap::GetMouseHit (CameraPtr *_camera,* math::Vector2i *_mousePos*)

Calculate a mouse ray hit on the terrain.

Parameters

in	*_camera*	**Camera** (p. 166) associated with the mouse press.
in	*_mousePos*	Position of the mouse in viewport coordinates.

Returns

The result of the mouse ray hit.

10.57.3.6 Ogre::TerrainGroup* gazebo::rendering::Heightmap::GetOgreTerrain ( ) const

Get a pointer to the OGRE terrain group object.

Returns

Pointer to the OGRE terrain.

10.57.3.7 void gazebo::rendering::Heightmap::Load ( )

Load the heightmap.

10.57.3.8 void gazebo::rendering::Heightmap::LoadFromMsg (ConstVisualPtr & *_msg*)

Load the heightmap from a visual message.

Parameters

in	*_msg*	The visual message containing heightmap info

10.57.3.9 bool gazebo::rendering::Heightmap::Lower (CameraPtr *_camera,* math::Vector2i *_mousePos,* double *_outsideRadius,* double *_insideRadius,* double *_weight* = 0.1)

Lower the terrain based on a mouse press.

Parameters

in	*_camera*	**Camera** (p. 166) associated with the mouse press.
in	*_mousePos*	Position of the mouse in viewport coordinates.
in	*_outsideRadius*	Controls the radius of effect.
in	*_insideRadius*	Controls the size of the radius with the maximum effect (value between 0 and 1).
in	*_weight*	Controls modification magnitude.

Returns

True if the terrain was modified

10.57.3.10 bool gazebo::rendering::Heightmap::Raise (CameraPtr *_camera*, math::Vector2i *_mousePos*, double *_outsideRadius*, double *_insideRadius*, double *_weight* = 0.1)

Raise the terrain based on a mouse press.

Parameters

in	*_camera*	**Camera** (p. 166) associated with the mouse press.
in	*_mousePos*	Position of the mouse in viewport coordinates.
in	*_outsideRadius*	Controls the radius of effect.
in	*_insideRadius*	Controls the size of the radius with the maximum effect (value between 0 and 1).
in	*_weight*	Controls modification magnitude.

Returns

True if the terrain was modified

10.57.3.11 void gazebo::rendering::Heightmap::SetWireframe (bool *_show*)

Set the heightmap to render in wireframe mode.

Parameters

in	*_show*	True to render wireframe, false to render solid.

10.57.3.12 bool gazebo::rendering::Heightmap::Smooth (CameraPtr *_camera*, math::Vector2i *_mousePos*, double *_outsideRadius*, double *_insideRadius*, double *_weight* = 0.1)

Smooth the terrain based on a mouse press.

Parameters

in	*_camera*	**Camera** (p. 166) associated with the mouse press.
in	*_mousePos*	Position of the mouse in viewport coordinates.
in	*_outsideRadius*	Controls the radius of effect.
in	*_insideRadius*	Controls the size of the radius with the maximum effect (value between 0 and 1).
in	*_weight*	Controls modification magnitude.

Returns

True if the terrain was modified

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

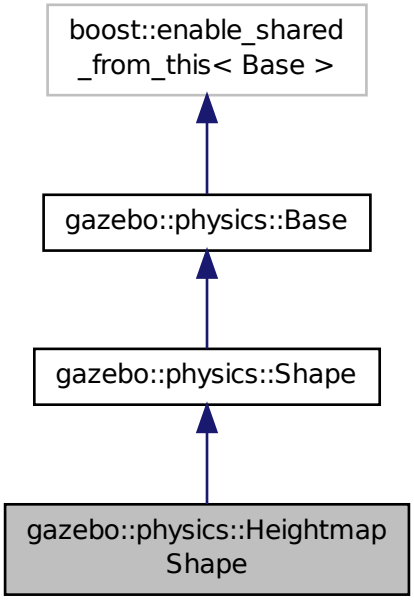
- **Heightmap.hh**

10.58 gazebo::physics::HeightmapShape Class Reference

HeightmapShape (p. 371) collision shape builds a heightmap from an image.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::HeightmapShape:



Public Member Functions

- **HeightmapShape** (**CollisionPtr** _parent)
  Constructor.
- virtual ~**HeightmapShape** ()
  Destructor.
- void **FillMsg** (msgs::Geometry &_msg)
  Fill a geometry message with this shape's data.
- float **GetHeight** (int _x, int _y) const
  Get a height at a position.
- **common::Image GetImage** () const
  Return an image representation of the heightmap.
- float **GetMaxHeight** () const
  Get the maximum height.
- float **GetMinHeight** () const
  Get the minimum height.
- **math::Vector3 GetPos** () const

Get the origin in world coordinate frame.

- **math::Vector3 GetSize** () const

  Get the size in meters.

- int **GetSubSampling** () const

  Get the amount of subsampling.

- std::string **GetURI** () const

  Get the URI of the heightmap image.

- **math::Vector2i GetVertexCount** () const

  Return the number of vertices, which equals the size of the image used to load the heightmap.

- virtual void **Init** ()

  Initialize the heightmap.

- virtual void **Load** (**sdf::ElementPtr** _sdf)

  Load the heightmap.

- virtual void **ProcessMsg** (const msgs::Geometry &_msg)

  Update the heightmap from a message.

Protected Attributes

- bool **flipY**

  True to flip the heights along the y direction.

- std::vector< float > **heights**

  Lookup table of heights.

- **common::Image img**

  Image used to generate the heights.

- **math::Vector3 scale**

  Scaling factor.

- int **subSampling**

  The amount of subsampling. Default is 2.

- unsigned int **vertSize**

  Size of the height lookup table.

Additional Inherited Members

10.58.1 Detailed Description

HeightmapShape (p. 371) collision shape builds a heightmap from an image.

The supplied image must be square with N∗N+1 pixels per side, where N is an integer.

10.58.2 Constructor & Destructor Documentation

10.58.2.1 gazebo::physics::HeightmapShape::HeightmapShape (**CollisionPtr** *_parent*) [explicit]

Constructor.

Parameters

in	*_parent*	Parent **Collision** (p. 199) object.

10.58.2.2 virtual gazebo::physics::HeightmapShape::~HeightmapShape ( ) [virtual]

Destructor.

10.58.3 Member Function Documentation

10.58.3.1 void gazebo::physics::HeightmapShape::FillMsg (msgs::Geometry & _msg) [virtual]

Fill a geometry message with this shape's data.

Parameters

in	*_msg*	Message to fill.

Implements **gazebo::physics::Shape** (p. 756).

10.58.3.2 float gazebo::physics::HeightmapShape::GetHeight (int _x, int _y) const

Get a height at a position.

Parameters

in	*_x*	X position.
in	*_y*	Y position.

Returns

The height at a the specified location.

10.58.3.3 common::Image gazebo::physics::HeightmapShape::GetImage ( ) const

Return an image representation of the heightmap.

Returns

Image where white pixels represents the highest locations, and black pixels the lowest.

10.58.3.4 float gazebo::physics::HeightmapShape::GetMaxHeight ( ) const

Get the maximum height.

Returns

The maximum height.

10.58.3.5 float gazebo::physics::HeightmapShape::GetMinHeight ( ) const

Get the minimum height.

Returns

The minimum height.

10.58.3.6 math::Vector3 gazebo::physics::HeightmapShape::GetPos ( ) const

Get the origin in world coordinate frame.

Returns

The origin in world coordinate frame.

10.58.3.7 math::Vector3 gazebo::physics::HeightmapShape::GetSize ( ) const

Get the size in meters.

Returns

The size in meters.

10.58.3.8 int gazebo::physics::HeightmapShape::GetSubSampling ( ) const

Get the amount of subsampling.

Returns

Amount of subsampling.

10.58.3.9 std::string gazebo::physics::HeightmapShape::GetURI ( ) const

Get the URI of the heightmap image.

Returns

The heightmap image URI.

10.58.3.10 math::Vector2i gazebo::physics::HeightmapShape::GetVertexCount ( ) const

Return the number of vertices, which equals the size of the image used to load the heightmap.

Returns

math::Vector2i (p. 881), result.x = width, result.y = length/height.

10.58.3.11 virtual void gazebo::physics::HeightmapShape::Init ( ) [virtual]

Initialize the heightmap.

Implements **gazebo::physics::Shape** (p. 756).

10.58.3.12 virtual void gazebo::physics::HeightmapShape::Load (sdf::ElementPtr _sdf) [virtual]

Load the heightmap.

Parameters

in	*_sdf*	SDF value to load from.

Reimplemented from **gazebo::physics::Base** (p. 149).

10.58.3.13 virtual void gazebo::physics::HeightmapShape::ProcessMsg (const msgs::Geometry & _msg) [virtual]

Update the heightmap from a message.

Parameters

in	*_msg*	Message to update from.

Implements **gazebo::physics::Shape** (p. 756).

10.58.4 Member Data Documentation

10.58.4.1 bool gazebo::physics::HeightmapShape::flipY [protected]

True to flip the heights along the y direction.

10.58.4.2 std::vector<float> gazebo::physics::HeightmapShape::heights [protected]

Lookup table of heights.

10.58.4.3 common::Image gazebo::physics::HeightmapShape::img [protected]

Image used to generate the heights.

10.58.4.4 math::Vector3 gazebo::physics::HeightmapShape::scale [protected]

Scaling factor.

10.58.4.5 int gazebo::physics::HeightmapShape::subSampling [protected]

The amount of subsampling. Default is 2.

10.58.4.6 unsigned int gazebo::physics::HeightmapShape::vertSize [protected]

Size of the height lookup table.

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

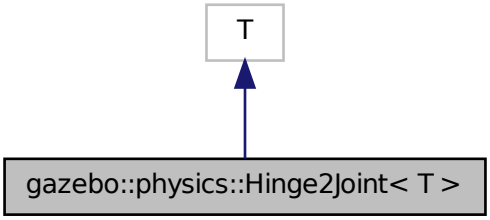
- **HeightmapShape.hh**

10.59 gazebo::physics::Hinge2Joint< T > Class Template Reference

A two axis hinge joint.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::Hinge2Joint< T >:



Public Member Functions

- **Hinge2Joint** (**BasePtr** _parent)
  Constructor.
- virtual ~**Hinge2Joint** ()
  Destructor.
- virtual unsigned int **GetAngleCount** () const
- virtual void **Load** (**sdf::ElementPtr** _sdf)
  Load the joint.

10.59.1 Detailed Description

template<class T>class gazebo::physics::Hinge2Joint< T >

A two axis hinge joint.

10.59.2 Constructor & Destructor Documentation

10.59.2.1 template<class T > gazebo::physics::Hinge2Joint< T >::Hinge2Joint (BasePtr *_parent*) [inline], [explicit]

Constructor.

Parameters

in	*_parent*	Parent link.

References gazebo::physics::Base::HINGE2_JOINT.

10.59.2.2 template<class T > virtual gazebo::physics::Hinge2Joint< T >::~Hinge2Joint ( ) [inline], [virtual]

Destructor.

10.59.3 Member Function Documentation

10.59.3.1 template<class T > virtual unsigned int gazebo::physics::Hinge2Joint< T >::GetAngleCount ( ) const [inline], [virtual]

10.59.3.2 template<class T > virtual void gazebo::physics::Hinge2Joint< T >::Load (sdf::ElementPtr *_sdf*) [inline], [virtual]

Load the joint.

Parameters

in	*_sdf*	SDF values to load from.

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

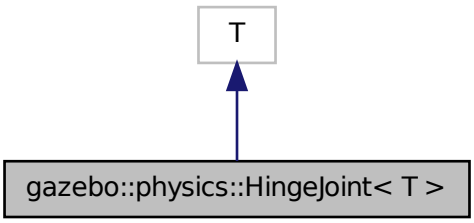
- **Hinge2Joint.hh**

10.60 gazebo::physics::HingeJoint< T > Class Template Reference

A single axis hinge joint.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::HingeJoint< T >:



Public Member Functions

- **HingeJoint** (**BasePtr** _parent)

Constructor.

- virtual ~**HingeJoint** ()

  Destructor.

- virtual unsigned int **GetAngleCount** () const
- virtual void **Load** (**sdf::ElementPtr** _sdf)

  Load joint.

Protected Member Functions

- virtual void **Init** ()

  Initialize joint.

10.60.1 Detailed Description

template<class T>class gazebo::physics::HingeJoint< T >

A single axis hinge joint.

10.60.2 Constructor & Destructor Documentation

10.60.2.1 template<class T > gazebo::physics::HingeJoint< T >::HingeJoint (BasePtr *_parent*) [inline]

Constructor.

Parameters

in	*_parent*	Parent link

References gazebo::physics::Base::HINGE_JOINT.

10.60.2.2 template<class T > virtual gazebo::physics::HingeJoint< T >::~HingeJoint ( ) [inline], [virtual]

Destructor.

10.60.3 Member Function Documentation

10.60.3.1 template<class T > virtual unsigned int gazebo::physics::HingeJoint< T >::GetAngleCount ( ) const [inline], [virtual]

10.60.3.2 template<class T > virtual void gazebo::physics::HingeJoint< T >::Init ( ) [inline], [protected], [virtual]

Initialize joint.

References gazebo::msgs::Init().

10.60.3.3 template<class T > virtual void gazebo::physics::HingeJoint< T >::Load (sdf::ElementPtr _sdf) [inline], [virtual]

Load joint.

Parameters

in	*_sdf*	Pointer to SDF element

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

- **HingeJoint.hh**

10.61 gazebo::common::Image Class Reference

Encapsulates an image.

`#include <common/common.hh>`

Public Types

- enum **PixelFormat** {
  UNKNOWN_PIXEL_FORMAT = 0, **L_INT8**, **L_INT16**, **RGB_INT8**,
  RGBA_INT8, **BGRA_INT8**, **RGB_INT16**, **RGB_INT32**,
  BGR_INT8, **BGR_INT16**, **BGR_INT32**, **R_FLOAT16**,
  RGB_FLOAT16, **R_FLOAT32**, **RGB_FLOAT32**, **BAYER_RGGB8**,
  BAYER_RGGR8, **BAYER_GBRG8**, **BAYER_GRBG8**, **PIXEL_FORMAT_COUNT** }

  Pixel formats enumeration.

Public Member Functions

- **Image** (const std::string &_filename="")

  Constructor.
- virtual ~**Image** ()

  Destructor.
- **Color GetAvgColor** ()

  Get the average color.
- unsigned int **GetBPP** () const

  Get the size of one pixel in bits.
- void **GetData** (unsigned char **_data, unsigned int &_count) const

  Get the image as a data array.
- std::string **GetFilename** () const

  Get the full filename of the image.
- unsigned int **GetHeight** () const

  Get the height.
- **Color GetMaxColor** ()

  Get the max color.
- int **GetPitch** () const
- **Color GetPixel** (unsigned int _x, unsigned int _y)

Get a pixel color value.

- **PixelFormat GetPixelFormat** () const

  Get the pixel format.

- void **GetRGBData** (unsigned char **_data, unsigned int &_count) const

  Get only the RGB data from the image.

- unsigned int **GetWidth** () const

  Get the width.

- int **Load** (const std::string &_filename)

  Load an image.

- void **Rescale** (int _width, int _height)

  Rescale the image.

- void **SavePNG** (const std::string &_filename)

  Save the image in PNG format.

- void **SetFromData** (const unsigned char *_data, unsigned int _width, unsigned int _height, **Image::PixelFormat** _format)

  Set the image from raw data.

- bool **Valid** () const

  Returns whether this is a valid image.

Static Public Member Functions

- static **Image::PixelFormat ConvertPixelFormat** (const std::string &_format)

  *Convert a string to a **Image::PixelFormat** (p. 380).*

10.61.1 Detailed Description

Encapsulates an image.

10.61.2 Member Enumeration Documentation

10.61.2.1 enum gazebo::common::Image::PixelFormat

Pixel formats enumeration.

Enumerator:

UNKNOWN_PIXEL_FORMAT

L_INT8

L_INT16

RGB_INT8

RGBA_INT8

BGRA_INT8

RGB_INT16

RGB_INT32

BGR_INT8

BGR_INT16

BGR_INT32

R_FLOAT16

RGB_FLOAT16

R_FLOAT32

RGB_FLOAT32

BAYER_RGGB8

BAYER_RGGR8

BAYER_GBRG8

BAYER_GRBG8

PIXEL_FORMAT_COUNT

10.61.3 Constructor & Destructor Documentation

10.61.3.1 gazebo::common::Image::Image (const std::string & *_filename* = " ") [explicit]

Constructor.

Parameters

in	*_filename*	the path to the image

10.61.3.2 virtual gazebo::common::Image::~Image ( ) [virtual]

Destructor.

10.61.4 Member Function Documentation

10.61.4.1 static Image::PixelFormat gazebo::common::Image::ConvertPixelFormat (const std::string & *_format*) [static]

Convert a string to a **Image::PixelFormat** (p. 380).

Parameters

in	*_format*	Pixel format string.

See Also

Image::PixelFormatNames

Returns

Image::PixelFormat (p. 380)

10.61.4.2 Color gazebo::common::Image::GetAvgColor ( )

Get the average color.

Returns

The average color

10.61.4.3 unsigned int gazebo::common::Image::GetBPP ( ) const

Get the size of one pixel in bits.

Returns

The BPP of the image

10.61.4.4 void gazebo::common::Image::GetData (unsigned char ** *_data*, unsigned int & *_count*) const

Get the image as a data array.

Parameters

out	*_data*	Pointer to a NULL array of char.
out	*_count*	The resulting data array size

10.61.4.5 std::string gazebo::common::Image::GetFilename ( ) const

Get the full filename of the image.

Returns

The filename used to load the image

10.61.4.6 unsigned int gazebo::common::Image::GetHeight ( ) const

Get the height.

Returns

The image height

10.61.4.7 Color gazebo::common::Image::GetMaxColor ( )

Get the max color.

Returns

The max color

10.61.4.8 int gazebo::common::Image::GetPitch ( ) const

Returns

The pitch of the image

10.61.4.9 Color gazebo::common::Image::GetPixel (unsigned int _x, unsigned int _y)

Get a pixel color value.

Parameters

in	_x	Column location in the image
in	_y	Row location in the image

10.61.4.10 PixelFormat gazebo::common::Image::GetPixelFormat ( ) const

Get the pixel format.

Returns

PixelFormat

10.61.4.11 void gazebo::common::Image::GetRGBData (unsigned char ** _data, unsigned int & _count) const

Get only the RGB data from the image.

This will drop the alpha channel if one is present.

Parameters

out	_data	Pointer to a NULL array of char.
out	_count	The resulting data array size

10.61.4.12 unsigned int gazebo::common::Image::GetWidth ( ) const

Get the width.

Returns

The image width

10.61.4.13 int gazebo::common::Image::Load (const std::string & _filename)

Load an image.

Return 0 on success

Parameters

in	_filename	the path to the image file

10.61.4.14 void gazebo::common::Image::Rescale (int _width, int _height)

Rescale the image.

Parameters

in	*_width*	New image width
in	*_height*	New image height

10.61.4.15 void gazebo::common::Image::SavePNG (const std::string & *_filename*)

Save the image in PNG format.

Parameters

in	*_filename*	The name of the saved image

10.61.4.16 void gazebo::common::Image::SetFromData (const unsigned char * *_data*, unsigned int *_width*, unsigned int *_height*, Image::PixelFormat *_format*)

Set the image from raw data.

Parameters

in	*_data*	Pointer to the raw image data
in	*_width*	Width in pixels
in	*_height*	Height in pixels
in	*_format*	Pixel format of the provided data

10.61.4.17 bool gazebo::common::Image::Valid ( ) const

Returns whether this is a valid image.

Returns

true if image has a bitmap

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

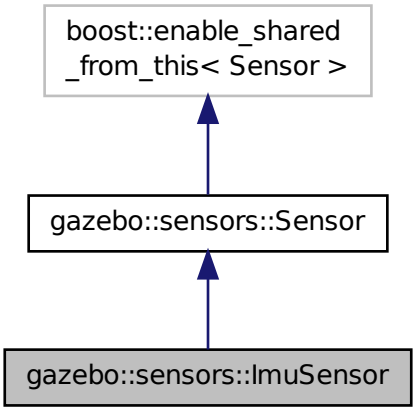
- **Image.hh**

10.62 gazebo::sensors::ImuSensor Class Reference

An IMU sensor.

`#include <sensors/sensors.hh>`

Inheritance diagram for gazebo::sensors::ImuSensor:



Public Member Functions

- **ImuSensor** ()

  Constructor.
- virtual ~**ImuSensor** ()

  Destructor.
- **math::Vector3 GetAngularVelocity** () const

  Returns the angular velocity.
- msgs::IMU **GetImuMessage** () const

  Returns the imu message.
- **math::Vector3 GetLinearAcceleration** () const

  Returns the imu linear acceleration.
- **math::Quaternion GetOrientation** () const

  get orientation of the IMU relative to the reference pose
- virtual void **Init** ()

  Initialize the IMU.
- virtual bool **IsActive** ()

  Returns true if sensor generation is active.
- void **SetReferencePose** ()

  Sets the current pose as the IMU reference pose.

Protected Member Functions

- virtual void **Fini** ()

  Finalize the sensor.
- void **Load** (const std::string &_worldName, **sdf::ElementPtr** _sdf)

Load the sensor with SDF parameters.

- virtual void **Load** (const std::string &_worldName)

  Load the sensor with default parameters.

- virtual void **UpdateImpl** (bool _force)

  This gets overwritten by derived sensor types.

Additional Inherited Members

10.62.1 Detailed Description

An IMU sensor.

10.62.2 Constructor & Destructor Documentation

10.62.2.1 gazebo::sensors::ImuSensor::ImuSensor ( )

Constructor.

10.62.2.2 virtual gazebo::sensors::ImuSensor::~ImuSensor ( ) [virtual]

Destructor.

10.62.3 Member Function Documentation

10.62.3.1 virtual void gazebo::sensors::ImuSensor::Fini ( ) [protected], [virtual]

Finalize the sensor.

Reimplemented from **gazebo::sensors::Sensor** (p. 735).

10.62.3.2 math::Vector3 gazebo::sensors::ImuSensor::GetAngularVelocity ( ) const

Returns the angular velocity.

Returns

Angular velocity.

10.62.3.3 msgs::IMU gazebo::sensors::ImuSensor::GetImuMessage ( ) const

Returns the imu message.

Returns

Imu message.

10.62.3.4 math::Vector3 gazebo::sensors::ImuSensor::GetLinearAcceleration ( ) const

Returns the imu linear acceleration.

Returns

Linear acceleration.

10.62.3.5 math::Quaternion gazebo::sensors::ImuSensor::GetOrientation ( ) const

get orientation of the IMU relative to the reference pose

Returns

returns the orientation quaternion of the IMU relative to the imu reference pose.

10.62.3.6 virtual void gazebo::sensors::ImuSensor::Init ( ) [virtual]

Initialize the IMU.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.62.3.7 virtual bool gazebo::sensors::ImuSensor::IsActive ( ) [virtual]

Returns true if sensor generation is active.

Returns

True if active, false if not.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.62.3.8 void gazebo::sensors::ImuSensor::Load (const std::string & *_worldName*, sdf::ElementPtr *_sdf*) [protected], [virtual]

Load the sensor with SDF parameters.

Parameters

in	*_sdf*	SDF **Sensor** (p. 731) parameters.
in	*_worldName*	Name of world to load from.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.62.3.9 virtual void gazebo::sensors::ImuSensor::Load (const std::string & *_worldName*) [protected], [virtual]

Load the sensor with default parameters.

Parameters

in	*_worldName*	Name of world to load from.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.62.3.10 void gazebo::sensors::ImuSensor::SetReferencePose ( )

Sets the current pose as the IMU reference pose.

10.62.3.11 virtual void gazebo::sensors::ImuSensor::UpdateImpl (bool) [protected], [virtual]

This gets overwritten by derived sensor types.

```
This function is called during Sensor::Update.
And in turn, Sensor::Update is called by
SensorManager::Update
```

Parameters

in	*_force*	True if update is forced, false if not

Reimplemented from **gazebo::sensors::Sensor** (p. 739).

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

- **ImuSensor.hh**

10.63 gazebo::physics::Inertial Class Reference

A class for inertial information about a link.

`#include <physics/physics.hh>`

Public Member Functions

- **Inertial** ()

  Default Constructor.
- **Inertial** (double _mass)

  Constructor.
- **Inertial** (const **Inertial** &_inertial)

  Copy constructor.
- virtual ~**Inertial** ()

  Destructor.
- const **math::Vector3** & **GetCoG** () const

  Get the center of gravity.
- **Inertial GetInertial** (const **math::Pose** &_frameOffset) const

  *Get equivalent Inertia values with the **Link** (p. 439) frame offset, while holding the Pose of CoG constant in the world frame.*
- double **GetIXX** () const

Get IXX.

- double **GetIXY** () const

  Get IXY.

- double **GetIXZ** () const

  Get IXZ.

- double **GetIYY** () const

  Get IYY.

- double **GetIYZ** () const

  Get IXZ.

- double **GetIZZ** () const

  Get IZZ.

- double **GetMass** () const

  Get the mass.

- **math::Matrix3 GetMOI** (const **math::Pose** &_pose) const

  *Get the equivalent inertia from a point in local **Link** (p. 439) frame If you specify GetMOI(this->**GetPose()** (p. 393)), you should get back the Moment of Inertia (MOI) exactly as specified in the SDF.*

- **math::Matrix3 GetMOI** () const

  returns Moments of Inertia as a Matrix3

- const **math::Pose GetPose** () const

  *Get the pose about which the mass and inertia matrix is specified in the **Link** (p. 439) frame.*

- **math::Vector3 GetPrincipalMoments** () const

  Get the principal moments of inertia (Ixx, Iyy, Izz).

- **math::Vector3 GetProductsofInertia** () const

  Get the products of inertia (Ixy, Ixz, Iyz).

- void **Load** (**sdf::ElementPtr** _sdf)

  Load from SDF values.

- **Inertial operator+** (const **Inertial** &_inertial) const

  Addition operator.

- const **Inertial** & **operator+=** (const **Inertial** &_inertial)

  Addition equal operator.

- **Inertial** & **operator=** (const **Inertial** &_inertial)

  Equal operator.

- void **ProcessMsg** (const msgs::Inertial &_msg)

  Update parameters from a message.

- void **Reset** ()

  Reset all the mass properties.

- void **Rotate** (const **math::Quaternion** &_rot)

  Rotate this mass.

- void **SetCoG** (double _cx, double _cy, double _cz)

  Set the center of gravity.

- void **SetCoG** (const **math::Vector3** &_center)

  Set the center of gravity.

- void **SetCoG** (double _cx, double _cy, double _cz, double _rx, double _ry, double _rz)

  *Set the center of gravity and rotation offset of inertial coordinate frame relative to **Link** (p. 439) frame.*

- void **SetCoG** (const **math::Pose** &_c)

  Set the center of gravity.

- void **SetInertiaMatrix** (double _ixx, double _iyy, double _izz, double _ixy, double _ixz, double iyz)

Set the mass matrix.

- void **SetIXX** (double _v)

  Set IXX.
- void **SetIXY** (double _v)

  Set IXY.
- void **SetIXZ** (double _v)

  Set IXZ.
- void **SetIYY** (double _v)

  Set IYY.
- void **SetIYZ** (double _v)

  Set IYZ.
- void **SetIZZ** (double _v)

  Set IZZ.
- void **SetMass** (double m)

  Set the mass.
- void **SetMOI** (const **math::Matrix3** &_moi)

  Sets Moments of Inertia (MOI) from a Matrix3.
- void **UpdateParameters** (**sdf::ElementPtr** _sdf)

  update the parameters using new sdf values.

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::physics::Inertial** &_inertial)

  Output operator.

10.63.1 Detailed Description

A class for inertial information about a link.

10.63.2 Constructor & Destructor Documentation

10.63.2.1 gazebo::physics::Inertial::Inertial ( )

Default Constructor.

10.63.2.2 gazebo::physics::Inertial::Inertial (double *_mass*) [explicit]

Constructor.

Parameters

in	*_mass*	Mass value in kg if using metric.

10.63.2.3 gazebo::physics::Inertial::Inertial (const **Inertial** & *_inertial*)

Copy constructor.

Parameters

in	*_inertial*	**Inertial** (p. 388) element to copy

10.63.2.4 virtual gazebo::physics::Inertial::~Inertial ( ) [virtual]

Destructor.

10.63.3 Member Function Documentation

10.63.3.1 const math::Vector3& gazebo::physics::Inertial::GetCoG ( ) const [inline]

Get the center of gravity.

Returns

The center of gravity.

References gazebo::math::Pose::pos.

10.63.3.2 Inertial gazebo::physics::Inertial::GetInertial (const math::Pose & *_frameOffset*) const

Get equivalent Inertia values with the **Link** (p. 439) frame offset, while holding the Pose of CoG constant in the world frame.

Parameters

in	*_frameOffset*	amount to offset the **Link** (p. 439) frame by, this is a transform defined in the **Link** (p. 439) frame.

Returns

Inertial (p. 388) parameters with the shifted frame.

10.63.3.3 double gazebo::physics::Inertial::GetIXX ( ) const

Get IXX.

Returns

IXX value

10.63.3.4 double gazebo::physics::Inertial::GetIXY ( ) const

Get IXY.

Returns

IXY value

10.63.3.5 double gazebo::physics::Inertial::GetIXZ ( ) const

Get IXZ.

Returns

IXZ value

10.63.3.6 double gazebo::physics::Inertial::GetIYY ( ) const

Get IYY.

Returns

IYY value

10.63.3.7 double gazebo::physics::Inertial::GetIYZ ( ) const

Get IXZ.

Returns

IYZ value

10.63.3.8 double gazebo::physics::Inertial::GetIZZ ( ) const

Get IZZ.

Returns

IZZ value

10.63.3.9 double gazebo::physics::Inertial::GetMass ( ) const

Get the mass.

10.63.3.10 math::Matrix3 gazebo::physics::Inertial::GetMOI (const math::Pose & *_pose*) const

Get the equivalent inertia from a point in local **Link** (p. 439) frame If you specify GetMOI(this->**GetPose()** (p. 393)), you should get back the Moment of Inertia (MOI) exactly as specified in the SDF.

If _pose is different from pose of the **Inertial** (p. 388) block, then the MOI is rotated accordingly, and contributions from changes in MOI location location due to point mass is added to the final MOI.

Parameters

in	*_pose*	location in **Link** (p. 439) local frame

Returns

equivalent inertia at _pose

10.63.3.11 math::Matrix3 gazebo::physics::Inertial::GetMOI ( ) const

returns Moments of Inertia as a Matrix3

Returns

Moments of Inertia as a Matrix3

10.63.3.12 const math::Pose gazebo::physics::Inertial::GetPose ( ) const [inline]

Get the pose about which the mass and inertia matrix is specified in the **Link** (p. 439) frame.

Returns

The inertial pose.

10.63.3.13 math::Vector3 gazebo::physics::Inertial::GetPrincipalMoments ( ) const

Get the principal moments of inertia (Ixx, Iyy, Izz).

Returns

The principal moments.

10.63.3.14 math::Vector3 gazebo::physics::Inertial::GetProductsofInertia ( ) const

Get the products of inertia (Ixy, Ixz, Iyz).

Returns

The products of inertia.

10.63.3.15 void gazebo::physics::Inertial::Load (sdf::ElementPtr *_sdf*)

Load from SDF values.

Parameters

in	*_sdf*	SDF value to load from.

10.63.3.16 Inertial gazebo::physics::Inertial::operator+ (const Inertial & *_inertial*) const

Addition operator.

Assuming both CG and Moment of Inertia (MOI) are defined in the same reference **Link** (p. 439) frame. New CG is computed from masses and perspective offsets, and both MOI contributions relocated to the new cog.

Parameters

in	*_inertial*	**Inertial** (p. 388) to add.

Returns

The result of the addition.

10.63.3.17 const Inertial& gazebo::physics::Inertial::operator+= (const Inertial & *_inertial*)

Addition equal operator.

Parameters

in	*_inertial*	**Inertial** (p. 388) to add.

Returns

Reference to this object.

10.63.3.18 Inertial& gazebo::physics::Inertial::operator= (const Inertial & *_inertial*)

Equal operator.

Parameters

in	*_inertial*	**Inertial** (p. 388) to copy.

Returns

Reference to this object.

10.63.3.19 void gazebo::physics::Inertial::ProcessMsg (const msgs::Inertial & *_msg*)

Update parameters from a message.

Parameters

in	*_msg*	Message to read

10.63.3.20 void gazebo::physics::Inertial::Reset ( )

Reset all the mass properties.

10.63.3.21 void gazebo::physics::Inertial::Rotate (const math::Quaternion & _rot)

Rotate this mass.

Parameters

in	*_rot*	Rotation amount.

10.63.3.22 void gazebo::physics::Inertial::SetCoG (double _cx, double _cy, double _cz)

Set the center of gravity.

Parameters

in	*_cx*	X position.
in	*_cy*	Y position.
in	*_cz*	Z position.

10.63.3.23 void gazebo::physics::Inertial::SetCoG (const math::Vector3 & _center)

Set the center of gravity.

Parameters

in	*_center*	Center of the gravity.

10.63.3.24 void gazebo::physics::Inertial::SetCoG (double _cx, double _cy, double _cz, double _rx, double _ry, double _rz)

Set the center of gravity and rotation offset of inertial coordinate frame relative to **Link** (p. 439) frame.

Parameters

in	*_cx*	Center offset in x-direction in **Link** (p. 439) frame
in	*_cy*	Center offset in y-direction in **Link** (p. 439) frame
in	*_cz*	Center offset in z-direction in **Link** (p. 439) frame
in	*_rx*	Roll angle offset of inertial coordinate frame.
in	*_ry*	Pitch angle offset of inertial coordinate frame.
in	*_rz*	Yaw angle offset of inertial coordinate frame.

10.63.3.25 void gazebo::physics::Inertial::SetCoG (const math::Pose & _c)

Set the center of gravity.

Parameters

in	*_c*	Transform to center of gravity.

10.63.3.26 void gazebo::physics::Inertial::SetInertiaMatrix (double *_ixx*, double *_iyy*, double *_izz*, double *_ixy*, double *_ixz*, double *iyz*)

Set the mass matrix.

Parameters

in	*_ixx*	X second moment of inertia (MOI) about x axis.
in	*_iyy*	Y second moment of inertia about y axis.
in	*_izz*	Z second moment of inertia about z axis.
in	*_ixy*	XY inertia.
in	*_ixz*	XZ inertia.
in	*_iyz*	YZ inertia.

10.63.3.27 void gazebo::physics::Inertial::SetIXX (double *_v*)

Set IXX.

Parameters

in	*_v*	IXX value

10.63.3.28 void gazebo::physics::Inertial::SetIXY (double *_v*)

Set IXY.

Parameters

in	*_v*	IXY value

10.63.3.29 void gazebo::physics::Inertial::SetIXZ (double *_v*)

Set IXZ.

Parameters

in	*_v*	IXZ value

10.63.3.30 void gazebo::physics::Inertial::SetIYY (double *_v*)

Set IYY.

Parameters

in	*_v*	IYY value

10.63.3.31 void gazebo::physics::Inertial::SetIYZ (double *_v*)

Set IYZ.

Parameters

in	*_v*	IXX value

10.63.3.32 void gazebo::physics::Inertial::SetIZZ (double *_v*)

Set IZZ.

Parameters

in	*_v*	IZZ value

10.63.3.33 void gazebo::physics::Inertial::SetMass (double *m*)

Set the mass.

10.63.3.34 void gazebo::physics::Inertial::SetMOI (const **math::Matrix3** & *_moi*)

Sets Moments of Inertia (MOI) from a Matrix3.

Parameters

in	*Moments*	of Inertia as a Matrix3

10.63.3.35 void gazebo::physics::Inertial::UpdateParameters (**sdf::ElementPtr** *_sdf*)

update the parameters using new sdf values.

Parameters

in	*_sdf*	Update values from.

10.63.4 Friends And Related Function Documentation

10.63.4.1 std::ostream& operator<< (std::ostream & *_out,* const **gazebo::physics::Inertial** & *_inertial*) [friend]

Output operator.

Parameters

in	*_out*	Output stream.
in	*_inertial*	**Inertial** (p. 388) object to output.

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

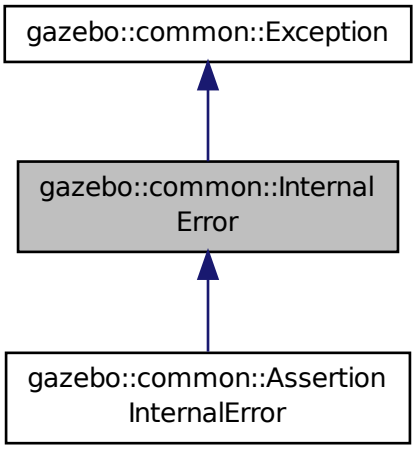
- **Inertial.hh**

10.64 gazebo::common::InternalError Class Reference

Class for generating Internal Gazebo Errors: those errors which should never happend and represent programming bugs.

`#include <common/common.hh>`

Inheritance diagram for gazebo::common::InternalError:



Public Member Functions

- **InternalError** ()

  Constructor.
- **InternalError** (const char ∗_file, int _line, const std::string &_msg)

  Default constructor.
- virtual ∼**InternalError** ()

  Destructor.

10.64.1 Detailed Description

Class for generating Internal Gazebo Errors: those errors which should never happend and represent programming bugs.

10.64.2 Constructor & Destructor Documentation

10.64.2.1 gazebo::common::InternalError::InternalError ( )

Constructor.

10.64.2.2 gazebo::common::InternalError::InternalError (const char * _file, int _line, const std::string & _msg)

Default constructor.

Parameters

in	*_file*	File name
in	*_line*	Line number where the error occurred
in	*_msg*	Error message

10.64.2.3 virtual gazebo::common::InternalError::~InternalError ( ) [virtual]

Destructor.

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

- **Exception.hh**

10.65 gazebo::transport::IOManager Class Reference

Manages boost::asio IO.

`#include <transport/transport.hh>`

Public Member Functions

- **IOManager** ()

  Constructor.
- ~**IOManager** ()

  Destructor.
- void **DecCount** ()

  Decrement the event count by 1.
- unsigned int **GetCount** () const

  Get the event count.
- boost::asio::io_service & **GetIO** ()

  Get handle to boost::asio IO service.
- void **IncCount** ()

  Increment the event count by 1.
- void **Stop** ()

  Stop the IO service.

10.65.1 Detailed Description

Manages boost::asio IO.

10.65.2 Constructor & Destructor Documentation

10.65.2.1 gazebo::transport::IOManager::IOManager ( )

Constructor.

10.65.2.2 gazebo::transport::IOManager::~IOManager ( )

Destructor.

10.65.3 Member Function Documentation

10.65.3.1 void gazebo::transport::IOManager::DecCount ( )

Decrement the event count by 1.

10.65.3.2 unsigned int gazebo::transport::IOManager::GetCount ( ) const

Get the event count.

Returns

The event count

10.65.3.3 boost::asio::io_service& gazebo::transport::IOManager::GetIO ( )

Get handle to boost::asio IO service.

Returns

Handle to boost::asio IO service

10.65.3.4 void gazebo::transport::IOManager::IncCount ( )

Increment the event count by 1.

10.65.3.5 void gazebo::transport::IOManager::Stop ( )

Stop the IO service.

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

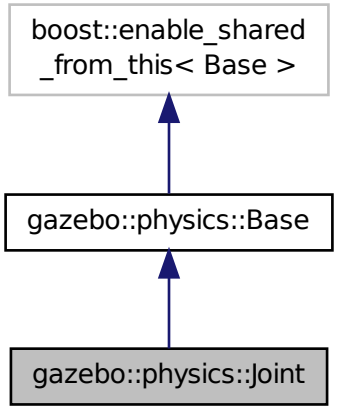
- **IOManager.hh**

10.66 gazebo::physics::Joint Class Reference

Base (p. 141) class for all joints.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::Joint:



Public Types

- enum **Attribute** {
  FUDGE_FACTOR, **SUSPENSION_ERP**, **SUSPENSION_CFM**, **STOP_ERP**,
  STOP_CFM, **ERP**, **CFM**, **FMAX**,
  VEL, **HI_STOP**, **LO_STOP** }

  Joint *(p. 401) attribute types.*

Public Member Functions

- **Joint** (**BasePtr** _parent)

  Constructor.
- virtual ~**Joint** ()

  Destructor.
- virtual void **ApplyDamping** ()

  Callback to apply damping force to joint.
- virtual bool **AreConnected** (**LinkPtr** _one, **LinkPtr** _two) const =0

  Determines of the two bodies are connected by a joint.
- virtual void **Attach** (**LinkPtr** _parent, **LinkPtr** _child)

  Attach the two bodies with this joint.
- template<typename T >
  event::ConnectionPtr ConnectJointUpdate (T _subscriber)

Connect a boost::slot the the joint update signal.

- virtual void **Detach** ()

  Detach this joint from all links.

- void **DisconnectJointUpdate** (**event::ConnectionPtr** &_conn)

  Disconnect a boost::slot the the joint update signal.

- void **FillMsg** (msgs::Joint &_msg)

  Fill a joint message.

- virtual **math::Vector3 GetAnchor** (int _index) const =0

  Get the anchor point.

- **math::Angle GetAngle** (int _index) const

  Get the angle of rotation of an axis(index)

- virtual unsigned int **GetAngleCount** () const =0

  Get the angle count.

- virtual double **GetAttribute** (const std::string &_key, unsigned int _index)=0

  Get a non-generic parameter for the joint.

- **LinkPtr GetChild** () const

  Get the child link.

- double **GetDamping** (int _index)

  Returns the current joint damping coefficient.

- virtual double **GetEffortLimit** (int _index)

  Get the effort limit on axis(index).

- virtual double **GetForce** (int _index) **GAZEBO_DEPRECATED**(1.5)
- virtual double **GetForce** (unsigned int _index)
- virtual **JointWrench GetForceTorque** (int _index) **GAZEBO_DEPRECATED**(1.5)=0

  get internal force and torque values at a joint Note that you must set <provide_feedback>true<provide_feedback> in the joint sdf to use this.

- virtual **JointWrench GetForceTorque** (unsigned int _index)=0

  get internal force and torque values at a joint Note that you must set <provide_feedback>true<provide_feedback> in the joint sdf to use this.

- virtual **math::Vector3 GetGlobalAxis** (int _index) const =0

  Get the axis of rotation in global cooridnate frame.

- virtual **math::Angle GetHighStop** (int _index)=0

  Get the high stop of an axis(index).

- double **GetInertiaRatio** (unsigned int _index) const

  Accessor to inertia ratio across this joint.

- virtual **LinkPtr GetJointLink** (int _index) const =0

  Get the link to which the joint is attached according the _index.

- virtual **math::Vector3 GetLinkForce** (unsigned int _index) const =0

  *Get the forces applied to the center of mass of a **physics::Link*** (p. *439*) *due to the existence of this **Joint*** (p. *401*).

- virtual **math::Vector3 GetLinkTorque** (unsigned int _index) const =0

  *Get the torque applied to the center of mass of a **physics::Link*** (p. *439*) *due to the existence of this **Joint*** (p. *401*).

- **math::Vector3 GetLocalAxis** (int _index) const

  Get the axis of rotation.

- **math::Angle GetLowerLimit** (unsigned int _index) const

  : get the joint upper limit (replaces GetLowStop and GetHighStop)

- virtual **math::Angle GetLowStop** (int _index)=0

  Get the low stop of an axis(index).

- virtual double **GetMaxForce** (int _index)=0

Get the max allowed force of an axis(index).

- **LinkPtr GetParent** () const

  Get the parent link.

- **math::Angle GetUpperLimit** (unsigned int _index) const

  : get the joint lower limit (replacee GetLowStop and GetHighStop)

- virtual double **GetVelocity** (int _index) const =0

  Get the rotation rate of an axis(index)

- virtual double **GetVelocityLimit** (int _index)

  Get the velocity limit on axis(index).

- virtual void **Init** ()

  Initialize a joint.

- void **Load** (**LinkPtr** _parent, **LinkPtr** _child, const **math::Pose** &_pose)

  *Set pose, parent and child links of a **physics::Joint*** (p. *401).*

- void **Load** (**LinkPtr** _parent, **LinkPtr** _child, const **math::Vector3** &_pos) **GAZEBO_DEPRECATED**(1.5)

  *Set parent and child links of a **physics::Joint*** (p. *401) and its anchor offset position.*

- virtual void **Load** (**sdf::ElementPtr** _sdf)

  *Load **physics::Joint*** (p. *401) from a SDF **sdf::Element*** (p. *280).*

- virtual void **Reset** ()

  Reset the joint.

- virtual void **SetAnchor** (int _index, const **math::Vector3** &_anchor)=0

  Set the anchor point.

- void **SetAngle** (int _index, **math::Angle** _angle)

  *If the **Joint*** (p. *401) is static, Gazebo stores the state of this **Joint*** (p. *401) as a scalar inside the **Joint*** (p. *401) class, so this call will NOT move the joint dynamically for a static **Model*** (p. *516).*

- virtual void **SetAttribute** (const std::string &_key, int _index, const boost::any &_value)=0

  Set a non-generic parameter for the joint.

- virtual void **SetAxis** (int _index, const **math::Vector3** &_axis)=0

  Set the axis of rotation where axis is specified in local joint frame.

- virtual void **SetDamping** (int _index, double _damping)=0

  Set the joint damping.

- virtual void **SetForce** (int _index, double _force)

  *Set the force applied to this **physics::Joint*** (p. *401).*

- virtual void **SetHighStop** (int _index, const **math::Angle** &_angle)

  Set the high stop of an axis(index).

- virtual void **SetLowStop** (int _index, const **math::Angle** &_angle)

  Set the low stop of an axis(index).

- virtual void **SetMaxForce** (int _index, double _force)=0

  Set the max allowed force of an axis(index).

- void **SetModel** (**ModelPtr** _model)

  Set the model this joint belongs too.

- virtual void **SetProvideFeedback** (bool _enable)

  Set whether the joint should generate feedback.

- void **SetState** (const **JointState** &_state)

  Set the joint state.

- virtual void **SetVelocity** (int _index, double _vel)=0

  Set the velocity of an axis(index).

- void **Update** ()

  Update the joint.

- virtual void **UpdateParameters** (**sdf::ElementPtr** _sdf)

  Update the parameters using new sdf values.

Protected Member Functions

- virtual **math::Angle GetAngleImpl** (int _index) const =0

  Get the angle of an axis helper function.

Protected Attributes

- **LinkPtr anchorLink**

  Anchor link.
- **math::Vector3 anchorPos**

  Anchor pose.
- **math::Pose anchorPose**

  Anchor pose specified in SDF <joint><pose> tag.
- **gazebo::event::ConnectionPtr applyDamping**

  apply damping for adding viscous damping forces on updates
- **LinkPtr childLink**

  The first link this joint connects to.
- double **dampingCoefficient**

  joint dampingCoefficient
- double **effortLimit** [2]

  *Store **Joint** (p. 401) effort limit as specified in SDF.*
- double **forceApplied** [2]

  Save force applied by user This plus the joint feedback (joint contstraint forces) is the equivalent of simulated force torque sensor reading Allocate a 2 vector in case hinge2 joint is used.
- double **inertiaRatio** [2]

  *Store **Joint** (p. 401) inertia ratio.*
- **math::Angle lowerLimit** [2]

  *Store **Joint** (p. 401) position lower limit as specified in SDF.*
- **ModelPtr model**

  Pointer to the parent model.
- **LinkPtr parentLink**

  The second link this joint connects to.
- bool **provideFeedback**

  Provide Feedback data for contact forces.
- **math::Angle upperLimit** [2]

  *Store **Joint** (p. 401) position upper limit as specified in SDF.*
- bool **useCFMDamping**

  option to use CFM damping
- double **velocityLimit** [2]

  *Store **Joint** (p. 401) velocity limit as specified in SDF.*

10.66.1 Detailed Description

Base (p. 141) class for all joints.

10.66.2 Member Enumeration Documentation

10.66.2.1 enum gazebo::physics::Joint::Attribute

Joint (p. 401) attribute types.

Enumerator:

FUDGE_FACTOR Fudge factor.

SUSPENSION_ERP Suspension error reduction parameter.

SUSPENSION_CFM Suspension constraint force mixing.

STOP_ERP Stop limit error reduction parameter.

STOP_CFM Stop limit constraint force mixing.

ERP Error reduction parameter.

CFM Constraint force mixing.

FMAX Maximum force.

VEL Velocity.

HI_STOP High stop angle.

LO_STOP Low stop angle.

10.66.3 Constructor & Destructor Documentation

10.66.3.1 gazebo::physics::Joint::Joint (BasePtr *_parent*) [explicit]

Constructor.

Parameters

in	***Joint*** (p. *401*)	parent

10.66.3.2 virtual gazebo::physics::Joint::~Joint ( ) [virtual]

Destructor.

10.66.4 Member Function Documentation

10.66.4.1 virtual void gazebo::physics::Joint::ApplyDamping ( ) [virtual]

Callback to apply damping force to joint.

10.66.4.2 virtual bool gazebo::physics::Joint::AreConnected (LinkPtr *_one*, LinkPtr *_two*) const [pure virtual]

Determines of the two bodies are connected by a joint.

Parameters

in	*_one*	First link.
in	*_two*	Second link.

Returns

True if the two links are connected by a joint.

10.66.4.3 virtual void gazebo::physics::Joint::Attach (LinkPtr *_parent*, LinkPtr *_child*) [virtual]

Attach the two bodies with this joint.

Parameters

in	*_parent*	Parent link.
in	*_child*	Child link.

10.66.4.4 template<typename T > event::ConnectionPtr gazebo::physics::Joint::ConnectJointUpdate (T *_subscriber*) [inline]

Connect a boost::slot the the joint update signal.

Parameters

in	*_subscriber*	Callback for the connection.

Returns

Connection pointer, which must be kept in scope.

References gazebo::event::EventT< T >::Connect().

10.66.4.5 virtual void gazebo::physics::Joint::Detach ( ) [virtual]

Detach this joint from all links.

10.66.4.6 void gazebo::physics::Joint::DisconnectJointUpdate (event::ConnectionPtr & *_conn*) [inline]

Disconnect a boost::slot the the joint update signal.

Parameters

in	*_conn*	Connection to disconnect.

References gazebo::event::EventT< T >::Disconnect().

10.66.4.7 void gazebo::physics::Joint::FillMsg (msgs::Joint & *_msg*)

Fill a joint message.

Parameters

out	*_msg*	Message to fill with this joint's properties.

10.66.4.8 virtual math::Vector3 gazebo::physics::Joint::GetAnchor (int _index) const [pure virtual]

Get the anchor point.

Parameters

in	_index	Index of the axis.

Returns

Anchor value for the axis.

10.66.4.9 math::Angle gazebo::physics::Joint::GetAngle (int _index) const

Get the angle of rotation of an axis(index)

Parameters

in	_index	Index of the axis.

Returns

Angle of the axis.

10.66.4.10 virtual unsigned int gazebo::physics::Joint::GetAngleCount ( ) const [pure virtual]

Get the angle count.

Returns

The number of DOF for the joint.

10.66.4.11 virtual math::Angle gazebo::physics::Joint::GetAngleImpl (int _index) const [protected], [pure virtual]

Get the angle of an axis helper function.

Parameters

in	_index	Index of the axis.

Returns

Angle of the axis.

10.66.4.12 virtual double gazebo::physics::Joint::GetAttribute (const std::string & _key, unsigned int _index) [pure virtual]

Get a non-generic parameter for the joint.

Parameters

in	*_key*	String key.
in	*_index*	Index of the axis.
in	*_value*	Value of the attribute.

10.66.4.13 LinkPtr gazebo::physics::Joint::GetChild ( ) const

Get the child link.

Returns

Pointer to the child link.

10.66.4.14 double gazebo::physics::Joint::GetDamping (int *_index*)

Returns the current joint damping coefficient.

Parameters

in	*_index*	Index of the axis to get, currently ignored, to be implemented.

Returns

Joint (p. 401) viscous damping coefficient for this joint.

10.66.4.15 virtual double gazebo::physics::Joint::GetEffortLimit (int *_index*) [virtual]

Get the effort limit on axis(index).

Parameters

in	*_index*	Index of axis, where 0=first axis and 1=second axis

Returns

Effort limit specified in SDF

10.66.4.16 virtual double gazebo::physics::Joint::GetForce (int *_index*) [virtual]

Todo : not yet implemented. Get the forces applied at this **Joint** (p. 401). Note that the unit of force should be consistent with the rest of the simulation scales. E.g. if you are using metric units, the unit for force is Newtons. If using imperial units (sorry), then unit of force is lb-force not (lb-mass), etc.

Parameters

in	*_index*	Index of the axis.

Returns

The force applied to an axis.

10.66.4.17 virtual double gazebo::physics::Joint::GetForce (unsigned int *_index*) [virtual]

Todo : not yet implemented. Get the forces applied at this **Joint** (p. 401). Note that the unit of force should be consistent with the rest of the simulation scales. E.g. if you are using metric units, the unit for force is Newtons. If using imperial units (sorry), then unit of force is lb-force not (lb-mass), etc.

Parameters

in	*_index*	Index of the axis.

Returns

The force applied to an axis.

10.66.4.18 virtual JointWrench gazebo::physics::Joint::GetForceTorque (int *_index*) [pure virtual]

get internal force and torque values at a joint Note that you must set <provide_feedback>true<provide_feedback> in the joint sdf to use this.

Parameters

in	*_index*	Force and torque on child link if _index = 0 and on parent link of _index = 1

Returns

The force and torque at the joint

10.66.4.19 virtual JointWrench gazebo::physics::Joint::GetForceTorque (unsigned int *_index*) [pure virtual]

get internal force and torque values at a joint Note that you must set <provide_feedback>true<provide_feedback> in the joint sdf to use this.

Parameters

in	*_index*	Force and torque on child link if _index = 0 and on parent link of _index = 1

Returns

The force and torque at the joint

10.66.4.20 virtual math::Vector3 gazebo::physics::Joint::GetGlobalAxis (int *_index*) const [pure virtual]

Get the axis of rotation in global cooridnate frame.

Parameters

in	*_index*	Index of the axis to get.

Returns

Axis value for the provided index.

10.66.4.21 virtual math::Angle gazebo::physics::Joint::GetHighStop (int *_index*) [pure virtual]

Get the high stop of an axis(index).

This function is replaced by GetUpperLimit(unsigned int). If you are interested in getting the value of dParamHiStop∗, use GetAttribute(hi_stop, _index)

Parameters

in	*_index*	Index of the axis.

Returns

Angle of the high stop value.

10.66.4.22 double gazebo::physics::Joint::GetInertiaRatio (unsigned int *_index*) const

Accessor to inertia ratio across this joint.

Parameters

in	*_index*	**Joint** (p. 401) axis index.

Returns

returns the inertia ratio across specified joint axis.

10.66.4.23 virtual LinkPtr gazebo::physics::Joint::GetJointLink (int *_index*) const [pure virtual]

Get the link to which the joint is attached according the _index.

Parameters

in	*_index*	Index of the link to retreive.

Returns

Pointer to the request link. NULL if the index was invalid.

10.66.4.24 virtual math::Vector3 gazebo::physics::Joint::GetLinkForce (unsigned int *_index*) const [pure virtual]

Get the forces applied to the center of mass of a **physics::Link** (p. 439) due to the existence of this **Joint** (p. 401).

Note that the unit of force should be consistent with the rest of the simulation scales. E.g. if you are using metric units, the unit for force is Newtons. If using imperial units (sorry), then unit of force is lb-force not (lb-mass), etc.

Parameters

in	*index*	The index of the link(0 or 1).

Returns

Force applied to the link.

10.66.4.25 virtual math::Vector3 gazebo::physics::Joint::GetLinkTorque (unsigned int _index) const [pure virtual]

Get the torque applied to the center of mass of a **physics::Link** (p. 439) due to the existence of this **Joint** (p. 401).

Note that the unit of torque should be consistent with the rest of the simulation scales. E.g. if you are using metric units, the unit for force is Newtons-Meters. If using imperial units (sorry), then unit of force is lb-force-inches not (lb-mass-inches), etc.

Parameters

in	*index*	The index of the link(0 or 1)

Returns

Torque applied to the link.

10.66.4.26 math::Vector3 gazebo::physics::Joint::GetLocalAxis (int _index) const

Get the axis of rotation.

Parameters

in	*_index*	Index of the axis to get.

Returns

Axis value for the provided index.

10.66.4.27 math::Angle gazebo::physics::Joint::GetLowerLimit (unsigned int _index) const

: get the joint upper limit (replaces GetLowStop and GetHighStop)

Parameters

in	*_index*	Index of the axis.

Returns

Upper limit of the axis.

10.66.4.28 virtual math::Angle gazebo::physics::Joint::GetLowStop (int _index) [pure virtual]

Get the low stop of an axis(index).

This function is replaced by GetLowerLimit(unsigned int). If you are interested in getting the value of dParamHiStop∗, use GetAttribute(hi_stop, _index)

Parameters

in	_index	Index of the axis.

Returns

Angle of the low stop value.

10.66.4.29 virtual double gazebo::physics::Joint::GetMaxForce (int _index) [pure virtual]

Get the max allowed force of an axis(index).

Note that the unit of force should be consistent with the rest of the simulation scales. E.g. if you are using metric units, the unit for force is Newtons. If using imperial units (sorry), then unit of force is lb-force not (lb-mass), etc.

Parameters

in	_index	Index of the axis.

Returns

The maximum force.

10.66.4.30 LinkPtr gazebo::physics::Joint::GetParent ( ) const

Get the parent link.

Returns

Pointer to the parent link.

10.66.4.31 math::Angle gazebo::physics::Joint::GetUpperLimit (unsigned int _index) const

: get the joint lower limit (replacee GetLowStop and GetHighStop)

Parameters

in	_index	Index of the axis.

Returns

Upper limit of the axis.

10.66.4.32 virtual double gazebo::physics::Joint::GetVelocity (int _index) const [pure virtual]

Get the rotation rate of an axis(index)

Parameters

in	*_index*	Index of the axis.

Returns

The rotaional velocity of the joint axis.

10.66.4.33 virtual double gazebo::physics::Joint::GetVelocityLimit (int _index) [virtual]

Get the velocity limit on axis(index).

Parameters

in	*_index*	Index of axis, where 0=first axis and 1=second axis

Returns

Velocity limit specified in SDF

10.66.4.34 virtual void gazebo::physics::Joint::Init ( ) [virtual]

Initialize a joint.

Reimplemented from **gazebo::physics::Base** (p. 149).

10.66.4.35 void gazebo::physics::Joint::Load (LinkPtr _parent, LinkPtr _child, const math::Pose & _pose)

Set pose, parent and child links of a **physics::Joint** (p. 401).

Parameters

in	*_parent*	Parent link.
in	*_child*	Child link.
in	*_pose*	Pose containing **Joint** (p. 401) Anchor offset from child link.

10.66.4.36 void gazebo::physics::Joint::Load (LinkPtr _parent, LinkPtr _child, const math::Vector3 & _pos)

Set parent and child links of a **physics::Joint** (p. 401) and its anchor offset position.

This funciton is deprecated, use **Load(LinkPtr _parent, LinkPtr _child, const math::Pose &_pose)** (p. 413)

Parameters

in	*_parent*	Parent link.
in	*_child*	Child link.
in	*_pos*	**Joint** (p. 401) Anchor offset from child link.

10.66.4.37 virtual void gazebo::physics::Joint::Load (sdf::ElementPtr _sdf) [virtual]

Load **physics::Joint** (p. 401) from a SDF **sdf::Element** (p. 280).

Parameters

in	*_sdf*	SDF values to load from.

Reimplemented from **gazebo::physics::Base** (p. 149).

10.66.4.38 virtual void gazebo::physics::Joint::Reset ( ) [virtual]

Reset the joint.

Reimplemented from **gazebo::physics::Base** (p. 151).

10.66.4.39 virtual void gazebo::physics::Joint::SetAnchor (int _index, const math::Vector3 & _anchor) [pure virtual]

Set the anchor point.

Parameters

in	*_index*	Indx of the axis.
in	*_anchor*	Anchor value.

10.66.4.40 void gazebo::physics::Joint::SetAngle (int _index, math::Angle _angle)

If the **Joint** (p. 401) is static, Gazebo stores the state of this **Joint** (p. 401) as a scalar inside the **Joint** (p. 401) class, so this call will NOT move the joint dynamically for a static **Model** (p. 516).

But if this **Model** (p. 516) is not static, then it is updated dynamically, all the conencted children **Link** (p. 439)'s are moved as a result of the **Joint** (p. 401) angle setting. Dynamic **Joint** (p. 401) angle update is accomplished by calling **JointController::SetJointPosition** (p. 420).

Parameters

in	*_index*	Index of the axis.
in	*_angle*	Angle to set the joint to.

10.66.4.41 virtual void gazebo::physics::Joint::SetAttribute (const std::string & _key, int _index, const boost::any & _value) [pure virtual]

Set a non-generic parameter for the joint.

replaces SetAttribute(Attribute, int, double)

Parameters

in	*_key*	String key.
in	*_index*	Index of the axis.
in	*_value*	Value of the attribute.

10.66.4.42 virtual void gazebo::physics::Joint::SetAxis (int *_index*, const math::Vector3 & *_axis*) [pure virtual]

Set the axis of rotation where axis is specified in local joint frame.

Parameters

in	*_index*	Index of the axis to set.
in	*_axis*	Vector in local joint frame of axis direction (must have length greater than zero).

10.66.4.43 virtual void gazebo::physics::Joint::SetDamping (int *_index*, double *_damping*) [pure virtual]

Set the joint damping.

Parameters

in	*_index*	Index of the axis to set, currently ignored, to be implemented.
in	*_damping*	Damping value for the axis.

10.66.4.44 virtual void gazebo::physics::Joint::SetForce (int *_index*, double *_force*) [virtual]

Set the force applied to this **physics::Joint** (p. 401).

Note that the unit of force should be consistent with the rest of the simulation scales. E.g. if you are using metric units, the unit for force is Newtons. If using imperial units (sorry), then unit of force is lb-force not (lb-mass), etc.

Parameters

in	*_index*	Index of the axis.
in	*_force*	Force value.

10.66.4.45 virtual void gazebo::physics::Joint::SetHighStop (int *_index*, const math::Angle & *_angle*) [virtual]

Set the high stop of an axis(index).

Parameters

in	*_index*	Index of the axis.
in	*_angle*	High stop angle.

10.66.4.46 virtual void gazebo::physics::Joint::SetLowStop (int *_index*, const math::Angle & *_angle*) [virtual]

Set the low stop of an axis(index).

Parameters

in	*_index*	Index of the axis.
in	*_angle*	Low stop angle.

10.66.4.47 virtual void gazebo::physics::Joint::SetMaxForce (int *_index*, double *_force*) [pure virtual]

Set the max allowed force of an axis(index).

Note that the unit of force should be consistent with the rest of the simulation scales. E.g. if you are using metric units, the unit for force is Newtons. If using imperial units (sorry), then unit of force is lb-force not (lb-mass), etc.

Parameters

in	*_index*	Index of the axis.
in	*_force*	Maximum force that can be applied to the axis.

10.66.4.48 void gazebo::physics::Joint::SetModel (ModelPtr *_model*)

Set the model this joint belongs too.

Parameters

in	*_model*	Pointer to a model.

10.66.4.49 virtual void gazebo::physics::Joint::SetProvideFeedback (bool *_enable*) [virtual]

Set whether the joint should generate feedback.

Parameters

in	*_enable*	True to enable joint feedback.

10.66.4.50 void gazebo::physics::Joint::SetState (const JointState & *_state*)

Set the joint state.

Parameters

in	*_state*	**Joint** (p. 401) state

10.66.4.51 virtual void gazebo::physics::Joint::SetVelocity (int *_index*, double *_vel*) [pure virtual]

Set the velocity of an axis(index).

Parameters

in	*_index*	Index of the axis.
in	*_vel*	Velocity.

10.66.4.52 void gazebo::physics::Joint::Update ( ) [virtual]

Update the joint.

Reimplemented from **gazebo::physics::Base** (p. 152).

10.66.4.53 virtual void gazebo::physics::Joint::UpdateParameters (sdf::ElementPtr _sdf) [virtual]

Update the parameters using new sdf values.

Parameters

in	*_sdf*	SDF values to update from.

Reimplemented from **gazebo::physics::Base** (p. 152).

10.66.5 Member Data Documentation

10.66.5.1 LinkPtr gazebo::physics::Joint::anchorLink [protected]

Anchor link.

10.66.5.2 math::Vector3 gazebo::physics::Joint::anchorPos [protected]

Anchor pose.

This is the xyz offset of the joint frame from child frame specified in the parent link frame

10.66.5.3 math::Pose gazebo::physics::Joint::anchorPose [protected]

Anchor pose specified in SDF <joint><pose> tag.

AnchorPose is the transform from child link frame to joint frame specified in the child link frame. AnchorPos is more relevant in normal usage, but sometimes, we do need this (e.g. GetForceTorque and joint visualization).

10.66.5.4 gazebo::event::ConnectionPtr gazebo::physics::Joint::applyDamping [protected]

apply damping for adding viscous damping forces on updates

10.66.5.5 LinkPtr gazebo::physics::Joint::childLink [protected]

The first link this joint connects to.

10.66.5.6 double gazebo::physics::Joint::dampingCoefficient [protected]

joint dampingCoefficient

10.66.5.7 double gazebo::physics::Joint::effortLimit[2] [protected]

Store **Joint** (p. 401) effort limit as specified in SDF.

10.66.5.8 double gazebo::physics::Joint::forceApplied[2] [protected]

Save force applied by user This plus the joint feedback (joint contstraint forces) is the equivalent of simulated force torque sensor reading Allocate a 2 vector in case hinge2 joint is used.

10.66.5.9 double gazebo::physics::Joint::inertiaRatio[2] [protected]

Store **Joint** (p. 401) inertia ratio.

This is a measure of how well this model behaves using interative LCP solvers.

10.66.5.10 math::Angle gazebo::physics::Joint::lowerLimit[2] [protected]

Store **Joint** (p. 401) position lower limit as specified in SDF.

10.66.5.11 ModelPtr gazebo::physics::Joint::model [protected]

Pointer to the parent model.

10.66.5.12 LinkPtr gazebo::physics::Joint::parentLink [protected]

The second link this joint connects to.

10.66.5.13 bool gazebo::physics::Joint::provideFeedback [protected]

Provide Feedback data for contact forces.

10.66.5.14 math::Angle gazebo::physics::Joint::upperLimit[2] [protected]

Store **Joint** (p. 401) position upper limit as specified in SDF.

10.66.5.15 bool gazebo::physics::Joint::useCFMDamping [protected]

option to use CFM damping

10.66.5.16 double gazebo::physics::Joint::velocityLimit[2] [protected]

Store **Joint** (p. 401) velocity limit as specified in SDF.

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

- **Joint.hh**

10.67 gazebo::physics::JointController Class Reference

A class for manipulating **physics::Joint** (p. 401).

`#include <physics/physics.hh>`

Public Member Functions

- **JointController** (**ModelPtr** _model)

  Constructor.
- void **AddJoint** (**JointPtr** _joint)

  Add a joint to control.
- void **Reset** ()

  Reset all commands.
- void **SetJointPosition** (const std::string &_name, double _position, int _index=0)

  *Set the positions of a **Joint** (p. 401) by name.*
- void **SetJointPosition** (**JointPtr** _joint, double _position, int _index=0)

  *Set the positions of a **Joint** (p. 401) by name The position is specified in native units, which means, if you are using metric system, it's meters for **SliderJoint** (p. 780) and radians for **HingeJoint** (p. 377), etc.*
- void **SetJointPositions** (const std::map< std::string, double > &_jointPositions)

  *Set the positions of a set of **Joint** (p. 401)'s.*
- void **Update** ()

  Update the joint control.

10.67.1 Detailed Description

A class for manipulating **physics::Joint** (p. 401).

10.67.2 Constructor & Destructor Documentation

10.67.2.1 gazebo::physics::JointController::JointController (**ModelPtr** *_model*) [explicit]

Constructor.

Parameters

in	*_model*	**Model** (p. 516) that uses this joint controller.

10.67.3 Member Function Documentation

10.67.3.1 void gazebo::physics::JointController::AddJoint (**JointPtr** *_joint*)

Add a joint to control.

Parameters

in	*_joint*	**Joint** (p. 401) to control.

10.67.3.2 void gazebo::physics::JointController::Reset ( )

Reset all commands.

10.67.3.3 void gazebo::physics::JointController::SetJointPosition (const std::string & _name, double _position, int _index = 0)

Set the positions of a **Joint** (p. 401) by name.

See Also

JointController::SetJointPosition(JointPtr, double)

10.67.3.4 void gazebo::physics::JointController::SetJointPosition (JointPtr _joint, double _position, int _index = 0)

Set the positions of a **Joint** (p. 401) by name The position is specified in native units, which means, if you are using metric system, it's meters for **SliderJoint** (p. 780) and radians for **HingeJoint** (p. 377), etc.

Implementation: In order to change the position of a **Joint** (p. 401) inside a **Model** (p. 516), this call must recursively crawl through all the connected children **Link** (p. 439)'s in this **Model** (p. 516), and update each **Link** (p. 439) Pose affected by this **Joint** (p. 401) angle update. Warning: There is no constraint satisfaction being done here, traversal through the kinematic graph has unexpected behavior if you try to set the joint position of a link inside a loop structure.

Parameters

in	*_joint*	**Joint** (p. 401) to set.
in	*_position*	Position of the joint.

10.67.3.5 void gazebo::physics::JointController::SetJointPositions (const std::map< std::string, double > & _jointPositions)

Set the positions of a set of **Joint** (p. 401)'s.

See Also

JointController::SetJointPosition(JointPtr, double)

10.67.3.6 void gazebo::physics::JointController::Update ( )

Update the joint control.

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

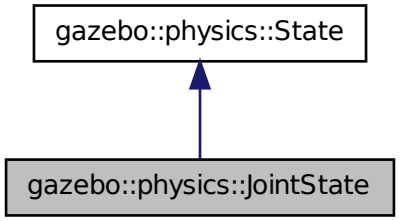
- **JointController.hh**

10.68 gazebo::physics::JointState Class Reference

keeps track of state of a **physics::Joint** (p. 401)

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::JointState:



Public Member Functions

- **JointState** ()
  Default constructor.
- **JointState** (**JointPtr** _joint, const **common::Time** &_realTime, const **common::Time** &_simTime)
  Constructor.
- **JointState** (**JointPtr** _joint)
  Constructor.
- **JointState** (const **sdf::ElementPtr** _sdf)
  Constructor.
- virtual ∼**JointState** ()
  Destructor.
- void **FillSDF** (**sdf::ElementPtr** _sdf)
  Populate a state SDF element with data from the object.
- **math::Angle GetAngle** (unsigned int _axis) const
  Get the joint angle.
- unsigned int **GetAngleCount** () const
  Get the number of angles.
- const std::vector< **math::Angle** > & **GetAngles** () const
  Get the angles.
- bool **IsZero** () const
  Return true if the values in the state are zero.
- void **Load** (**JointPtr** _joint, const **common::Time** &_realTime, const **common::Time** &_simTime)
  Load.
- virtual void **Load** (const **sdf::ElementPtr** _elem)
  Load state from SDF element.
- **JointState operator+** (const **JointState** &_state) const
  Addition operator.
- **JointState operator-** (const **JointState** &_state) const
  Subtraction operator.
- **JointState** & **operator=** (const **JointState** &_state)
  Assignment operator.

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::physics::JointState** &_state)
  Stream insertion operator.

Additional Inherited Members

10.68.1 Detailed Description

keeps track of state of a **physics::Joint** (p. 401)

10.68.2 Constructor & Destructor Documentation

10.68.2.1 gazebo::physics::JointState::JointState ( )

Default constructor.

10.68.2.2 gazebo::physics::JointState::JointState (JointPtr *_joint,* const common::Time & *_realTime,* const common::Time & *_simTime*)

Constructor.

Parameters

in	*_joint*	**Joint** (p. 401) to get the state of.
in	*_realTime*	Real time stamp.
in	*_simTime*	Sim time stamp.

10.68.2.3 gazebo::physics::JointState::JointState (JointPtr *_joint*) [explicit]

Constructor.

Parameters

in	*_joint*	**Joint** (p. 401) to get the state of.

10.68.2.4 gazebo::physics::JointState::JointState (const sdf::ElementPtr *_sdf*) [explicit]

Constructor.

Build a **JointState** (p. 420) from SDF data

Parameters

in	*_sdf*	SDF data to load a joint state from.

10.68.2.5 virtual gazebo::physics::JointState::~JointState ( ) [virtual]

Destructor.

10.68.3 Member Function Documentation

10.68.3.1 void gazebo::physics::JointState::FillSDF (sdf::ElementPtr _sdf)

Populate a state SDF element with data from the object.

Parameters

out	*_sdf*	SDF element to populate.

10.68.3.2 math::Angle gazebo::physics::JointState::GetAngle (unsigned int _axis) const

Get the joint angle.

Parameters

in	*_axis*	The axis index.

Returns

Angle of the axis.

Exceptions

common::Exception *(p. 325)*	When _axis is invalid.

10.68.3.3 unsigned int gazebo::physics::JointState::GetAngleCount ( ) const

Get the number of angles.

Returns

The number of angles.

10.68.3.4 const std::vector<math::Angle>& gazebo::physics::JointState::GetAngles ( ) const

Get the angles.

Returns

Vector of angles.

10.68.3.5 bool gazebo::physics::JointState::IsZero ( ) const

Return true if the values in the state are zero.

Returns

True if the values in the state are zero.

10.68.3.6 void gazebo::physics::JointState::Load (JointPtr *_joint,* const common::Time & *_realTime,* const common::Time & *_simTime*)

Load.

Parameters

in	*_joint*	**Joint** (p. 401) to get the state of.
in	*_realTime*	Real time stamp.
in	*_simTime*	Sim time stamp.

10.68.3.7 virtual void gazebo::physics::JointState::Load (const sdf::ElementPtr *_elem*) [virtual]

Load state from SDF element.

Parameters

in	*_elem*	SDf values to load from.

Reimplemented from **gazebo::physics::State** (p. 800).

10.68.3.8 JointState gazebo::physics::JointState::operator+ (const JointState & *_state*) const

Addition operator.

Parameters

in	*_pt*	A state to add.

Returns

The resulting state.

10.68.3.9 JointState gazebo::physics::JointState::operator- (const JointState & *_state*) const

Subtraction operator.

Parameters

in	*_pt*	A state to substract.

Returns

The resulting state.

10.68.3.10 JointState& gazebo::physics::JointState::operator= (const JointState & *_state*)

Assignment operator.

Parameters

in	*_state*	**State** (p. 797) value

Returns

this

10.68.4 Friends And Related Function Documentation

10.68.4.1 std::ostream& operator<< (std::ostream & *_out,* const gazebo::physics::JointState & *_state*) [friend]

Stream insertion operator.

Parameters

in	*_out*	output stream.
in	*_state*	**Joint** (p. 401) state to output.

Returns

The stream.

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

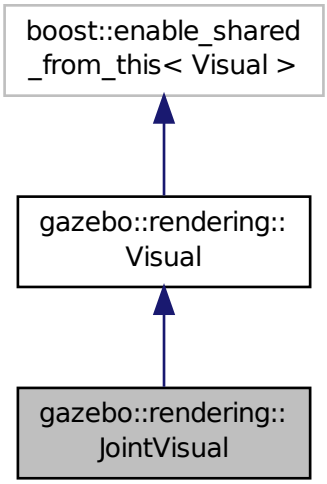
- **JointState.hh**

10.69 gazebo::rendering::JointVisual Class Reference

Visualization for joints.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::JointVisual:



Public Member Functions

- **JointVisual** (const std::string &_name, **VisualPtr** _vis)

  Constructor.
- virtual ~**JointVisual** ()

  Destructor.
- void **Load** (ConstJointPtr &_msg)

  Load the visual based on a message.

Additional Inherited Members

10.69.1 Detailed Description

Visualization for joints.

10.69.2 Constructor & Destructor Documentation

10.69.2.1 gazebo::rendering::JointVisual::JointVisual (const std::string & *_name*, VisualPtr *_vis*)

Constructor.

Parameters

in	*_name*	Name of the visual
in	*_vis*	Pointer to the parent visual

10.69.2.2 virtual gazebo::rendering::JointVisual::~JointVisual ( ) [virtual]

Destructor.

10.69.3 Member Function Documentation

10.69.3.1 void gazebo::rendering::JointVisual::Load (ConstJointPtr & *_msg*)

Load the visual based on a message.

Parameters

in	*_msg*	Joint message

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

- **JointVisual.hh**

10.70 gazebo::physics::JointWrench Class Reference

Wrench information from a joint.

`#include <physics/physics.hh>`

Public Member Functions

- **JointWrench** & **operator+** (const **JointWrench** &_wrench)
  Operator +.
- **JointWrench** & **operator-** (const **JointWrench** &_wrench)
  Operator -.
- **JointWrench** & **operator=** (const **JointWrench** &_wrench)
  Operator =.

Public Attributes

- **math::Vector3 body1Force**
  Force on the first link.
- **math::Vector3 body1Torque**
  Torque on the first link.
- **math::Vector3 body2Force**
  Force on the second link.
- **math::Vector3 body2Torque**
  Torque on the second link.

10.70.1 Detailed Description

Wrench information from a joint.

These are forces and torques on parent and child Links, relative to the **Joint** (p. 401) frame immediately after rotation.

10.70.2 Member Function Documentation

10.70.2.1 JointWrench& gazebo::physics::JointWrench::operator+ (const JointWrench & _wrench) [inline]

Operator +.

Parameters

in	*_wrench*	**Joint** (p. 401) wrench to add

Returns

∗this

References body1Force, body1Torque, body2Force, and body2Torque.

10.70.2.2 JointWrench& gazebo::physics::JointWrench::operator- (const JointWrench & _wrench) [inline]

Operator -.

Parameters

in	*_wrench*	**Joint** (p. 401) wrench to subtract

Returns

∗this

References body1Force, body1Torque, body2Force, and body2Torque.

10.70.2.3 JointWrench& gazebo::physics::JointWrench::operator= (const JointWrench & _wrench) [inline]

Operator =.

Parameters

in	*_wrench*	**Joint** (p. 401) wrench to set from.

Returns

∗this

References body1Force, body1Torque, body2Force, and body2Torque.

10.70.3 Member Data Documentation

10.70.3.1 math::Vector3 gazebo::physics::JointWrench::body1Force

Force on the first link.

Referenced by operator+(), operator-(), and operator=().

10.70.3.2 math::Vector3 gazebo::physics::JointWrench::body1Torque

Torque on the first link.

Referenced by operator+(), operator-(), and operator=().

10.70.3.3 math::Vector3 gazebo::physics::JointWrench::body2Force

Force on the second link.

Referenced by operator+(), operator-(), and operator=().

10.70.3.4 math::Vector3 gazebo::physics::JointWrench::body2Torque

Torque on the second link.

Referenced by operator+(), operator-(), and operator=().

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

- **JointWrench.hh**

10.71 gazebo::common::KeyEvent Class Reference

Generic description of a keyboard event.

`#include <common/common.hh>`

Public Types

- enum **EventType** { **NO_EVENT**, **PRESS**, **RELEASE** }

  Key event types enumeration.

Public Member Functions

- **KeyEvent** ()

  Constructor.

Public Attributes

- int **key**
- **EventType type**

  Event type.

10.71.1 Detailed Description

Generic description of a keyboard event.

10.71.2 Member Enumeration Documentation

10.71.2.1 enum gazebo::common::KeyEvent::EventType

Key event types enumeration.

Enumerator:

NO_EVENT

PRESS

RELEASE

10.71.3 Constructor & Destructor Documentation

10.71.3.1 gazebo::common::KeyEvent::KeyEvent ( ) [inline]

Constructor.

10.71.4 Member Data Documentation

10.71.4.1 int gazebo::common::KeyEvent::key

10.71.4.2 EventType gazebo::common::KeyEvent::type

Event type.

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

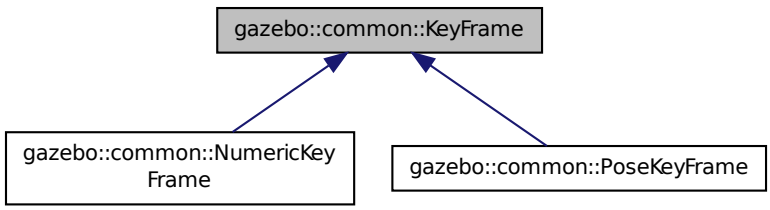
- **KeyEvent.hh**

10.72 gazebo::common::KeyFrame Class Reference

A key frame in an animation.

`#include <common/common.hh>`

Inheritance diagram for gazebo::common::KeyFrame:



Public Member Functions

- **KeyFrame** (double _time)

  Constructor.
- virtual ~**KeyFrame** ()

  Destructor.
- double **GetTime** () const

  Get the time of the keyframe.

Protected Attributes

- double **time**

  time of key frame

10.72.1 Detailed Description

A key frame in an animation.

10.72.2 Constructor & Destructor Documentation

10.72.2.1 gazebo::common::KeyFrame::KeyFrame (double *_time*)

Constructor.

Parameters

in	*_time*	**Time** (p. 831) of the keyframe in seconds

10.72.2.2 virtual gazebo::common::KeyFrame::~KeyFrame ( ) [virtual]

Destructor.

10.72.3 Member Function Documentation

10.72.3.1 double gazebo::common::KeyFrame::GetTime ( ) const

Get the time of the keyframe.

Returns

the time

10.72.4 Member Data Documentation

10.72.4.1 double gazebo::common::KeyFrame::time [protected]

time of key frame

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

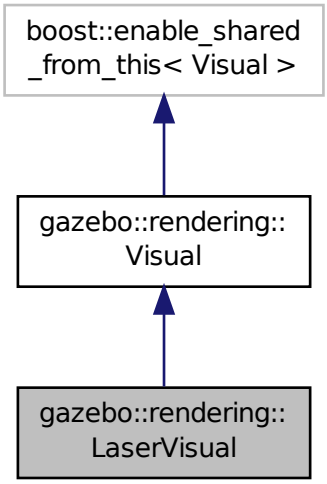
- **KeyFrame.hh**

10.73 gazebo::rendering::LaserVisual Class Reference

Visualization for laser data.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::LaserVisual:



Public Member Functions

- **LaserVisual** (const std::string &_name, **VisualPtr** _vis, const std::string &_topicName)

  Constructor.
- virtual ~**LaserVisual** ()

  Destructor.
- virtual void **SetEmissive** (const **common::Color** &_color)

  Documentation inherited from parent.

Additional Inherited Members

10.73.1 Detailed Description

Visualization for laser data.

10.73.2 Constructor & Destructor Documentation

10.73.2.1 gazebo::rendering::LaserVisual::LaserVisual (const std::string & *_name*, VisualPtr *_vis*, const std::string & *_topicName*)

Constructor.

Parameters

in	*_name*	Name of the visual.
in	*_vis*	Pointer to the parent **Visual** (p. 920).
in	*_topicName*	Name of the topic that has laser data.

10.73.2.2 virtual gazebo::rendering::LaserVisual::~LaserVisual ( ) [virtual]

Destructor.

10.73.3 Member Function Documentation

10.73.3.1 virtual void gazebo::rendering::LaserVisual::SetEmissive (const common::Color & *_color*) [virtual]

Documentation inherited from parent.

Reimplemented from **gazebo::rendering::Visual** (p. 934).

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

- **LaserVisual.hh**

10.74 gazebo::rendering::Light Class Reference

A light source.

`#include <rendering/rendering.hh>`

Public Member Functions

- **Light** (**ScenePtr** _scene)
  Constructor.
- virtual ~**Light** ()
  Destructor.
- void **FillMsg** (msgs::Light &_msg) const
  Fill the contents of a light message.
- **common::Color GetDiffuseColor** () const
  Get the diffuse color.
- **math::Vector3 GetDirection** () const
  Get the direction.
- std::string **GetName** () const
  Get the name of the visual.

- **math::Vector3 GetPosition** () const

  Get the position of the light.
- **common::Color GetSpecularColor** () const

  Get the specular color.
- std::string **GetType** () const

  Get the type of the light.
- void **Load** (**sdf::ElementPtr** _sdf)

  Load the light using a set of SDF parameters.
- void **Load** ()

  Load the light using default parameters.
- void **LoadFromMsg** (ConstLightPtr &_msg)

  Load from a light message.
- void **SetAttenuation** (double _constant, double _linear, double _quadratic)

  Set the attenuation.
- void **SetCastShadows** (const bool &_cast)

  Set cast shadows.
- void **SetDiffuseColor** (const **common::Color** &_color)

  Set the diffuse color.
- void **SetDirection** (const **math::Vector3** &_dir)

  Set the direction.
- void **SetLightType** (const std::string &_type)

  Set the light type.
- void **SetName** (const std::string &_name)

  Set the name of the visual.
- void **SetPosition** (const **math::Vector3** &_p)

  Set the position of the light.
- void **SetRange** (const double &_range)

  Set the range.
- virtual bool **SetSelected** (bool _s)

  Set whether this entity has been selected by the user through the gui.
- void **SetSpecularColor** (const **common::Color** &_color)

  Set the specular color.
- void **SetSpotFalloff** (const double &_value)

  Set the spot light falloff.
- void **SetSpotInnerAngle** (const double &_angle)

  Set the spot light inner angle.
- void **SetSpotOuterAngle** (const double &_angle)

  Set the spot light outer angle.
- void **ShowVisual** (bool _s)

  Set whether to show the visual.
- void **ToggleShowVisual** ()
- void **UpdateFromMsg** (ConstLightPtr &_msg)

  Update a light source from a message.

Protected Member Functions

- virtual void **OnPoseChange** ()

  On pose change callback.

10.74.1 Detailed Description

A light source.

There are three types of lights: Point, Spot, and Directional. This class encapsulates all three. Point lights are light light bulbs, spot lights project a cone of light, and directional lights are light sun light.

10.74.2 Constructor & Destructor Documentation

10.74.2.1 gazebo::rendering::Light::Light (ScenePtr _scene)

Constructor.

Parameters

in	*_scene*	Pointer to the scene that contains the **Light** (p. 433).

10.74.2.2 virtual gazebo::rendering::Light::~Light ( ) [virtual]

Destructor.

10.74.3 Member Function Documentation

10.74.3.1 void gazebo::rendering::Light::FillMsg (msgs::Light & _msg) const

Fill the contents of a light message.

Parameters

out	*_msg*	Message to fill.

10.74.3.2 common::Color gazebo::rendering::Light::GetDiffuseColor ( ) const

Get the diffuse color.

Returns

The light's diffuse color.

10.74.3.3 math::Vector3 gazebo::rendering::Light::GetDirection ( ) const

Get the direction.

Returns

The light's direction.

10.74.3.4 std::string gazebo::rendering::Light::GetName ( ) const

Get the name of the visual.

Returns

The light's name.

10.74.3.5 math::Vector3 gazebo::rendering::Light::GetPosition ( ) const

Get the position of the light.

Returns

The position of the light

10.74.3.6 common::Color gazebo::rendering::Light::GetSpecularColor ( ) const

Get the specular color.

Returns

The specular color

10.74.3.7 std::string gazebo::rendering::Light::GetType ( ) const

Get the type of the light.

Returns

The light type: "point", "spot", "directional".

10.74.3.8 void gazebo::rendering::Light::Load (sdf::ElementPtr *_sdf*)

Load the light using a set of SDF parameters.

Parameters

in	*_sdf*	Pointer to the SDF containing the **Light** (p. 433) description.

10.74.3.9 void gazebo::rendering::Light::Load ( )

Load the light using default parameters.

10.74.3.10 void gazebo::rendering::Light::LoadFromMsg (ConstLightPtr & *_msg*)

Load from a light message.

Parameters

in	*_msg*	Containing the light information.

10.74.3.11 virtual void gazebo::rendering::Light::OnPoseChange ( ) [inline], [protected], [virtual]

On pose change callback.

10.74.3.12 void gazebo::rendering::Light::SetAttenuation (double *_constant,* double *_linear,* double *_quadratic*)

Set the attenuation.

Parameters

in	*_contant*	Constant attenuation
in	*_linear*	Linear attenuation
in	*_quadratic*	Quadratic attenuation

10.74.3.13 void gazebo::rendering::Light::SetCastShadows (const bool & *_cast*)

Set cast shadows.

Parameters

in	*_cast*	Set to true to cast shadows.

10.74.3.14 void gazebo::rendering::Light::SetDiffuseColor (const **common::Color** & *_color*)

Set the diffuse color.

Parameters

in	*_color*	**Light** (p. 433) diffuse color.

10.74.3.15 void gazebo::rendering::Light::SetDirection (const **math::Vector3** & *_dir*)

Set the direction.

Parameters

in	*_dir*	Set the light's direction. Only applicable to spot and directional lights.

10.74.3.16 void gazebo::rendering::Light::SetLightType (const std::string & *_type*)

Set the light type.

Parameters

in	*_type*	The light type: "point", "spot", "directional"

10.74.3.17 void gazebo::rendering::Light::SetName (const std::string & _name)

Set the name of the visual.

Parameters

in	*_name*	Name of the light source.

10.74.3.18 void gazebo::rendering::Light::SetPosition (const math::Vector3 & _p)

Set the position of the light.

Parameters

in	*_p*	New position for the light

10.74.3.19 void gazebo::rendering::Light::SetRange (const double & _range)

Set the range.

Parameters

in	*_range*	Rage of the light in meters.

10.74.3.20 virtual bool gazebo::rendering::Light::SetSelected (bool _s) [virtual]

Set whether this entity has been selected by the user through the gui.

Parameters

in	*_s*	Set to True when the light is selected by the user.

10.74.3.21 void gazebo::rendering::Light::SetSpecularColor (const common::Color & _color)

Set the specular color.

Parameters

in	*_color*	The specular color

10.74.3.22 void gazebo::rendering::Light::SetSpotFalloff (const double & _value)

Set the spot light falloff.

Parameters

in	*_value*	Falloff value

10.74.3.23 void gazebo::rendering::Light::SetSpotInnerAngle (const double & _angle)

Set the spot light inner angle.

Parameters

in	*_angle*	Inner angle in radians

10.74.3.24 void gazebo::rendering::Light::SetSpotOuterAngle (const double & _angle)

Set the spot light outer angle.

Parameters

in	*_angle*	Outer angle in radians

10.74.3.25 void gazebo::rendering::Light::ShowVisual (bool _s)

Set whether to show the visual.

Parameters

in	*_s*	Set to true to draw a representation of the light.

10.74.3.26 void gazebo::rendering::Light::ToggleShowVisual ( )

10.74.3.27 void gazebo::rendering::Light::UpdateFromMsg (ConstLightPtr & _msg)

Update a light source from a message.

Parameters

in	*_msg*	**Light** (p. 433) message to update from

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

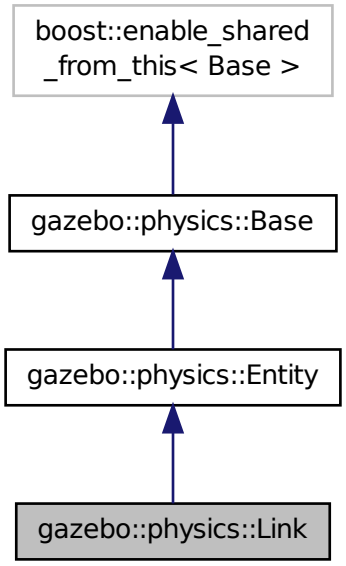
- **Light.hh**

10.75 gazebo::physics::Link Class Reference

Link (p. 439) class defines a rigid body entity, containing information on inertia, visual and collision properties of a rigid body.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::Link:



Public Member Functions

- **Link** (**EntityPtr** _parent)

  Constructor.
- virtual ~**Link** ()

  Destructor.
- void **AddChildJoint** (**JointPtr** _joint)

  *Joints that have this **Link** (p. 439) as a parent **Link** (p. 439).*
- virtual void **AddForce** (const **math::Vector3** &_force)=0

  Add a force to the body.
- virtual void **AddForceAtRelativePosition** (const **math::Vector3** &_force, const **math::Vector3** &_relPos)=0

  Add a force to the body at position expressed to the body's own frame of reference.
- virtual void **AddForceAtWorldPosition** (const **math::Vector3** &_force, const **math::Vector3** &_pos)=0

  Add a force to the body using a global position.
- void **AddParentJoint** (**JointPtr** _joint)

  *Joints that have this **Link** (p. 439) as a child **Link** (p. 439).*
- virtual void **AddRelativeForce** (const **math::Vector3** &_force)=0

  Add a force to the body, components are relative to the body's own frame of reference.
- virtual void **AddRelativeTorque** (const **math::Vector3** &_torque)=0

  Add a torque to the body, components are relative to the body's own frame of reference.
- virtual void **AddTorque** (const **math::Vector3** &_torque)=0

  Add a torque to the body.

- void **AttachStaticModel** (**ModelPtr** &_model, const **math::Pose** &_offset)

  Attach a static model to this link.
- template<typename T >
  event::ConnectionPtr ConnectEnabled (T _subscriber)

  Connect to the add entity signal.
- void **DetachAllStaticModels** ()

  Detach all static models from this link.
- void **DetachStaticModel** (const std::string &_modelName)

  Detach a static model from this link.
- void **DisconnectEnabled** (**event::ConnectionPtr** &_conn)

  Disconnect to the add entity signal.
- void **FillMsg** (msgs::Link &_msg)

  Fill a link message.
- void **Fini** ()

  Finalize the body.
- double **GetAngularDamping** () const

  Get the angular damping factor.
- virtual **math::Box GetBoundingBox** () const

  Get the bounding box for the link and all the child elements.
- **Link_V GetChildJointsLinks** () const

  Returns a vector of children Links connected by joints.
- **CollisionPtr GetCollision** (const std::string &_name)

  Get a child collision by name.
- **CollisionPtr GetCollision** (unsigned int _index) const

  Get a child collision by index.
- **CollisionPtr GetCollisionById** (unsigned int _id) const

  This is an internal function
- **Collision_V GetCollisions** () const

  Get all the child collisions.
- virtual bool **GetEnabled** () const =0

  Get whether this body is enabled in the physics engine.
- virtual bool **GetGravityMode** () const =0

  Get the gravity mode.
- **InertialPtr GetInertial** () const

  Get the inertia of the link.
- virtual bool **GetKinematic** () const

  Implement this function.
- double **GetLinearDamping** () const

  Get the linear damping factor.
- **ModelPtr GetModel** () const

  Get the model that this body belongs to.
- **Link_V GetParentJointsLinks** () const

  Returns a vector of parent Links connected by joints.
- **math::Vector3 GetRelativeAngularAccel** () const

  Get the angular acceleration of the body.
- **math::Vector3 GetRelativeAngularVel** () const

  Get the angular velocity of the body.

- **math::Vector3 GetRelativeForce** () const

  Get the force applied to the body.
- **math::Vector3 GetRelativeLinearAccel** () const

  Get the linear acceleration of the body.
- **math::Vector3 GetRelativeLinearVel** () const

  Get the linear velocity of the body.
- **math::Vector3 GetRelativeTorque** () const

  Get the torque applied to the body.
- bool **GetSelfCollide** ()

  Get Self-Collision Flag, if this is true, this body will collide with other bodies even if they share the same parent.
- unsigned int **GetSensorCount** () const

  Get sensor count.
- std::string **GetSensorName** (unsigned int _index) const

  Get sensor name.
- **math::Vector3 GetWorldAngularAccel** () const

  Get the angular acceleration of the body in the world frame.
- virtual **math::Vector3 GetWorldCoGLinearVel** () const =0

  Get the linear velocity at the body's center of gravity in the world frame.
- **math::Pose GetWorldCoGPose** () const

  Get the pose of the body's center of gravity in the world coordinate frame.
- virtual **math::Vector3 GetWorldForce** () const =0

  Get the force applied to the body in the world frame.
- **math::Vector3 GetWorldLinearAccel** () const

  Get the linear acceleration of the body in the world frame.
- virtual **math::Vector3 GetWorldLinearVel** (const **math::Vector3** &_offset=**math::Vector3**(0, 0, 0)) const =0

  Get the linear velocity of a point on the body in the world frame, using an offset expressed in a body-fixed frame.
- virtual **math::Vector3 GetWorldLinearVel** (const **math::Vector3** &_offset, const **math::Quaternion** &_q) const =0

  Get the linear velocity of a point on the body in the world frame, using an offset expressed in an arbitrary frame.
- virtual **math::Vector3 GetWorldTorque** () const =0

  Get the torque applied to the body in the world frame.
- virtual void **Init** ()

  Initialize the body.
- virtual void **Load** (**sdf::ElementPtr** _sdf)

  Load the body based on an SDF element.
- virtual void **OnPoseChange** ()

  This function is called when the entity's (or one of its parents) pose of the parent has changed.
- void **ProcessMsg** (const msgs::Link &_msg)

  Update parameters from a message.
- void **RemoveChildJoint** (**JointPtr** _joint) **GAZEBO_DEPRECATED**(1.5)

  *Remove Joints that have this **Link** (p. 439) as a parent **Link** (p. 439).*
- void **RemoveChildJoint** (const std::string &_jointName)

  *Remove Joints that have this **Link** (p. 439) as a parent **Link** (p. 439).*
- void **RemoveParentJoint** (**JointPtr** _joint) **GAZEBO_DEPRECATED**(1.5)

  *Remove Joints that have this **Link** (p. 439) as a child **Link** (p. 439).*
- void **RemoveParentJoint** (const std::string &_jointName)

  *Remove Joints that have this **Link** (p. 439) as a child **Link** (p. 439).*

- void **Reset** ()

  Reset the link.
- void **ResetPhysicsStates** ()

  Reset the link.
- void **SetAngularAccel** (const **math::Vector3** &_accel)

  Set the angular acceleration of the body.
- virtual void **SetAngularDamping** (double _damping)=0

  Set the angular damping factor.
- virtual void **SetAngularVel** (const **math::Vector3** &_vel)=0

  Set the angular velocity of the body.
- virtual void **SetAutoDisable** (bool _disable)=0

  Allow the link to auto disable.
- void **SetCollideMode** (const std::string &_mode)

  Set the collide mode of the body.
- virtual void **SetEnabled** (bool _enable) const =0

  Set whether this body is enabled.
- virtual void **SetForce** (const **math::Vector3** &_force)=0

  Set the force applied to the body.
- virtual void **SetGravityMode** (bool _mode)=0

  Set whether gravity affects this body.
- void **SetInertial** (const **InertialPtr** &_inertial)

  Set the mass of the link.
- virtual void **SetKinematic** (const bool &_kinematic)

  Implement this function.
- void **SetLaserRetro** (float _retro)

  Set the laser retro reflectiveness.
- void **SetLinearAccel** (const **math::Vector3** &_accel)

  Set the linear acceleration of the body.
- virtual void **SetLinearDamping** (double _damping)=0

  Set the linear damping factor.
- virtual void **SetLinearVel** (const **math::Vector3** &_vel)=0

  Set the linear velocity of the body.
- void **SetPublishData** (bool _enable)

  Enable/Disable link data publishing.
- virtual bool **SetSelected** (bool _set)

  Set whether this entity has been selected by the user through the gui.
- virtual void **SetSelfCollide** (bool _collide)=0

  Set whether this body will collide with others in the model.
- void **SetState** (const **LinkState** &_state)

  Set the current link state.
- virtual void **SetTorque** (const **math::Vector3** &_torque)=0

  Set the torque applied to the body.
- virtual void **Update** ()

  Update the body.
- virtual void **UpdateMass** ()

  Update the mass matrix.
- virtual void **UpdateParameters** (**sdf::ElementPtr** _sdf)

  Update the parameters using new sdf values.
- virtual void **UpdateSurface** ()

  Update surface parameters.

Protected Attributes

- **math::Vector3 angularAccel**

  Angular acceleration.
- std::vector< **math::Pose** > **attachedModelsOffset**

  Offsets for the attached models.
- std::vector< std::string > **cgVisuals**

  Center of gravity visual elements.
- **InertialPtr inertial**

  Inertial *(p. 388) properties.*
- **math::Vector3 linearAccel**

  Linear acceleration.
- std::vector< std::string > **visuals**

  Link *(p. 439) visual elements.*

Additional Inherited Members

10.75.1 Detailed Description

Link (p. 439) class defines a rigid body entity, containing information on inertia, visual and collision properties of a rigid body.

10.75.2 Constructor & Destructor Documentation

10.75.2.1 gazebo::physics::Link::Link (EntityPtr *_parent*) [explicit]

Constructor.

Parameters

in	*_parent*	Parent of this link.

10.75.2.2 virtual gazebo::physics::Link::~Link ( ) [virtual]

Destructor.

10.75.3 Member Function Documentation

10.75.3.1 void gazebo::physics::Link::AddChildJoint (JointPtr *_joint*)

Joints that have this **Link** (p. 439) as a parent **Link** (p. 439).

Parameters

in	*_joint*	**Joint** (p. 401) that is a child of this link.

10.75.3.2 virtual void gazebo::physics::Link::AddForce (const math::Vector3 & _force) [pure virtual]

Add a force to the body.

Parameters

in	*_force*	Force to add.

10.75.3.3 virtual void gazebo::physics::Link::AddForceAtRelativePosition (const math::Vector3 & _force, const math::Vector3 & _relPos) [pure virtual]

Add a force to the body at position expressed to the body's own frame of reference.

Parameters

in	*_force*	Force to add.
in	*_relPos*	Position on the link to add the force.

10.75.3.4 virtual void gazebo::physics::Link::AddForceAtWorldPosition (const math::Vector3 & _force, const math::Vector3 & _pos) [pure virtual]

Add a force to the body using a global position.

Parameters

in	*_force*	Force to add.
in	*_pos*	Position in global coord frame to add the force.

10.75.3.5 void gazebo::physics::Link::AddParentJoint (JointPtr _joint)

Joints that have this **Link** (p. 439) as a child **Link** (p. 439).

Parameters

in	*_joint*	**Joint** (p. 401) that is a parent of this link.

10.75.3.6 virtual void gazebo::physics::Link::AddRelativeForce (const math::Vector3 & _force) [pure virtual]

Add a force to the body, components are relative to the body's own frame of reference.

Parameters

in	*_force*	Force to add.

10.75.3.7 virtual void gazebo::physics::Link::AddRelativeTorque (const math::Vector3 & _torque) [pure virtual]

Add a torque to the body, components are relative to the body's own frame of reference.

Parameters

in	*_torque*	Torque value to add.

10.75.3.8 virtual void gazebo::physics::Link::AddTorque (const math::Vector3 & *_torque*) [pure virtual]

Add a torque to the body.

Parameters

in	*_torque*	Torque value to add to the link.

10.75.3.9 void gazebo::physics::Link::AttachStaticModel (ModelPtr & *_model,* const math::Pose & *_offset*)

Attach a static model to this link.

Parameters

in	*_model*	Pointer to a static model.
in	*_offset*	Pose relative to this link to place the model.

10.75.3.10 template<typename T > event::ConnectionPtr gazebo::physics::Link::ConnectEnabled (T *_subscriber*) [inline]

Connect to the add entity signal.

Parameters

in	*_subscriber*	Subsciber callback function.

Returns

Pointer to the connection, which must be kept in scope.

References gazebo::event::EventT< T >::Connect().

10.75.3.11 void gazebo::physics::Link::DetachAllStaticModels ( )

Detach all static models from this link.

10.75.3.12 void gazebo::physics::Link::DetachStaticModel (const std::string & *_modelName*)

Detach a static model from this link.

Parameters

in	*_modelName*	Name of an attached model to detach.

10.75.3.13 void gazebo::physics::Link::DisconnectEnabled (event::ConnectionPtr & _conn) [inline]

Disconnect to the add entity signal.

Parameters

in	*_conn*	Connection pointer to disconnect.

References gazebo::event::EventT< T >::Disconnect().

10.75.3.14 void gazebo::physics::Link::FillMsg (msgs::Link & _msg)

Fill a link message.

Parameters

out	*_msg*	Message to fill

10.75.3.15 void gazebo::physics::Link::Fini ( ) [virtual]

Finalize the body.

Reimplemented from **gazebo::physics::Entity** (p. 291).

10.75.3.16 double gazebo::physics::Link::GetAngularDamping ( ) const

Get the angular damping factor.

Returns

Angular damping.

10.75.3.17 virtual math::Box gazebo::physics::Link::GetBoundingBox ( ) const [virtual]

Get the bounding box for the link and all the child elements.

Returns

The link's bounding box.

Reimplemented from **gazebo::physics::Entity** (p. 291).

10.75.3.18 Link_V gazebo::physics::Link::GetChildJointsLinks ( ) const

Returns a vector of children Links connected by joints.

Returns

A vector of children Links connected by joints.

10.75.3.19 CollisionPtr gazebo::physics::Link::GetCollision (const std::string & _name)

Get a child collision by name.

Parameters

in	*_name*	Name of the collision object.

Returns

Pointer to the collision, NULL if the name was not found.

10.75.3.20 CollisionPtr gazebo::physics::Link::GetCollision (unsigned int _index) const

Get a child collision by index.

Parameters

in	*_index*	Index of the collision object.

Returns

Pointer to the collision, NULL if the name was not found.

10.75.3.21 CollisionPtr gazebo::physics::Link::GetCollisionById (unsigned int _id) const

This is an internal function

Get a collision by id.

Parameters

in	*_id*	Id of the collision object to find.

Returns

Pointer to the collision, NULL if the id is invalid.

10.75.3.22 Collision_V gazebo::physics::Link::GetCollisions ( ) const

Get all the child collisions.

Returns

A std::vector of all the child collisions.

10.75.3.23 virtual bool gazebo::physics::Link::GetEnabled ( ) const [pure virtual]

Get whether this body is enabled in the physics engine.

Returns

True if the link is enabled.

10.75.3.24 virtual bool gazebo::physics::Link::GetGravityMode ( ) const [pure virtual]

Get the gravity mode.

Returns

True if gravity is enabled.

10.75.3.25 InertialPtr gazebo::physics::Link::GetInertial ( ) const [inline]

Get the inertia of the link.

Returns

Inertia of the link.

References inertial.

10.75.3.26 virtual bool gazebo::physics::Link::GetKinematic ( ) const [inline], [virtual]

Implement this function.

Get whether this body is in the kinematic state.

Returns

True if the link is kinematic only.

10.75.3.27 double gazebo::physics::Link::GetLinearDamping ( ) const

Get the linear damping factor.

Returns

Linear damping.

10.75.3.28 ModelPtr gazebo::physics::Link::GetModel ( ) const

Get the model that this body belongs to.

Returns

Model (p. 516) that this body belongs to.

10.75.3.29 Link_V gazebo::physics::Link::GetParentJointsLinks ( ) const

Returns a vector of parent Links connected by joints.

Returns

Vector of parent Links connected by joints.

10.75.3.30 math::Vector3 gazebo::physics::Link::GetRelativeAngularAccel ( ) const [virtual]

Get the angular acceleration of the body.

Returns

Angular acceleration of the body.

Reimplemented from **gazebo::physics::Entity** (p. 293).

10.75.3.31 math::Vector3 gazebo::physics::Link::GetRelativeAngularVel ( ) const [virtual]

Get the angular velocity of the body.

Returns

Angular velocity of the body.

Reimplemented from **gazebo::physics::Entity** (p. 293).

10.75.3.32 math::Vector3 gazebo::physics::Link::GetRelativeForce ( ) const

Get the force applied to the body.

Returns

Force applied to the body.

10.75.3.33 math::Vector3 gazebo::physics::Link::GetRelativeLinearAccel ( ) const [virtual]

Get the linear acceleration of the body.

Returns

Linear acceleration of the body.

Reimplemented from **gazebo::physics::Entity** (p. 293).

10.75.3.34 math::Vector3 gazebo::physics::Link::GetRelativeLinearVel ( ) const [virtual]

Get the linear velocity of the body.

Returns

Linear velocity of the body.

Reimplemented from **gazebo::physics::Entity** (p. 294).

10.75.3.35 math::Vector3 gazebo::physics::Link::GetRelativeTorque ( ) const

Get the torque applied to the body.

Returns

Torque applied to the body.

10.75.3.36 bool gazebo::physics::Link::GetSelfCollide ( )

Get Self-Collision Flag, if this is true, this body will collide with other bodies even if they share the same parent.

Returns

True if self collision is enabled.

10.75.3.37 unsigned int gazebo::physics::Link::GetSensorCount ( ) const

Get sensor count.

This will return the number of sensors created by the link when it was loaded. This function is commonly used with **Link::GetSensorName** (p. 451).

Returns

The number of sensors created by the link.

10.75.3.38 std::string gazebo::physics::Link::GetSensorName (unsigned int *_index*) const

Get sensor name.

Get the name of a sensor based on an index. The index should be in the range of 0...**Link::GetSensorCount()** (p. 451).

Note

A **Link** (p. 439) does not manage or maintain a pointer to a **sensors::Sensor** (p. 731). Access to a Sensor object is accomplished through the **sensors::SensorManager** (p. 743). This was done to separate the physics engine from the sensor engine.

Parameters

in	*_index*	Index of the sensor name.

Returns

The name of the sensor, or empty string if the index is out of bounds.

10.75.3.39 math::Vector3 gazebo::physics::Link::GetWorldAngularAccel ( ) const [virtual]

Get the angular acceleration of the body in the world frame.

Returns

Angular acceleration of the body in the world frame.

Reimplemented from **gazebo::physics::Entity** (p. 294).

10.75.3.40 virtual math::Vector3 gazebo::physics::Link::GetWorldCoGLinearVel ( ) const [pure virtual]

Get the linear velocity at the body's center of gravity in the world frame.

Returns

Linear velocity at the body's center of gravity in the world frame.

10.75.3.41 math::Pose gazebo::physics::Link::GetWorldCoGPose ( ) const

Get the pose of the body's center of gravity in the world coordinate frame.

Returns

Pose of the body's center of gravity in the world coordinate frame.

10.75.3.42 virtual math::Vector3 gazebo::physics::Link::GetWorldForce ( ) const [pure virtual]

Get the force applied to the body in the world frame.

Returns

Force applied to the body in the world frame.

10.75.3.43 math::Vector3 gazebo::physics::Link::GetWorldLinearAccel ( ) const [virtual]

Get the linear acceleration of the body in the world frame.

Returns

Linear acceleration of the body in the world frame.

Reimplemented from **gazebo::physics::Entity** (p. 294).

10.75.3.44 virtual math::Vector3 gazebo::physics::Link::GetWorldLinearVel (const math::Vector3 & *_offset* = math::Vector3(0, 0, 0)) const [pure virtual]

Get the linear velocity of a point on the body in the world frame, using an offset expressed in a body-fixed frame.

If no offset is given, the velocity at the origin of the **Link** (p. 439) frame will be returned.

Parameters

in	*_offset*	Offset of the point from the origin of the **Link** (p. 439) frame, expressed in the body-fixed frame.

Returns

Linear velocity of the point on the body

10.75.3.45 virtual math::Vector3 gazebo::physics::Link::GetWorldLinearVel (const math::Vector3 & *_offset*, const math::Quaternion & *_q*) const [pure virtual]

Get the linear velocity of a point on the body in the world frame, using an offset expressed in an arbitrary frame.

Parameters

in	*_offset*	Offset from the origin of the link frame expressed in a frame defined by _q.
in	*_q*	Describes the rotation of a reference frame relative to the world reference frame.

Returns

Linear velocity of the point on the body in the world frame.

10.75.3.46 virtual math::Vector3 gazebo::physics::Link::GetWorldTorque ( ) const [pure virtual]

Get the torque applied to the body in the world frame.

Returns

Torque applied to the body in the world frame.

10.75.3.47 virtual void gazebo::physics::Link::Init ( ) [virtual]

Initialize the body.

Reimplemented from **gazebo::physics::Base** (p. 149).

10.75.3.48 virtual void gazebo::physics::Link::Load (sdf::ElementPtr *_sdf*) [virtual]

Load the body based on an SDF element.

Parameters

in	*_sdf*	SDF parameters.

Reimplemented from **gazebo::physics::Entity** (p. 295).

10.75.3.49 virtual void gazebo::physics::Link::OnPoseChange ( ) [virtual]

This function is called when the entity's (or one of its parents) pose of the parent has changed.

Implements **gazebo::physics::Entity** (p. 295).

10.75.3.50 void gazebo::physics::Link::ProcessMsg (const msgs::Link & *_msg*)

Update parameters from a message.

Parameters

in	*_msg*	Message to read.

10.75.3.51 void gazebo::physics::Link::RemoveChildJoint (JointPtr *_joint*)

Remove Joints that have this **Link** (p. 439) as a parent **Link** (p. 439).

Parameters

in	*_joint*	**Joint** (p. 401) that is a child of this link.

10.75.3.52 void gazebo::physics::Link::RemoveChildJoint (const std::string & *_jointName*)

Remove Joints that have this **Link** (p. 439) as a parent **Link** (p. 439).

Parameters

in	*_jointName*	Child **Joint** (p. 401) name.

10.75.3.53 void gazebo::physics::Link::RemoveParentJoint (JointPtr *_joint*)

Remove Joints that have this **Link** (p. 439) as a child **Link** (p. 439).

Parameters

in	*_joint*	**Joint** (p. 401) that is a parent of this link.

10.75.3.54 void gazebo::physics::Link::RemoveParentJoint (const std::string & *_jointName*)

Remove Joints that have this **Link** (p. 439) as a child **Link** (p. 439).

Parameters

in	*_jointName*	Parent **Joint** (p. 401) name.

10.75.3.55 void gazebo::physics::Link::Reset ( ) [virtual]

Reset the link.

Reimplemented from **gazebo::physics::Entity** (p. 296).

10.75.3.56 void gazebo::physics::Link::ResetPhysicsStates ( )

Reset the link.

10.75.3.57 void gazebo::physics::Link::SetAngularAccel (const math::Vector3 & _accel)

Set the angular acceleration of the body.

Parameters

in	*_accel*	Angular acceleration.

10.75.3.58 virtual void gazebo::physics::Link::SetAngularDamping (double _damping) [pure virtual]

Set the angular damping factor.

Parameters

in	*_damping*	Angular damping factor.

10.75.3.59 virtual void gazebo::physics::Link::SetAngularVel (const math::Vector3 & _vel) [pure virtual]

Set the angular velocity of the body.

Parameters

in	*_vel*	Angular velocity.

10.75.3.60 virtual void gazebo::physics::Link::SetAutoDisable (bool _disable) [pure virtual]

Allow the link to auto disable.

Parameters

in	*_disable*	If true, the link is allowed to auto disable.

10.75.3.61 void gazebo::physics::Link::SetCollideMode (const std::string & _mode)

Set the collide mode of the body.

Parameters

in	*_mode*	**Collision** (p. 199) Mode, this can be: [all|none|sensors|fixed|ghost] all: collides with everything none: collides with nothing sensors: collides with everything else but other sensors fixed: collides with everything else but other fixed ghost: collides with everything else but other ghost

10.75.3.62 virtual void gazebo::physics::Link::SetEnabled (bool *_enable*) const [pure virtual]

Set whether this body is enabled.

Parameters

in	*_enable*	True to enable the link in the physics engine.

10.75.3.63 virtual void gazebo::physics::Link::SetForce (const math::Vector3 & *_force*) [pure virtual]

Set the force applied to the body.

Parameters

in	*_force*	Force value.

10.75.3.64 virtual void gazebo::physics::Link::SetGravityMode (bool *_mode*) [pure virtual]

Set whether gravity affects this body.

Parameters

in	*_mode*	True to enable gravity.

10.75.3.65 void gazebo::physics::Link::SetInertial (const InertialPtr & *_inertial*)

Set the mass of the link.

[in] _inertial **Inertial** (p. 388) value for the link.

10.75.3.66 virtual void gazebo::physics::Link::SetKinematic (const bool & *_kinematic*) [virtual]

Implement this function.

Set whether this body is in the kinematic state.

Parameters

in	*_kinematic*	True to make the link kinematic only.

10.75.3.67 void gazebo::physics::Link::SetLaserRetro (float *_retro*)

Set the laser retro reflectiveness.

Parameters

in	*_retro*	Retro value for all child collisions.

10.75.3.68 void gazebo::physics::Link::SetLinearAccel (const math::Vector3 & _accel)

Set the linear acceleration of the body.

Parameters

in	_accel	Linear acceleration.

10.75.3.69 virtual void gazebo::physics::Link::SetLinearDamping (double _damping) [pure virtual]

Set the linear damping factor.

Parameters

in	_damping	Linear damping factor.

10.75.3.70 virtual void gazebo::physics::Link::SetLinearVel (const math::Vector3 & _vel) [pure virtual]

Set the linear velocity of the body.

Parameters

in	_vel	Linear velocity.

10.75.3.71 void gazebo::physics::Link::SetPublishData (bool _enable)

Enable/Disable link data publishing.

Parameters

in	_enable	True to enable publishing, false to stop publishing

10.75.3.72 virtual bool gazebo::physics::Link::SetSelected (bool _set) [virtual]

Set whether this entity has been selected by the user through the gui.

Parameters

in	_set	True to set the link as selected.

Reimplemented from **gazebo::physics::Base** (p. 151).

10.75.3.73 virtual void gazebo::physics::Link::SetSelfCollide (bool _collide) [pure virtual]

Set whether this body will collide with others in the model.

Parameters

in	*_collid*	True to enable collisions.

10.75.3.74 void gazebo::physics::Link::SetState (const LinkState & *_state*)

Set the current link state.

Parameters

in	*_state*	The state to set the link to.

10.75.3.75 virtual void gazebo::physics::Link::SetTorque (const math::Vector3 & *_torque*) [pure virtual]

Set the torque applied to the body.

Parameters

in	*_torque*	Torque value.

10.75.3.76 virtual void gazebo::physics::Link::Update ( ) [virtual]

Update the body.

Reimplemented from **gazebo::physics::Base** (p. 152).

10.75.3.77 virtual void gazebo::physics::Link::UpdateMass ( ) [inline], [virtual]

Update the mass matrix.

10.75.3.78 virtual void gazebo::physics::Link::UpdateParameters (sdf::ElementPtr *_sdf*) [virtual]

Update the parameters using new sdf values.

Parameters

in	*_sdf*	SDF values to load from.

Reimplemented from **gazebo::physics::Entity** (p. 298).

10.75.3.79 virtual void gazebo::physics::Link::UpdateSurface ( ) [inline], [virtual]

Update surface parameters.

10.75.4 Member Data Documentation

10.75.4.1 math::Vector3 gazebo::physics::Link::angularAccel [protected]

Angular acceleration.

10.75.4.2 std::vector<math::Pose> gazebo::physics::Link::attachedModelsOffset [protected]

Offsets for the attached models.

10.75.4.3 std::vector<std::string> gazebo::physics::Link::cgVisuals [protected]

Center of gravity visual elements.

10.75.4.4 InertialPtr gazebo::physics::Link::inertial [protected]

Inertial (p. 388) properties.

Referenced by GetInertial().

10.75.4.5 math::Vector3 gazebo::physics::Link::linearAccel [protected]

Linear acceleration.

10.75.4.6 std::vector<std::string> gazebo::physics::Link::visuals [protected]

Link (p. 439) visual elements.

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

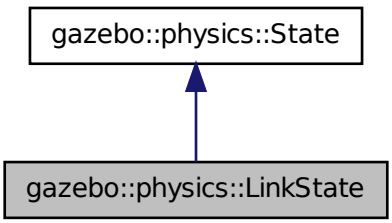
- **Link.hh**

10.76 gazebo::physics::LinkState Class Reference

Store state information of a **physics::Link** (p. 439) object.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::LinkState:



Public Member Functions

- **LinkState** ()

  Default constructor.
- **LinkState** (const **LinkPtr** _link, const **common::Time** &_realTime, const **common::Time** &_simTime)

  Constructor.
- **LinkState** (const **LinkPtr** _link)

  Constructor.
- **LinkState** (const **sdf::ElementPtr** _sdf)

  Constructor.
- virtual ∼**LinkState** ()

  Destructor.
- void **FillSDF** (**sdf::ElementPtr** _sdf)

  Populate a state SDF element with data from the object.
- const **math::Pose** & **GetAcceleration** () const

  Get the link acceleration.
- **CollisionState GetCollisionState** (unsigned int _index) const

  Get a collision state.
- **CollisionState GetCollisionState** (const std::string &_collisionName) const

  Get a link state by link name.
- unsigned int **GetCollisionStateCount** () const

  Get the number of link states.
- const std::vector
  < **CollisionState** > & **GetCollisionStates** () const

  Get the collision states.
- const **math::Pose** & **GetPose** () const

  Get the link pose.
- const **math::Pose** & **GetVelocity** () const

  Get the link velocity.
- const **math::Pose** & **GetWrench** () const

  *Get the force applied to the **Link** (p. 439).*
- bool **IsZero** () const

  Return true if the values in the state are zero.
- void **Load** (const **LinkPtr** _link, const **common::Time** &_realTime, const **common::Time** &_simTime)

  *Load a **LinkState** (p. 459) from a **Link** (p. 439) pointer.*
- virtual void **Load** (const **sdf::ElementPtr** _elem)

  Load state from SDF element.
- **LinkState operator+** (const **LinkState** &_state) const

  Addition operator.
- **LinkState operator-** (const **LinkState** &_state) const

  Subtraction operator.
- **LinkState** & **operator=** (const **LinkState** &_state)

  Assignment operator.
- virtual void **SetRealTime** (const **common::Time** &_time)

  Set the real time when this state was generated.
- virtual void **SetSimTime** (const **common::Time** &_time)

  Set the sim time when this state was generated.
- virtual void **SetWallTime** (const **common::Time** &_time)

  Set the wall time when this state was generated.

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::physics::LinkState** &_state)
  Stream insertion operator.

Additional Inherited Members

10.76.1 Detailed Description

Store state information of a **physics::Link** (p. 439) object.

This class captures the entire state of a **Link** (p. 439) at one specific time during a simulation run.

State (p. 797) of a **Link** (p. 439) includes the state of itself all its child **Collision** (p. 199) entities.

10.76.2 Constructor & Destructor Documentation

10.76.2.1 gazebo::physics::LinkState::LinkState ( )

Default constructor.

10.76.2.2 gazebo::physics::LinkState::LinkState (const **LinkPtr** *_link,* const **common::Time** & *_realTime,* const **common::Time** & *_simTime*)

Constructor.

Build a **LinkState** (p. 459) from an existing **Link** (p. 439).

Parameters

in	*_model*	Pointer to the **Link** (p. 439) from which to gather state info.
in	*_realTime*	Real time stamp.
in	*_simTime*	Sim time stamp

10.76.2.3 gazebo::physics::LinkState::LinkState (const **LinkPtr** *_link*) [explicit]

Constructor.

Build a **LinkState** (p. 459) from an existing **Link** (p. 439).

Parameters

in	*_model*	Pointer to the **Link** (p. 439) from which to gather state info.

10.76.2.4 gazebo::physics::LinkState::LinkState (const **sdf::ElementPtr** *_sdf*) [explicit]

Constructor.

Build a **LinkState** (p. 459) from SDF data

Parameters

in	*_sdf*	SDF data to load a link state from.

10.76.2.5 virtual gazebo::physics::LinkState::~LinkState ( ) [virtual]

Destructor.

10.76.3 Member Function Documentation

10.76.3.1 void gazebo::physics::LinkState::FillSDF (sdf::ElementPtr *_sdf*)

Populate a state SDF element with data from the object.

Parameters

out	*_sdf*	SDF element to populate.

10.76.3.2 const math::Pose& gazebo::physics::LinkState::GetAcceleration ( ) const

Get the link acceleration.

Returns

The acceleration represented as a **math::Pose** (p. 626).

10.76.3.3 CollisionState gazebo::physics::LinkState::GetCollisionState (unsigned int *_index*) const

Get a collision state.

Get a **Collision** (p. 199) **State** (p. 797) based on an index, where index is in the range of 0...**LinkState::GetCollisionStateCount** (p. 463).

Parameters

in	*_index*	Index of the **CollisionState** (p. 209).

Returns

State (p. 797) of the **Collision** (p. 199).

Exceptions

common::Exception *(p. 325)*	When _index is invalid.

10.76.3.4 CollisionState gazebo::physics::LinkState::GetCollisionState (const std::string & *_collisionName*) const

Get a link state by link name.

Searches through all CollisionStates. Returns the **CollisionState** (p. 209) with the matching name, if any.

Parameters

in	*_collisionName*	Name of the **CollisionState** (p. 209)

Returns

State (p. 797) of the **Collision** (p. 199).

Exceptions

common::Exception *(p. 325)*	When _collisionName is invalid

10.76.3.5 unsigned int gazebo::physics::LinkState::GetCollisionStateCount ( ) const

Get the number of link states.

This returns the number of Collisions recorded.

Returns

Number of **CollisionState** (p. 209) recorded.

10.76.3.6 const std::vector<CollisionState>& gazebo::physics::LinkState::GetCollisionStates ( ) const

Get the collision states.

Returns

A vector of collision states.

10.76.3.7 const math::Pose& gazebo::physics::LinkState::GetPose ( ) const

Get the link pose.

Returns

The **math::Pose** (p. 626) of the **Link** (p. 439).

10.76.3.8 const math::Pose& gazebo::physics::LinkState::GetVelocity ( ) const

Get the link velocity.

Returns

The velocity represented as a **math::Pose** (p. 626).

10.76.3.9 const math::Pose& gazebo::physics::LinkState::GetWrench ( ) const

Get the force applied to the **Link** (p. 439).

Returns

Magnitude of the force.

10.76.3.10 bool gazebo::physics::LinkState::IsZero ( ) const

Return true if the values in the state are zero.

Returns

True if the values in the state are zero.

10.76.3.11 void gazebo::physics::LinkState::Load (const LinkPtr *_link,* const common::Time & *_realTime,* const common::Time & *_simTime*)

Load a **LinkState** (p. 459) from a **Link** (p. 439) pointer.

Build a **LinkState** (p. 459) from an existing **Link** (p. 439).

Parameters

in	*_model*	Pointer to the **Link** (p. 439) from which to gather state info.
in	*_realTime*	Real time stamp.
in	*_simTime*	Sim time stamp

10.76.3.12 virtual void gazebo::physics::LinkState::Load (const sdf::ElementPtr *_elem*) [virtual]

Load state from SDF element.

Load **LinkState** (p. 459) information from stored data in and SDF::Element.

Parameters

in	*_elem*	Pointer to the SDF::Element containing state info.

Reimplemented from **gazebo::physics::State** (p. 800).

10.76.3.13 LinkState gazebo::physics::LinkState::operator+ (const LinkState & *_state*) const

Addition operator.

Parameters

in	*_pt*	A state to add.

Returns

The resulting state.

10.76.3.14 LinkState gazebo::physics::LinkState::operator- (const LinkState & *_state*) const

Subtraction operator.

Parameters

in	*_pt*	A state to substract.

Returns

The resulting state.

10.76.3.15 LinkState& gazebo::physics::LinkState::operator= (const LinkState & *_state*)

Assignment operator.

Parameters

in	*_state*	**State** (p. 797) value

Returns

this

10.76.3.16 virtual void gazebo::physics::LinkState::SetRealTime (const common::Time & *_time*) [virtual]

Set the real time when this state was generated.

Parameters

in	*_time*	Clock time since simulation was stated.

Reimplemented from **gazebo::physics::State** (p. 801).

10.76.3.17 virtual void gazebo::physics::LinkState::SetSimTime (const common::Time & *_time*) [virtual]

Set the sim time when this state was generated.

Parameters

in	*_time*	Simulation time when the data was recorded.

Reimplemented from **gazebo::physics::State** (p. 801).

10.76.3.18 virtual void gazebo::physics::LinkState::SetWallTime (const common::Time & *_time*) [virtual]

Set the wall time when this state was generated.

Parameters

in	*_time*	The absolute clock time when the **State** (p. 797) data was recorded.

Reimplemented from **gazebo::physics::State** (p. 801).

10.76.4 Friends And Related Function Documentation

10.76.4.1 std::ostream& operator<< (std::ostream & *_out,* const gazebo::physics::LinkState & *_state*) [friend]

Stream insertion operator.

Parameters

in	*_out*	output stream
in	*_state*	**Link** (p. 439) state to output

Returns

the stream

Disabling this for efficiency.

Disabling this for efficiency.

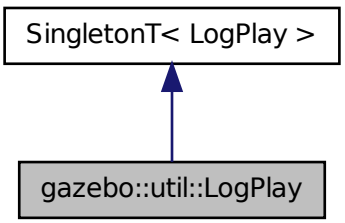
The documentation for this class was generated from the following file:

- **LinkState.hh**

10.77 gazebo::util::LogPlay Class Reference

`#include <LogPlay.hh>`

Inheritance diagram for gazebo::util::LogPlay:



Public Member Functions

- bool **GetChunk** (unsigned int _index, std::string &_data)

  Get data for a particular chunk index.
- unsigned int **GetChunkCount** () const

  Get the number of chunks (steps) in the open log file.
- std::string **GetEncoding** () const

  Get the type of encoding used for current chunck in the open log file.
- std::string **GetGazeboVersion** () const

  Get the Gazebo version number of the open log file.
- std::string **GetHeader** () const

  Get the header that was read from a log file.
- std::string **GetLogVersion** () const

  Get the log version number of the open log file.
- uint32_t **GetRandSeed** () const

  Get the random number seed of the open log file.
- bool **IsOpen** () const

  Return true if a file is open.
- void **Open** (const std::string &_logFile)

  Open a log file for reading.
- bool **Step** (std::string &_data)

  Step through the open log file.

Additional Inherited Members

10.77.1 Member Function Documentation

10.77.1.1 bool gazebo::util::LogPlay::GetChunk (unsigned int *_index*, std::string & *_data*)

Get data for a particular chunk index.

Parameters

in	*_index*	Index of the chunk.
out	*_data*	Storage for the chunk's data.

Returns

True if the _index was valid.

10.77.1.2 unsigned int gazebo::util::LogPlay::GetChunkCount ( ) const

Get the number of chunks (steps) in the open log file.

Returns

The number of recorded states in the log file.

10.77.1.3 std::string gazebo::util::LogPlay::GetEncoding ( ) const

Get the type of encoding used for current chunck in the open log file.

Returns

The type of encoding. An empty string will be returned if **LogPlay::Step** (p. 469) has not been called at least once.

10.77.1.4 std::string gazebo::util::LogPlay::GetGazeboVersion ( ) const

Get the Gazebo version number of the open log file.

Returns

The Gazebo version of the open log file. Empty string if a log file is not open.

10.77.1.5 std::string gazebo::util::LogPlay::GetHeader ( ) const

Get the header that was read from a log file.

Should call **LogPlay::Open** (p. 469) first.

Returns

Header of the open log file.

10.77.1.6 std::string gazebo::util::LogPlay::GetLogVersion ( ) const

Get the log version number of the open log file.

Returns

The log version of the open log file. Empty string if a log file is not open.

10.77.1.7 uint32_t gazebo::util::LogPlay::GetRandSeed ( ) const

Get the random number seed of the open log file.

Returns

The random number seed the open log file. The current random number seed, as defined in **math::Rand::GetSeed** (p. 669).

10.77.1.8 bool gazebo::util::LogPlay::IsOpen ( ) const

Return true if a file is open.

Returns

True if a log file is open.

10.77.1.9 void gazebo::util::LogPlay::Open (const std::string & _logFile)

Open a log file for reading.

Open a log file that was previously recorded.

Parameters

in	*_logFile*	The file to load

Exceptions

Exception	

10.77.1.10 bool gazebo::util::LogPlay::Step (std::string & _data)

Step through the open log file.

Parameters

out	*_data*	Data from next entry in the log file.

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

- **LogPlay.hh**

10.78 Logplay Class Reference

Open and playback log files that were recorded using LogRecord.

10.78.1 Detailed Description

Open and playback log files that were recorded using LogRecord.

Use **Logplay** (p. 469) to open a log file (Logplay::Open), and access the recorded state information. Iterators are available to step through the state information. It is also possible to replay the data in a World using the Play functions. Replay involves reading and applying state information to a World.

See Also

LogRecord, State

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

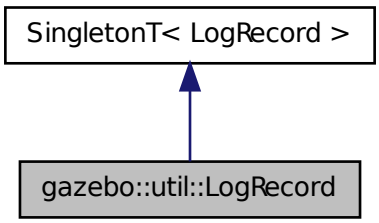
- **LogPlay.hh**

10.79 gazebo::util::LogRecord Class Reference

addtogroup gazebo_util

`#include <util/util.hh>`

Inheritance diagram for gazebo::util::LogRecord:



Public Member Functions

- void **Add** (const std::string &_name, const std::string &_filename, boost::function< bool(std::ostringstream &)> _logCallback)

  Add an object to a log file.
- void **Fini** ()

  Finialize, and shutdown.
- std::string **GetBasePath** () const

  Get the base path for a log recording.
- unsigned int **GetBufferSize** () const

  Get the size of the buffer.
- const std::string & **GetEncoding** () const

  Get the encoding used.
- std::string **GetFilename** (const std::string &_name="") const

  Get the filename for a log object.
- unsigned int **GetFileSize** (const std::string &_name="") const

  Get the file size for a log object.
- bool **GetFirstUpdate** () const

  Return true if an Update has not yet been completed.
- bool **GetPaused** () const

  Get whether logging is paused.
- bool **GetRunning** () const

  Get whether logging is running.
- **common::Time GetRunTime** () const

  Get the run time in sim time.
- bool **Init** (const std::string &_subdir)

  Initialize logging into a subdirectory.
- bool **IsReadyToStart** () const

  Get whether the logger is ready to start, which implies that any previous runs have finished.
- void **Notify** ()

Tell the recorder that an update should occur.

- bool **Remove** (const std::string &_name)

  Remove an entity from a log.
- void **SetBasePath** (const std::string &_path)

  Set the base path.
- void **SetPaused** (bool _paused)

  Set whether logging should pause.
- bool **Start** (const std::string &_encoding="zlib", const std::string &_path="")

  Start the logger.
- void **Stop** ()

  Stop the logger.
- void **Write** (bool _force=false)

  Write all logs.

Additional Inherited Members

10.79.1 Detailed Description

addtogroup gazebo_util

Handles logging of data to disk

The **LogRecord** (p. 469) class is a Singleton that manages data logging of any entity within a running simulation. An entity may be a World, Model, or any of their child entities. This class only writes log files, see **LogPlay** (p. 466) for playback functionality.

State information for an entity may be logged through the **LogRecord::Add** (p. 471) function, and stopped through the **LogRecord::Remove** (p. 474) function. Data may be logged into a single file, or split into many separate files by specifying different filenames for the **LogRecord::Add** (p. 471) function.

The **LogRecord** (p. 469) is updated at the start of each simulation step. This guarantees that all data is stored.

See Also

Logplay (p. 469), State

10.79.2 Member Function Documentation

10.79.2.1 void gazebo::util::LogRecord::Add (const std::string & *_name*, const std::string & *_filename*, boost::function< bool(std::ostringstream &)> *_logCallback*)

Add an object to a log file.

Add a new object to a log. An object can be any valid named object in simulation, including the world itself. Duplicate additions are ignored. Objects can be added to the same file by specifying the same _filename.

Parameters

in	*_name*	Name of the object to log.
in	*_filename*	Filename of the log file.
in	*_logCallback*	Function used to log data for the object. Typically an object will have a log function that outputs data to the provided ofstream.

Exceptions

Exception	

10.79.2.2 void gazebo::util::LogRecord::Fini ( )

Finialize, and shutdown.

10.79.2.3 std::string gazebo::util::LogRecord::GetBasePath ( ) const

Get the base path for a log recording.

Returns

Path for log recording.

10.79.2.4 unsigned int gazebo::util::LogRecord::GetBufferSize ( ) const

Get the size of the buffer.

Returns

Size of the buffer, in bytes.

10.79.2.5 const std::string& gazebo::util::LogRecord::GetEncoding ( ) const

Get the encoding used.

Returns

Either [txt, zlib, or bz2], where txt is plain txt and bz2 and zlib are compressed data with Base64 encoding.

10.79.2.6 std::string gazebo::util::LogRecord::GetFilename (const std::string & *_name* = " ") const

Get the filename for a log object.

Parameters

in	*_name*	Name of the log object.

Returns

Filename, empty string if not found.

10.79.2.7 unsigned int gazebo::util::LogRecord::GetFileSize (const std::string & *_name* = " ") const

Get the file size for a log object.

Parameters

in	*_name*	Name of the log object.

Returns

Size in bytes.

10.79.2.8 bool gazebo::util::LogRecord::GetFirstUpdate ( ) const

Return true if an Update has not yet been completed.

Returns

True if an Update has not yet been completed.

10.79.2.9 bool gazebo::util::LogRecord::GetPaused ( ) const

Get whether logging is paused.

Returns

True if logging is paused.

See Also

LogRecord::SetPaused (p. 474)

10.79.2.10 bool gazebo::util::LogRecord::GetRunning ( ) const

Get whether logging is running.

Returns

True if logging has been started.

10.79.2.11 common::Time gazebo::util::LogRecord::GetRunTime ( ) const

Get the run time in sim time.

Returns

Run sim time.

10.79.2.12 bool gazebo::util::LogRecord::Init (const std::string & *_subdir*)

Initialize logging into a subdirectory.

Init may only be called once, False will be returned if called multiple times.

Parameters

in	*_subdir*	Directory to record to

Returns

True if successful.

10.79.2.13 bool gazebo::util::LogRecord::IsReadyToStart ( ) const

Get whether the logger is ready to start, which implies that any previous runs have finished.

10.79.2.14 void gazebo::util::LogRecord::Notify ( )

Tell the recorder that an update should occur.

10.79.2.15 bool gazebo::util::LogRecord::Remove (const std::string & *_name*)

Remove an entity from a log.

Removes an entity from the logger. The stops data recording for the entity and all its children. For example, specifying a world will stop all data logging.

Parameters

in	*_name*	Name of the log

Returns

True if the entity existed and was removed. False if the entity was not registered with the logger.

10.79.2.16 void gazebo::util::LogRecord::SetBasePath (const std::string & *_path*)

Set the base path.

Parameters

in	*_path*	Path to the new logging location.

10.79.2.17 void gazebo::util::LogRecord::SetPaused (bool *_paused*)

Set whether logging should pause.

A paused state means the log file is still open, but data is not written to it.

Parameters

in	*_paused*	True to pause data logging.

See Also

LogRecord::GetPaused (p. 473)

10.79.2.18 bool gazebo::util::LogRecord::Start (const std::string & _encoding = "zlib", const std::string & _path = "")

Start the logger.

Parameters

in	*_encoding*	The type of encoding (txt, zlib, or bz2).
in	*_path*	Path in which to store log files.

10.79.2.19 void gazebo::util::LogRecord::Stop ( )

Stop the logger.

10.79.2.20 void gazebo::util::LogRecord::Write (bool _force = false)

Write all logs.

Parameters

in	*_force*	True to skip waiting on dataAvailableCondition.

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

- **LogRecord.hh**

10.80 gazebo::Master Class Reference

A ROS Master-like manager that directs gztopic connections, enables each gazebo network client to locate one another for peer-to-peer communication.

`#include <gazebo_core.hh>`

Public Member Functions

- **Master** ()
  Constructor.
- virtual ~**Master** ()
  Destructor.
- void **Fini** ()
  Finalize the master.
- void **Init** (uint16_t _port)
  Initialize.
- void **Run** ()
  Run the master.

- void **RunOnce** ()

  Run the master one iteration.
- void **RunThread** ()

  Run the master in a new thread.
- void **Stop** ()

  Stop the master.

10.80.1 Detailed Description

A ROS Master-like manager that directs gztopic connections, enables each gazebo network client to locate one another for peer-to-peer communication.

Base class for simulation server that handles commandline options, starts a **Master** (p. 475), runs World update and sensor generation loops.

10.80.2 Constructor & Destructor Documentation

10.80.2.1 gazebo::Master::Master ( )

Constructor.

10.80.2.2 virtual gazebo::Master::~Master ( ) [virtual]

Destructor.

10.80.3 Member Function Documentation

10.80.3.1 void gazebo::Master::Fini ( )

Finalize the master.

10.80.3.2 void gazebo::Master::Init (uint16_t *_port*)

Initialize.

Parameters

in	*_port*	The master's port

10.80.3.3 void gazebo::Master::Run ( )

Run the master.

10.80.3.4 void gazebo::Master::RunOnce ( )

Run the master one iteration.

10.80.3.5 void gazebo::Master::RunThread ( )

Run the master in a new thread.

10.80.3.6 void gazebo::Master::Stop ( )

Stop the master.

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

- **Master.hh**

10.81 gazebo::common::Material Class Reference

Encapsulates description of a material.

`#include <common/common.hh>`

Public Types

- enum **BlendMode** { **ADD**, **MODULATE**, **REPLACE**, **BLEND_COUNT** }
- enum **ShadeMode** {
  FLAT, **GOURAUD**, **PHONG**, **BLINN**,
  SHADE_COUNT }

Public Member Functions

- **Material** ()
  Constructor.
- **Material** (const **Color** &_clr)
  Create a material with a default color.
- virtual ~**Material** ()
  Destructor.
- **Color GetAmbient** () const
  Get the ambient color.
- void **GetBlendFactors** (double &_srcFactor, double &_dstFactor)
  Get the blend factors.
- **BlendMode GetBlendMode** () const
  Get the blending mode.
- bool **GetDepthWrite** () const
  Get depth write.
- **Color GetDiffuse** () const
  Get the diffuse color.
- **Color GetEmissive** () const
  Get the emissive color.
- bool **GetLighting** () const
  Get lighting enabled.
- std::string **GetName** () const

Get the name of the material.

- double **GetPointSize** () const

  Get the point size.

- **ShadeMode GetShadeMode** () const

  Get the shading mode.

- double **GetShininess** () const

  Get the shininess.

- **Color GetSpecular** () const

  Get the specular color.

- std::string **GetTextureImage** () const

  Get a texture image.

- double **GetTransparency** () const

  Get the transparency percentage (0..1)

- void **SetAmbient** (const **Color** &_clr)

  Set the ambient color.

- void **SetBlendFactors** (double _srcFactor, double _dstFactor)

  Set the blende factors.

- void **SetBlendMode** (**BlendMode** _b)

  Set the blending mode.

- void **SetDepthWrite** (bool _value)

  Set depth write.

- void **SetDiffuse** (const **Color** &_clr)

  Set the diffuse color.

- void **SetEmissive** (const **Color** &_clr)

  Set the emissive color.

- void **SetLighting** (bool _value)

  Set lighting enabled.

- void **SetPointSize** (double _size)

  Set the point size.

- void **SetShadeMode** (**ShadeMode** _b)

  Set the shading mode param[in] the shading mode.

- void **SetShininess** (double _t)

  Set the shininess.

- void **SetSpecular** (const **Color** &_clr)

  Set the specular color.

- void **SetTextureImage** (const std::string &_tex)

  Set a texture image.

- void **SetTextureImage** (const std::string &_tex, const std::string &_resourcePath)

  Set a texture image.

- void **SetTransparency** (double _t)

  Set the transparency percentage (0..1)

Static Public Attributes

- static std::string **BlendModeStr** [**BLEND_COUNT**]
- static std::string **ShadeModeStr** [**SHADE_COUNT**]

Protected Attributes

- **Color ambient**

  the ambient light color
- **BlendMode blendMode**

  blend mode
- **Color diffuse**

  the diffuse ligth color
- **Color emissive**

  the emissive light color
- std::string **name**

  the name of the material
- double **pointSize**

  point size
- **ShadeMode shadeMode**

  the shade mode
- double **shininess**

  shininess value (0 to 1)
- **Color specular**

  the specular light color
- std::string **texImage**

  the texture image file name
- double **transparency**

  transparency value in the range 0 to 1

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::common::Material** &_m)

  Stream insertion operator param[in] _out the output stream to extract from param[out] _m the material information.

10.81.1 Detailed Description

Encapsulates description of a material.

10.81.2 Member Enumeration Documentation

10.81.2.1 enum gazebo::common::Material::BlendMode

Enumerator:

ADD

MODULATE

REPLACE

BLEND_COUNT

10.81.2.2 enum gazebo::common::Material::ShadeMode

Enumerator:

FLAT

GOURAUD

PHONG

BLINN

SHADE_COUNT

10.81.3 Constructor & Destructor Documentation

10.81.3.1 gazebo::common::Material::Material ( )

Constructor.

10.81.3.2 virtual gazebo::common::Material::~Material ( ) [virtual]

Destructor.

10.81.3.3 gazebo::common::Material::Material (const Color & *_clr*)

Create a material with a default color.

Parameters

in	*_clr*	**Color** (p. 213) of the material

10.81.4 Member Function Documentation

10.81.4.1 Color gazebo::common::Material::GetAmbient ( ) const

Get the ambient color.

Returns

The ambient color

10.81.4.2 void gazebo::common::Material::GetBlendFactors (double & *_srcFactor,* double & *_dstFactor*)

Get the blend factors.

Parameters

in	*_srcFactor*	Source factor is returned in this variable
in	*_dstFactor*	Destination factor is returned in this variable

10.81.4.3 BlendMode gazebo::common::Material::GetBlendMode ( ) const

Get the blending mode.

Returns

the blend mode

10.81.4.4 bool gazebo::common::Material::GetDepthWrite ( ) const

Get depth write.

Returns

the depth write enabled state

10.81.4.5 Color gazebo::common::Material::GetDiffuse ( ) const

Get the diffuse color.

Returns

The diffuse color

10.81.4.6 Color gazebo::common::Material::GetEmissive ( ) const

Get the emissive color.

Returns

The emissive color

10.81.4.7 bool gazebo::common::Material::GetLighting ( ) const

Get lighting enabled.

Returns

the lighting enabled state

10.81.4.8 std::string gazebo::common::Material::GetName ( ) const

Get the name of the material.

Returns

The name of the material

10.81.4.9 double gazebo::common::Material::GetPointSize ( ) const

Get the point size.

Returns

the point size

10.81.4.10 ShadeMode gazebo::common::Material::GetShadeMode ( ) const

Get the shading mode.

Returns

the shading mode

10.81.4.11 double gazebo::common::Material::GetShininess ( ) const

Get the shininess.

Returns

The shininess value

10.81.4.12 Color gazebo::common::Material::GetSpecular ( ) const

Get the specular color.

Returns

The specular color

10.81.4.13 std::string gazebo::common::Material::GetTextureImage ( ) const

Get a texture image.

Returns

The name of the texture image (if one exists) or an empty string

10.81.4.14 double gazebo::common::Material::GetTransparency ( ) const

Get the transparency percentage (0..1)

Returns

The transparency percentage

10.81.4.15 void gazebo::common::Material::SetAmbient (const Color & _clr)

Set the ambient color.

Parameters

in	*_clr*	The ambient color

10.81.4.16 void gazebo::common::Material::SetBlendFactors (double _srcFactor, double _dstFactor)

Set the blende factors.

Will be interpreted as: (texture ∗ _srcFactor) + (scene_pixel ∗ _dstFactor)

Parameters

in	*_srcFactor*	The source factor
in	*_dstFactor*	The destination factor

10.81.4.17 void gazebo::common::Material::SetBlendMode (BlendMode _b)

Set the blending mode.

Parameters

in	*_b*	the blend mode

10.81.4.18 void gazebo::common::Material::SetDepthWrite (bool _value)

Set depth write.

Parameters

in	*_value*	the depth write enabled state

10.81.4.19 void gazebo::common::Material::SetDiffuse (const Color & _clr)

Set the diffuse color.

Parameters

in	*_clr*	The diffuse color

10.81.4.20 void gazebo::common::Material::SetEmissive (const Color & _clr)

Set the emissive color.

Parameters

in	*_clr*	The emissive color

10.81.4.21 void gazebo::common::Material::SetLighting (bool *_value*)

Set lighting enabled.

Parameters

in	*_value*	the lighting enabled state

10.81.4.22 void gazebo::common::Material::SetPointSize (double *_size*)

Set the point size.

Parameters

in	*_size*	the size

10.81.4.23 void gazebo::common::Material::SetShadeMode (ShadeMode *_b*)

Set the shading mode param[in] the shading mode.

10.81.4.24 void gazebo::common::Material::SetShininess (double *_t*)

Set the shininess.

Parameters

in	*_t*	The shininess value

10.81.4.25 void gazebo::common::Material::SetSpecular (const Color & *_clr*)

Set the specular color.

Parameters

in	*_clr*	The specular color

10.81.4.26 void gazebo::common::Material::SetTextureImage (const std::string & *_tex*)

Set a texture image.

Parameters

in	*_tex*	The name of the texture, which must be in Gazebo's resource path

10.81.4.27 void gazebo::common::Material::SetTextureImage (const std::string & _tex, const std::string & _resourcePath)

Set a texture image.

Parameters

in	_tex	The name of the texture
in	_resourcePath	Path which contains _tex

10.81.4.28 void gazebo::common::Material::SetTransparency (double _t)

Set the transparency percentage (0..1)

Parameters

in	_t	The amount of transparency (0..1)

10.81.5 Friends And Related Function Documentation

10.81.5.1 std::ostream& operator<< (std::ostream & _out, const gazebo::common::Material & _m) [friend]

Stream insertion operator param[in] _out the output stream to extract from param[out] _m the material information.

10.81.6 Member Data Documentation

10.81.6.1 Color gazebo::common::Material::ambient [protected]

the ambient light color

10.81.6.2 BlendMode gazebo::common::Material::blendMode [protected]

blend mode

10.81.6.3 std::string gazebo::common::Material::BlendModeStr[BLEND_COUNT] [static]

10.81.6.4 Color gazebo::common::Material::diffuse [protected]

the diffuse ligth color

10.81.6.5 Color gazebo::common::Material::emissive [protected]

the emissive light color

10.81.6.6 std::string gazebo::common::Material::name [protected]

the name of the material

10.81.6.7 double gazebo::common::Material::pointSize [protected]

point size

10.81.6.8 ShadeMode gazebo::common::Material::shadeMode [protected]

the shade mode

10.81.6.9 std::string gazebo::common::Material::ShadeModeStr[SHADE_COUNT] [static]

10.81.6.10 double gazebo::common::Material::shininess [protected]

shininess value (0 to 1)

10.81.6.11 Color gazebo::common::Material::specular [protected]

the specular light color

10.81.6.12 std::string gazebo::common::Material::texImage [protected]

the texture image file name

10.81.6.13 double gazebo::common::Material::transparency [protected]

transparency value in the range 0 to 1

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

- **common/Material.hh**

10.82 gazebo::math::Matrix3 Class Reference

A 3x3 matrix class.

`#include <Matrix3.hh>`

Public Member Functions

- **Matrix3** ()

  Constructor.
- **Matrix3** (const **Matrix3** &_m)

  Copy constructor.
- **Matrix3** (double _v00, double _v01, double _v02, double _v10, double _v11, double _v12, double _v20, double _v21, double _v22)

  Constructor.
- virtual ~**Matrix3** ()

  Desctructor.

- **Matrix3 operator**∗ (const double &_s) const

  returns the element wise scalar multiplication
- **Matrix3 operator**∗ (const **Matrix3** &_m) const

  Matrix multiplication operator.
- **Matrix3 operator+** (const **Matrix3** &_m) const

  returns the element wise sum of two matrices
- **Matrix3 operator-** (const **Matrix3** &_m) const

  returns the element wise difference of two matrices
- bool **operator==** (const **Matrix3** &_m) const

  Equality test operator.
- const double ∗ **operator[]** (size_t _row) const

  Array subscript operator.
- double ∗ **operator[]** (size_t _row)

  Array subscript operator.
- void **SetCol** (unsigned int _c, const **Vector3** &_v)

  Set a column.
- void **SetFromAxes** (const **Vector3** &_xAxis, const **Vector3** &_yAxis, const **Vector3** &_zAxis)

  Set the matrix from three axis (1 per column)
- void **SetFromAxis** (const **Vector3** &_axis, double _angle)

  Set the matrix from an axis and angle.

Protected Attributes

- double **m** [3][3]

  the 3x3 matrix

Friends

- **Matrix3 operator**∗ (double _s, const **Matrix3** &_m)

  Multiplication operators.
- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::math::Matrix3** &_m)

  Stream insertion operator.

10.82.1 Detailed Description

A 3x3 matrix class.

10.82.2 Constructor & Destructor Documentation

10.82.2.1 gazebo::math::Matrix3::Matrix3 ( )

Constructor.

Referenced by operator∗(), operator+(), and operator-().

10.82.2.2 gazebo::math::Matrix3::Matrix3 (const Matrix3 & _m)

Copy constructor.

Parameters

_m	Matrix to copy

10.82.2.3 gazebo::math::Matrix3::Matrix3 (double _v00, double _v01, double _v02, double _v10, double _v11, double _v12, double _v20, double _v21, double _v22)

Constructor.

Parameters

in	_v00	Row 0, Col 0 value
in	_v01	Row 0, Col 1 value
in	_v02	Row 0, Col 2 value
in	_v10	Row 1, Col 0 value
in	_v11	Row 1, Col 1 value
in	_v12	Row 1, Col 2 value
in	_v20	Row 2, Col 0 value
in	_v21	Row 2, Col 1 value
in	_v22	Row 2, Col 2 value

10.82.2.4 virtual gazebo::math::Matrix3::~Matrix3 ( ) [virtual]

Desctructor.

10.82.3 Member Function Documentation

10.82.3.1 Matrix3 gazebo::math::Matrix3::operator* (const double & _s) const [inline]

returns the element wise scalar multiplication

References m, and Matrix3().

10.82.3.2 Matrix3 gazebo::math::Matrix3::operator* (const Matrix3 & _m) const [inline]

Matrix multiplication operator.

Parameters

in	_m	**Matrix3** (p. 486) to multiply

Returns

product of this * _m

References m, and Matrix3().

10.82.3.3 Matrix3 gazebo::math::Matrix3::operator+ (const Matrix3 & _m) const [inline]

returns the element wise sum of two matrices

References m, and Matrix3().

10.82.3.4 Matrix3 gazebo::math::Matrix3::operator- (const Matrix3 & _m) const [inline]

returns the element wise difference of two matrices

References m, and Matrix3().

10.82.3.5 bool gazebo::math::Matrix3::operator== (const Matrix3 & _m) const

Equality test operator.

Parameters

in	*_m*	**Matrix3** (p. 486) to test

Returns

True if equal (using the default tolerance of 1e-6)

10.82.3.6 const double* gazebo::math::Matrix3::operator[] (size_t _row) const [inline]

Array subscript operator.

Parameters

in	*_row*	row index

Returns

a pointer to the row

References m.

10.82.3.7 double* gazebo::math::Matrix3::operator[] (size_t _row) [inline]

Array subscript operator.

Parameters

in	*_row*	row index

Returns

a pointer to the row

References m.

10.82.3.8 void gazebo::math::Matrix3::SetCol (unsigned int _c, const Vector3 & _v)

Set a column.

Parameters

in	_c	The colum index (0, 1, 2)
in	_v	The value to set in each row of the column

10.82.3.9 void gazebo::math::Matrix3::SetFromAxes (const Vector3 & _xAxis, const Vector3 & _yAxis, const Vector3 & _zAxis)

Set the matrix from three axis (1 per column)

Parameters

in	_xAxis	The x axis
in	_yAxis	The y axis
in	_zAxis	The z axis

10.82.3.10 void gazebo::math::Matrix3::SetFromAxis (const Vector3 & _axis, double _angle)

Set the matrix from an axis and angle.

Parameters

in	_axis	the axis
in	_angle	ccw rotation around the axis in radians

10.82.4 Friends And Related Function Documentation

10.82.4.1 Matrix3 operator* (double _s, const Matrix3 & _m) [friend]

Multiplication operators.

Parameters

in	_s	the scaling factor
in	_m	input matrix

Returns

a scaled matrix

10.82.4.2 std::ostream& operator<< (std::ostream & _out, const gazebo::math::Matrix3 & _m) [friend]

Stream insertion operator.

Parameters

in	_out	Output stream
in	_m	Matrix to output

Returns

the stream

10.82.5 Member Data Documentation

10.82.5.1 double gazebo::math::Matrix3::m[3][3] [protected]

the 3x3 matrix

Referenced by operator∗(), operator+(), operator-(), and operator[]().

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

- **Matrix3.hh**

10.83 gazebo::math::Matrix4 Class Reference

A 3x3 matrix class.

`#include <math/gzmath.hh>`

Public Member Functions

- **Matrix4** ()

  Constructor.
- **Matrix4** (const **Matrix4** &_m)

  Copy constructor.
- **Matrix4** (double _v00, double _v01, double _v02, double _v03, double _v10, double _v11, double _v12, double _v13, double _v20, double _v21, double _v22, double _v23, double _v30, double _v31, double _v32, double _v33)

  Constructor.
- virtual ∼**Matrix4** ()

  Destructor.
- **math::Pose GetAsPose** () const

  *Get the transformation as **math::Pose** (*p. *626).*
- **Vector3 GetEulerRotation** (unsigned int solution_number=1) const

  Get the rotation as a Euler angles.
- **Quaternion GetRotation** () const

  Get the rotation as a quaternion.
- **Vector3 GetTranslation** () const

  *Get the translational values as a **Vector3** (*p. *890).*
- **Matrix4 Inverse** () const

  Return the inverse matrix.
- bool **IsAffine** () const

  Return true if the matrix is affine.
- **Matrix4 operator**∗ (const **Matrix4** &_mat) const

  Multiplication operator.
- **Matrix4 operator**∗ (const **Matrix3** &_mat) const

Multiplication operator.

- **Vector3 operator**∗ (const **Vector3** &_vec) const

  Multiplication operator.

- **Matrix4** & **operator=** (const **Matrix4** &_mat)

  Equal operator.

- const **Matrix4** & **operator=** (const **Matrix3** &_mat)

  Equal operator for 3x3 matrix.

- bool **operator==** (const **Matrix4** &_m) const

  Equality operator.

- double ∗ **operator[]** (size_t _row)

  Array subscript operator.

- const double ∗ **operator[]** (size_t _row) const
- void **Set** (double _v00, double _v01, double _v02, double _v03, double _v10, double _v11, double _v12, double _v13, double _v20, double _v21, double _v22, double _v23, double _v30, double _v31, double _v32, double _v33)

  Change the values.

- void **SetScale** (const **Vector3** &_s)

  Set the scale.

- void **SetTranslate** (const **Vector3** &_t)

  Set the translational values [(0, 3) (1, 3) (2, 3)].

- **Vector3 TransformAffine** (const **Vector3** &_v) const

  Perform an affine transformation.

Static Public Attributes

- static const **Matrix4 IDENTITY**

  Identity matrix.

- static const **Matrix4 ZERO**

  Zero matrix.

Protected Attributes

- double **m** [4][4]

  The 4x4 matrix.

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::math::Matrix4** &_m)

  Stream insertion operator.

10.83.1 Detailed Description

A 3x3 matrix class.

10.83.2 Constructor & Destructor Documentation

10.83.2.1 gazebo::math::Matrix4::Matrix4 ( )

Constructor.

10.83.2.2 gazebo::math::Matrix4::Matrix4 (const Matrix4 & _m)

Copy constructor.

Parameters

_m	Matrix to copy

10.83.2.3 gazebo::math::Matrix4::Matrix4 (double _v00, double _v01, double _v02, double _v03, double _v10, double _v11, double _v12, double _v13, double _v20, double _v21, double _v22, double _v23, double _v30, double _v31, double _v32, double _v33)

Constructor.

Parameters

in	*_v00*	Row 0, Col 0 value
in	*_v01*	Row 0, Col 1 value
in	*_v02*	Row 0, Col 2 value
in	*_v03*	Row 0, Col 3 value
in	*_v10*	Row 1, Col 0 value
in	*_v11*	Row 1, Col 1 value
in	*_v12*	Row 1, Col 2 value
in	*_v13*	Row 1, Col 3 value
in	*_v20*	Row 2, Col 0 value
in	*_v21*	Row 2, Col 1 value
in	*_v22*	Row 2, Col 2 value
in	*_v23*	Row 2, Col 3 value
in	*_v30*	Row 3, Col 0 value
in	*_v31*	Row 3, Col 1 value
in	*_v32*	Row 3, Col 2 value
in	*_v33*	Row 3, Col 3 value

10.83.2.4 virtual gazebo::math::Matrix4::~Matrix4 ( ) [virtual]

Destructor.

10.83.3 Member Function Documentation

10.83.3.1 math::Pose gazebo::math::Matrix4::GetAsPose ( ) const

Get the transformation as **math::Pose** (p. 626).

Returns

the pose

10.83.3.2 Vector3 gazebo::math::Matrix4::GetEulerRotation (unsigned int *solution_number* = `1`) const

Get the rotation as a Euler angles.

Returns

the rotation

10.83.3.3 Quaternion gazebo::math::Matrix4::GetRotation ( ) const

Get the rotation as a quaternion.

Returns

the rotation

10.83.3.4 Vector3 gazebo::math::Matrix4::GetTranslation ( ) const

Get the translational values as a **Vector3** (p. 890).

Returns

x,y,z

10.83.3.5 Matrix4 gazebo::math::Matrix4::Inverse ( ) const

Return the inverse matrix.

10.83.3.6 bool gazebo::math::Matrix4::IsAffine ( ) const

Return true if the matrix is affine.

Returns

true if the matrix is affine, false otherwise

10.83.3.7 Matrix4 gazebo::math::Matrix4::operator* (const Matrix4 & *_mat*) const

Multiplication operator.

Parameters

_mat	Incoming matrix

Returns

This matrix * _mat

10.83.3.8 Matrix4 gazebo::math::Matrix4::operator* (const Matrix3 & _mat) const

Multiplication operator.

Parameters

_mat	Incoming matrix

Returns

This matrix * _mat

10.83.3.9 Vector3 gazebo::math::Matrix4::operator* (const Vector3 & _vec) const

Multiplication operator.

Parameters

_vec	**Vector3** (p. 890)

Returns

Resulting vector from multiplication

10.83.3.10 Matrix4& gazebo::math::Matrix4::operator= (const Matrix4 & _mat)

Equal operator.

this = _mat

Parameters

_mat	Incoming matrix

Returns

itself

10.83.3.11 const Matrix4& gazebo::math::Matrix4::operator= (const Matrix3 & _mat)

Equal operator for 3x3 matrix.

Parameters

_mat	Incoming matrix

Returns

itself

10.83.3.12 bool gazebo::math::Matrix4::operator== (const Matrix4 & *_m*) const

Equality operator.

Parameters

in	*_m*	**Matrix3** (p. 486) to test

Returns

true if the 2 matrices are equal (using the tolerance 1e-6), false otherwise

10.83.3.13 double* gazebo::math::Matrix4::operator[] (size_t *_row*) [inline]

Array subscript operator.

Parameters

in	*_row*	the row index

Returns

the row

References m.

10.83.3.14 const double* gazebo::math::Matrix4::operator[] (size_t *_row*) const [inline]

Parameters

in	*_row*	the row index

Returns

the row

References m.

10.83.3.15 void gazebo::math::Matrix4::Set (double *_v00*, double *_v01*, double *_v02*, double *_v03*, double *_v10*, double *_v11*, double *_v12*, double *_v13*, double *_v20*, double *_v21*, double *_v22*, double *_v23*, double *_v30*, double *_v31*, double *_v32*, double *_v33*)

Change the values.

Parameters

in	*_v00*	Row 0, Col 0 value
in	*_v01*	Row 0, Col 1 value
in	*_v02*	Row 0, Col 2 value
in	*_v03*	Row 0, Col 3 value
in	*_v10*	Row 1, Col 0 value
in	*_v11*	Row 1, Col 1 value
in	*_v12*	Row 1, Col 2 value
in	*_v13*	Row 1, Col 3 value
in	*_v20*	Row 2, Col 0 value
in	*_v21*	Row 2, Col 1 value
in	*_v22*	Row 2, Col 2 value
in	*_v23*	Row 2, Col 3 value
in	*_v30*	Row 3, Col 0 value
in	*_v31*	Row 3, Col 1 value
in	*_v32*	Row 3, Col 2 value
in	*_v33*	Row 3, Col 3 value

10.83.3.16 void gazebo::math::Matrix4::SetScale (const Vector3 & *_s*)

Set the scale.

Parameters

in	*_s*	scale

10.83.3.17 void gazebo::math::Matrix4::SetTranslate (const Vector3 & *_t*)

Set the translational values [(0, 3) (1, 3) (2, 3)].

Parameters

in	*_t*	Values to set

10.83.3.18 Vector3 gazebo::math::Matrix4::TransformAffine (const Vector3 & *_v*) const

Perform an affine transformation.

Parameters

_v	**Vector3** (p. 890) value for the transformation

Returns

The result of the transformation

10.83.4 Friends And Related Function Documentation

10.83.4.1 std::ostream& operator<< (std::ostream & _out, const gazebo::math::Matrix4 & _m) [friend]

Stream insertion operator.

Parameters

_out	output stream
_m	Matrix to output

Returns

the stream

10.83.5 Member Data Documentation

10.83.5.1 const Matrix4 gazebo::math::Matrix4::IDENTITY [static]

Identity matrix.

10.83.5.2 double gazebo::math::Matrix4::m[4][4] [protected]

The 4x4 matrix.

Referenced by operator[]().

10.83.5.3 const Matrix4 gazebo::math::Matrix4::ZERO [static]

Zero matrix.

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

- **Matrix4.hh**

10.84 gazebo::common::Mesh Class Reference

A 3D mesh.

`#include <common/common.hh>`

Public Member Functions

- **Mesh** ()
  Constructor.
- virtual ~**Mesh** ()
  Destructor.
- int **AddMaterial** (**Material** *_mat)
  Add a material to the mesh.
- void **AddSubMesh** (**SubMesh** *_child)
  Add a submesh mesh.

- void **Center** (const **math::Vector3** &_center=**math::Vector3::Zero**)

  Move the center of the mesh to the given coordinate.
- void **FillArrays** (float ∗∗_vertArr, int ∗∗_indArr) const

  Put all the data into flat arrays.
- void **GenSphericalTexCoord** (const **math::Vector3** &_center)

  Generate texture coordinates using spherical projection from center.
- void **GetAABB** (**math::Vector3** &_center, **math::Vector3** &_min_xyz, **math::Vector3** &_max_xyz) const

  Get AABB coordinate.
- unsigned int **GetIndexCount** () const

  Return the number of indices.
- const **Material** ∗ **GetMaterial** (int _index) const

  Get a material.
- unsigned int **GetMaterialCount** () const

  Get the number of materials.
- **math::Vector3 GetMax** () const

  Get the maximun X, Y, Z values.
- **math::Vector3 GetMin** () const

  Get the minimum X, Y, Z values.
- std::string **GetName** () const

  Get the name of this mesh.
- unsigned int **GetNormalCount** () const

  Return the number of normals.
- std::string **GetPath** () const

  Get the path which contains the mesh resource.
- **Skeleton** ∗ **GetSkeleton** () const

  Get the skeleton to which this mesh is attached.
- const **SubMesh** ∗ **GetSubMesh** (unsigned int _i) const

  Get a child mesh.
- const **SubMesh** ∗ **GetSubMesh** (const std::string &_name) const

  Get a child mesh by name.
- unsigned int **GetSubMeshCount** () const

  Get the number of children.
- unsigned int **GetTexCoordCount** () const

  Return the number of texture coordinates.
- unsigned int **GetVertexCount** () const

  Return the number of vertices.
- bool **HasSkeleton** () const

  Return true if mesh is attached to a skeleton.
- void **RecalculateNormals** ()

  Recalculate all the normals of each face defined by three indices.
- void **Scale** (double _factor)

  Scale all vertices by _factor.
- void **SetName** (const std::string &_n)

  Set the name of this mesh.
- void **SetPath** (const std::string &_path)

  Set the path which contains the mesh resource.
- void **SetScale** (const **math::Vector3** &_factor)

Scale all vertices by the _factor vector.

- void **SetSkeleton** (**Skeleton** ∗_skel)

  Set the mesh skeleton.

- void **Translate** (const **math::Vector3** &_vec)

  Move all vertices in all submeshes by _vec.

10.84.1 Detailed Description

A 3D mesh.

10.84.2 Constructor & Destructor Documentation

10.84.2.1 gazebo::common::Mesh::Mesh ( )

Constructor.

10.84.2.2 virtual gazebo::common::Mesh::∼Mesh ( ) [virtual]

Destructor.

10.84.3 Member Function Documentation

10.84.3.1 int gazebo::common::Mesh::AddMaterial (**Material** ∗ *_mat*)

Add a material to the mesh.

Parameters

in	*_mat*	the material

Returns

Index of this material

10.84.3.2 void gazebo::common::Mesh::AddSubMesh (**SubMesh** ∗ *_child*)

Add a submesh mesh.

The **Mesh** (p. 498) object takes ownership of the submesh.

Parameters

in	*_child*	the submesh

10.84.3.3 void gazebo::common::Mesh::Center (const **math::Vector3** & *_center* = **math::Vector3::Zero**)

Move the center of the mesh to the given coordinate.

This will move all the vertices in all submeshes.

Parameters

in	*_center*	Location of the mesh center.

10.84.3.4 void gazebo::common::Mesh::FillArrays (float ** *_vertArr*, int ** *_indArr*) const

Put all the data into flat arrays.

Parameters

out	*_vertArr*	the vertex array
out	*_indArr*	the index array

10.84.3.5 void gazebo::common::Mesh::GenSphericalTexCoord (const math::Vector3 & *_center*)

Generate texture coordinates using spherical projection from center.

Parameters

in	*_center*	the center of the projection

10.84.3.6 void gazebo::common::Mesh::GetAABB (math::Vector3 & *_center*, math::Vector3 & *_min_xyz*, math::Vector3 & *_max_xyz*) const

Get AABB coordinate.

Parameters

out	*_center*	of the bounding box
out	*_min_xyz*	bounding box minimum values
out	*_max_xyz*	bounding box maximum values

10.84.3.7 unsigned int gazebo::common::Mesh::GetIndexCount ( ) const

Return the number of indices.

Returns

the count

10.84.3.8 const Material* gazebo::common::Mesh::GetMaterial (int *_index*) const

Get a material.

Parameters

in	*_index*	the index

Returns

the material or NULL if the index is out of bounds

10.84.3.9 unsigned int gazebo::common::Mesh::GetMaterialCount ( ) const

Get the number of materials.

Returns

the count

10.84.3.10 math::Vector3 gazebo::common::Mesh::GetMax ( ) const

Get the maximun X, Y, Z values.

Returns

the upper bounds of the bounding box

10.84.3.11 math::Vector3 gazebo::common::Mesh::GetMin ( ) const

Get the minimum X, Y, Z values.

Returns

the lower bounds of the bounding box

10.84.3.12 std::string gazebo::common::Mesh::GetName ( ) const

Get the name of this mesh.

Returns

the name

10.84.3.13 unsigned int gazebo::common::Mesh::GetNormalCount ( ) const

Return the number of normals.

Returns

the count

10.84.3.14 std::string gazebo::common::Mesh::GetPath ( ) const

Get the path which contains the mesh resource.

Returns

the path to the mesh resource

10.84.3.15 Skeleton* gazebo::common::Mesh::GetSkeleton ( ) const

Get the skeleton to which this mesh is attached.

Returns

pointer to skeleton, or NULL if none is present.

10.84.3.16 const SubMesh* gazebo::common::Mesh::GetSubMesh (unsigned int _i) const

Get a child mesh.

Parameters

in	*_i*	the index

Returns

the submesh. An exception is thrown if the index is out of bounds

10.84.3.17 const SubMesh* gazebo::common::Mesh::GetSubMesh (const std::string & _name) const

Get a child mesh by name.

Parameters

in	*_name*	Name of the submesh.

Returns

The submesh, NULL if the _name is not found.

10.84.3.18 unsigned int gazebo::common::Mesh::GetSubMeshCount ( ) const

Get the number of children.

Returns

the count

10.84.3.19 unsigned int gazebo::common::Mesh::GetTexCoordCount ( ) const

Return the number of texture coordinates.

Returns

the count

10.84.3.20 unsigned int gazebo::common::Mesh::GetVertexCount ( ) const

Return the number of vertices.

Returns

the count

10.84.3.21 bool gazebo::common::Mesh::HasSkeleton ( ) const

Return true if mesh is attached to a skeleton.

10.84.3.22 void gazebo::common::Mesh::RecalculateNormals ( )

Recalculate all the normals of each face defined by three indices.

10.84.3.23 void gazebo::common::Mesh::Scale (double *_factor*)

Scale all vertices by _factor.

Parameters

_factor	Scaling factor

10.84.3.24 void gazebo::common::Mesh::SetName (const std::string & *_n*)

Set the name of this mesh.

Parameters

in	*_n*	the name to set

10.84.3.25 void gazebo::common::Mesh::SetPath (const std::string & *_path*)

Set the path which contains the mesh resource.

Parameters

in	*_path*	the file path

10.84.3.26 void gazebo::common::Mesh::SetScale (const **math::Vector3** & *_factor*)

Scale all vertices by the _factor vector.

Parameters

in	*_factor*	Scaling vector

10.84.3.27 void gazebo::common::Mesh::SetSkeleton (Skeleton * _skel)

Set the mesh skeleton.

10.84.3.28 void gazebo::common::Mesh::Translate (const math::Vector3 & _vec)

Move all vertices in all submeshes by _vec.

Parameters

in	*_vec*	Amount to translate vertices.

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

- **Mesh.hh**

10.85 gazebo::common::MeshCSG Class Reference

Creates CSG meshes.

`#include <common/common.hh>`

Public Types

- enum **BooleanOperation** { **UNION**, **INTERSECTION**, **DIFFERENCE** }

  An enumeration of the boolean operations.

Public Member Functions

- **MeshCSG** ()

  Constructor.
- virtual ~**MeshCSG** ()

  Destructor.
- **Mesh** * **CreateBoolean** (const **Mesh** *_m1, const **Mesh** *_m2, const int _operation, const **math::Pose** &_offset=**math::Pose::Zero**)

  Create a boolean mesh from two meshes.

10.85.1 Detailed Description

Creates CSG meshes.

10.85.2 Member Enumeration Documentation

10.85.2.1 enum gazebo::common::MeshCSG::BooleanOperation

An enumeration of the boolean operations.

Enumerator:

UNION

INTERSECTION

DIFFERENCE

10.85.3 Constructor & Destructor Documentation

10.85.3.1 gazebo::common::MeshCSG::MeshCSG ( )

Constructor.

10.85.3.2 virtual gazebo::common::MeshCSG::~MeshCSG ( ) [virtual]

Destructor.

10.85.4 Member Function Documentation

10.85.4.1 Mesh* gazebo::common::MeshCSG::CreateBoolean (const Mesh * *_m1*, const Mesh * *_m2*, const int *_operation*, const math::Pose & *_offset* = math::Pose::Zero)

Create a boolean mesh from two meshes.

Parameters

in	*_m1*	the parent mesh in the boolean operation
in	*_m2*	the child mesh in the boolean operation
in	*_operation*	the boolean operation applied to the two meshes
in	*_offset*	_m2's pose offset from _m1

Returns

a pointer to the created mesh

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

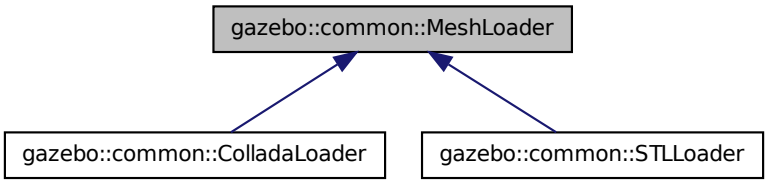
- **MeshCSG.hh**

10.86 gazebo::common::MeshLoader Class Reference

Base class for loading meshes.

`#include <common/common.hh>`

Inheritance diagram for gazebo::common::MeshLoader:



Public Member Functions

- **MeshLoader** ()
  Constructor.
- virtual ∼**MeshLoader** ()
  Destructor.
- virtual **Mesh** ∗ **Load** (const std::string &_filename)=0
  Load a 3D mesh.

10.86.1 Detailed Description

Base class for loading meshes.

10.86.2 Constructor & Destructor Documentation

10.86.2.1 gazebo::common::MeshLoader::MeshLoader ( )

Constructor.

10.86.2.2 virtual gazebo::common::MeshLoader::∼MeshLoader ( ) [virtual]

Destructor.

10.86.3 Member Function Documentation

10.86.3.1 virtual Mesh∗ gazebo::common::MeshLoader::Load (const std::string & *_filename*) [pure virtual]

Load a 3D mesh.

Parameters

in	*_filename*	the path to the mesh

Returns

a pointer to the created mesh

Implemented in **gazebo::common::ColladaLoader** (p. 199), and **gazebo::common::STLLoader** (p. 803).

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

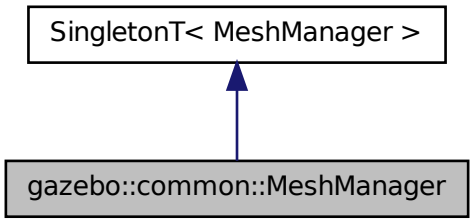
- **MeshLoader.hh**

10.87 gazebo::common::MeshManager Class Reference

Maintains and manages all meshes.

`#include <common/common.hh>`

Inheritance diagram for gazebo::common::MeshManager:



Public Member Functions

- void **AddMesh** (**Mesh** *_mesh)

  Add a mesh to the manager.
- void **CreateBox** (const std::string &_name, const **math::Vector3** &_sides, const **math::Vector2d** &_uvCoords)

  Create a Box mesh.
- void **CreateCamera** (const std::string &_name, float _scale)

  Create a Camera mesh.
- void **CreateCone** (const std::string &_name, float _radius, float _height, int _rings, int _segments)

  Create a cone mesh.
- void **CreateCylinder** (const std::string &_name, float _radius, float _height, int _rings, int _segments)

  Create a cylinder mesh.
- void **CreatePlane** (const std::string &_name, const **math::Plane** &_plane, const **math::Vector2d** &_segments, const **math::Vector2d** &_uvTile)

  Create mesh for a plane.
- void **CreatePlane** (const std::string &_name, const **math::Vector3** &_normal, double _d, const **math::Vector2d** &_size, const **math::Vector2d** &_segments, const **math::Vector2d** &_uvTile)

  Create mesh for a plane.

- void **CreateSphere** (const std::string &_name, float _radius, int _rings, int _segments)

  Create a sphere mesh.
- void **CreateTube** (const std::string &_name, float _innerRadius, float _outterRadius, float _height, int _rings, int _segments)

  Create a tube mesh.
- void **GenSphericalTexCoord** (const **Mesh** *_mesh, **math::Vector3** _center)

  generate spherical texture coordinates
- const **Mesh** * **GetMesh** (const std::string &_name) const

  Get a mesh by name.
- void **GetMeshAABB** (const **Mesh** *_mesh, **math::Vector3** &_center, **math::Vector3** &_min_xyz, **math::Vector3** &_max_xyz)

  Get mesh aabb and center.
- bool **HasMesh** (const std::string &_name) const

  Return true if the mesh exists.
- bool **IsValidFilename** (const std::string &_filename)

  Checks a path extension against the list of valid extensions.
- const **Mesh** * **Load** (const std::string &_filename)

  Load a mesh from a file.

Additional Inherited Members

10.87.1 Detailed Description

Maintains and manages all meshes.

10.87.2 Member Function Documentation

10.87.2.1 void gazebo::common::MeshManager::AddMesh (Mesh * *_mesh*)

Add a mesh to the manager.

This **MeshManager** (p. 508) takes ownership of the mesh and will destroy it. See ~MeshManager.

Parameters

in	*the*	mesh to add.

10.87.2.2 void gazebo::common::MeshManager::CreateBox (const std::string & *_name,* const math::Vector3 & *_sides,* const math::Vector2d & *_uvCoords*)

Create a Box mesh.

Parameters

in	*_name*	the name of the new mesh
in	*_sides*	the x y x dimentions of eah side in meter
in	*_uvCoords*	the texture coordinates

10.87.2.3 void gazebo::common::MeshManager::CreateCamera (const std::string & _name, float _scale)

Create a Camera mesh.

Parameters

in	*_name*	name of the new mesh
in	*_scale*	scaling factor for the camera

10.87.2.4 void gazebo::common::MeshManager::CreateCone (const std::string & _name, float _radius, float _height, int _rings, int _segments)

Create a cone mesh.

Parameters

in	*_name*	the name of the new mesh
in	*_radius*	the radius of the cylinder in the x y plane
in	*_height*	the height along z
in	*_rings*	the number of circles along the height
in	*_segments*	the number of segment per circle

10.87.2.5 void gazebo::common::MeshManager::CreateCylinder (const std::string & _name, float _radius, float _height, int _rings, int _segments)

Create a cylinder mesh.

Parameters

in	*_name*	the name of the new mesh
in	*_radius*	the radius of the cylinder in the x y plane
in	*_height*	the height along z
in	*_rings*	the number of circles along the height
in	*_segments*	the number of segment per circle

10.87.2.6 void gazebo::common::MeshManager::CreatePlane (const std::string & _name, const math::Plane & _plane, const math::Vector2d & _segments, const math::Vector2d & _uvTile)

Create mesh for a plane.

Parameters

in	*_name*	
in	*_plane*	plane parameters
in	*_segments*	number of segments in x and y
in	*_uvTile*	the texture tile size in x and y

10.87.2.7 void gazebo::common::MeshManager::CreatePlane (const std::string & *_name*, const math::Vector3 & *_normal*, double *_d*, const math::Vector2d & *_size*, const math::Vector2d & *_segments*, const math::Vector2d & *_uvTile*)

Create mesh for a plane.

Parameters

in	*_name*	the name of the new mesh
in	*_normal*	the normal to the plane
in	*_d*	distance from the origin along normal
in	*_size*	the size of the plane in x and y
in	*_segments*	the number of segments in x and y
in	*_uvTile*	the texture tile size in x and y

10.87.2.8 void gazebo::common::MeshManager::CreateSphere (const std::string & *_name*, float *_radius*, int *_rings*, int *_segments*)

Create a sphere mesh.

Parameters

in	*_name*	the name of the mesh
in	*_radius*	radius of the sphere in meter
in	*_rings*	number of circles on th y axis
in	*_segments*	number of segment per circle

10.87.2.9 void gazebo::common::MeshManager::CreateTube (const std::string & *_name*, float *_innerRadius*, float *_outterRadius*, float *_height*, int *_rings*, int *_segments*)

Create a tube mesh.

Generates rings inside and outside the cylinder Needs at least two rings and 3 segments

Parameters

in	*_name*	the name of the new mesh
in	*_innerRadius*	the inner radius of the tube in the x y plane
in	*_outterRadius*	the outer radius of the tube in the x y plane
in	*_height*	the height along z
in	*_rings*	the number of circles along the height
in	*_segments*	the number of segment per circle

10.87.2.10 void gazebo::common::MeshManager::GenSphericalTexCoord (const Mesh * *_mesh*, math::Vector3 *_center*)

generate spherical texture coordinates

10.87.2.11 const Mesh* gazebo::common::MeshManager::GetMesh (const std::string & *_name*) const

Get a mesh by name.

Parameters

in	*_name*	the name of the mesh to look for

Returns

the mesh or NULL if not found

10.87.2.12 void gazebo::common::MeshManager::GetMeshAABB (const Mesh * *_mesh*, math::Vector3 & *_center*, math::Vector3 & *_min_xyz*, math::Vector3 & *_max_xyz*)

Get mesh aabb and center.

Parameters

in	*_mesh*	the mesh
out	*_center*	the AAB center position
out	*_min_xyz*	the bounding box minimum
out	*_max_xyz*	the bounding box maximum

10.87.2.13 bool gazebo::common::MeshManager::HasMesh (const std::string & *_name*) const

Return true if the mesh exists.

Parameters

in	*_name*	the name of the mesh

10.87.2.14 bool gazebo::common::MeshManager::IsValidFilename (const std::string & *_filename*)

Checks a path extension against the list of valid extensions.

Returns

true if the file extension is loadable

10.87.2.15 const Mesh* gazebo::common::MeshManager::Load (const std::string & *_filename*)

Load a mesh from a file.

Parameters

in	*_filename*	the path to the mesh

Returns

a pointer to the created mesh

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

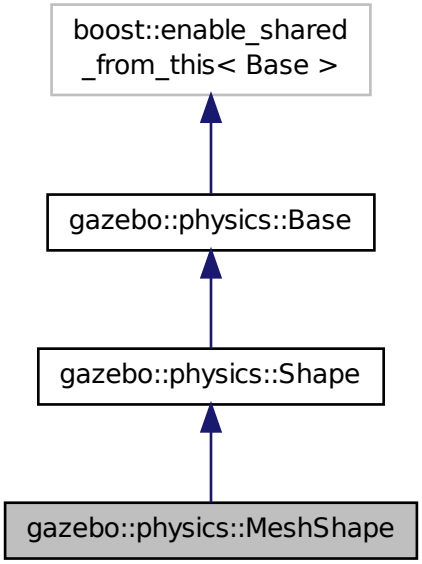
- **MeshManager.hh**

10.88 gazebo::physics::MeshShape Class Reference

Triangle mesh collision shape.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::MeshShape:



Public Member Functions

- **MeshShape** (**CollisionPtr** _parent)

  Constructor.
- virtual ~**MeshShape** ()

  Destructor.
- void **FillMsg** (msgs::Geometry &_msg)

  Populate a msgs::Geometry message with data from this shape.
- std::string **GetFilename** () const **GAZEBO_DEPRECATED**(1.5)

  Deprecated.
- std::string **GetMeshURI** () const

  Get the URI of the mesh data.
- virtual **math::Vector3 GetSize** () const

  Get the size of the triangle mesh.
- virtual void **Init** ()

  Initialize the shape.
- virtual void **ProcessMsg** (const msgs::Geometry &_msg)

  Update this shape from a message.

- void **SetFilename** (const std::string &_filename) **GAZEBO_DEPRECATED**(1.5)

  Deprecated.
- void **SetMesh** (const std::string &_uri, const std::string &_submesh="", bool _center=false)

  Set the mesh uri and submesh name.
- void **SetScale** (const **math::Vector3** &_scale)

  Set the scaling factor.
- virtual void **Update** ()

  Update the tri mesh.

Protected Attributes

- const **common::Mesh** ∗ **mesh**

  Pointer to the mesh data.
- **common::SubMesh** ∗ **submesh**

  The submesh to use from within the parent mesh.

Additional Inherited Members

10.88.1 Detailed Description

Triangle mesh collision shape.

10.88.2 Constructor & Destructor Documentation

10.88.2.1 gazebo::physics::MeshShape::MeshShape (CollisionPtr *_parent*) [explicit]

Constructor.

Parameters

in	*_parent*	Parent collision.

10.88.2.2 virtual gazebo::physics::MeshShape::∼MeshShape ( ) [virtual]

Destructor.

10.88.3 Member Function Documentation

10.88.3.1 void gazebo::physics::MeshShape::FillMsg (msgs::Geometry & *_msg*) [virtual]

Populate a msgs::Geometry message with data from this shape.

Parameters

out	*_msg*	Message to fill.

Implements **gazebo::physics::Shape** (p. 756).

10.88.3.2 std::string gazebo::physics::MeshShape::GetFilename ( ) const

Deprecated.

See Also

GetMeshURI (p. 515)

10.88.3.3 std::string gazebo::physics::MeshShape::GetMeshURI ( ) const

Get the URI of the mesh data.

Returns

The URI of the mesh data.

10.88.3.4 virtual math::Vector3 gazebo::physics::MeshShape::GetSize ( ) const [virtual]

Get the size of the triangle mesh.

Returns

The size of the triangle mesh.

10.88.3.5 virtual void gazebo::physics::MeshShape::Init ( ) [virtual]

Initialize the shape.

Implements **gazebo::physics::Shape** (p. 756).

10.88.3.6 virtual void gazebo::physics::MeshShape::ProcessMsg (const msgs::Geometry & *_msg*) [virtual]

Update this shape from a message.

Parameters

in	*_msg*	Message that contains triangle mesh info.

Implements **gazebo::physics::Shape** (p. 756).

10.88.3.7 void gazebo::physics::MeshShape::SetFilename (const std::string & *_filename*)

Deprecated.

See Also

SetMesh (p. 516)

10.88.3.8 void gazebo::physics::MeshShape::SetMesh (const std::string & *_uri,* const std::string & *_submesh* = " ", bool *_center* = false)

Set the mesh uri and submesh name.

Parameters

in	*_uri*	Filename of the mesh file to load from.
in	*_submesh*	Name of the submesh to use within the mesh
in	*_center*	True to center the submesh. specified in the _uri.

10.88.3.9 void gazebo::physics::MeshShape::SetScale (const math::Vector3 & *_scale*)

Set the scaling factor.

Parameters

in	*_scale*	Scaling factor.

10.88.3.10 virtual void gazebo::physics::MeshShape::Update ( ) [inline], [virtual]

Update the tri mesh.

Reimplemented from **gazebo::physics::Base** (p. 152).

10.88.4 Member Data Documentation

10.88.4.1 const common::Mesh* gazebo::physics::MeshShape::mesh [protected]

Pointer to the mesh data.

10.88.4.2 common::SubMesh* gazebo::physics::MeshShape::submesh [protected]

The submesh to use from within the parent mesh.

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

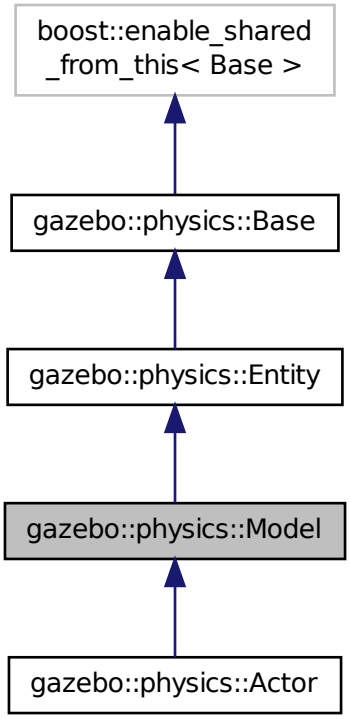
- **MeshShape.hh**

10.89 gazebo::physics::Model Class Reference

A model is a collection of links, joints, and plugins.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::Model:



Public Member Functions

- **Model** (**BasePtr** _parent)

  Constructor.
- virtual ~**Model** ()

  Destructor.
- void **AttachStaticModel** (**ModelPtr** &_model, **math::Pose** _offset)

  Attach a static model to this model.
- void **DetachStaticModel** (const std::string &_model)

  Detach a static model from this model.
- void **FillMsg** (msgs::Model &_msg)

  Fill a model message.
- virtual void **Fini** ()

  Finalize the model.
- bool **GetAutoDisable** () const

  Return the value of the SDF <allow_auto_disable> element.
- virtual **math::Box GetBoundingBox** () const

  Get the size of the bounding box.

- **JointPtr GetJoint** (const std::string &name)

  Get a joint.
- **JointControllerPtr GetJointController** ()

  Get a handle to the Controller for the joints in this model.
- unsigned int **GetJointCount** () const

  Get the number of joints.
- const **Joint_V** & **GetJoints** () const

  Get the joints.
- **LinkPtr GetLink** (const std::string &_name="canonical") const

  Get a link by name.
- **LinkPtr GetLinkById** (unsigned int _id) const

  This is an internal function
- **Link_V GetLinks** () const

  *Construct and return a vector of **Link** (p. 439)'s in this model Note this constructs the vector of **Link** (p. 439)'s on the fly, could be costly.*
- unsigned int **GetPluginCount** () const

  Get the number of plugins this model has.
- virtual **math::Vector3 GetRelativeAngularAccel** () const

  Get the angular acceleration of the entity.
- virtual **math::Vector3 GetRelativeAngularVel** () const

  Get the angular velocity of the entity.
- virtual **math::Vector3 GetRelativeLinearAccel** () const

  Get the linear acceleration of the entity.
- virtual **math::Vector3 GetRelativeLinearVel** () const

  Get the linear velocity of the entity.
- virtual const **sdf::ElementPtr GetSDF** ()

  Get the SDF values for the model.
- unsigned int **GetSensorCount** () const

  Get the number of sensors attached to this model.
- virtual **math::Vector3 GetWorldAngularAccel** () const

  Get the angular acceleration of the entity in the world frame.
- virtual **math::Vector3 GetWorldAngularVel** () const

  Get the angular velocity of the entity in the world frame.
- virtual **math::Vector3 GetWorldLinearAccel** () const

  Get the linear acceleration of the entity in the world frame.
- virtual **math::Vector3 GetWorldLinearVel** () const

  Get the linear velocity of the entity in the world frame.
- virtual void **Init** ()

  Initialize the model.
- void **Load** (**sdf::ElementPtr** _sdf)

  Load the model.
- void **LoadJoints** ()

  Load all the joints.
- void **LoadPlugins** ()

  Load all plugins.
- void **ProcessMsg** (const msgs::Model &_msg)

  Update parameters from a model message.

- virtual void **RemoveChild** (**EntityPtr** _child)

  Remove a child.
- void **Reset** ()

  Reset the model.
- void **SetAngularAccel** (const **math::Vector3** &_vel)

  Set the angular acceleration of the model, and all its links.
- void **SetAngularVel** (const **math::Vector3** &_vel)

  Set the angular velocity of the model, and all its links.
- void **SetAutoDisable** (bool _disable)

  Allow the model the auto disable.
- void **SetCollideMode** (const std::string &_mode)

  *This is not implemented in **Link** (p. 439), which means this function doesn't do anything.*
- void **SetEnabled** (bool _enabled)

  Enable all the links in all the models.
- void **SetGravityMode** (const bool &_value)

  Set the gravity mode of the model.
- void **SetJointAnimation** (const std::map< std::string, **common::NumericAnimationPtr** > _anim, boost::function< void()> _onComplete=**NULL**)

  ***Joint** (p. 401) Animation.*
- void **SetJointPosition** (const std::string &_jointName, double _position, int _index=0)

  *Set the positions of a **Joint** (p. 401) by name.*
- void **SetJointPositions** (const std::map< std::string, double > &_jointPositions)

  Set the positions of a set of joints.
- void **SetLaserRetro** (const float _retro)

  Set the laser retro reflectiveness of the model.
- void **SetLinearAccel** (const **math::Vector3** &_vel)

  Set the linear acceleration of the model, and all its links.
- void **SetLinearVel** (const **math::Vector3** &_vel)

  Set the linear velocity of the model, and all its links.
- void **SetLinkWorldPose** (const **math::Pose** &_pose, std::string _linkName)

  *Set the Pose of the entire **Model** (p. 516) by specifying desired Pose of a **Link** (p. 439) within the **Model** (p. 516).*
- void **SetLinkWorldPose** (const **math::Pose** &_pose, const **LinkPtr** &_link)

  *Set the Pose of the entire **Model** (p. 516) by specifying desired Pose of a **Link** (p. 439) within the **Model** (p. 516).*
- void **SetState** (const **ModelState** &_state)

  Set the current model state.
- virtual void **StopAnimation** ()

  Stop the current animations.
- void **Update** ()

  Update the model.
- virtual void **UpdateParameters** (**sdf::ElementPtr** _sdf)

  Update the parameters using new sdf values.

Protected Member Functions

- virtual void **OnPoseChange** ()

  Callback when the pose of the model has been changed.

Protected Attributes

- std::vector< **ModelPtr** > **attachedModels**

  used by ***Model::AttachStaticModel*** *(p. 520)*
- std::vector< **math::Pose** > **attachedModelsOffset**

  used by ***Model::AttachStaticModel*** *(p. 520)*

Additional Inherited Members

10.89.1 Detailed Description

A model is a collection of links, joints, and plugins.

10.89.2 Constructor & Destructor Documentation

10.89.2.1 gazebo::physics::Model::Model (BasePtr *_parent*) [explicit]

Constructor.

Parameters

in	*_parent*	Parent object.

10.89.2.2 virtual gazebo::physics::Model::~Model ( ) [virtual]

Destructor.

10.89.3 Member Function Documentation

10.89.3.1 void gazebo::physics::Model::AttachStaticModel (ModelPtr & *_model,* math::Pose *_offset*)

Attach a static model to this model.

This function takes as input a static **Model** (p. 516), which is a **Model** (p. 516) that has been marked as static (no physics simulation), and attaches it to this **Model** (p. 516) with a given offset.

This function is useful when you want to simulate a grasp of a static object, or move a static object around using a dynamic model.

If you are in doubt, do not use this function.

Parameters

in	*_model*	Pointer to the static model.
in	*_offset*	Offset, relative to this **Model** (p. 516), to place _model.

10.89.3.2 void gazebo::physics::Model::DetachStaticModel (const std::string & *_model*)

Detach a static model from this model.

Parameters

in	*_model*	Name of an attached static model to remove.

See Also

Model::AttachStaticModel (p. 520).

10.89.3.3 void gazebo::physics::Model::FillMsg (msgs::Model & *_msg*)

Fill a model message.

Parameters

in	*_msg*	Message to fill using this model's data.

10.89.3.4 virtual void gazebo::physics::Model::Fini ( ) [virtual]

Finalize the model.

Reimplemented from **gazebo::physics::Entity** (p. 291).

Reimplemented in **gazebo::physics::Actor** (p. 118).

10.89.3.5 bool gazebo::physics::Model::GetAutoDisable ( ) const

Return the value of the SDF <allow_auto_disable> element.

Returns

True if auto disable is allowed for this model.

10.89.3.6 virtual math::Box gazebo::physics::Model::GetBoundingBox ( ) const [virtual]

Get the size of the bounding box.

Returns

The bounding box.

Reimplemented from **gazebo::physics::Entity** (p. 291).

10.89.3.7 JointPtr gazebo::physics::Model::GetJoint (const std::string & *name*)

Get a joint.

Parameters

name	The name of the joint, specified in the world file

Returns

Pointer to the joint

10.89.3.8 JointControllerPtr gazebo::physics::Model::GetJointController ( ) [inline]

Get a handle to the Controller for the joints in this model.

Returns

A handle to the Controller for the joints in this model.

10.89.3.9 unsigned int gazebo::physics::Model::GetJointCount ( ) const

Get the number of joints.

Returns

Get the number of joints.

10.89.3.10 const Joint_V& gazebo::physics::Model::GetJoints ( ) const

Get the joints.

Returns

Vector of joints.

10.89.3.11 LinkPtr gazebo::physics::Model::GetLink (const std::string & *_name* = "canonical") const

Get a link by name.

Parameters

in	*_name*	Name of the link to get.

Returns

Pointer to the link, NULL if the name is invalid.

10.89.3.12 LinkPtr gazebo::physics::Model::GetLinkById (unsigned int *_id*) const

This is an internal function

Get a link by id.

Returns

Pointer to the link, NULL if the id is invalid.

10.89.3.13 Link_V gazebo::physics::Model::GetLinks ( ) const

Construct and return a vector of **Link** (p. 439)'s in this model Note this constructs the vector of **Link** (p. 439)'s on the fly, could be costly.

Returns

a vector of **Link** (p. 439)'s in this model

10.89.3.14 unsigned int gazebo::physics::Model::GetPluginCount ( ) const

Get the number of plugins this model has.

Returns

Number of plugins associated with this model.

10.89.3.15 virtual math::Vector3 gazebo::physics::Model::GetRelativeAngularAccel ( ) const [virtual]

Get the angular acceleration of the entity.

Returns

math::Vector3 (p. 890), set to 0, 0, 0 if the model has no body.

Reimplemented from **gazebo::physics::Entity** (p. 293).

10.89.3.16 virtual math::Vector3 gazebo::physics::Model::GetRelativeAngularVel ( ) const [virtual]

Get the angular velocity of the entity.

Returns

math::Vector3 (p. 890), set to 0, 0, 0 if the model has no body.

Reimplemented from **gazebo::physics::Entity** (p. 293).

10.89.3.17 virtual math::Vector3 gazebo::physics::Model::GetRelativeLinearAccel ( ) const [virtual]

Get the linear acceleration of the entity.

Returns

math::Vector3 (p. 890), set to 0, 0, 0 if the model has no body.

Reimplemented from **gazebo::physics::Entity** (p. 293).

10.89.3.18 virtual math::Vector3 gazebo::physics::Model::GetRelativeLinearVel ( ) const [virtual]

Get the linear velocity of the entity.

Returns

math::Vector3 (p. 890), set to 0, 0, 0 if the model has no body.

Reimplemented from **gazebo::physics::Entity** (p. 294).

10.89.3.19 virtual const sdf::ElementPtr gazebo::physics::Model::GetSDF ( ) [virtual]

Get the SDF values for the model.

Returns

The SDF value for this model.

Reimplemented from **gazebo::physics::Base** (p. 148).

Reimplemented in **gazebo::physics::Actor** (p. 118).

10.89.3.20 unsigned int gazebo::physics::Model::GetSensorCount ( ) const

Get the number of sensors attached to this model.

This will count all the sensors attached to all the links.

Returns

Number of sensors.

10.89.3.21 virtual math::Vector3 gazebo::physics::Model::GetWorldAngularAccel ( ) const [virtual]

Get the angular acceleration of the entity in the world frame.

Returns

math::Vector3 (p. 890), set to 0, 0, 0 if the model has no body.

Reimplemented from **gazebo::physics::Entity** (p. 294).

10.89.3.22 virtual math::Vector3 gazebo::physics::Model::GetWorldAngularVel ( ) const [virtual]

Get the angular velocity of the entity in the world frame.

Returns

math::Vector3 (p. 890), set to 0, 0, 0 if the model has no body.

Reimplemented from **gazebo::physics::Entity** (p. 294).

10.89.3.23 virtual math::Vector3 gazebo::physics::Model::GetWorldLinearAccel ( ) const [virtual]

Get the linear acceleration of the entity in the world frame.

Returns

math::Vector3 (p. 890), set to 0, 0, 0 if the model has no body.

Reimplemented from **gazebo::physics::Entity** (p. 294).

10.89.3.24 virtual math::Vector3 gazebo::physics::Model::GetWorldLinearVel ( ) const [virtual]

Get the linear velocity of the entity in the world frame.

Returns

math::Vector3 (p. 890), set to 0, 0, 0 if the model has no body.

Reimplemented from **gazebo::physics::Entity** (p. 295).

10.89.3.25 virtual void gazebo::physics::Model::Init ( ) [virtual]

Initialize the model.

Reimplemented from **gazebo::physics::Base** (p. 149).

Reimplemented in **gazebo::physics::Actor** (p. 118).

10.89.3.26 void gazebo::physics::Model::Load (sdf::ElementPtr *_sdf*) [virtual]

Load the model.

Parameters

in	*_sdf*	SDF parameters to load from.

Reimplemented from **gazebo::physics::Entity** (p. 295).

10.89.3.27 void gazebo::physics::Model::LoadJoints ( )

Load all the joints.

10.89.3.28 void gazebo::physics::Model::LoadPlugins ( )

Load all plugins.

Load all plugins specified in the SDF for the model.

10.89.3.29 virtual void gazebo::physics::Model::OnPoseChange ( ) [protected], [virtual]

Callback when the pose of the model has been changed.

Implements **gazebo::physics::Entity** (p. 295).

10.89.3.30 void gazebo::physics::Model::ProcessMsg (const msgs::Model & *_msg*)

Update parameters from a model message.

Parameters

in	*_msg*	Message to process.

10.89.3.31 virtual void gazebo::physics::Model::RemoveChild (EntityPtr *_child*) [virtual]

Remove a child.

Parameters

in	*_child*	Remove a child entity.

10.89.3.32 void gazebo::physics::Model::Reset ( ) [virtual]

Reset the model.

Reimplemented from **gazebo::physics::Entity** (p. 296).

10.89.3.33 void gazebo::physics::Model::SetAngularAccel (const math::Vector3 & *_vel*)

Set the angular acceleration of the model, and all its links.

Parameters

in	*_vel*	The new angular acceleration

10.89.3.34 void gazebo::physics::Model::SetAngularVel (const math::Vector3 & *_vel*)

Set the angular velocity of the model, and all its links.

Parameters

in	*_vel*	The new angular velocity.

10.89.3.35 void gazebo::physics::Model::SetAutoDisable (bool *_disable*)

Allow the model the auto disable.

This is ignored if the model has joints.

Parameters

in	*_disable*	If true, the model is allowed to auto disable.

10.89.3.36 void gazebo::physics::Model::SetCollideMode (const std::string & _mode)

This is not implemented in **Link** (p. 439), which means this function doesn't do anything.

Set the collide mode of the model.

Parameters

in	*_mode*	The collision mode

10.89.3.37 void gazebo::physics::Model::SetEnabled (bool _enabled)

Enable all the links in all the models.

Parameters

in	*_enabled*	True to enable all the links.

10.89.3.38 void gazebo::physics::Model::SetGravityMode (const bool & _value)

Set the gravity mode of the model.

Parameters

in	*_value*	False to turn gravity on for the model.

10.89.3.39 void gazebo::physics::Model::SetJointAnimation (const std::map< std::string, common::NumericAnimationPtr > _anim, boost::function< void() > _onComplete = NULL)

Joint (p. 401) Animation.

Parameters

in	*_anim*	Map of joint names to their position animation.
in	*_onComplete*	Callback function for when the animation completes.

10.89.3.40 void gazebo::physics::Model::SetJointPosition (const std::string & _jointName, double _position, int _index = 0)

Set the positions of a **Joint** (p. 401) by name.

See Also

JointController::SetJointPosition (p. 420)

Parameters

in	*_jointName*	Name of the joint to set.
in	*_position*	Position to set the joint to.

10.89.3.41 void gazebo::physics::Model::SetJointPositions (const std::map< std::string, double > & *_jointPositions*)

Set the positions of a set of joints.

See Also

JointController::SetJointPositions (p. 420).

Parameters

in	*_jointPositions*	Map of joint names to their positions.

10.89.3.42 void gazebo::physics::Model::SetLaserRetro (const float *_retro*)

Set the laser retro reflectiveness of the model.

Parameters

in	*_retro*	Retro reflectance value.

10.89.3.43 void gazebo::physics::Model::SetLinearAccel (const **math::Vector3** & *_vel*)

Set the linear acceleration of the model, and all its links.

Parameters

in	*_vel*	The new linear acceleration.

10.89.3.44 void gazebo::physics::Model::SetLinearVel (const **math::Vector3** & *_vel*)

Set the linear velocity of the model, and all its links.

Parameters

in	*_vel*	The new linear velocity.

10.89.3.45 void gazebo::physics::Model::SetLinkWorldPose (const **math::Pose** & *_pose*, std::string *_linkName*)

Set the Pose of the entire **Model** (p. 516) by specifying desired Pose of a **Link** (p. 439) within the **Model** (p. 516).

Doing so, keeps the configuration of the **Model** (p. 516) unchanged, i.e. all **Joint** (p. 401) angles are unchanged.

Parameters

in	*_pose*	Pose to set the link to.
in	*_linkName*	Name of the link to set.

10.89.3.46 void gazebo::physics::Model::SetLinkWorldPose (const math::Pose & _pose, const LinkPtr & _link)

Set the Pose of the entire **Model** (p. 516) by specifying desired Pose of a **Link** (p. 439) within the **Model** (p. 516).

Doing so, keeps the configuration of the **Model** (p. 516) unchanged, i.e. all **Joint** (p. 401) angles are unchanged.

Parameters

in	*_pose*	Pose to set the link to.
in	*_link*	Pointer to the link to set.

10.89.3.47 void gazebo::physics::Model::SetState (const ModelState & _state)

Set the current model state.

Parameters

in	*_state*	**State** (p. 797) to set the model to.

10.89.3.48 virtual void gazebo::physics::Model::StopAnimation ( ) [virtual]

Stop the current animations.

Reimplemented from **gazebo::physics::Entity** (p. 298).

10.89.3.49 void gazebo::physics::Model::Update ( ) [virtual]

Update the model.

Reimplemented from **gazebo::physics::Base** (p. 152).

10.89.3.50 virtual void gazebo::physics::Model::UpdateParameters (sdf::ElementPtr _sdf) [virtual]

Update the parameters using new sdf values.

Parameters

in	*_sdf*	SDF values to update from.

Reimplemented from **gazebo::physics::Entity** (p. 298).

Reimplemented in **gazebo::physics::Actor** (p. 119).

10.89.4 Member Data Documentation

10.89.4.1 std::vector<ModelPtr> gazebo::physics::Model::attachedModels [protected]

used by **Model::AttachStaticModel** (p. 520)

10.89.4.2 std::vector<math::Pose> gazebo::physics::Model::attachedModelsOffset [protected]

used by **Model::AttachStaticModel** (p. 520)

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

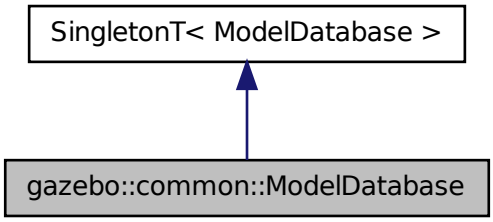
- **Model.hh**

10.90 gazebo::common::ModelDatabase Class Reference

Connects to model database, and has utility functions to find models.

`#include <common/common.hh>`

Inheritance diagram for gazebo::common::ModelDatabase:



Public Member Functions

- void **DownloadDependencies** (const std::string &_path)

  Download all dependencies for a give model path.
- void **Fini** ()

  Finalize the model database.
- std::string **GetDBConfig** (const std::string &_uri)

  Return the database.config file as a string.
- std::string **GetManifest** (const std::string &_uri) **GAZEBO_DEPRECATED**(1.5)

  Deprecated.
- std::string **GetModelConfig** (const std::string &_uri)

  Return the model.config file as a string.
- std::string **GetModelFile** (const std::string &_uri)

  Get a model's SDF file based on a URI.
- std::string **GetModelName** (const std::string &_uri)

  Get the name of a model based on a URI.
- std::string **GetModelPath** (const std::string &_uri, bool _forceDownload=false)

  Get the local path to a model.

- std::map< std::string, std::string > **GetModels** ()

  Returns the dictionary of all the model names.
- void **GetModels** (boost::function< void(const std::map< std::string, std::string > &)> _func)

  Get the dictionary of all model names via a callback.
- std::string **GetURI** ()

  Returns the the global model database URI.
- bool **HasModel** (const std::string &_modelName)

  Returns true if the model exists on the database.
- void **Start** (bool _fetchImmediately=false)

  Start the model database.

Additional Inherited Members

10.90.1 Detailed Description

Connects to model database, and has utility functions to find models.

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

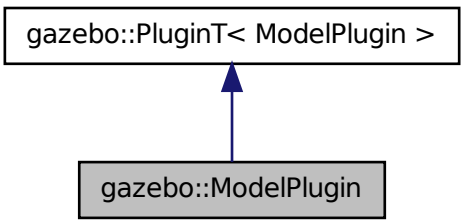
- **ModelDatabase.hh**

10.91 gazebo::ModelPlugin Class Reference

A plugin with access to **physics::Model** (p. 516).

`#include <Plugin.hh>`

Inheritance diagram for gazebo::ModelPlugin:



Public Member Functions

- **ModelPlugin** ()

  Constructor.
- virtual ~**ModelPlugin** ()

Destructor.

- virtual void **Init** ()

  Override this method for custom plugin initialization behavior.

- virtual void **Load** (**physics::ModelPtr** _model, **sdf::ElementPtr** _sdf)=0

  Load function.

- virtual void **Reset** ()

  Override this method for custom plugin reset behavior.

Additional Inherited Members

10.91.1 Detailed Description

A plugin with access to **physics::Model** (p. 516).

See `reference`.

10.91.2 Constructor & Destructor Documentation

10.91.2.1 gazebo::ModelPlugin::ModelPlugin ( ) [inline]

Constructor.

References gazebo::MODEL_PLUGIN, and gazebo::PluginT< ModelPlugin >::type.

10.91.2.2 virtual gazebo::ModelPlugin::~ModelPlugin ( ) [inline], [virtual]

Destructor.

10.91.3 Member Function Documentation

10.91.3.1 virtual void gazebo::ModelPlugin::Init ( ) [inline], [virtual]

Override this method for custom plugin initialization behavior.

10.91.3.2 virtual void gazebo::ModelPlugin::Load (**physics::ModelPtr** *_model,* **sdf::ElementPtr** *_sdf*) [pure virtual]

Load function.

Called when a Plugin is first created, and after the World has been loaded. This function should not be blocking.

Parameters

in	*_model*	Pointer to the Model
in	*_sdf*	Pointer to the SDF element of the plugin.

10.91.3.3 virtual void gazebo::ModelPlugin::Reset ( ) [inline], [virtual]

Override this method for custom plugin reset behavior.

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

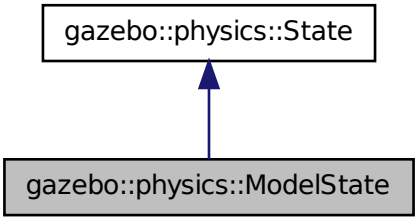
- **common/Plugin.hh**

10.92 gazebo::physics::ModelState Class Reference

Store state information of a **physics::Model** (p. 516) object.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::ModelState:



Public Member Functions

- **ModelState** ()
  Default constructor.
- **ModelState** (const **ModelPtr** _model, const **common::Time** &_realTime, const **common::Time** &_simTime)
  Constructor.
- **ModelState** (const **ModelPtr** _model)
  Constructor.
- **ModelState** (const **sdf::ElementPtr** _sdf)
  Constructor.
- virtual ~**ModelState** ()
  Destructor.
- void **FillSDF** (**sdf::ElementPtr** _sdf)
  Populate a state SDF element with data from the object.
- **JointState GetJointState** (unsigned int _index) const
  *Get a **Joint** (p. 401) state.*
- **JointState GetJointState** (const std::string &_jointName) const
  *Get a **Joint** (p. 401) state by **Joint** (p. 401) name.*
- unsigned int **GetJointStateCount** () const
  Get the number of joint states.
- **JointState_M GetJointStates** (const boost::regex &_regex) const
  Get joint states based on a regular expression.

- const **JointState_M** & **GetJointStates** () const

  Get the joint states.
- **LinkState GetLinkState** (unsigned int _index) const **GAZEBO_DEPRECATED**(1.7)

  Get a link state.
- **LinkState GetLinkState** (const std::string &_linkName) const

  *Get a link state by **Link** (p. 439) name.*
- unsigned int **GetLinkStateCount** () const

  Get the number of link states.
- **LinkState_M GetLinkStates** (const boost::regex &_regex) const

  Get link states based on a regular expression.
- const **LinkState_M** & **GetLinkStates** () const

  Get the link states.
- const **math::Pose** & **GetPose** () const

  Get the stored model pose.
- bool **HasJointState** (const std::string &_jointName) const

  Return true if there is a joint with the specified name.
- bool **HasLinkState** (const std::string &_linkName) const

  Return true if there is a link with the specified name.
- bool **IsZero** () const

  Return true if the values in the state are zero.
- void **Load** (const **ModelPtr** _model, const **common::Time** &_realTime, const **common::Time** &_simTime)

  *Load state from **Model** (p. 516) pointer.*
- virtual void **Load** (const **sdf::ElementPtr** _elem)

  Load state from SDF element.
- **ModelState operator+** (const **ModelState** &_state) const

  Addition operator.
- **ModelState operator-** (const **ModelState** &_state) const

  Subtraction operator.
- **ModelState** & **operator=** (const **ModelState** &_state)

  Assignment operator.
- virtual void **SetRealTime** (const **common::Time** &_time)

  Set the real time when this state was generated.
- virtual void **SetSimTime** (const **common::Time** &_time)

  Set the sim time when this state was generated.
- virtual void **SetWallTime** (const **common::Time** &_time)

  Set the wall time when this state was generated.

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::physics::ModelState** &_state)

  Stream insertion operator.

Additional Inherited Members

10.92.1 Detailed Description

Store state information of a **physics::Model** (p. 516) object.

This class captures the entire state of a **Model** (p. 516) at one specific time during a simulation run.

State (p. 797) of a **Model** (p. 516) includes the state of all its child Links and Joints.

10.92.2 Constructor & Destructor Documentation

10.92.2.1 gazebo::physics::ModelState::ModelState ( )

Default constructor.

10.92.2.2 gazebo::physics::ModelState::ModelState (const ModelPtr *_model,* const common::Time & *_realTime,* const common::Time & *_simTime*)

Constructor.

Build a **ModelState** (p. 533) from an existing **Model** (p. 516).

Parameters

in	*_model*	Pointer to the model from which to gather state info.
in	*_realTime*	Real time stamp.
in	*_simTime*	Sim time stamp.

10.92.2.3 gazebo::physics::ModelState::ModelState (const ModelPtr *_model*) [explicit]

Constructor.

Build a **ModelState** (p. 533) from an existing **Model** (p. 516).

Parameters

in	*_model*	Pointer to the model from which to gather state info.

10.92.2.4 gazebo::physics::ModelState::ModelState (const sdf::ElementPtr *_sdf*) [explicit]

Constructor.

Build a **ModelState** (p. 533) from SDF data

Parameters

in	*_sdf*	SDF data to load a model state from.

10.92.2.5 virtual gazebo::physics::ModelState::~ModelState ( ) [virtual]

Destructor.

10.92.3 Member Function Documentation

10.92.3.1 void gazebo::physics::ModelState::FillSDF (sdf::ElementPtr _sdf)

Populate a state SDF element with data from the object.

Parameters

out	*_sdf*	SDF element to populate.

10.92.3.2 JointState gazebo::physics::ModelState::GetJointState (unsigned int _index) const

Get a **Joint** (p. 401) state.

Return a **JointState** (p. 420) based on a index, where index is between 0...**ModelState::GetJointStateCount()** (p. 537).

Parameters

in	*_index*	Index of a **JointState** (p. 420).

Returns

State (p. 797) of a **Joint** (p. 401).

Exceptions

common::Exception *(p. 325)*	When _index is out of range.

10.92.3.3 JointState gazebo::physics::ModelState::GetJointState (const std::string & _jointName) const

Get a **Joint** (p. 401) state by **Joint** (p. 401) name.

Searches through all JointStates. Returns the **JointState** (p. 420) with the matching name, if any.

Parameters

in	*_jointName*	Name of the **JointState** (p. 420).

Returns

State (p. 797) of the **Joint** (p. 401).

Exceptions

common::Exception *(p. 325)*	When _jointName is invalid.

10.92.3.4 unsigned int gazebo::physics::ModelState::GetJointStateCount ( ) const

Get the number of joint states.

Returns the number of JointStates recorded.

Returns

Number of JointStates.

10.92.3.5 JointState_M gazebo::physics::ModelState::GetJointStates (const boost::regex & *_regex*) const

Get joint states based on a regular expression.

Parameters

in	*_regex*	The regular expression.

Returns

List of joint states whose names match the regular expression.

10.92.3.6 const JointState_M& gazebo::physics::ModelState::GetJointStates ( ) const

Get the joint states.

Returns

A map of joint states.

10.92.3.7 LinkState gazebo::physics::ModelState::GetLinkState (unsigned int *_index*) const

Get a link state.

Get a **Link** (p. 439) **State** (p. 797) based on an index, where index is in the range of 0...**ModelState::GetLinkStateCount** (p. 538)

Parameters

in	*_index*	Index of the **LinkState** (p. 459)

Returns

State (p. 797) of the **Link** (p. 439).

Exceptions

common::Exception *(p. 325)*	When _index is out of range.

10.92.3.8 LinkState gazebo::physics::ModelState::GetLinkState (const std::string & _linkName) const

Get a link state by **Link** (p. 439) name.

Searches through all LinkStates. Returns the **LinkState** (p. 459) with the matching name, if any.

Parameters

in	*_linkName*	Name of the **LinkState** (p. 459)

Returns

State (p. 797) of the **Link** (p. 439).

Exceptions

common::Exception *(p. 325)*	When _linkName is invalid.

10.92.3.9 unsigned int gazebo::physics::ModelState::GetLinkStateCount ( ) const

Get the number of link states.

This returns the number of Links recorded.

Returns

Number of **LinkState** (p. 459) recorded.

10.92.3.10 LinkState_M gazebo::physics::ModelState::GetLinkStates (const boost::regex & _regex) const

Get link states based on a regular expression.

Parameters

in	*_regex*	The regular expression.

Returns

List of link states whose names match the regular expression.

10.92.3.11 const LinkState_M& gazebo::physics::ModelState::GetLinkStates ( ) const

Get the link states.

Returns

A map of link states.

10.92.3.12 const math::Pose& gazebo::physics::ModelState::GetPose ( ) const

Get the stored model pose.

Returns

The **math::Pose** (p. 626) of the **Model** (p. 516).

10.92.3.13 bool gazebo::physics::ModelState::HasJointState (const std::string & *_jointName*) const

Return true if there is a joint with the specified name.

Parameters

in	*_jointName*	Name of the Jointtate.

Returns

True if the joint exists in the model.

10.92.3.14 bool gazebo::physics::ModelState::HasLinkState (const std::string & *_linkName*) const

Return true if there is a link with the specified name.

Parameters

in	*_linkName*	Name of the **LinkState** (p. 459).

Returns

True if the link exists in the model.

10.92.3.15 bool gazebo::physics::ModelState::IsZero ( ) const

Return true if the values in the state are zero.

Returns

True if the values in the state are zero.

10.92.3.16 void gazebo::physics::ModelState::Load (const ModelPtr *_model,* const common::Time & *_realTime,* const common::Time & *_simTime*)

Load state from **Model** (p. 516) pointer.

Build a **ModelState** (p. 533) from an existing **Model** (p. 516).

Parameters

in	*_model*	Pointer to the model from which to gather state info.
in	*_realTime*	Real time stamp.
in	*_simTime*	Sim time stamp.

10.92.3.17 virtual void gazebo::physics::ModelState::Load (const sdf::ElementPtr _elem) [virtual]

Load state from SDF element.

Load **ModelState** (p. 533) information from stored data in and SDF::Element

Parameters

in	*_elem*	Pointer to the SDF::Element containing state info.

Reimplemented from **gazebo::physics::State** (p. 800).

10.92.3.18 ModelState gazebo::physics::ModelState::operator+ (const ModelState & _state) const

Addition operator.

Parameters

in	*_pt*	A state to substract.

Returns

The resulting state.

10.92.3.19 ModelState gazebo::physics::ModelState::operator- (const ModelState & _state) const

Subtraction operator.

Parameters

in	*_pt*	A state to substract.

Returns

The resulting state.

10.92.3.20 ModelState& gazebo::physics::ModelState::operator= (const ModelState & _state)

Assignment operator.

Parameters

in	*_state*	**State** (p. 797) value

Returns

this

10.92.3.21 virtual void gazebo::physics::ModelState::SetRealTime (const common::Time & _time) [virtual]

Set the real time when this state was generated.

Parameters

in	*_time*	Clock time since simulation was stated.

Reimplemented from **gazebo::physics::State** (p. 801).

10.92.3.22 virtual void gazebo::physics::ModelState::SetSimTime (const common::Time & *_time*) [virtual]

Set the sim time when this state was generated.

Parameters

in	*_time*	Simulation time when the data was recorded.

Reimplemented from **gazebo::physics::State** (p. 801).

10.92.3.23 virtual void gazebo::physics::ModelState::SetWallTime (const common::Time & *_time*) [virtual]

Set the wall time when this state was generated.

Parameters

in	*_time*	The absolute clock time when the **State** (p. 797) data was recorded.

Reimplemented from **gazebo::physics::State** (p. 801).

10.92.4 Friends And Related Function Documentation

10.92.4.1 std::ostream& operator<< (std::ostream & *_out,* const gazebo::physics::ModelState & *_state*) [friend]

Stream insertion operator.

Parameters

in	*_out*	output stream.
in	*_state*	**Model** (p. 516) state to output.

Returns

The stream.

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

- **ModelState.hh**

10.93 gazebo::common::MouseEvent Class Reference

Generic description of a mouse event.

`#include <common/common.hh>`

Public Types

- enum **Buttons** { **NO_BUTTON** = 0x0, **LEFT** = 0x1, **MIDDLE** = 0x2, **RIGHT** = 0x4 }

  Standard mouse buttons enumeration.
- enum **EventType** {
  NO_EVENT, **MOVE**, **PRESS**, **RELEASE**,
  SCROLL }

  Mouse event types enumeration.

Public Member Functions

- **MouseEvent** ()

  Constructor.

Public Attributes

- bool **alt**

  Alt key press flag.
- unsigned int **button**

  The button which caused the event.
- unsigned int **buttons**

  State of the buttons when the event was generated.
- bool **control**

  Control key press flag.
- bool **dragging**

  Flag for mouse drag motion.
- float **moveScale**

  Scaling factor.
- **math::Vector2i pos**

  Mouse pointer position on the screen.
- **math::Vector2i pressPos**

  Position of button press.
- **math::Vector2i prevPos**

  Previous position.
- **math::Vector2i scroll**

  Scroll position.
- bool **shift**

  Shift key press flag.
- **EventType type**

  Event type.

10.93.1 Detailed Description

Generic description of a mouse event.

10.93.2 Member Enumeration Documentation

10.93.2.1 enum gazebo::common::MouseEvent::Buttons

Standard mouse buttons enumeration.

Enumerator:

NO_BUTTON
LEFT
MIDDLE
RIGHT

10.93.2.2 enum gazebo::common::MouseEvent::EventType

Mouse event types enumeration.

Enumerator:

NO_EVENT
MOVE
PRESS
RELEASE
SCROLL

10.93.3 Constructor & Destructor Documentation

10.93.3.1 gazebo::common::MouseEvent::MouseEvent ( ) [inline]

Constructor.

10.93.4 Member Data Documentation

10.93.4.1 bool gazebo::common::MouseEvent::alt

Alt key press flag.

10.93.4.2 unsigned int gazebo::common::MouseEvent::button

The button which caused the event.

10.93.4.3 unsigned int gazebo::common::MouseEvent::buttons

State of the buttons when the event was generated.

10.93.4.4 bool gazebo::common::MouseEvent::control

Control key press flag.

10.93.4.5 bool gazebo::common::MouseEvent::dragging

Flag for mouse drag motion.

10.93.4.6 float gazebo::common::MouseEvent::moveScale

Scaling factor.

10.93.4.7 math::Vector2i gazebo::common::MouseEvent::pos

Mouse pointer position on the screen.

10.93.4.8 math::Vector2i gazebo::common::MouseEvent::pressPos

Position of button press.

10.93.4.9 math::Vector2i gazebo::common::MouseEvent::prevPos

Previous position.

10.93.4.10 math::Vector2i gazebo::common::MouseEvent::scroll

Scroll position.

10.93.4.11 bool gazebo::common::MouseEvent::shift

Shift key press flag.

10.93.4.12 EventType gazebo::common::MouseEvent::type

Event type.

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

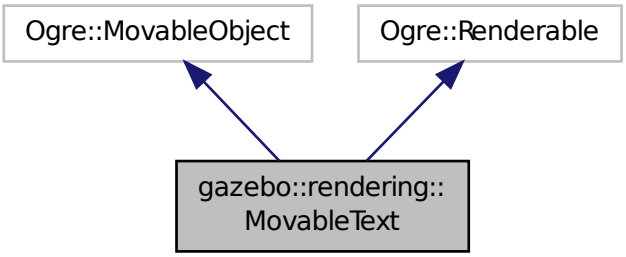
- **MouseEvent.hh**

10.94 gazebo::rendering::MovableText Class Reference

Movable text.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::MovableText:



Public Types

- enum **HorizAlign** { **H_LEFT**, **H_CENTER** }

  Horizontal alignment.
- enum **VertAlign** { **V_BELOW**, **V_ABOVE** }

  vertical alignment

Public Member Functions

- **MovableText** ()

  Constructor.
- virtual ~**MovableText** ()

  Destructor.
- **math::Box GetAABB** ()

  Get the axis aligned bounding box of the text.
- float **GetBaseline** () const

  Get the baseline height.
- float **GetCharHeight** () const

  Set the height of a characters return Height of the characters.
- const **common::Color** & **GetColor** () const

  Get the text color.
- const std::string & **GetFont** () const

  Get the font.
- bool **GetShowOnTop** () const

  True = text is displayed on top.
- float **GetSpaceWidth** () const

  Get the width of a space.
- const std::string & **GetText** () const

  Get the displayed text.
- void **Load** (const std::string &_name, const std::string &_text, const std::string &_fontName="Arial", float _charHeight=1.0, const **common::Color** &_color=**common::Color::White**)

Loads text and font info.

- void **SetBaseline** (float _height)

  Set the baseline height of the text.
- void **SetCharHeight** (float _height)

  Set the height of a character.
- void **SetColor** (const **common::Color** &_color)

  Set the text color.
- void **SetFontName** (const std::string &_font)

  Set the font.
- void **SetShowOnTop** (bool _show)

  True = text always is displayed ontop.
- void **SetSpaceWidth** (float _width)

  Set the width of a space.
- void **SetText** (const std::string &_text)

  Set the text to display.
- void **SetTextAlignment** (const **HorizAlign** &_hAlign, const **VertAlign** &_vAlign)

  Set the alignment of the text.
- void **Update** ()

  Update the text.
- virtual void **visitRenderables** (Ogre::Renderable::Visitor ∗_visitor, bool _debug=false)

Protected Member Functions

- void **_setupGeometry** ()
- void **_updateColors** ()
- float **getBoundingRadius** () const
- const Ogre::LightList & **getLights** (void) const
- const Ogre::MaterialPtr & **getMaterial** (void) const
- void **getRenderOperation** (Ogre::RenderOperation &op)
- float **getSquaredViewDepth** (const Ogre::Camera ∗cam) const
- void **getWorldTransforms** (Ogre::Matrix4 ∗xform) const

10.94.1 Detailed Description

Movable text.

10.94.2 Member Enumeration Documentation

10.94.2.1 enum gazebo::rendering::MovableText::HorizAlign

Horizontal alignment.

Enumerator:

H_LEFT Left alignment.

H_CENTER Center alignment.

10.94.2.2 enum gazebo::rendering::MovableText::VertAlign

vertical alignment

Enumerator:

V_BELOW Align below.

V_ABOVE Align above.

10.94.3 Constructor & Destructor Documentation

10.94.3.1 gazebo::rendering::MovableText::MovableText ( )

Constructor.

10.94.3.2 virtual gazebo::rendering::MovableText::~MovableText ( ) [virtual]

Destructor.

10.94.4 Member Function Documentation

10.94.4.1 void gazebo::rendering::MovableText::_setupGeometry ( ) [protected]

10.94.4.2 void gazebo::rendering::MovableText::_updateColors ( ) [protected]

10.94.4.3 math::Box gazebo::rendering::MovableText::GetAABB ( )

Get the axis aligned bounding box of the text.

Returns

The axis aligned bounding box.

10.94.4.4 float gazebo::rendering::MovableText::GetBaseline ( ) const

Get the baseline height.

Returns

Baseline height

10.94.4.5 float gazebo::rendering::MovableText::getBoundingRadius ( ) const [protected]

10.94.4.6 float gazebo::rendering::MovableText::GetCharHeight ( ) const

Set the height of a characters return Height of the characters.

10.94.4.7 const common::Color& gazebo::rendering::MovableText::GetColor ( ) const

Get the text color.

Returns

Texture color.

10.94.4.8 const std::string& gazebo::rendering::MovableText::GetFont ( ) const

Get the font.

Returns

The font name

10.94.4.9 const Ogre::LightList& gazebo::rendering::MovableText::getLights (void) const [protected]

10.94.4.10 const Ogre::MaterialPtr& gazebo::rendering::MovableText::getMaterial (void) const [protected]

10.94.4.11 void gazebo::rendering::MovableText::getRenderOperation (Ogre::RenderOperation & *op*) [protected]

10.94.4.12 bool gazebo::rendering::MovableText::GetShowOnTop ( ) const

True = text is displayed on top.

Returns

True if MovableText::SetShownOnTop(true) was called.

10.94.4.13 float gazebo::rendering::MovableText::GetSpaceWidth ( ) const

Get the width of a space.

Returns

Space width

10.94.4.14 float gazebo::rendering::MovableText::getSquaredViewDepth (const Ogre::Camera * *cam*) const [protected]

10.94.4.15 const std::string& gazebo::rendering::MovableText::GetText ( ) const

Get the displayed text.

Returns

The displayed text.

10.94.4.16 void gazebo::rendering::MovableText::getWorldTransforms (Ogre::Matrix4 * *xform*) const [protected]

10.94.4.17 void gazebo::rendering::MovableText::Load (const std::string & *_name*, const std::string & *_text*, const std::string & *_fontName* = "Arial", float *_charHeight* = 1.0, const common::Color & *_color* = common::Color::White)

Loads text and font info.

Parameters

in	*_name*	Name of the text object
in	*_text*	Text to render
in	*_fontName*	Font to use
in	*_charHeight*	Height of the characters
in	*_color*	Text color

10.94.4.18 void gazebo::rendering::MovableText::SetBaseline (float *_height*)

Set the baseline height of the text.

Parameters

in	*_height*	Baseline height

10.94.4.19 void gazebo::rendering::MovableText::SetCharHeight (float *_height*)

Set the height of a character.

Parameters

in	*_height*	Height of the characters.

10.94.4.20 void gazebo::rendering::MovableText::SetColor (const common::Color & *_color*)

Set the text color.

Parameters

in	*_color*	Text color.

10.94.4.21 void gazebo::rendering::MovableText::SetFontName (const std::string & *_font*)

Set the font.

Parameters

in	*_font*	Name of the font

10.94.4.22 void gazebo::rendering::MovableText::SetShowOnTop (bool *_show*)

True = text always is displayed ontop.

Parameters

in	*_show*	Set to true to render the text on top of all other drawables.

10.94.4.23 void gazebo::rendering::MovableText::SetSpaceWidth (float *_width*)

Set the width of a space.

Parameters

in	*_width*	space width

10.94.4.24 void gazebo::rendering::MovableText::SetText (const std::string & *_text*)

Set the text to display.

Parameters

in	*_text*	The text to display.

10.94.4.25 void gazebo::rendering::MovableText::SetTextAlignment (const HorizAlign & *_hAlign,* const VertAlign & *_vAlign*)

Set the alignment of the text.

Parameters

in	*_hAlign*	Horizontal alignment
in	*_vAlign*	Vertical alignment

10.94.4.26 void gazebo::rendering::MovableText::Update ( )

Update the text.

10.94.4.27 virtual void gazebo::rendering::MovableText::visitRenderables (Ogre::Renderable::Visitor * *_visitor,* bool *_debug* = `false`) [virtual]

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

- **MovableText.hh**

10.95 gazebo::msgs::MsgFactory Class Reference

A factory that generates protobuf message based on a string type.

`#include <msgs/msgs.hh>`

Static Public Member Functions

- static void **GetMsgTypes** (std::vector< std::string > &_types)

  Get all the message types.
- static boost::shared_ptr < google::protobuf::Message > **NewMsg** (const std::string &_msgType)

  Create a new instance of a message.
- static void **RegisterMsg** (const std::string &_msgType, **MsgFactoryFn** _factoryfn)

  Register a message.

10.95.1 Detailed Description

A factory that generates protobuf message based on a string type.

10.95.2 Member Function Documentation

10.95.2.1 static void gazebo::msgs::MsgFactory::GetMsgTypes (std::vector< std::string > & *_types*) [static]

Get all the message types.

Parameters

out	*_types*	Vector of strings of the message types.

10.95.2.2 static boost::shared_ptr<google::protobuf::Message> gazebo::msgs::MsgFactory::NewMsg (const std::string & *_msgType*) [static]

Create a new instance of a message.

Parameters

in	*_msgType*	Type of message to create.

Returns

Pointer to a google protobuf message. Null if the message type could not be handled.

10.95.2.3 static void gazebo::msgs::MsgFactory::RegisterMsg (const std::string & *_msgType,* **MsgFactoryFn** *_factoryfn*) [static]

Register a message.

Parameters

in	*_msgType*	Type of message to register.
in	*_factoryfn*	Function that generates the message.

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

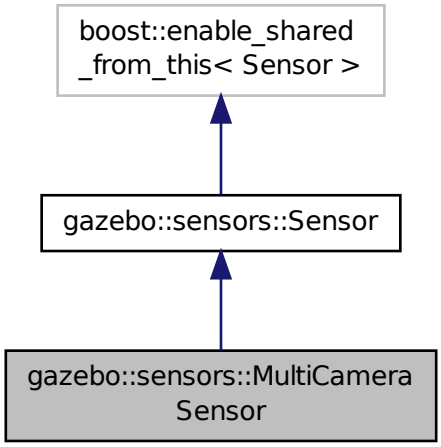
- **MsgFactory.hh**

10.96 gazebo::sensors::MultiCameraSensor Class Reference

Multiple camera sensor.

`#include <sensors/sensors.hh>`

Inheritance diagram for gazebo::sensors::MultiCameraSensor:



Public Member Functions

- **MultiCameraSensor** ()

  Constructor.
- virtual ∼**MultiCameraSensor** ()

  Destructor.
- **rendering::CameraPtr GetCamera** (unsigned int _index) const

  *Returns a pointer to a **rendering::Camera** (p. 166).*
- unsigned int **GetCameraCount** () const

  Get the number of cameras.
- const unsigned char ∗ **GetImageData** (unsigned int _index)

  Gets the raw image data from the sensor.
- unsigned int **GetImageHeight** (unsigned int _index) const

  Gets the height of the image in pixels.
- unsigned int **GetImageWidth** (unsigned int _index) const

  Gets the width of the image in pixels.

- virtual std::string **GetTopic** () const

  Returns the topic name as set in SDF.
- virtual void **Init** ()

  Initialize the sensor.
- virtual bool **IsActive** ()

  Returns true if sensor generation is active.
- virtual void **Load** (const std::string &_worldName)

  Load the sensor with default parameters.
- bool **SaveFrame** (const std::vector< std::string > &_filenames)

  Saves the camera image(s) to the disk.

Protected Member Functions

- virtual void **Fini** ()

  Finalize the sensor.
- virtual void **UpdateImpl** (bool _force)

  This gets overwritten by derived sensor types.

Additional Inherited Members

10.96.1 Detailed Description

Multiple camera sensor.

This sensor type can create one or more synchronized cameras.

10.96.2 Constructor & Destructor Documentation

10.96.2.1 gazebo::sensors::MultiCameraSensor::MultiCameraSensor ( )

Constructor.

10.96.2.2 virtual gazebo::sensors::MultiCameraSensor::~MultiCameraSensor ( ) [virtual]

Destructor.

10.96.3 Member Function Documentation

10.96.3.1 virtual void gazebo::sensors::MultiCameraSensor::Fini ( ) [protected], [virtual]

Finalize the sensor.

Reimplemented from **gazebo::sensors::Sensor** (p. 735).

10.96.3.2 rendering::CameraPtr gazebo::sensors::MultiCameraSensor::GetCamera (unsigned int _index) const

Returns a pointer to a **rendering::Camera** (p. 166).

Parameters

in	*_index*	Index of the camera to get

Returns

The Pointer to the camera sensor.

See Also

MultiCameraSensor::GetCameraCount (p. 554)

10.96.3.3 unsigned int gazebo::sensors::MultiCameraSensor::GetCameraCount ( ) const

Get the number of cameras.

Returns

The number of cameras.

10.96.3.4 const unsigned char* gazebo::sensors::MultiCameraSensor::GetImageData (unsigned int _index)

Gets the raw image data from the sensor.

Parameters

in	*_index*	Index of the camera

Returns

The pointer to the image data array.

See Also

MultiCameraSensor::GetCameraCount (p. 554)

10.96.3.5 unsigned int gazebo::sensors::MultiCameraSensor::GetImageHeight (unsigned int _index) const

Gets the height of the image in pixels.

Parameters

in	*_index*	Index of the camera

Returns

The image height in pixels.

See Also

MultiCameraSensor::GetCameraCount (p. 554)

10.96.3.6 unsigned int gazebo::sensors::MultiCameraSensor::GetImageWidth (unsigned int *_index*) const

Gets the width of the image in pixels.

Parameters

in	*_index*	Index of the camera

Returns

The image width in pixels.

See Also

MultiCameraSensor::GetCameraCount (p. 554)

10.96.3.7 virtual std::string gazebo::sensors::MultiCameraSensor::GetTopic ( ) const [virtual]

Returns the topic name as set in SDF.

Returns

Topic name.

Reimplemented from **gazebo::sensors::Sensor** (p. 737).

10.96.3.8 virtual void gazebo::sensors::MultiCameraSensor::Init ( ) [virtual]

Initialize the sensor.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.96.3.9 virtual bool gazebo::sensors::MultiCameraSensor::IsActive ( ) [virtual]

Returns true if sensor generation is active.

Returns

True if active, false if not.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.96.3.10 virtual void gazebo::sensors::MultiCameraSensor::Load (const std::string & *_worldName*) [virtual]

Load the sensor with default parameters.

Parameters

in	*_worldName*	Name of world to load from.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.96.3.11 bool gazebo::sensors::MultiCameraSensor::SaveFrame (const std::vector< std::string > & *_filenames*)

Saves the camera image(s) to the disk.

Parameters

in	*_filenames*	The name of the files for each camera.

Returns

True if successful, false if unsuccessful.

See Also

MultiCameraSensor::GetCameraCount (p. 554)

10.96.3.12 virtual void gazebo::sensors::MultiCameraSensor::UpdateImpl (bool) [protected], [virtual]

This gets overwritten by derived sensor types.

```
This function is called during Sensor::Update.
And in turn, Sensor::Update is called by
SensorManager::Update
```

Parameters

in	*_force*	True if update is forced, false if not

Reimplemented from **gazebo::sensors::Sensor** (p. 739).

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

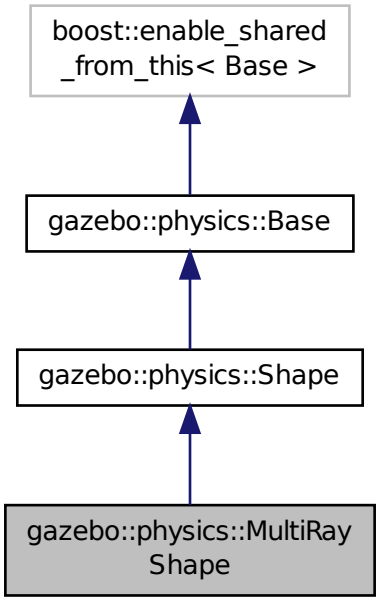
- **MultiCameraSensor.hh**

10.97 gazebo::physics::MultiRayShape Class Reference

Laser collision contains a set of ray-collisions, structured to simulate a laser range scanner.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::MultiRayShape:



Public Member Functions

- **MultiRayShape** (**CollisionPtr** _parent)

  Constructor.
- virtual ~**MultiRayShape** ()

  Destructor.
- template<typename T >
  event::ConnectionPtr ConnectNewLaserScans (T _subscriber)

  Connect a to the new laser scan signal.
- void **DisconnectNewLaserScans** (**event::ConnectionPtr** &_conn)

  Disconnect from the new laser scans signal.
- void **FillMsg** (msgs::Geometry &_msg)

  This function is not implemented.
- int **GetFiducial** (int _index)

  Get detected fiducial value for a ray.
- **math::Angle GetMaxAngle** () const

  Get the maximum angle.
- double **GetMaxRange** () const

  Get the maximum range.
- **math::Angle GetMinAngle** () const

  Get the minimum angle.

- double **GetMinRange** () const

  Get the minimum range.
- double **GetRange** (int _index)

  Get detected range for a ray.
- double **GetResRange** () const

  Get the range resolution.
- double **GetRetro** (int _index)

  Get detected retro (intensity) value for a ray.
- int **GetSampleCount** () const

  Get the horizontal sample count.
- double **GetScanResolution** () const

  Get the horizontal resolution.
- **math::Angle GetVerticalMaxAngle** () const

  Get the vertical max angle.
- **math::Angle GetVerticalMinAngle** () const

  Get the vertical min angle.
- int **GetVerticalSampleCount** () const

  Get the vertical sample count.
- double **GetVerticalScanResolution** () const

  Get the vertical range resolution.
- virtual void **Init** ()

  Init the shape.
- virtual void **ProcessMsg** (const msgs::Geometry &_msg)

  This function is not implemented.
- void **Update** ()

  Update the ray collisions.

Protected Member Functions

- virtual void **AddRay** (const **math::Vector3** &_start, const **math::Vector3** &_end)

  Add a ray to the collision.
- virtual void **UpdateRays** ()=0

  Physics engine specific method for updating the rays.

Protected Attributes

- **sdf::ElementPtr horzElem**

  Horizontal SDF element pointer.
- **event::EventT**< void()> **newLaserScans**

  New laser scans event.
- **math::Pose offset**

  Pose offset of all the rays.
- **sdf::ElementPtr rangeElem**

  Range SDF element pointer.
- **sdf::ElementPtr rayElem**

  Ray SDF element pointer.

- std::vector< **RayShapePtr** > **rays**

  Ray data.
- **sdf::ElementPtr scanElem**

  Scan SDF element pointer.
- **sdf::ElementPtr vertElem**

  Vertical SDF element pointer.

Additional Inherited Members

10.97.1 Detailed Description

Laser collision contains a set of ray-collisions, structured to simulate a laser range scanner.

10.97.2 Constructor & Destructor Documentation

10.97.2.1 gazebo::physics::MultiRayShape::MultiRayShape (CollisionPtr *_parent*) [explicit]

Constructor.

Parameters

in	*_parent*	Parent collision shape.

10.97.2.2 virtual gazebo::physics::MultiRayShape::~MultiRayShape ( ) [virtual]

Destructor.

10.97.3 Member Function Documentation

10.97.3.1 virtual void gazebo::physics::MultiRayShape::AddRay (const math::Vector3 & *_start*, const math::Vector3 & *_end*) [protected], [virtual]

Add a ray to the collision.

Parameters

in	*_start*	Start of the ray.
in	*_end*	End of the ray.

10.97.3.2 template<typename T > event::ConnectionPtr gazebo::physics::MultiRayShape::ConnectNewLaserScans (T *_subscriber*) [inline]

Connect a to the new laser scan signal.

Parameters

in	*_subscriber*	Callback function.

Returns

The connection, which must be kept in scope.

References gazebo::event::EventT< T >::Connect(), and newLaserScans.

10.97.3.3 void gazebo::physics::MultiRayShape::DisconnectNewLaserScans (event::ConnectionPtr & *_conn*) [inline]

Disconnect from the new laser scans signal.

Parameters

in	*_conn*	Connection to remove.

References gazebo::event::EventT< T >::Disconnect(), and newLaserScans.

10.97.3.4 void gazebo::physics::MultiRayShape::FillMsg (msgs::Geometry & *_msg*) [virtual]

This function is not implemented.

Fill a message with this shape's values.

Parameters

out	*_msg*	Message that contains the shape's values.

Implements **gazebo::physics::Shape** (p. 756).

10.97.3.5 int gazebo::physics::MultiRayShape::GetFiducial (int *_index*)

Get detected fiducial value for a ray.

Parameters

in	*_index*	Index of the ray.

Returns

Fiducial value for the ray.

10.97.3.6 math::Angle gazebo::physics::MultiRayShape::GetMaxAngle ( ) const

Get the maximum angle.

Returns

Maximum angle of ray scan.

10.97.3.7 double gazebo::physics::MultiRayShape::GetMaxRange ( ) const

Get the maximum range.

Returns

Maximum range of all the rays.

10.97.3.8 math::Angle gazebo::physics::MultiRayShape::GetMinAngle ( ) const

Get the minimum angle.

Returns

Minimum angle of ray scan.

10.97.3.9 double gazebo::physics::MultiRayShape::GetMinRange ( ) const

Get the minimum range.

Returns

Minimum range of all the rays.

10.97.3.10 double gazebo::physics::MultiRayShape::GetRange (int *_index*)

Get detected range for a ray.

Parameters

in	*_index*	Index of the ray.

Returns

Returns DBL_MAX for no detection.

10.97.3.11 double gazebo::physics::MultiRayShape::GetResRange ( ) const

Get the range resolution.

Returns

Range resolution of all the rays.

10.97.3.12 double gazebo::physics::MultiRayShape::GetRetro (int *_index*)

Get detected retro (intensity) value for a ray.

Parameters

in	*_index*	Index of the ray.

Returns

Retro value for the ray.

10.97.3.13 int gazebo::physics::MultiRayShape::GetSampleCount ( ) const

Get the horizontal sample count.

Returns

Horizontal sample count.

10.97.3.14 double gazebo::physics::MultiRayShape::GetScanResolution ( ) const

Get the horizontal resolution.

Returns

Horizontal resolution.

10.97.3.15 math::Angle gazebo::physics::MultiRayShape::GetVerticalMaxAngle ( ) const

Get the vertical max angle.

Returns

Verticam max angle.

10.97.3.16 math::Angle gazebo::physics::MultiRayShape::GetVerticalMinAngle ( ) const

Get the vertical min angle.

Returns

Vertical min angle.

10.97.3.17 int gazebo::physics::MultiRayShape::GetVerticalSampleCount ( ) const

Get the vertical sample count.

Returns

Verical sample count.

10.97.3.18 double gazebo::physics::MultiRayShape::GetVerticalScanResolution ( ) const

Get the vertical range resolution.

Returns

Vertical range resolution.

10.97.3.19 virtual void gazebo::physics::MultiRayShape::Init ( ) [virtual]

Init the shape.

Implements **gazebo::physics::Shape** (p. 756).

10.97.3.20 virtual void gazebo::physics::MultiRayShape::ProcessMsg (const msgs::Geometry & *_msg*) [virtual]

This function is not implemented.

Update the ray based on a message.

Parameters

in	*_msg*	Message to update from.

Implements **gazebo::physics::Shape** (p. 756).

10.97.3.21 void gazebo::physics::MultiRayShape::Update ( ) [virtual]

Update the ray collisions.

Reimplemented from **gazebo::physics::Base** (p. 152).

10.97.3.22 virtual void gazebo::physics::MultiRayShape::UpdateRays ( ) [protected], [pure virtual]

Physics engine specific method for updating the rays.

10.97.4 Member Data Documentation

10.97.4.1 sdf::ElementPtr gazebo::physics::MultiRayShape::horzElem [protected]

Horizontal SDF element pointer.

10.97.4.2 event::EventT<void()> gazebo::physics::MultiRayShape::newLaserScans [protected]

New laser scans event.

Referenced by ConnectNewLaserScans(), and DisconnectNewLaserScans().

10.97.4.3 math::Pose gazebo::physics::MultiRayShape::offset [protected]

Pose offset of all the rays.

10.97.4.4 sdf::ElementPtr gazebo::physics::MultiRayShape::rangeElem [protected]

Range SDF element pointer.

10.97.4.5 sdf::ElementPtr gazebo::physics::MultiRayShape::rayElem [protected]

Ray SDF element pointer.

10.97.4.6 std::vector<RayShapePtr> gazebo::physics::MultiRayShape::rays [protected]

Ray data.

10.97.4.7 sdf::ElementPtr gazebo::physics::MultiRayShape::scanElem [protected]

Scan SDF element pointer.

10.97.4.8 sdf::ElementPtr gazebo::physics::MultiRayShape::vertElem [protected]

Vertical SDF element pointer.

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

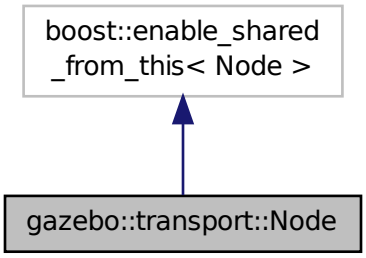
- **MultiRayShape.hh**

10.98 gazebo::transport::Node Class Reference

A node can advertise and subscribe topics, publish on advertised topics and listen to subscribed topics.

`#include <transport/transport.hh>`

Inheritance diagram for gazebo::transport::Node:



Public Member Functions

- **Node** ()

  Constructor.
- virtual ~**Node** ()

  Destructor.

- template<typename M >
  transport::PublisherPtr Advertise (const std::string &_topic, unsigned int _queueLimit=1000, double _hzRate=0)

  Adverise a topic.
- std::string **DecodeTopicName** (const std::string &_topic)

  Decode a topic name.
- std::string **EncodeTopicName** (const std::string &_topic)

  Encode a topic name.
- void **Fini** ()

  Finalize the node.
- unsigned int **GetId** () const

  Get the unique ID of the node.
- std::string **GetMsgType** (const std::string &_topic) const

  Get the message type for a topic.
- std::string **GetTopicNamespace** () const

  Get the topic namespace for this node.
- bool **HandleData** (const std::string &_topic, const std::string &_msg)

  Handle incoming data.
- bool **HandleMessage** (const std::string &_topic, **MessagePtr** _msg)

  Handle incoming msg.
- bool **HasLatchedSubscriber** (const std::string &_topic) const

  Return true if a subscriber on a specific topic is latched.
- void **Init** (const std::string &_space="")

  Init the node.
- void **InsertLatchedMsg** (const std::string &_topic, const std::string &_msg)

  Add a latched message to the node for publication.
- void **InsertLatchedMsg** (const std::string &_topic, **MessagePtr** _msg)

  Add a latched message to the node for publication.
- void **ProcessIncoming** ()

  Process incoming messages.
- void **ProcessPublishers** ()

  Process all publishers, which has each publisher send it's most recent message over the wire.
- template<typename M >
  void **Publish** (const std::string &_topic, const google::protobuf::Message &_message)

  A convenience function for a one-time publication of a message.
- void **RemoveCallback** (const std::string &_topic, unsigned int _id)
- template<typename M , typename T >
  SubscriberPtr Subscribe (const std::string &_topic, void(T::*_fp)(const boost::shared_ptr< M const > &), T *_obj, bool _latching=false)

  Subscribe to a topic using a class method as the callback.
- template<typename M >
  SubscriberPtr Subscribe (const std::string &_topic, void(*_fp)(const boost::shared_ptr< M const > &), bool _latching=false)

  Subscribe to a topic using a bare function as the callback.
- template<typename T >
  SubscriberPtr Subscribe (const std::string &_topic, void(T::*_fp)(const std::string &), T *_obj, bool _latching=false)

  Subscribe to a topic using a class method as the callback.
- **SubscriberPtr Subscribe** (const std::string &_topic, void(*_fp)(const std::string &), bool _latching=false)

  Subscribe to a topic using a bare function as the callback.

10.98.1 Detailed Description

A node can advertise and subscribe topics, publish on advertised topics and listen to subscribed topics.

10.98.2 Constructor & Destructor Documentation

10.98.2.1 gazebo::transport::Node::Node ( )

Constructor.

10.98.2.2 virtual gazebo::transport::Node::~Node ( ) [virtual]

Destructor.

10.98.3 Member Function Documentation

10.98.3.1 template<typename M > transport::PublisherPtr gazebo::transport::Node::Advertise (const std::string & *_topic*, unsigned int *_queueLimit* = 1000, double *_hzRate* = 0) [inline]

Adverise a topic.

Parameters

in	*_topic*	The topic to advertise
in	*_queueLimit*	The maximum number of outgoing messages to queue for delivery
in	*_hz*	Update rate for the publisher. Units are 1.0/seconds.

Returns

Pointer to new publisher object

References DecodeTopicName(), and SingletonT< T >::Instance().

10.98.3.2 std::string gazebo::transport::Node::DecodeTopicName (const std::string & *_topic*)

Decode a topic name.

Parameters

in	*The*	encoded name

Returns

The decoded name

Referenced by Advertise(), and Subscribe().

10.98.3.3 std::string gazebo::transport::Node::EncodeTopicName (const std::string & *_topic*)

Encode a topic name.

Parameters

in	*The*	decoded name

Returns

The encoded name

10.98.3.4 void gazebo::transport::Node::Fini ( )

Finalize the node.

10.98.3.5 unsigned int gazebo::transport::Node::GetId ( ) const

Get the unique ID of the node.

Returns

The unique ID of the node

Referenced by Subscribe().

10.98.3.6 std::string gazebo::transport::Node::GetMsgType (const std::string & *_topic*) const

Get the message type for a topic.

Parameters

in	*_topic*	The topic

Returns

The message type

10.98.3.7 std::string gazebo::transport::Node::GetTopicNamespace ( ) const

Get the topic namespace for this node.

Returns

The namespace

10.98.3.8 bool gazebo::transport::Node::HandleData (const std::string & *_topic*, const std::string & *_msg*)

Handle incoming data.

Parameters

in	*_topic*	Topic for which the data was received
in	*_msg*	The message that was received

Returns

true if the message was handled successfully, false otherwise

10.98.3.9 bool gazebo::transport::Node::HandleMessage (const std::string & *_topic,* MessagePtr *_msg*)

Handle incoming msg.

Parameters

in	*_topic*	Topic for which the data was received
in	*_msg*	The message that was received

Returns

true if the message was handled successfully, false otherwise

10.98.3.10 bool gazebo::transport::Node::HasLatchedSubscriber (const std::string & *_topic*) const

Return true if a subscriber on a specific topic is latched.

Parameters

in	*_topic*	Name of the topic to check.

Returns

True if a latched subscriber exists.

10.98.3.11 void gazebo::transport::Node::Init (const std::string & *_space* = " ")

Init the node.

Parameters

in	*_space*	Set the global namespace of all topics. If left blank, the topic will initialize to the first namespace on the **Master** (p. 475)

10.98.3.12 void gazebo::transport::Node::InsertLatchedMsg (const std::string & *_topic,* const std::string & *_msg*)

Add a latched message to the node for publication.

This is called when a subscription is connected to a publication.

Parameters

in	*_topic*	Name of the topic to publish data on.
in	*_msg*	The message to publish.

10.98.3.13 void gazebo::transport::Node::InsertLatchedMsg (const std::string & _topic, MessagePtr _msg)

Add a latched message to the node for publication.

This is called when a subscription is connected to a publication.

Parameters

in	*_topic*	Name of the topic to publish data on.
in	*_msg*	The message to publish.

10.98.3.14 void gazebo::transport::Node::ProcessIncoming ( )

Process incoming messages.

10.98.3.15 void gazebo::transport::Node::ProcessPublishers ( )

Process all publishers, which has each publisher send it's most recent message over the wire.

This is for internal use only

10.98.3.16 template<typename M > void gazebo::transport::Node::Publish (const std::string & _topic, const google::protobuf::Message & _message) [inline]

A convenience function for a one-time publication of a message.

This is inefficient, compared to **Node::Advertise** (p. 566) followed by **Publisher::Publish** (p. 651). This function should only be used when sending a message very infrequently.

Parameters

in	*_topic*	The topic to advertise
in	*_message*	Message to be published

10.98.3.17 void gazebo::transport::Node::RemoveCallback (const std::string & _topic, unsigned int _id)

10.98.3.18 template<typename M , typename T > SubscriberPtr gazebo::transport::Node::Subscribe (const std::string & _topic, void(T::*)(const boost::shared_ptr< M const > &) _fp, T * _obj, bool _latching = false) [inline]

Subscribe to a topic using a class method as the callback.

Parameters

in	*_topic*	The topic to subscribe to
in	*_fp*	Class method to be called on receipt of new message
in	*_obj*	Class instance to be used on receipt of new message
in	*_latching*	If true, latch latest incoming message; otherwise don't latch

Returns

Pointer to new **Subscriber** (p. 816) object

References DecodeTopicName(), GetId(), and SingletonT< T >::Instance().

10.98.3.19 template<typename M > SubscriberPtr gazebo::transport::Node::Subscribe (const std::string & *_topic*, void(*)(const boost::shared_ptr< M const > &) *_fp*, bool *_latching* = false) [inline]

Subscribe to a topic using a bare function as the callback.

Parameters

in	*_topic*	The topic to subscribe to
in	*_fp*	Function to be called on receipt of new message
in	*_latching*	If true, latch latest incoming message; otherwise don't latch

Returns

Pointer to new **Subscriber** (p. 816) object

References DecodeTopicName(), GetId(), and SingletonT< T >::Instance().

10.98.3.20 template<typename T > SubscriberPtr gazebo::transport::Node::Subscribe (const std::string & *_topic*, void(T::*)(const std::string &) *_fp*, T * *_obj*, bool *_latching* = false) [inline]

Subscribe to a topic using a class method as the callback.

Parameters

in	*_topic*	The topic to subscribe to
in	*_fp*	Class method to be called on receipt of new message
in	*_obj*	Class instance to be used on receipt of new message
in	*_latching*	If true, latch latest incoming message; otherwise don't latch

Returns

Pointer to new **Subscriber** (p. 816) object

References DecodeTopicName(), GetId(), gazebo::transport::SubscribeOptions::Init(), and SingletonT< T >::Instance().

10.98.3.21 SubscriberPtr gazebo::transport::Node::Subscribe (const std::string & *_topic*, void(*)(const std::string &) *_fp*, bool *_latching* = false) [inline]

Subscribe to a topic using a bare function as the callback.

Parameters

in	*_topic*	The topic to subscribe to
in	*_fp*	Function to be called on receipt of new message
in	*_latching*	If true, latch latest incoming message; otherwise don't latch

Returns

Pointer to new **Subscriber** (p. 816) object

References DecodeTopicName(), GetId(), gazebo::transport::SubscribeOptions::Init(), and SingletonT< T >::Instance().

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

- **Node.hh**

10.99 gazebo::common::NodeAnimation Class Reference

Node animation.

`#include <common/common.hh>`

Public Member Functions

- **NodeAnimation** (const std::string &_name)

  constructor
- ~**NodeAnimation** ()

  Destructor. It empties the key frames list.
- void **AddKeyFrame** (const double _time, const **math::Matrix4** _trans)

  Adds a key frame at a specific time.
- void **AddKeyFrame** (const double _time, const **math::Pose** _pose)

  Adds a key fram at a specific time.
- **math::Matrix4 GetFrameAt** (double _time, bool _loop=true) const

  Returns a frame transformation at a specific time if a node does not exist at that time (with tolerance of 1e-6 sec), the transformation is interpolated.
- unsigned int **GetFrameCount** () const

  Returns the number of key frames.
- void **GetKeyFrame** (const unsigned int _i, double &_time, **math::Matrix4** &_trans) const

  Finds a key frame using the index.
- std::pair< double, **math::Matrix4** > **GetKeyFrame** (const unsigned int _i) const

  Returns a key frame using the index.
- double **GetLength** () const

  Returns the duration of the animations.
- std::string **GetName** () const

  Returns the name.
- double **GetTimeAtX** (const double _x) const

  Returns the time where a transformation's translational value along the X axis is equal to _x.
- void **Scale** (const double _scale)

  Scales each transformation in the key frames.
- void **SetName** (const std::string &_name)

  Changes the name of the animation.

Protected Attributes

- std::map< double, **math::Matrix4** > **keyFrames**

  the dictionary of key frames, indexed by time
- double **length**

  the duration of the animations (time of last key frame)
- std::string **name**

  the name of the animation

10.99.1 Detailed Description

Node animation.

10.99.2 Constructor & Destructor Documentation

10.99.2.1 gazebo::common::NodeAnimation::NodeAnimation (const std::string & *_name*)

constructor

Parameters

in	*_name*	the name of the node

10.99.2.2 gazebo::common::NodeAnimation::~NodeAnimation ( )

Destructor. It empties the key frames list.

10.99.3 Member Function Documentation

10.99.3.1 void gazebo::common::NodeAnimation::AddKeyFrame (const double *_time,* const **math::Matrix4** *_trans*)

Adds a key frame at a specific time.

Parameters

in	*_time*	the time of the key frame
in	*_trans*	the transformation

10.99.3.2 void gazebo::common::NodeAnimation::AddKeyFrame (const double *_time,* const **math::Pose** *_pose*)

Adds a key fram at a specific time.

Parameters

in	*_time*	the tiem of the key frame
in	*_pose*	the pose

10.99.3.3 math::Matrix4 gazebo::common::NodeAnimation::GetFrameAt (double _time, bool _loop = true) const

Returns a frame transformation at a specific time if a node does not exist at that time (with tolerance of 1e-6 sec), the transformation is interpolated.

Parameters

in	*_time*	the time
in	*_loop*	when true, the time is divided by the duration (see GetLength)

10.99.3.4 unsigned int gazebo::common::NodeAnimation::GetFrameCount ( ) const

Returns the number of key frames.

Returns

the count

10.99.3.5 void gazebo::common::NodeAnimation::GetKeyFrame (const unsigned int _i, double & _time, math::Matrix4 & _trans) const

Finds a key frame using the index.

Note the index of a key frame can change as frames are added.

Parameters

in	*_i*	the index
out	*_time*	the time of the frame, or -1 if the index id is out of bounds
out	*_trans*	the transformation for this key frame

10.99.3.6 std::pair<double, math::Matrix4> gazebo::common::NodeAnimation::GetKeyFrame (const unsigned int _i) const

Returns a key frame using the index.

Note the index of a key frame can change as frames are added.

Parameters

in	*_i*	the index

Returns

a pair that contains the time and transformation. **Time** (p. 831) is -1 if the index is out of bounds

10.99.3.7 double gazebo::common::NodeAnimation::GetLength ( ) const

Returns the duration of the animations.

Returns

the time of the last animation

10.99.3.8 std::string gazebo::common::NodeAnimation::GetName ( ) const

Returns the name.

Returns

the name

10.99.3.9 double gazebo::common::NodeAnimation::GetTimeAtX (const double _x) const

Returns the time where a transformation's translational value along the X axis is equal to _x.

When no transformation is found (within a tolerance of 1e-6), the time is interpolated.

Parameters

in	*_x*	the value along x. You must ensure that _x is within a valid range.

10.99.3.10 void gazebo::common::NodeAnimation::Scale (const double _scale)

Scales each transformation in the key frames.

This only affects the translational values.

Parameters

in	*_scale*	the scaling factor

10.99.3.11 void gazebo::common::NodeAnimation::SetName (const std::string & _name)

Changes the name of the animation.

Parameters

in	*the*	new name

10.99.4 Member Data Documentation

10.99.4.1 std::map<double, math::Matrix4> gazebo::common::NodeAnimation::keyFrames [protected]

the dictionary of key frames, indexed by time

10.99.4.2 double gazebo::common::NodeAnimation::length [protected]

the duration of the animations (time of last key frame)

10.99.4.3 std::string gazebo::common::NodeAnimation::name [protected]

the name of the animation

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

- **SkeletonAnimation.hh**

10.100 gazebo::common::NodeAssignment Struct Reference

Vertex to node weighted assignement for skeleton animation visualization.

`#include <Mesh.hh>`

Public Attributes

- unsigned int **nodeIndex**

  node (or bone) index
- unsigned int **vertexIndex**

  index of the vertex
- float **weight**

  the weight (between 0 and 1)

10.100.1 Detailed Description

Vertex to node weighted assignement for skeleton animation visualization.

10.100.2 Member Data Documentation

10.100.2.1 unsigned int gazebo::common::NodeAssignment::nodeIndex

node (or bone) index

10.100.2.2 unsigned int gazebo::common::NodeAssignment::vertexIndex

index of the vertex

10.100.2.3 float gazebo::common::NodeAssignment::weight

the weight (between 0 and 1)

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

- **Mesh.hh**

10.101 gazebo::common::NodeTransform Class Reference

NodeTransform (p. 576) **Skeleton.hh** (p. 1132) common/common.hh

```
#include <Skeleton.hh>
```

Public Types

- enum **TransformType** { **TRANSLATE**, **ROTATE**, **SCALE**, **MATRIX** }

  Enumeration of the transform types.

Public Member Functions

- **NodeTransform** (**TransformType** _type=**MATRIX**)

  Constructor.
- **NodeTransform** (**math::Matrix4** _mat, std::string _sid="_default_", TransformType _type=**MATRIX**)

  Constructor.
- ∼**NodeTransform** ()

  Destructor. It does nothing.
- **math::Matrix4 Get** ()

  Returns the transformation matrix.
- std::string **GetSID** ()

  Returns thr SID.
- **TransformType GetType** ()

  Returns the transformation type.
- **math::Matrix4 operator()** ()

  Matrix cast operator.
- **math::Matrix4 operator**∗ (**NodeTransform** _t)

  Node transform multiplication operator.
- **math::Matrix4 operator**∗ (**math::Matrix4** _m)

  Matrix multiplication operator.
- void **PrintSource** ()

  Prints the transform matrix to std::err stream.
- void **RecalculateMatrix** ()

  Sets the transform matrix from the source according to the type.
- void **Set** (**math::Matrix4** _mat)

  Assign a transformation.
- void **SetComponent** (unsigned int _idx, double _value)

  Set a transformation matrix component value.
- void **SetSID** (std::string _sid)

  Set the SID.
- void **SetSourceValues** (**math::Matrix4** _mat)

  Set source data values _ param[in] _mat the values.
- void **SetSourceValues** (**math::Vector3** _vec)

  Set source data values.
- void **SetSourceValues** (**math::Vector3** _axis, double _angle)

  Sets source matrix values from roation.
- void **SetType** (**TransformType** _type)

  Set transform type.

Protected Attributes

- std::string **sid**

  the sid
- std::vector< double > **source**

  source data values (can be a matrix, a position or rotation)
- **math::Matrix4 transform**

  transform
- **TransformType type**

  transform type

10.101.1 Detailed Description

NodeTransform (p. 576) **Skeleton.hh** (p. 1132) common/common.hh

A transformation node

10.101.2 Member Enumeration Documentation

10.101.2.1 enum gazebo::common::NodeTransform::TransformType

Enumeration of the transform types.

Enumerator:

TRANSLATE

ROTATE

SCALE

MATRIX

10.101.3 Constructor & Destructor Documentation

10.101.3.1 gazebo::common::NodeTransform::NodeTransform (TransformType *_type* = MATRIX)

Constructor.

Parameters

in	*_type*	the type of transform

10.101.3.2 gazebo::common::NodeTransform::NodeTransform (math::Matrix4 *_mat*, std::string *_sid* = "_default_", TransformType *_type* = MATRIX)

Constructor.

Parameters

in	*_mat*	the matrix
in	*_sid*	identifier
in	*_type*	the type of transform

10.101.3.3 gazebo::common::NodeTransform::~NodeTransform ( )

Destructor. It does nothing.

10.101.4 Member Function Documentation

10.101.4.1 math::Matrix4 gazebo::common::NodeTransform::Get ( )

Returns the transformation matrix.

Returns

the matrix

10.101.4.2 std::string gazebo::common::NodeTransform::GetSID ( )

Returns thr SID.

Returns

the SID

10.101.4.3 TransformType gazebo::common::NodeTransform::GetType ( )

Returns the transformation type.

Returns

the type

10.101.4.4 math::Matrix4 gazebo::common::NodeTransform::operator() ( )

Matrix cast operator.

Returns

the transform

10.101.4.5 math::Matrix4 gazebo::common::NodeTransform::operator* (NodeTransform _t)

Node transform multiplication operator.

Parameters

in	_t	a transform

Returns

transform matrix multiplied by _t's transform

10.101.4.6 math::Matrix4 gazebo::common::NodeTransform::operator∗ (math::Matrix4 _m)

Matrix multiplication operator.

Parameters

in	*_m*	a matrix

Returns

transform matrix multiplied by _m

10.101.4.7 void gazebo::common::NodeTransform::PrintSource ( )

Prints the transform matrix to std::err stream.

10.101.4.8 void gazebo::common::NodeTransform::RecalculateMatrix ( )

Sets the transform matrix from the source according to the type.

10.101.4.9 void gazebo::common::NodeTransform::Set (math::Matrix4 _mat)

Assign a transformation.

Parameters

in	*_mat*	the transform

10.101.4.10 void gazebo::common::NodeTransform::SetComponent (unsigned int _idx, double _value)

Set a transformation matrix component value.

Parameters

in	*_idx*	the component index
in	*_value*	the value

10.101.4.11 void gazebo::common::NodeTransform::SetSID (std::string _sid)

Set the SID.

Parameters

in	*_sid*	the sid

10.101.4.12 void gazebo::common::NodeTransform::SetSourceValues (math::Matrix4 _mat)

Set source data values _ param[in] _mat the values.

10.101.4.13 void gazebo::common::NodeTransform::SetSourceValues (math::Vector3 _vec)

Set source data values.

10.101.4.14 void gazebo::common::NodeTransform::SetSourceValues (math::Vector3 _axis, double _angle)

Sets source matrix values from roation.

Parameters

in	*_axis*	of rotation
in	*_angle*	of rotation

10.101.4.15 void gazebo::common::NodeTransform::SetType (TransformType _type)

Set transform type.

Parameters

in	*_type*	the type

10.101.5 Member Data Documentation

10.101.5.1 std::string gazebo::common::NodeTransform::sid [protected]

the sid

10.101.5.2 std::vector<double> gazebo::common::NodeTransform::source [protected]

source data values (can be a matrix, a position or rotation)

10.101.5.3 math::Matrix4 gazebo::common::NodeTransform::transform [protected]

transform

10.101.5.4 TransformType gazebo::common::NodeTransform::type [protected]

transform type

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

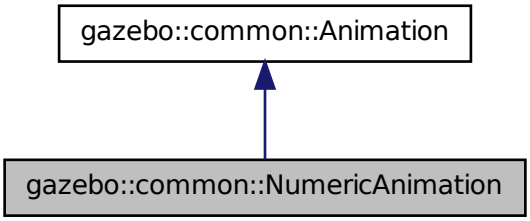
- **Skeleton.hh**

10.102 gazebo::common::NumericAnimation Class Reference

A numeric animation.

`#include <Animation.hh>`

Inheritance diagram for gazebo::common::NumericAnimation:



Public Member Functions

- **NumericAnimation** (const std::string &_name, double _length, bool _loop)

  Constructor.
- virtual ~**NumericAnimation** ()

  Destructor.
- **NumericKeyFrame** * **CreateKeyFrame** (double _time)

  Create a numeric keyframe at the given time.
- void **GetInterpolatedKeyFrame** (**NumericKeyFrame** &_kf) const

  Get a keyframe using the animation's current time.

Additional Inherited Members

10.102.1 Detailed Description

A numeric animation.

10.102.2 Constructor & Destructor Documentation

10.102.2.1 gazebo::common::NumericAnimation::NumericAnimation (const std::string & *_name,* double *_length,* bool *_loop*)

Constructor.

Parameters

in	*_name*	String name of the animation. This should be unique.
in	*_length*	Length of the animation in seconds
in	*_loop*	True == loop the animation

10.102.2.2 virtual gazebo::common::NumericAnimation::~NumericAnimation ( ) [virtual]

Destructor.

10.102.3 Member Function Documentation

10.102.3.1 NumericKeyFrame* gazebo::common::NumericAnimation::CreateKeyFrame (double *_time*)

Create a numeric keyframe at the given time.

Parameters

in	*_time*	**Time** (p. 831) at which to create the keyframe

Returns

Pointer to the new keyframe

10.102.3.2 void gazebo::common::NumericAnimation::GetInterpolatedKeyFrame (NumericKeyFrame & *_kf*) const

Get a keyframe using the animation's current time.

Parameters

out	*_kf*	**NumericKeyFrame** (p. 582) reference to hold the interpolated result

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

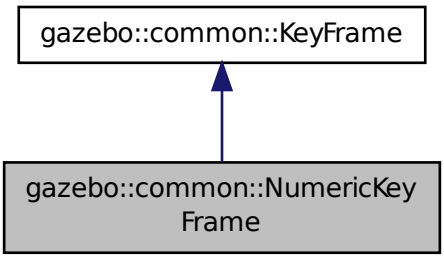
- **Animation.hh**

10.103 gazebo::common::NumericKeyFrame Class Reference

A keyframe for a **NumericAnimation** (p. 581).

`#include <KeyFrame.hh>`

Inheritance diagram for gazebo::common::NumericKeyFrame:



Public Member Functions

- **NumericKeyFrame** (double _time)

  Constructor.
- virtual ~**NumericKeyFrame** ()

  Destructor.
- const double & **GetValue** () const

  Get the value of the keyframe.
- void **SetValue** (const double &_value)

  Set the value of the keyframe.

Protected Attributes

- double **value**

  numeric value

10.103.1 Detailed Description

A keyframe for a **NumericAnimation** (p. 581).

10.103.2 Constructor & Destructor Documentation

10.103.2.1 gazebo::common::NumericKeyFrame::NumericKeyFrame (double *_time*)

Constructor.

Parameters

in	*_time*	**Time** (p. 831) of the keyframe

10.103.2.2 virtual gazebo::common::NumericKeyFrame::~NumericKeyFrame ( ) [virtual]

Destructor.

10.103.3 Member Function Documentation

10.103.3.1 const double& gazebo::common::NumericKeyFrame::GetValue ( ) const

Get the value of the keyframe.

Returns

the value of the keyframe

10.103.3.2 void gazebo::common::NumericKeyFrame::SetValue (const double & *_value*)

Set the value of the keyframe.

Parameters

in	*_value*	The new value

10.103.4 Member Data Documentation

10.103.4.1 double gazebo::common::NumericKeyFrame::value [protected]

numeric value

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

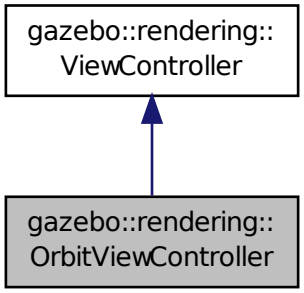
- **KeyFrame.hh**

10.104 gazebo::rendering::OrbitViewController Class Reference

Orbit view controller.

`#include <OrbitViewController.hh>`

Inheritance diagram for gazebo::rendering::OrbitViewController:



Public Member Functions

- **OrbitViewController** (**UserCameraPtr** _camera)

  Constructor.
- virtual ~**OrbitViewController** ()

  Destructor.
- **math::Vector3 GetFocalPoint** () const

  Get the focal point.
- virtual void **HandleKeyPressEvent** (const std::string &_key)

  Handle a key press event.
- void **HandleKeyReleaseEvent** (const std::string &_key)

  Handle a key release event.
- virtual void **HandleMouseEvent** (const **common::MouseEvent** &_event)

  Handle a mouse event.
- virtual void **Init** ()

  Initialize the controller.
- virtual void **Init** (const **math::Vector3** &_focalPoint)

  Initialze the controller with a focal point.
- void **SetDistance** (float _d)

  Set the distance to the focal point.
- void **SetFocalPoint** (const **math::Vector3** &_fp)

  Set the focal point.
- virtual void **Update** ()

  Update.

Static Public Member Functions

- static std::string **GetTypeString** ()

  Get the type name of this view controller.

Additional Inherited Members

10.104.1 Detailed Description

Orbit view controller.

10.104.2 Constructor & Destructor Documentation

10.104.2.1 gazebo::rendering::OrbitViewController::OrbitViewController (UserCameraPtr *_camera*)

Constructor.

Parameters

in	*_camera*	Pointer to the camera to control.

10.104.2.2 virtual gazebo::rendering::OrbitViewController::~OrbitViewController ( ) [virtual]

Destructor.

10.104.3 Member Function Documentation

10.104.3.1 math::Vector3 gazebo::rendering::OrbitViewController::GetFocalPoint ( ) const

Get the focal point.

Returns

The focal point

10.104.3.2 static std::string gazebo::rendering::OrbitViewController::GetTypeString ( ) [static]

Get the type name of this view controller.

Returns

The view controller name: "orbit".

10.104.3.3 virtual void gazebo::rendering::OrbitViewController::HandleKeyPressEvent (const std::string & *_key*) [virtual]

Handle a key press event.

Parameters

in	*_key*	The key that was pressed.

Implements **gazebo::rendering::ViewController** (p. 918).

10.104.3.4 void gazebo::rendering::OrbitViewController::HandleKeyReleaseEvent (const std::string & _key) [virtual]

Handle a key release event.

Parameters

in	*_key*	The key that was released.

Implements **gazebo::rendering::ViewController** (p. 918).

10.104.3.5 virtual void gazebo::rendering::OrbitViewController::HandleMouseEvent (const common::MouseEvent & _event) [virtual]

Handle a mouse event.

Parameters

in	*_event*	The mouse event.

Implements **gazebo::rendering::ViewController** (p. 918).

10.104.3.6 virtual void gazebo::rendering::OrbitViewController::Init ( ) [virtual]

Initialize the controller.

Implements **gazebo::rendering::ViewController** (p. 919).

10.104.3.7 virtual void gazebo::rendering::OrbitViewController::Init (const math::Vector3 & _focalPoint) [virtual]

Initialze the controller with a focal point.

Parameters

in	*_focalPoint*	Point to look at.

Reimplemented from **gazebo::rendering::ViewController** (p. 919).

10.104.3.8 void gazebo::rendering::OrbitViewController::SetDistance (float _d)

Set the distance to the focal point.

Parameters

in	*_d*	The distance from the focal point.

10.104.3.9 void gazebo::rendering::OrbitViewController::SetFocalPoint (const math::Vector3 & _fp)

Set the focal point.

Parameters

in	*_fp*	The focal point

10.104.3.10 virtual void gazebo::rendering::OrbitViewController::Update ( ) [virtual]

Update.

Implements **gazebo::rendering::ViewController** (p. 919).

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

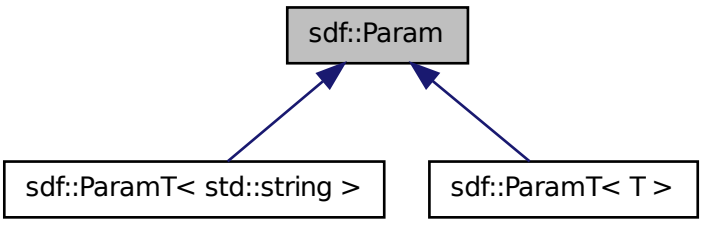
- **OrbitViewController.hh**

10.105 sdf::Param Class Reference

A parameter class.

`#include <Param.hh>`

Inheritance diagram for sdf::Param:



Public Member Functions

- **Param** (**Param** *_newParam) GAZEBO_DEPRECATED(1.6)

  Constructor.
- virtual ~**Param** () GAZEBO_DEPRECATED(1.6)

  Destructor.
- virtual boost::shared_ptr< **Param** > **Clone** () const GAZEBO_DEPRECATED(1.6)=0
- template<typename T >
  bool **Get** (T &_value)

  Get the value of the parameter.
- bool **Get** (bool &_value) GAZEBO_DEPRECATED(1.6)
- bool **Get** (int &_value) GAZEBO_DEPRECATED(1.6)
- bool **Get** (unsigned int &_value) GAZEBO_DEPRECATED(1.6)
- bool **Get** (float &_value) GAZEBO_DEPRECATED(1.6)

- bool **Get** (double &_value) GAZEBO_DEPRECATED(1.6)
- bool **Get** (char &_value) GAZEBO_DEPRECATED(1.6)
- bool **Get** (std::string &_value) GAZEBO_DEPRECATED(1.6)
- bool **Get** (**gazebo::math::Vector3** &_value) GAZEBO_DEPRECATED(1.6)
- bool **Get** (**gazebo::math::Vector2i** &_value) GAZEBO_DEPRECATED(1.6)
- bool **Get** (**gazebo::math::Vector2d** &_value) GAZEBO_DEPRECATED(1.6)
- bool **Get** (**gazebo::math::Quaternion** &_value) GAZEBO_DEPRECATED(1.6)
- bool **Get** (**gazebo::math::Pose** &_value) GAZEBO_DEPRECATED(1.6)
- bool **Get** (**gazebo::common::Color** &_value) GAZEBO_DEPRECATED(1.6)
- bool **Get** (**gazebo::common::Time** &_value) GAZEBO_DEPRECATED(1.6)
- virtual std::string **GetAsString** () const GAZEBO_DEPRECATED(1.6)

  Get the type.
- virtual std::string **GetDefaultAsString** () const GAZEBO_DEPRECATED(1.6)
- std::string **GetDescription** () const GAZEBO_DEPRECATED(1.6)

  Get the description of the parameter.
- const std::string & **GetKey** () const GAZEBO_DEPRECATED(1.6)
- bool **GetRequired** () const GAZEBO_DEPRECATED(1.6)
- bool **GetSet** () const GAZEBO_DEPRECATED(1.6)

  Return true if the parameter has been set.
- std::string **GetTypeName** () const GAZEBO_DEPRECATED(1.6)
- bool **IsBool** () const GAZEBO_DEPRECATED(1.6)
- bool **IsChar** () const GAZEBO_DEPRECATED(1.6)
- bool **IsColor** () const GAZEBO_DEPRECATED(1.6)
- bool **IsDouble** () const GAZEBO_DEPRECATED(1.6)
- bool **IsFloat** () const GAZEBO_DEPRECATED(1.6)
- bool **IsInt** () const GAZEBO_DEPRECATED(1.6)
- bool **IsPose** () const GAZEBO_DEPRECATED(1.6)
- bool **IsQuaternion** () const GAZEBO_DEPRECATED(1.6)
- bool **IsStr** () const GAZEBO_DEPRECATED(1.6)
- bool **IsTime** () const GAZEBO_DEPRECATED(1.6)
- bool **IsUInt** () const GAZEBO_DEPRECATED(1.6)
- bool **IsVector2d** () const GAZEBO_DEPRECATED(1.6)
- bool **IsVector2i** () const GAZEBO_DEPRECATED(1.6)
- bool **IsVector3** () const GAZEBO_DEPRECATED(1.6)
- virtual void **Reset** () GAZEBO_DEPRECATED(1.6)=0

  Reset the parameter.
- bool **Set** (const bool &_value) GAZEBO_DEPRECATED(1.6)
- bool **Set** (const int &_value) GAZEBO_DEPRECATED(1.6)
- bool **Set** (const unsigned int &_value) GAZEBO_DEPRECATED(1.6)
- bool **Set** (const float &_value) GAZEBO_DEPRECATED(1.6)
- bool **Set** (const double &_value) GAZEBO_DEPRECATED(1.6)
- bool **Set** (const char &_value) GAZEBO_DEPRECATED(1.6)
- bool **Set** (const std::string &_value) GAZEBO_DEPRECATED(1.6)
- bool **Set** (const char ∗_value) GAZEBO_DEPRECATED(1.6)
- bool **Set** (const **gazebo::math::Vector3** &_value) GAZEBO_DEPRECATED(1.6)
- bool **Set** (const **gazebo::math::Vector2i** &_value) GAZEBO_DEPRECATED(1.6)
- bool **Set** (const **gazebo::math::Vector2d** &_value) GAZEBO_DEPRECATED(1.6)
- bool **Set** (const **gazebo::math::Quaternion** &_value) GAZEBO_DEPRECATED(1.6)
- bool **Set** (const **gazebo::math::Pose** &_value) GAZEBO_DEPRECATED(1.6)
- bool **Set** (const **gazebo::common::Color** &_value) GAZEBO_DEPRECATED(1.6)

- bool **Set** (const **gazebo::common::Time** &_value) GAZEBO_DEPRECATED(1.6)
- void **SetDescription** (const std::string &_desc) GAZEBO_DEPRECATED(1.6)

  Set the description of the parameter.
- virtual bool **SetFromString** (const std::string &) GAZEBO_DEPRECATED(1.6)

  Set the parameter value from a string.
- template<typename T >
  void **SetUpdateFunc** (T _updateFunc)

  Update function.
- virtual void **Update** () GAZEBO_DEPRECATED(1.6)=0

Protected Attributes

- std::string **description**
- std::string **key**
- bool **required**
- bool **set**
- std::string **typeName**
- boost::function< boost::any()> **updateFunc**

10.105.1 Detailed Description

A parameter class.

10.105.2 Constructor & Destructor Documentation

10.105.2.1 sdf::Param::Param (Param * _newParam)

Constructor.

10.105.2.2 virtual sdf::Param::~Param ( ) [virtual]

Destructor.

10.105.3 Member Function Documentation

10.105.3.1 virtual boost::shared_ptr<Param> sdf::Param::Clone ( ) const [pure virtual]

Implemented in **sdf::ParamT< T >** (p. 596), and **sdf::ParamT< std::string >** (p. 596).

10.105.3.2 template<typename T > bool sdf::Param::Get (T & _value) [inline]

Get the value of the parameter.

Parameters

out	*_value*	The value of the parameter.

Returns

True if parameter was successfully cast to the value type passed in.

References GetAsString(), gzerr, key, and typeName.

10.105.3.3 bool sdf::Param::Get (bool & *_value*)

10.105.3.4 bool sdf::Param::Get (int & *_value*)

10.105.3.5 bool sdf::Param::Get (unsigned int & *_value*)

10.105.3.6 bool sdf::Param::Get (float & *_value*)

10.105.3.7 bool sdf::Param::Get (double & *_value*)

10.105.3.8 bool sdf::Param::Get (char & *_value*)

10.105.3.9 bool sdf::Param::Get (std::string & *_value*)

10.105.3.10 bool sdf::Param::Get (gazebo::math::Vector3 & *_value*)

10.105.3.11 bool sdf::Param::Get (gazebo::math::Vector2i & *_value*)

10.105.3.12 bool sdf::Param::Get (gazebo::math::Vector2d & *_value*)

10.105.3.13 bool sdf::Param::Get (gazebo::math::Quaternion & *_value*)

10.105.3.14 bool sdf::Param::Get (gazebo::math::Pose & *_value*)

10.105.3.15 bool sdf::Param::Get (gazebo::common::Color & *_value*)

10.105.3.16 bool sdf::Param::Get (gazebo::common::Time & *_value*)

10.105.3.17 virtual std::string sdf::Param::GetAsString ( ) const [inline], [virtual]

Get the type.

Reimplemented in **sdf::ParamT< T >** (p. 596), and **sdf::ParamT< std::string >** (p. 596).

Referenced by sdf::ParamT< std::string >::Clone(), and Get().

10.105.3.18 virtual std::string sdf::Param::GetDefaultAsString ( ) const [inline], [virtual]

Reimplemented in **sdf::ParamT< T >** (p. 596), and **sdf::ParamT< std::string >** (p. 596).

10.105.3.19 std::string sdf::Param::GetDescription ( ) const

Get the description of the parameter.

10.105.3.20 const std::string& sdf::Param::GetKey ( ) const [inline]

References key.

Referenced by sdf::ParamT< std::string >::Clone(), and sdf::ParamT< std::string >::Set().

10.105.3.21 bool sdf::Param::GetRequired ( ) const [inline]

References required.

10.105.3.22 bool sdf::Param::GetSet ( ) const [inline]

Return true if the parameter has been set.

10.105.3.23 std::string sdf::Param::GetTypeName ( ) const

10.105.3.24 bool sdf::Param::IsBool ( ) const

10.105.3.25 bool sdf::Param::IsChar ( ) const

10.105.3.26 bool sdf::Param::IsColor ( ) const

10.105.3.27 bool sdf::Param::IsDouble ( ) const

10.105.3.28 bool sdf::Param::IsFloat ( ) const

10.105.3.29 bool sdf::Param::IsInt ( ) const

10.105.3.30 bool sdf::Param::IsPose ( ) const

10.105.3.31 bool sdf::Param::IsQuaternion ( ) const

10.105.3.32 bool sdf::Param::IsStr ( ) const

10.105.3.33 bool sdf::Param::IsTime ( ) const

10.105.3.34 bool sdf::Param::IsUInt ( ) const

10.105.3.35 bool sdf::Param::IsVector2d ( ) const

10.105.3.36 bool sdf::Param::IsVector2i ( ) const

10.105.3.37 bool sdf::Param::IsVector3 ( ) const

10.105.3.38 virtual void sdf::Param::Reset ( ) [pure virtual]

Reset the parameter.

Implemented in **sdf::ParamT< T >** (p. 597), and **sdf::ParamT< std::string >** (p. 597).

10.105.3.39 bool sdf::Param::Set (const bool & _value)

Referenced by sdf::ParamT< std::string >::ParamT(), sdf::ParamT< std::string >::SetFromString(), and sdf::ParamT< std::string >::Update().

10.105.3.40 bool sdf::Param::Set (const int & _value)

10.105.3.41 bool sdf::Param::Set (const unsigned int & _value)

10.105.3.42 bool sdf::Param::Set (const float & _value)

10.105.3.43 bool sdf::Param::Set (const double & _value)

10.105.3.44 bool sdf::Param::Set (const char & _value)

10.105.3.45 bool sdf::Param::Set (const std::string & _value)

10.105.3.46 bool sdf::Param::Set (const char * _value)

10.105.3.47 bool sdf::Param::Set (const gazebo::math::Vector3 & _value)

10.105.3.48 bool sdf::Param::Set (const gazebo::math::Vector2i & _value)

10.105.3.49 bool sdf::Param::Set (const gazebo::math::Vector2d & _value)

10.105.3.50 bool sdf::Param::Set (const gazebo::math::Quaternion & _value)

10.105.3.51 bool sdf::Param::Set (const gazebo::math::Pose & _value)

10.105.3.52 bool sdf::Param::Set (const gazebo::common::Color & _value)

10.105.3.53 bool sdf::Param::Set (const gazebo::common::Time & _value)

10.105.3.54 void sdf::Param::SetDescription (const std::string & _desc)

Set the description of the parameter.

10.105.3.55 virtual bool sdf::Param::SetFromString (const std::string &) [inline], [virtual]

Set the parameter value from a string.

Reimplemented in **sdf::ParamT< T >** (p. 597), and **sdf::ParamT< std::string >** (p. 597).

10.105.3.56 template<typename T > void sdf::Param::SetUpdateFunc (T _updateFunc) [inline]

Update function.

References updateFunc.

10.105.3.57 virtual void sdf::Param::Update ( ) [pure virtual]

Implemented in **sdf::ParamT< T >** (p. 597), and **sdf::ParamT< std::string >** (p. 597).

10.105.4 Member Data Documentation

10.105.4.1 std::string sdf::Param::description [protected]

10.105.4.2 std::string sdf::Param::key [protected]

Referenced by Get(), and GetKey().

10.105.4.3 bool sdf::Param::required [protected]

Referenced by GetRequired(), and sdf::ParamT< std::string >::Set().

10.105.4.4 bool sdf::Param::set [protected]

10.105.4.5 std::string sdf::Param::typeName [protected]

Referenced by Get().

10.105.4.6 boost::function<boost::any ()> sdf::Param::updateFunc [protected]

Referenced by SetUpdateFunc(), and sdf::ParamT< std::string >::Update().

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

- **Param.hh**

10.106 ParamT< T > Class Template Reference

`#include <CommonTypes.hh>`

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

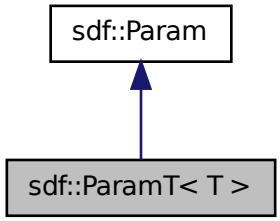
- **CommonTypes.hh**

10.107 sdf::ParamT< T > Class Template Reference

Templatized parameter class.

`#include <Param.hh>`

Inheritance diagram for sdf::ParamT< T >:



Public Member Functions

- **ParamT** (const std::string &_key, const std::string &_default, bool _required, const std::string &_typeName="", const std::string &_description="") GAZEBO_DEPRECATED(1.6)

  Constructor.
- virtual ~**ParamT** () GAZEBO_DEPRECATED(1.6)

  Destructor.
- virtual boost::shared_ptr< **Param** > **Clone** () const GAZEBO_DEPRECATED(1.6)
- virtual std::string **GetAsString** () const GAZEBO_DEPRECATED(1.6)

  Get the parameter value as a string.
- virtual std::string **GetDefaultAsString** () const GAZEBO_DEPRECATED(1.6)
- T **GetDefaultValue** () const GAZEBO_DEPRECATED(1.6)

  Get the value.
- T **GetValue** () const GAZEBO_DEPRECATED(1.6)

  Get the value.
- T **operator*** () const GAZEBO_DEPRECATED(1.6)
- virtual void **Reset** () GAZEBO_DEPRECATED(1.6)

  Reset to default value.
- virtual bool **Set** (const std::string &_str) GAZEBO_DEPRECATED(1.6)

  Set the parameter value from a string.
- virtual bool **SetFromString** (const std::string &_value) GAZEBO_DEPRECATED(1.6)

  Set the parameter value from a string.
- void **SetValue** (const T &_value) GAZEBO_DEPRECATED(1.6)

  Set the value of the parameter.
- virtual void **Update** () GAZEBO_DEPRECATED(1.6)

  Update param value.

Protected Attributes

- T **defaultValue**
- T **value**

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **ParamT**< T > &_p) GAZEBO_DEPRECATED(1.6)

10.107.1 Detailed Description

template<**typename T**>**class sdf::ParamT**< T >

Templatized parameter class.

10.107.2 Constructor & Destructor Documentation

10.107.2.1 template<typename T> sdf::ParamT< T >::ParamT (const std::string & *_key,* const std::string & *_default,* bool *_required,* const std::string & *_typeName = " ",* const std::string & *_description = " ")* [inline]

Constructor.

10.107.2.2 template<typename T> virtual sdf::ParamT< T >::~ParamT ( ) [inline], [virtual]

Destructor.

10.107.3 Member Function Documentation

10.107.3.1 template<typename T> virtual boost::shared_ptr<Param> sdf::ParamT< T >::Clone ( ) const [inline], [virtual]

Implements **sdf::Param** (p. 590).

10.107.3.2 template<typename T> virtual std::string sdf::ParamT< T >::GetAsString ( ) const [inline], [virtual]

Get the parameter value as a string.

Reimplemented from **sdf::Param** (p. 591).

10.107.3.3 template<typename T> virtual std::string sdf::ParamT< T >::GetDefaultAsString ( ) const [inline], [virtual]

Reimplemented from **sdf::Param** (p. 591).

10.107.3.4 template<typename T> T sdf::ParamT< T >::GetDefaultValue ( ) const [inline]

Get the value.

10.107.3.5 template<typename T> T sdf::ParamT< T >::GetValue ( ) const [inline]

Get the value.

10.107.3.6 template<typename T> T sdf::ParamT< T >::operator*( ) const [inline]

10.107.3.7 template<typename T> virtual void sdf::ParamT< T >::Reset( ) [inline], [virtual]

Reset to default value.

Implements **sdf::Param** (p. 592).

10.107.3.8 template<typename T> virtual bool sdf::ParamT< T >::Set(const std::string & _str) [inline], [virtual]

Set the parameter value from a string.

10.107.3.9 template<typename T> virtual bool sdf::ParamT< T >::SetFromString(const std::string & _value) [inline], [virtual]

Set the parameter value from a string.

Reimplemented from **sdf::Param** (p. 593).

10.107.3.10 template<typename T> void sdf::ParamT< T >::SetValue(const T & _value) [inline]

Set the value of the parameter.

10.107.3.11 template<typename T> virtual void sdf::ParamT< T >::Update( ) [inline], [virtual]

Update param value.

Implements **sdf::Param** (p. 594).

10.107.4 Friends And Related Function Documentation

10.107.4.1 template<typename T> std::ostream& operator<<(std::ostream & _out, const ParamT< T > & _p) [friend]

10.107.5 Member Data Documentation

10.107.5.1 template<typename T> T sdf::ParamT< T >::defaultValue [protected]

10.107.5.2 template<typename T> T sdf::ParamT< T >::value [protected]

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

- **Param.hh**

10.108 gazebo::physics::PhysicsEngine Class Reference

Base (p. 141) class for a physics engine.

`#include <physics/physics.hh>`

Public Member Functions

- **PhysicsEngine** (**WorldPtr** _world)

  Default constructor.
- virtual ~**PhysicsEngine** ()

  Destructor.
- virtual **CollisionPtr CreateCollision** (const std::string &_shapeType, **LinkPtr** _link)=0

  Create a collision.
- **CollisionPtr CreateCollision** (const std::string &_shapeType, const std::string &_linkName)

  Create a collision.
- virtual **JointPtr CreateJoint** (const std::string &_type, **ModelPtr** _parent=**ModelPtr**())=0

  Create a new joint.
- virtual **LinkPtr CreateLink** (**ModelPtr** _parent)=0

  Create a new body.
- virtual **ShapePtr CreateShape** (const std::string &_shapeType, **CollisionPtr** _collision)=0

  *Create a **physics::Shape** (p. 754) object.*
- virtual void **DebugPrint** () const =0

  Debug print out of the physic engine state.
- virtual void **Fini** ()

  Finilize the physics engine.
- virtual bool **GetAutoDisableFlag** ()

  : Remove this function, and replace it with a more generic property map
- **ContactManager** * **GetContactManager** () const

  Get a pointer to the contact manger.
- virtual double **GetContactMaxCorrectingVel** ()

  : Remove this function, and replace it with a more generic property map.
- virtual double **GetContactSurfaceLayer** ()

  : Remove this function, and replace it with a more generic property map.
- virtual **math::Vector3 GetGravity** () const

  Return the gavity vector.
- virtual int **GetMaxContacts** ()

  : Remove this function, and replace it with a more generic property map.
- double **GetMaxStepSize** () const

  Get max step size.
- virtual boost::any **GetParam** (std::string _key) const

  Get an parameter of the physics engine.
- boost::recursive_mutex * **GetPhysicsUpdateMutex** () const

  *returns a pointer to the **PhysicsEngine::physicsUpdateMutex** (p. 610).*
- double **GetRealTimeUpdateRate** () const

  Get real time update rate.
- virtual int **GetSORPGSIters** ()

  : Remove this function, and replace it with a more generic property map
- virtual int **GetSORPGSPreconIters** ()

  : Remove this function, and replace it with a more generic property map
- virtual double **GetSORPGSW** ()

  : Remove this function, and replace it with a more generic property map.
- virtual double **GetStepTime** () **GAZEBO_DEPRECATED**(1.5)

Get the simulation step time.

- double **GetTargetRealTimeFactor** () const

  Get target real time factor.

- virtual std::string **GetType** () const =0

  Return the type of the physics engine (ode|bullet).

- double **GetUpdatePeriod** ()

  Get the simulation update period.

- double **GetUpdateRate** () **GAZEBO_DEPRECATED**(1.5)

  Get the simulation update rate.

- virtual double **GetWorldCFM** ()

  : Remove this function, and replace it with a more generic property map

- virtual double **GetWorldERP** ()

  : Remove this function, and replace it with a more generic property map

- virtual void **Init** ()=0

  Initialize the physics engine.

- virtual void **InitForThread** ()=0

  Init the engine for threads.

- virtual void **Load** (**sdf::ElementPtr** _sdf)

  Load the physics engine.

- virtual void **Reset** ()

  Rest the physics engine.

- virtual void **SetAutoDisableFlag** (bool _autoDisable)

  : Remove this function, and replace it with a more generic property map

- virtual void **SetContactMaxCorrectingVel** (double _vel)

  : Remove this function, and replace it with a more generic property map

- virtual void **SetContactSurfaceLayer** (double _layerDepth)

  : Remove this function, and replace it with a more generic property map

- virtual void **SetGravity** (const **gazebo::math::Vector3** &_gravity)=0

  Set the gavity vector.

- virtual void **SetMaxContacts** (double _maxContacts)

  : Remove this function, and replace it with a more generic property map

- void **SetMaxStepSize** (double _stepSize)

  Set max step size.

- virtual void **SetParam** (std::string _key, const boost::any &_value)

  Set a parameter of the physics engine.

- void **SetRealTimeUpdateRate** (double _rate)

  Set real time update rate.

- virtual void **SetSeed** (uint32_t _seed)=0

  Set the random number seed for the physics engine.

- virtual void **SetSORPGSIters** (unsigned int _iters)

  : Remove this function, and replace it with a more generic property map

- virtual void **SetSORPGSPreconIters** (unsigned int _iters)

  : Remove this function, and replace it with a more generic property map

- virtual void **SetSORPGSW** (double _w)

  : Remove this function, and replace it with a more generic property map

- virtual void **SetStepTime** (double _value) **GAZEBO_DEPRECATED**(1.5)

  Set the simulation step time.

- void **SetTargetRealTimeFactor** (double _factor)

  Set target real time factor.
- void **SetUpdateRate** (double _value) **GAZEBO_DEPRECATED**(1.5)

  Set the simulation update rate.
- virtual void **SetWorldCFM** (double _cfm)

  : Remove this function, and replace it with a more generic property map
- virtual void **SetWorldERP** (double _erp)

  : Remove this function, and replace it with a more generic property map
- virtual void **UpdateCollision** ()=0

  Update the physics engine collision.
- virtual void **UpdatePhysics** ()

  Update the physics engine.

Protected Member Functions

- virtual void **OnPhysicsMsg** (ConstPhysicsPtr &_msg)

  virtual callback for gztopic "~/physics".
- virtual void **OnRequest** (ConstRequestPtr &_msg)

  virtual callback for gztopic "~/request".

Protected Attributes

- **ContactManager** * **contactManager**

  Class that handles all contacts generated by the physics engine.
- double **maxStepSize**

  Real time update rate.
- **transport::NodePtr node**

  Node for communication.
- **transport::SubscriberPtr physicsSub**

  Subscribe to the physics topic.
- boost::recursive_mutex * **physicsUpdateMutex**

  Mutex to protect the update cycle.
- double **realTimeUpdateRate**

  Real time update rate.
- **transport::SubscriberPtr requestSub**

  Subscribe to the request topic.
- **transport::PublisherPtr responsePub**

  Response publisher.
- **sdf::ElementPtr sdf**

  Our SDF values.
- double **targetRealTimeFactor**

  Target real time factor.
- **WorldPtr world**

  Pointer to the world.

10.108.1 Detailed Description

Base (p. 141) class for a physics engine.

10.108.2 Constructor & Destructor Documentation

10.108.2.1 gazebo::physics::PhysicsEngine::PhysicsEngine (WorldPtr *_world*) [explicit]

Default constructor.

Parameters

in	*_world*	Pointer to the world.

10.108.2.2 virtual gazebo::physics::PhysicsEngine::~PhysicsEngine ( ) [virtual]

Destructor.

10.108.3 Member Function Documentation

10.108.3.1 virtual CollisionPtr gazebo::physics::PhysicsEngine::CreateCollision (const std::string & *_shapeType,* LinkPtr *_link*) [pure virtual]

Create a collision.

Parameters

in	*_shapeType*	Type of collision to create.
in	*_link*	Parent link.

10.108.3.2 CollisionPtr gazebo::physics::PhysicsEngine::CreateCollision (const std::string & *_shapeType,* const std::string & *_linkName*)

Create a collision.

Parameters

in	*_shapeType*	Type of collision to create.
in	*_linkName*	Name of the parent link.

10.108.3.3 virtual JointPtr gazebo::physics::PhysicsEngine::CreateJoint (const std::string & *_type,* ModelPtr *_parent =* ModelPtr()) [pure virtual]

Create a new joint.

Parameters

in	*_type*	Type of joint to create.
in	*_parent*	**Model** (p. 516) parent.

10.108.3.4 virtual LinkPtr gazebo::physics::PhysicsEngine::CreateLink (ModelPtr *_parent*) [pure virtual]

Create a new body.

Parameters

in	*_parent*	Parent model for the link.

10.108.3.5 virtual ShapePtr gazebo::physics::PhysicsEngine::CreateShape (const std::string & *_shapeType,* CollisionPtr *_collision*) [pure virtual]

Create a **physics::Shape** (p. 754) object.

Parameters

in	*_shapeType*	Type of shape to create.
in	*_collision*	**Collision** (p. 199) parent.

10.108.3.6 virtual void gazebo::physics::PhysicsEngine::DebugPrint ( ) const [pure virtual]

Debug print out of the physic engine state.

10.108.3.7 virtual void gazebo::physics::PhysicsEngine::Fini ( ) [virtual]

Finilize the physics engine.

10.108.3.8 virtual bool gazebo::physics::PhysicsEngine::GetAutoDisableFlag ( ) [inline], [virtual]

: Remove this function, and replace it with a more generic property map

access functions to set ODE parameters..

Returns

Auto disable flag.

10.108.3.9 ContactManager* gazebo::physics::PhysicsEngine::GetContactManager ( ) const

Get a pointer to the contact manger.

Returns

Pointer to the contact manager.

10.108.3.10 virtual double gazebo::physics::PhysicsEngine::GetContactMaxCorrectingVel ( ) [inline], [virtual]

: Remove this function, and replace it with a more generic property map.

access functions to set ODE parameters.

Returns

Max correcting velocity.

10.108.3.11 virtual double gazebo::physics::PhysicsEngine::GetContactSurfaceLayer ( ) [inline], [virtual]

: Remove this function, and replace it with a more generic property map.

access functions to set ODE parameters.

Returns

Contact (p. 241) suerface layer depth.

10.108.3.12 virtual math::Vector3 gazebo::physics::PhysicsEngine::GetGravity ( ) const [virtual]

Return the gavity vector.

Returns

The gavity vector.

10.108.3.13 virtual int gazebo::physics::PhysicsEngine::GetMaxContacts ( ) [inline], [virtual]

: Remove this function, and replace it with a more generic property map.

access functions to set ODE parameters.

Returns

Maximum number of allows contacts.

10.108.3.14 double gazebo::physics::PhysicsEngine::GetMaxStepSize ( ) const

Get max step size.

Returns

Max step size.

10.108.3.15 virtual boost::any gazebo::physics::PhysicsEngine::GetParam (std::string *_key*) const [virtual]

Get an parameter of the physics engine.

Parameters

in	*_attr*	String key

Returns

The value of the parameter

10.108.3.16 boost::recursive_mutex* gazebo::physics::PhysicsEngine::GetPhysicsUpdateMutex ( ) const [inline]

returns a pointer to the **PhysicsEngine::physicsUpdateMutex** (p. 610).

Returns

Pointer to the physics mutex.

References physicsUpdateMutex.

10.108.3.17 double gazebo::physics::PhysicsEngine::GetRealTimeUpdateRate ( ) const

Get real time update rate.

Returns

Update rate

10.108.3.18 virtual int gazebo::physics::PhysicsEngine::GetSORPGSIters ( ) [inline], [virtual]

: Remove this function, and replace it with a more generic property map
access functions to set ODE parameters.

Returns

SORPGS iterations.

10.108.3.19 virtual int gazebo::physics::PhysicsEngine::GetSORPGSPreconIters ( ) [inline], [virtual]

: Remove this function, and replace it with a more generic property map
access functions to set ODE parameters.

Returns

SORPGS precondition iterations.

10.108.3.20 virtual double gazebo::physics::PhysicsEngine::GetSORPGSW ( ) [inline], [virtual]

: Remove this function, and replace it with a more generic property map.
access functions to set ODE parameters

Returns

SORPGSW value.

10.108.3.21 virtual double gazebo::physics::PhysicsEngine::GetStepTime ( ) [virtual]

Get the simulation step time.

This funciton is deprecated, use World::GetMaxStepSize.

Returns

Simulation step time.

10.108.3.22 double gazebo::physics::PhysicsEngine::GetTargetRealTimeFactor ( ) const

Get target real time factor.

Returns

Target real time factor

10.108.3.23 virtual std::string gazebo::physics::PhysicsEngine::GetType ( ) const [pure virtual]

Return the type of the physics engine (ode|bullet).

Returns

Type of the physics engine.

10.108.3.24 double gazebo::physics::PhysicsEngine::GetUpdatePeriod ( )

Get the simulation update period.

Returns

Simulation update period.

10.108.3.25 double gazebo::physics::PhysicsEngine::GetUpdateRate ( )

Get the simulation update rate.

This funciton is deprecated, use **PhysicsEngine::GetRealTimeUpdateRate** (p. 604).

Returns

Update rate.

10.108.3.26 virtual double gazebo::physics::PhysicsEngine::GetWorldCFM ( ) [inline], [virtual]

: Remove this function, and replace it with a more generic property map

Get **World** (p. 945) CFM.

Returns

World (p. 945) CFM.

10.108.3.27 virtual double gazebo::physics::PhysicsEngine::GetWorldERP ( ) [inline], [virtual]

: Remove this function, and replace it with a more generic property map

Get **World** (p. 945) ERP.

Returns

World (p. 945) ERP.

10.108.3.28 virtual void gazebo::physics::PhysicsEngine::Init ( ) [pure virtual]

Initialize the physics engine.

10.108.3.29 virtual void gazebo::physics::PhysicsEngine::InitForThread ( ) [pure virtual]

Init the engine for threads.

10.108.3.30 virtual void gazebo::physics::PhysicsEngine::Load (sdf::ElementPtr *_sdf*) [virtual]

Load the physics engine.

Parameters

in	*_sdf*	Pointer to the SDF parameters.

10.108.3.31 virtual void gazebo::physics::PhysicsEngine::OnPhysicsMsg (ConstPhysicsPtr & *_msg*) [protected], [virtual]

virtual callback for gztopic "~/physics".

Parameters

in	*_msg*	Physics message.

10.108.3.32 virtual void gazebo::physics::PhysicsEngine::OnRequest (ConstRequestPtr & *_msg*) [protected], [virtual]

virtual callback for gztopic "~/request".

Parameters

in	*_msg*	Request message.

10.108.3.33 virtual void gazebo::physics::PhysicsEngine::Reset ( ) [inline], [virtual]

Rest the physics engine.

10.108.3.34 virtual void gazebo::physics::PhysicsEngine::SetAutoDisableFlag (bool *_autoDisable*) [virtual]

: Remove this function, and replace it with a more generic property map

Access functions to set ODE parameters.

Parameters

in	*_autoDisable*	True to enable auto disabling of bodies.

10.108.3.35 virtual void gazebo::physics::PhysicsEngine::SetContactMaxCorrectingVel (double *_vel*) [virtual]

: Remove this function, and replace it with a more generic property map

Access functions to set ODE parameters.

Parameters

in	*_vel*	Max correcting velocity.

10.108.3.36 virtual void gazebo::physics::PhysicsEngine::SetContactSurfaceLayer (double *_layerDepth*) [virtual]

: Remove this function, and replace it with a more generic property map

Access functions to set ODE parameters.

Parameters

in	*_layerDepth*	Surface layer depth

10.108.3.37 virtual void gazebo::physics::PhysicsEngine::SetGravity (const gazebo::math::Vector3 & *_gravity*) [pure virtual]

Set the gavity vector.

Parameters

in	*_gravity*	New gravity vector.

10.108.3.38 virtual void gazebo::physics::PhysicsEngine::SetMaxContacts (double *_maxContacts*) [virtual]

: Remove this function, and replace it with a more generic property map

access functions to set ODE parameters

Parameters

in	*_maxContacts*	Maximum number of contacts.

10.108.3.39 void gazebo::physics::PhysicsEngine::SetMaxStepSize (double *_stepSize*)

Set max step size.

Parameters

in	*_stepSize*	Max step size.

10.108.3.40 virtual void gazebo::physics::PhysicsEngine::SetParam (std::string *_key*, const boost::any & *_value*) [virtual]

Set a parameter of the physics engine.

Parameters

in	*_key*	String key
in	*_value*	The value to set to

10.108.3.41 void gazebo::physics::PhysicsEngine::SetRealTimeUpdateRate (double *_rate*)

Set real time update rate.

Parameters

in	*_rate*	Update rate

10.108.3.42 virtual void gazebo::physics::PhysicsEngine::SetSeed (uint32_t *_seed*) [pure virtual]

Set the random number seed for the physics engine.

Parameters

in	*_seed*	The random number seed.

10.108.3.43 virtual void gazebo::physics::PhysicsEngine::SetSORPGSIters (unsigned int *_iters*) [virtual]

: Remove this function, and replace it with a more generic property map

Access functions to set ODE parameters.

Parameters

in	*_iter*	Number of iterations.

10.108.3.44 virtual void gazebo::physics::PhysicsEngine::SetSORPGSPreconIters (unsigned int *_iters*) [virtual]

: Remove this function, and replace it with a more generic property map

Access functions to set ODE parameters.

Parameters

in	*_iter*	Number of iterations.

10.108.3.45 virtual void gazebo::physics::PhysicsEngine::SetSORPGSW (double *_w*) [virtual]

: Remove this function, and replace it with a more generic property map

Access functions to set ODE parameters.

Parameters

in	*_w*	SORPGSW value.

10.108.3.46 virtual void gazebo::physics::PhysicsEngine::SetStepTime (double *_value*) [virtual]

Set the simulation step time.

This funciton is deprecated, use World::SetMaxStepSize.

Parameters

in	*_value*	Value of the step time.

10.108.3.47 void gazebo::physics::PhysicsEngine::SetTargetRealTimeFactor (double *_factor*)

Set target real time factor.

Parameters

in	*_factor*	Target real time factor

10.108.3.48 void gazebo::physics::PhysicsEngine::SetUpdateRate (double *_value*)

Set the simulation update rate.

This funciton is deprecated, use **PhysicsEngine::SetRealTimeUpdateRate** (p. 608).

Parameters

in	*_value*	Value of the update rate.

10.108.3.49 virtual void gazebo::physics::PhysicsEngine::SetWorldCFM (double *_cfm*) [virtual]

: Remove this function, and replace it with a more generic property map

Access functions to set ODE parameters.

Parameters

in	*_cfm*	Constraint force mixing.

10.108.3.50 virtual void gazebo::physics::PhysicsEngine::SetWorldERP (double *_erp*) [virtual]

: Remove this function, and replace it with a more generic property map

Access functions to set ODE parameters.

Parameters

in	*_erp*	Error reduction parameter.

10.108.3.51 virtual void gazebo::physics::PhysicsEngine::UpdateCollision ( ) [pure virtual]

Update the physics engine collision.

10.108.3.52 virtual void gazebo::physics::PhysicsEngine::UpdatePhysics ( ) [inline], [virtual]

Update the physics engine.

10.108.4 Member Data Documentation

10.108.4.1 ContactManager* gazebo::physics::PhysicsEngine::contactManager [protected]

Class that handles all contacts generated by the physics engine.

10.108.4.2 double gazebo::physics::PhysicsEngine::maxStepSize [protected]

Real time update rate.

10.108.4.3 transport::NodePtr gazebo::physics::PhysicsEngine::node [protected]

Node for communication.

10.108.4.4 transport::SubscriberPtr gazebo::physics::PhysicsEngine::physicsSub [protected]

Subscribe to the physics topic.

10.108.4.5 boost::recursive_mutex* gazebo::physics::PhysicsEngine::physicsUpdateMutex [protected]

Mutex to protect the update cycle.

Referenced by GetPhysicsUpdateMutex().

10.108.4.6 double gazebo::physics::PhysicsEngine::realTimeUpdateRate [protected]

Real time update rate.

10.108.4.7 transport::SubscriberPtr gazebo::physics::PhysicsEngine::requestSub [protected]

Subscribe to the request topic.

10.108.4.8 transport::PublisherPtr gazebo::physics::PhysicsEngine::responsePub [protected]

Response publisher.

10.108.4.9 sdf::ElementPtr gazebo::physics::PhysicsEngine::sdf [protected]

Our SDF values.

10.108.4.10 double gazebo::physics::PhysicsEngine::targetRealTimeFactor [protected]

Target real time factor.

10.108.4.11 WorldPtr gazebo::physics::PhysicsEngine::world [protected]

Pointer to the world.

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

- **PhysicsEngine.hh**

10.109 gazebo::physics::PhysicsFactory Class Reference

The physics factory instantiates different physics engines.

`#include <physics/physics.hh>`

Static Public Member Functions

- static bool **IsRegistered** (const std::string &_name)

  Check if a physics engine is registered.
- static **PhysicsEnginePtr NewPhysicsEngine** (const std::string &_className, **WorldPtr** _world)

  Create a new instance of a physics engine.
- static void **RegisterAll** ()

  Register everything.
- static void **RegisterPhysicsEngine** (std::string _className, **PhysicsFactoryFn** _factoryfn)

  Register a physics class.

10.109.1 Detailed Description

The physics factory instantiates different physics engines.

10.109.2 Member Function Documentation

10.109.2.1 static bool gazebo::physics::PhysicsFactory::IsRegistered (const std::string & *_name*) [static]

Check if a physics engine is registered.

Parameters

in	*_name*	Name of the physics engine.

Returns

True if physics engine is registered, false otherwise.

10.109.2.2 static PhysicsEnginePtr gazebo::physics::PhysicsFactory::NewPhysicsEngine (const std::string & *_className,* WorldPtr *_world*) [static]

Create a new instance of a physics engine.

Parameters

in	*_className*	Name of the physics class.
in	*_world*	**World** (p. 945) to pass to the created physics engine.

10.109.2.3 static void gazebo::physics::PhysicsFactory::RegisterAll ( ) [static]

Register everything.

10.109.2.4 static void gazebo::physics::PhysicsFactory::RegisterPhysicsEngine (std::string *_className,* PhysicsFactoryFn *_factoryfn*) [static]

Register a physics class.

Parameters

in	*_className*	Name of the physics class.
in	*_factoryfn*	Function pointer used to create a physics engine.

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

- **PhysicsFactory.hh**

10.110 gazebo::common::PID Class Reference

Generic **PID** (p. 613) controller class.

`#include <common/common.hh>`

Public Member Functions

- **PID** (double _p=0.0, double _i=0.0, double _d=0.0, double _imax=0.0, double _imin=0.0, double _cmdMax=0.0, double _cmdMin=0.0)

  Constructor, zeros out Pid values when created and initialize Pid-gains and integral term limits:[iMax:iMin]-[I1:I2].
- virtual ~**PID** ()

  Destructor.
- double **GetCmd** ()

  *Return current command for this **PID** (p. 613) controller.*
- void **GetErrors** (double &_pe, double &_ie, double &_de)

  *Return **PID** (p. 613) error terms for the controller.*
- void **Init** (double _p=0.0, double _i=0.0, double _d=0.0, double _imax=0.0, double _imin=0.0, double _cmdMax=0.0, double _cmdMin=0.0)

  Initialize PID-gains and integral term limits:[iMax:iMin]-[I1:I2].
- **PID** & **operator=** (const **PID** &_p)

  Assignment operator.
- void **Reset** ()

  Reset the errors and command.
- void **SetCmd** (double _cmd)

  *Set current target command for this **PID** (p. 613) controller.*
- void **SetCmdMax** (double _c)

  Set the maximum value for the command.
- void **SetCmdMin** (double _c)

  Set the maximum value for the command.
- void **SetDGain** (double _d)

  Set the derivtive Gain.
- void **SetIGain** (double _i)

  Set the integral Gain.
- void **SetIMax** (double _i)

  Set the integral upper limit.
- void **SetIMin** (double _i)

  Set the integral lower limit.
- void **SetPGain** (double _p)

  Set the proportional Gain.
- double **Update** (double _error, **common::Time** _dt)

  Update the Pid loop with nonuniform time step size.

10.110.1 Detailed Description

Generic **PID** (p. 613) controller class.

Generic proportiolnal-integral-derivative controller class that keeps track of PID-error states and control inputs given the state of a system and a user specified target state.

10.110.2 Constructor & Destructor Documentation

10.110.2.1 gazebo::common::PID::PID (double _p = 0.0, double _i = 0.0, double _d = 0.0, double _imax = 0.0, double _imin = 0.0, double _cmdMax = 0.0, double _cmdMin = 0.0)

Constructor, zeros out Pid values when created and initialize Pid-gains and integral term limits:[iMax:iMin]-[I1:I2].

Parameters

in	*_p*	The proportional gain.
in	*_i*	The integral gain.
in	*_d*	The derivative gain.
in	*_imax*	The integral upper limit.
in	*_imin*	The integral lower limit.

10.110.2.2 virtual gazebo::common::PID::~PID ( ) [virtual]

Destructor.

10.110.3 Member Function Documentation

10.110.3.1 double gazebo::common::PID::GetCmd ( )

Return current command for this **PID** (p. 613) controller.

Returns

the command value

10.110.3.2 void gazebo::common::PID::GetErrors (double & _pe, double & _ie, double & _de)

Return **PID** (p. 613) error terms for the controller.

Parameters

in	*_pe*	The proportional error.
in	*_ie*	The integral error.
in	*_de*	The derivative error.

10.110.3.3 void gazebo::common::PID::Init (double _p = 0.0, double _i = 0.0, double _d = 0.0, double _imax = 0.0, double _imin = 0.0, double _cmdMax = 0.0, double _cmdMin = 0.0)

Initialize PID-gains and integral term limits:[iMax:iMin]-[I1:I2].

Parameters

in	*_p*	The proportional gain.
in	*_i*	The integral gain.
in	*_d*	The derivative gain.
in	*_imax*	The integral upper limit.
in	*_imin*	The integral lower limit.

10.110.3.4 PID& gazebo::common::PID::operator= (const PID & _p) [inline]

Assignment operator.

Parameters

in	*_p*	a reference to a **PID** (p. 613) to assign values from

Returns

reference to this instance

References Reset().

10.110.3.5 void gazebo::common::PID::Reset ( )

Reset the errors and command.

Referenced by operator=().

10.110.3.6 void gazebo::common::PID::SetCmd (double _cmd)

Set current target command for this **PID** (p. 613) controller.

Parameters

in	*_cmd*	New command

10.110.3.7 void gazebo::common::PID::SetCmdMax (double _c)

Set the maximum value for the command.

Parameters

in	*_c*	The maximum value

10.110.3.8 void gazebo::common::PID::SetCmdMin (double _c)

Set the maximum value for the command.

Parameters

in	*_c*	The maximum value

10.110.3.9 void gazebo::common::PID::SetDGain (double _d)

Set the derivtive Gain.

Parameters

in	*_p*	dertivative gain value

10.110.3.10 void gazebo::common::PID::SetIGain (double *_i*)

Set the integral Gain.

Parameters

in	*_p*	integral gain value

10.110.3.11 void gazebo::common::PID::SetIMax (double *_i*)

Set the integral upper limit.

Parameters

in	*_p*	integral upper limit value

10.110.3.12 void gazebo::common::PID::SetIMin (double *_i*)

Set the integral lower limit.

Parameters

in	*_p*	integral lower limit value

10.110.3.13 void gazebo::common::PID::SetPGain (double *_p*)

Set the proportional Gain.

Parameters

in	*_p*	proportional gain value

10.110.3.14 double gazebo::common::PID::Update (double *_error,* common::Time *_dt*)

Update the Pid loop with nonuniform time step size.

Parameters

_in]	_error Error since last call (p_state - p_target).
_in]	_dt Change in time since last update call. Normally, this is called at every time step, The return value is an updated command to be passed to the object being controlled.

Returns

the command value

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

- **PID.hh**

10.111 gazebo::math::Plane Class Reference

A plane and related functions.

`#include <math/gzmath.hh>`

Public Member Functions

- **Plane** ()
  Constructor.
- **Plane** (const **Vector3** &_normal, double _offset=0.0)
  Constructor from a normal and a distanec.
- **Plane** (const **Vector3** &_normal, const **Vector2d** &_size, double _offset)
  Constructor.
- virtual ~**Plane** ()
  Destructor.
- double **Distance** (const **Vector3** &_origin, const **Vector3** &_dir) const
  Get distance to the plane give an origin and direction.
- **Plane** & **operator=** (const **Plane** &_p)
  Equal operator.
- void **Set** (const **Vector3** &_normal, const **Vector2d** &_size, double offset)
  Set the plane.

Public Attributes

- double **d**
  Plane *(*p. *617) offset.*
- **Vector3 normal**
  Plane *(*p. *617) normal.*
- **Vector2d size**
  Plane *(*p. *617) size.*

10.111.1 Detailed Description

A plane and related functions.

10.111.2 Constructor & Destructor Documentation

10.111.2.1 gazebo::math::Plane::Plane ( )

Constructor.

10.111.2.2 gazebo::math::Plane::Plane (const Vector3 & _normal, double _offset = 0.0)

Constructor from a normal and a distanec.

Parameters

in	*_normal*	The plane normal
in	*_offset*	Offset along the normal

10.111.2.3 gazebo::math::Plane::Plane (const Vector3 & _normal, const Vector2d & _size, double _offset)

Constructor.

Parameters

in	*_normal*	The plane normal
in	*_size*	Size of the plane
in	*_offset*	Offset along the normal

10.111.2.4 virtual gazebo::math::Plane::~Plane ( ) [virtual]

Destructor.

10.111.3 Member Function Documentation

10.111.3.1 double gazebo::math::Plane::Distance (const Vector3 & _origin, const Vector3 & _dir) const

Get distance to the plane give an origin and direction.

Parameters

in	*_origin*	the origin
in	*_dir*	a direction

Returns

the shortest distance

10.111.3.2 Plane& gazebo::math::Plane::operator= (const Plane & _p)

Equal operator.

Parameters

_p	another plane

Returns

itself

10.111.3.3 void gazebo::math::Plane::Set (const Vector3 & *_normal,* const Vector2d & *_size,* double *offset*)

Set the plane.

Parameters

in	*_normal*	The plane normal
in	*_size*	Size of the plane
in	*_offset*	Offset along the normal

10.111.4 Member Data Documentation

10.111.4.1 double gazebo::math::Plane::d

Plane (p. 617) offset.

10.111.4.2 Vector3 gazebo::math::Plane::normal

Plane (p. 617) normal.

10.111.4.3 Vector2d gazebo::math::Plane::size

Plane (p. 617) size.

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

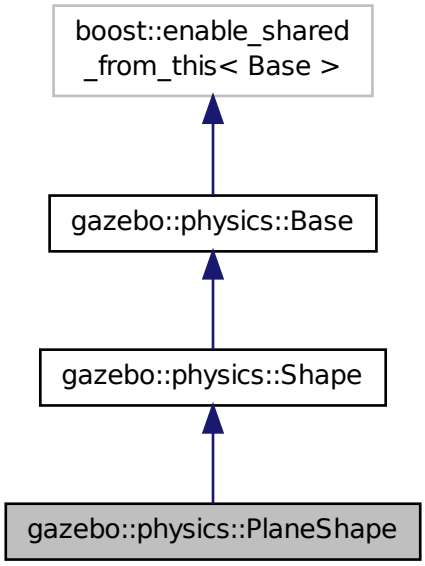
- **Plane.hh**

10.112 gazebo::physics::PlaneShape Class Reference

Collision (p. 199) for an infinite plane.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::PlaneShape:



Public Member Functions

- **PlaneShape** (**CollisionPtr** _parent)

  Constructor.
- virtual ~**PlaneShape** ()

  Destructor.
- virtual void **CreatePlane** ()

  Create the plane.
- void **FillMsg** (msgs::Geometry &_msg)

  Fill a geometry message with data from this object.
- **math::Vector3 GetNormal** () const

  Get the plane normal.
- **math::Vector2d GetSize** () const

  Get the size.
- virtual void **Init** ()

  Initialize the plane.
- virtual void **ProcessMsg** (const msgs::Geometry &_msg)

  Process a geometry message and use the data to update this object.
- virtual void **SetAltitude** (const **math::Vector3** &_pos)

  Set the altitude of the plane.
- void **SetNormal** (const **math::Vector3** &_norm)

  Set the normal.

- void **SetSize** (const **math::Vector2d** &_size)

  Set the size.

Additional Inherited Members

10.112.1 Detailed Description

Collision (p. 199) for an infinite plane.

This collision is used primarily for ground planes. Note that while the plane in infinite, only the part near the camera is drawn.

10.112.2 Constructor & Destructor Documentation

10.112.2.1 gazebo::physics::PlaneShape::PlaneShape (CollisionPtr *_parent*) [explicit]

Constructor.

Parameters

in	*_parent*	**Link** (p. 439) to which we are attached.

10.112.2.2 virtual gazebo::physics::PlaneShape::~PlaneShape ( ) [virtual]

Destructor.

10.112.3 Member Function Documentation

10.112.3.1 virtual void gazebo::physics::PlaneShape::CreatePlane ( ) [virtual]

Create the plane.

10.112.3.2 void gazebo::physics::PlaneShape::FillMsg (msgs::Geometry & *_msg*) [virtual]

Fill a geometry message with data from this object.

Parameters

out	*_msg*	Message to fill.

Implements **gazebo::physics::Shape** (p. 756).

10.112.3.3 math::Vector3 gazebo::physics::PlaneShape::GetNormal ( ) const

Get the plane normal.

Returns

The plane normal.

10.112.3.4 math::Vector2d gazebo::physics::PlaneShape::GetSize ( ) const

Get the size.

Returns

Size of the plane.

10.112.3.5 virtual void gazebo::physics::PlaneShape::Init ( ) [virtual]

Initialize the plane.

Implements **gazebo::physics::Shape** (p. 756).

10.112.3.6 virtual void gazebo::physics::PlaneShape::ProcessMsg (const msgs::Geometry & *_msg*) [virtual]

Process a geometry message and use the data to update this object.

Parameters

in	*_msg*	Message to update from.

Implements **gazebo::physics::Shape** (p. 756).

10.112.3.7 virtual void gazebo::physics::PlaneShape::SetAltitude (const math::Vector3 & *_pos*) [virtual]

Set the altitude of the plane.

Parameters

in	*_pos*	Position of the plane.

10.112.3.8 void gazebo::physics::PlaneShape::SetNormal (const math::Vector3 & *_norm*)

Set the normal.

Parameters

in	*_norm*	Plane normal.

10.112.3.9 void gazebo::physics::PlaneShape::SetSize (const math::Vector2d & *_size*)

Set the size.

Parameters

in	*_size*	2D size of the plane.

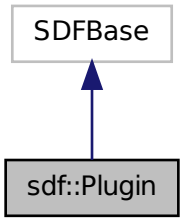
The documentation for this class was generated from the following file:

- **PlaneShape.hh**

10.113 sdf::Plugin Class Reference

`#include <Plugin.hh>`

Inheritance diagram for sdf::Plugin:



Public Member Functions

- **Plugin** ()
- void **Clear** ()
- void **Print** (const std::string &prefix)

Public Attributes

- std::vector< **ParamT** < std::string > > **data**
- **ParamT**< std::string > **filename**
- **ParamT**< std::string > **name**

10.113.1 Constructor & Destructor Documentation

10.113.1.1 sdf::Plugin::Plugin ( ) [inline]

10.113.2 Member Function Documentation

10.113.2.1 void sdf::Plugin::Clear ( ) [inline]

References data.

10.113.2.2 void sdf::Plugin::Print (const std::string & *prefix*) [inline]

References filename, and name.

10.113.3 Member Data Documentation

10.113.3.1 std::vector<ParamT<std::string> > sdf::Plugin::data

Referenced by Clear().

10.113.3.2 ParamT<std::string> sdf::Plugin::filename

Referenced by Print().

10.113.3.3 ParamT<std::string> sdf::Plugin::name

Referenced by Print().

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

- **sdf/interface/Plugin.hh**

10.114 gazebo::PluginT< T > Class Template Reference

A class which all plugins must inherit from.

`#include <common/common.hh>`

Public Types

- typedef boost::shared_ptr< T > **TPtr**

  plugin pointer type definition

Public Member Functions

- std::string **GetFilename** () const

  Get the name of the handler.
- std::string **GetHandle** () const

  Get the short name of the handler.
- **PluginType GetType** () const

  Returns the type of the plugin.

Static Public Member Functions

- static **TPtr Create** (const std::string &_filename, const std::string &_handle)

  a class method that creates a plugin from a file name.

Protected Attributes

- std::string **filename**

  Path to the shared library file.
- std::string **handle**

  Short name.
- **PluginType type**

  Type of plugin.

10.114.1 Detailed Description

template<class T>class gazebo::PluginT< T >

A class which all plugins must inherit from.

10.114.2 Member Typedef Documentation

10.114.2.1 template<class T> typedef boost::shared_ptr<T> gazebo::PluginT< T >::TPtr

plugin pointer type definition

10.114.3 Member Function Documentation

10.114.3.1 template<class T> static TPtr gazebo::PluginT< T >::Create (const std::string & *_filename,* const std::string & *_handle*) [inline], [static]

a class method that creates a plugin from a file name.

It locates the shared library and loads it dynamically.

Parameters

in	*_filename*	the path to the shared library.
in	*_handle*	short name of the handler

Returns

Shared Pointer to this class type

10.114.3.2 template<class T> std::string gazebo::PluginT< T >::GetFilename ( ) const [inline]

Get the name of the handler.

10.114.3.3 template<class T> std::string gazebo::PluginT< T >::GetHandle ( ) const [inline]

Get the short name of the handler.

10.114.3.4 template<class T> PluginType gazebo::PluginT< T >::GetType ( ) const [inline]

Returns the type of the plugin.

Returns

type of the plugin

10.114.4 Member Data Documentation

10.114.4.1 template<class T> std::string gazebo::PluginT< T >::filename [protected]

Path to the shared library file.

Referenced by gazebo::PluginT< ModelPlugin >::GetFilename().

10.114.4.2 template<class T> std::string gazebo::PluginT< T >::handle [protected]

Short name.

Referenced by gazebo::PluginT< ModelPlugin >::Create(), and gazebo::PluginT< ModelPlugin >::GetHandle().

10.114.4.3 template<class T> PluginType gazebo::PluginT< T >::type [protected]

Type of plugin.

Referenced by gazebo::PluginT< ModelPlugin >::GetType().

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

- **common/Plugin.hh**

10.115 gazebo::math::Pose Class Reference

Encapsulates a position and rotation in three space.

`#include <math/gzmath.hh>`

Public Member Functions

- **Pose** ()
  Default constructors.
- **Pose** (const **Vector3** &_pos, const **Quaternion** &_rot)
  Constructor.
- **Pose** (double _x, double _y, double _z, double _roll, double _pitch, double _yaw)
  Constructor.
- **Pose** (const **Pose** &_pose)
  Copy constructor.
- virtual ~**Pose** ()
  Destructor.

- **Pose CoordPoseSolve** (const **Pose** &_b) const

  Find the inverse of a pose; i.e., if b = this + a, given b and this, find a.
- **Vector3 CoordPositionAdd** (const **Vector3** &_pos) const

  Add one point to a vector: result = this + pos.
- **Vector3 CoordPositionAdd** (const **Pose** &_pose) const

  Add one point to another: result = this + pose.
- **Vector3 CoordPositionSub** (const **Pose** &_pose) const

  Subtract one position from another: result = this - pose.
- **Quaternion CoordRotationAdd** (const **Quaternion** &_rot) const

  Add one rotation to another: result = this->rot + rot.
- **Quaternion CoordRotationSub** (const **Quaternion** &_rot) const

  Subtract one rotation from another: result = this->rot - rot.
- void **Correct** ()

  Fix any nan values.
- **Pose GetInverse** () const

  Get the inverse of this pose.
- bool **IsFinite** () const

  See if a pose is finite (e.g., not nan)
- bool **operator!=** (const **Pose** &_pose) const

  Inequality operator.
- **Pose operator*** (const **Pose** &_pose)

  Multiplication operator.
- **Pose operator+** (const **Pose** &_pose) const

  Addition operator A is the transform from O to P specified in frame O B is the transform from P to Q specified in frame P then, B + A is the transform from O to Q specified in frame O.
- const **Pose** & **operator+=** (const **Pose** &_pose)

  Add-Equals operator.
- **Pose operator-** () const

  Negation operator A is the transform from O to P in frame O then -A is transform from P to O specified in frame P.
- **Pose operator-** (const **Pose** &_pose) const

  Subtraction operator A is the transform from O to P in frame O B is the transform from O to Q in frame O B - A is the transform from P to Q in frame P.
- const **Pose** & **operator-=** (const **Pose** &_pose)

  Subtraction operator.
- **Pose** & **operator=** (const **Pose** &_pose)

  Equal operator.
- bool **operator==** (const **Pose** &_pose) const

  Equality operator.
- void **Reset** ()

  Reset the pose.
- **Pose RotatePositionAboutOrigin** (const **Quaternion** &_rot) const

  Rotate vector part of a pose about the origin.
- void **Round** (int _precision)

  Round all values to _precision decimal places.
- void **Set** (const **Vector3** &_pos, const **Quaternion** &_rot)

  *Set the pose from a **Vector3** (p. 890) and a **Quaternion** (p. 654).*
- void **Set** (const **Vector3** &_pos, const **Vector3** &_rpy)

  Set the pose from pos and rpy vectors.
- void **Set** (double _x, double _y, double _z, double _roll, double _pitch, double _yaw)

  Set the pose from a six tuple.

Public Attributes

- **Vector3 pos**

  The position.
- **Quaternion rot**

  The rotation.

Static Public Attributes

- static const **Pose Zero**

  math::Pose(0, 0, 0, 0, 0, 0)

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::math::Pose** &_pose)

  Stream insertion operator.
- std::istream & **operator**>> (std::istream &_in, **gazebo::math::Pose** &_pose)

  Stream extraction operator.

10.115.1 Detailed Description

Encapsulates a position and rotation in three space.

10.115.2 Constructor & Destructor Documentation

10.115.2.1 gazebo::math::Pose::Pose ( )

Default constructors.

Referenced by operator-().

10.115.2.2 gazebo::math::Pose::Pose (const Vector3 & *_pos*, const Quaternion & *_rot*)

Constructor.

Parameters

in	*_pos*	A position
in	*_rot*	A rotation

10.115.2.3 gazebo::math::Pose::Pose (double *_x*, double *_y*, double *_z*, double *_roll*, double *_pitch*, double *_yaw*)

Constructor.

Parameters

in	*_x*	x position in meters.
in	*_y*	y position in meters.
in	*_z*	z position in meters.

in	*_roll*	Roll (rotation about X-axis) in radians.
in	*_pitch*	Pitch (rotation about y-axis) in radians.
in	*_yaw*	Yaw (rotation about z-axis) in radians.

10.115.2.4 gazebo::math::Pose::Pose (const Pose & *_pose*)

Copy constructor.

Parameters

in	*_pose*	**Pose** (p. 626) to copy

10.115.2.5 virtual gazebo::math::Pose::~Pose ( ) [virtual]

Destructor.

10.115.3 Member Function Documentation

10.115.3.1 Pose gazebo::math::Pose::CoordPoseSolve (const Pose & *_b*) const

Find the inverse of a pose; i.e., if b = this + a, given b and this, find a.

Parameters

in	*_b*	the other pose

10.115.3.2 Vector3 gazebo::math::Pose::CoordPositionAdd (const Vector3 & *_pos*) const

Add one point to a vector: result = this + pos.

Parameters

in	*_pos*	Position to add to this pose

Returns

the resulting position

10.115.3.3 Vector3 gazebo::math::Pose::CoordPositionAdd (const Pose & *_pose*) const

Add one point to another: result = this + pose.

Parameters

in	*_pose*	The **Pose** (p. 626) to add

Returns

The resulting position

10.115.3.4 Vector3 gazebo::math::Pose::CoordPositionSub (const Pose & *_pose*) const [inline]

Subtract one position from another: result = this - pose.

Parameters

in	*_pose*	**Pose** (p. 626) to subtract

Returns

The resulting position

References gazebo::math::Quaternion::GetInverse(), pos, rot, gazebo::math::Vector3::x, gazebo::math::Quaternion::x, gazebo::math::Vector3::y, gazebo::math::Quaternion::y, gazebo::math::Vector3::z, and gazebo::math::Quaternion::z.

Referenced by operator-().

10.115.3.5 Quaternion gazebo::math::Pose::CoordRotationAdd (const Quaternion & *_rot*) const

Add one rotation to another: result = this->rot + rot.

Parameters

in	*_rot*	Rotation to add

Returns

The resulting rotation

10.115.3.6 Quaternion gazebo::math::Pose::CoordRotationSub (const Quaternion & *_rot*) const [inline]

Subtract one rotation from another: result = this->rot - rot.

Parameters

in	*_rot*	The rotation to subtract

Returns

The resulting rotation

References gazebo::math::Quaternion::GetInverse(), gazebo::math::Quaternion::Normalize(), and rot.

Referenced by operator-().

10.115.3.7 void gazebo::math::Pose::Correct ( ) [inline]

Fix any nan values.

References gazebo::math::Vector3::Correct(), gazebo::math::Quaternion::Correct(), pos, and rot.

10.115.3.8 Pose gazebo::math::Pose::GetInverse ( ) const

Get the inverse of this pose.

Returns

the inverse pose

10.115.3.9 bool gazebo::math::Pose::IsFinite ( ) const

See if a pose is finite (e.g., not nan)

10.115.3.10 bool gazebo::math::Pose::operator!= (const Pose & _pose) const

Inequality operator.

Parameters

in	*_pose*	**Pose** (p. 626) for comparison

Returns

True if not equal

10.115.3.11 Pose gazebo::math::Pose::operator* (const Pose & _pose)

Multiplication operator.

Parameters

in	*_pose*	the other pose

Returns

itself

10.115.3.12 Pose gazebo::math::Pose::operator+ (const Pose & _pose) const

Addition operator A is the transform from O to P specified in frame O B is the transform from P to Q specified in frame P then, B + A is the transform from O to Q specified in frame O.

Parameters

in	*_pose*	**Pose** (p. 626) to add to this pose

Returns

The resulting pose

10.115.3.13 const Pose& gazebo::math::Pose::operator+= (const Pose & *_pose*)

Add-Equals operator.

Parameters

in	*_pose*	**Pose** (p. 626) to add to this pose

Returns

The resulting pose

10.115.3.14 Pose gazebo::math::Pose::operator- ( ) const [inline]

Negation operator A is the transform from O to P in frame O then -A is transform from P to O specified in frame P.

Returns

The resulting pose

References Pose().

10.115.3.15 Pose gazebo::math::Pose::operator- (const Pose & *_pose*) const [inline]

Subtraction operator A is the transform from O to P in frame O B is the transform from O to Q in frame O B - A is the transform from P to Q in frame P.

Parameters

in	*_pose*	**Pose** (p. 626) to subtract from this one

Returns

The resulting pose

References CoordPositionSub(), CoordRotationSub(), Pose(), and rot.

10.115.3.16 const Pose& gazebo::math::Pose::operator-= (const Pose & *_pose*)

Subtraction operator.

Parameters

in	*_pose*	**Pose** (p. 626) to subtract from this one

Returns

The resulting pose

10.115.3.17 Pose& gazebo::math::Pose::operator= (const Pose & _pose)

Equal operator.

Parameters

in	*_pose*	**Pose** (p. 626) to copy

10.115.3.18 bool gazebo::math::Pose::operator== (const Pose & _pose) const

Equality operator.

Parameters

in	*_pose*	**Pose** (p. 626) for comparison

Returns

True if equal

10.115.3.19 void gazebo::math::Pose::Reset ( )

Reset the pose.

10.115.3.20 Pose gazebo::math::Pose::RotatePositionAboutOrigin (const Quaternion & _rot) const

Rotate vector part of a pose about the origin.

Parameters

in	*_rot*	rotation

Returns

the rotated pose

10.115.3.21 void gazebo::math::Pose::Round (int _precision)

Round all values to _precision decimal places.

Parameters

in	*_precision*	

10.115.3.22 void gazebo::math::Pose::Set (const Vector3 & _pos, const Quaternion & _rot)

Set the pose from a **Vector3** (p. 890) and a **Quaternion** (p. 654).

Parameters

in	*_pos*	The position.
in	*_rot*	The rotation.

10.115.3.23 void gazebo::math::Pose::Set (const Vector3 & _pos, const Vector3 & _rpy)

Set the pose from pos and rpy vectors.

Parameters

in	*_pos*	The position.
in	*_rpy*	The rotation expressed as Euler angles.

10.115.3.24 void gazebo::math::Pose::Set (double _x, double _y, double _z, double _roll, double _pitch, double _yaw)

Set the pose from a six tuple.

Parameters

in	*_x*	x position in meters.
in	*_y*	y position in meters.
in	*_z*	z position in meters.
in	*_roll*	Roll (rotation about X-axis) in radians.
in	*_pitch*	Pitch (rotation about y-axis) in radians.
in	*_yaw*	Pitch (rotation about z-axis) in radians.

10.115.4 Friends And Related Function Documentation

10.115.4.1 std::ostream& operator<< (std::ostream & _out, const gazebo::math::Pose & _pose) [friend]

Stream insertion operator.

Parameters

in	*_out*	output stream
in	*_pose*	pose to output

Returns

the stream

10.115.4.2 std::istream& operator>> (std::istream & _in, gazebo::math::Pose & _pose) [friend]

Stream extraction operator.

Parameters

in	*_in*	the input stream
in	*_pose*	the pose

Returns

the stream

10.115.5 Member Data Documentation

10.115.5.1 Vector3 gazebo::math::Pose::pos

The position.

Referenced by CoordPositionSub(), Correct(), and gazebo::physics::Inertial::GetCoG().

10.115.5.2 Quaternion gazebo::math::Pose::rot

The rotation.

Referenced by CoordPositionSub(), CoordRotationSub(), Correct(), and operator-().

10.115.5.3 const Pose gazebo::math::Pose::Zero [static]

math::Pose(0, 0, 0, 0, 0, 0)

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

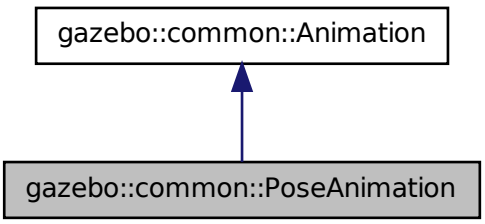
- **Pose.hh**

10.116 gazebo::common::PoseAnimation Class Reference

A pose animation.

`#include <Animation.hh>`

Inheritance diagram for gazebo::common::PoseAnimation:



Public Member Functions

- **PoseAnimation** (const std::string &_name, double _length, bool _loop)

  Constructor.
- virtual ∼**PoseAnimation** ()

  Destructor.
- **PoseKeyFrame** ∗ **CreateKeyFrame** (double _time)

  Create a pose keyframe at the given time.
- void **GetInterpolatedKeyFrame** (**PoseKeyFrame** &_kf) const

  Get a keyframe using the animation's current time.

Protected Member Functions

- void **BuildInterpolationSplines** () const

  Update the pose splines.
- void **GetInterpolatedKeyFrame** (double _time, **PoseKeyFrame** &_kf) const

  Get a keyframe using a passed in time.

Additional Inherited Members

10.116.1 Detailed Description

A pose animation.

10.116.2 Constructor & Destructor Documentation

10.116.2.1 gazebo::common::PoseAnimation::PoseAnimation (const std::string & *_name,* double *_length,* bool *_loop*)

Constructor.

Parameters

in	*_name*	String name of the animation. This should be unique.
in	*_length*	Length of the animation in seconds
in	*_loop*	True == loop the animation

10.116.2.2 virtual gazebo::common::PoseAnimation::∼PoseAnimation ( ) [virtual]

Destructor.

10.116.3 Member Function Documentation

10.116.3.1 void gazebo::common::PoseAnimation::BuildInterpolationSplines ( ) const [protected]

Update the pose splines.

10.116.3.2 PoseKeyFrame* gazebo::common::PoseAnimation::CreateKeyFrame (double _time)

Create a pose keyframe at the given time.

Parameters

in	*_time*	**Time** (p. 831) at which to create the keyframe

Returns

Pointer to the new keyframe

10.116.3.3 void gazebo::common::PoseAnimation::GetInterpolatedKeyFrame (PoseKeyFrame & _kf) const

Get a keyframe using the animation's current time.

Parameters

out	*_kf*	**PoseKeyFrame** (p. 637) reference to hold the interpolated result

10.116.3.4 void gazebo::common::PoseAnimation::GetInterpolatedKeyFrame (double _time, PoseKeyFrame & _kf) const [protected]

Get a keyframe using a passed in time.

Parameters

in	*_time*	**Time** (p. 831) in seconds
out	*_kf*	**PoseKeyFrame** (p. 637) reference to hold the interpolated result

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

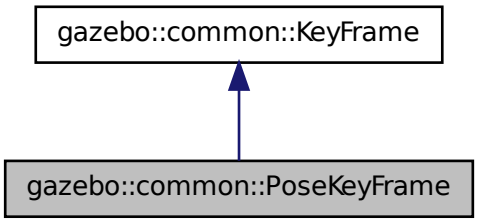
- **Animation.hh**

10.117 gazebo::common::PoseKeyFrame Class Reference

A keyframe for a **PoseAnimation** (p. 635).

`#include <KeyFrame.hh>`

Inheritance diagram for gazebo::common::PoseKeyFrame:



Public Member Functions

- **PoseKeyFrame** (double _time)

  Constructor.
- virtual ~**PoseKeyFrame** ()

  Destructor.
- const **math::Quaternion** & **GetRotation** () const

  Get the rotation of the keyframe.
- const **math::Vector3** & **GetTranslation** () const

  Get the translation of the keyframe.
- void **SetRotation** (const **math::Quaternion** &_rot)

  Set the rotation for the keyframe.
- void **SetTranslation** (const **math::Vector3** &_trans)

  Set the translation for the keyframe.

Protected Attributes

- **math::Quaternion rotate**

  the rotation quaternion
- **math::Vector3 translate**

  the translation vector

10.117.1 Detailed Description

A keyframe for a **PoseAnimation** (p. 635).

10.117.2 Constructor & Destructor Documentation

10.117.2.1 gazebo::common::PoseKeyFrame::PoseKeyFrame (double *_time*)

Constructor.

Parameters

in	*_time*	of the keyframe

10.117.2.2 virtual gazebo::common::PoseKeyFrame::~PoseKeyFrame ( ) [virtual]

Destructor.

10.117.3 Member Function Documentation

10.117.3.1 const math::Quaternion& gazebo::common::PoseKeyFrame::GetRotation ( ) const

Get the rotation of the keyframe.

Returns

The rotation amount

10.117.3.2 const math::Vector3& gazebo::common::PoseKeyFrame::GetTranslation ( ) const

Get the translation of the keyframe.

Returns

The translation amount

10.117.3.3 void gazebo::common::PoseKeyFrame::SetRotation (const math::Quaternion & *_rot*)

Set the rotation for the keyframe.

Parameters

in	*_rot*	Rotation amount

10.117.3.4 void gazebo::common::PoseKeyFrame::SetTranslation (const math::Vector3 & *_trans*)

Set the translation for the keyframe.

Parameters

in	*_trans*	Translation amount

10.117.4 Member Data Documentation

10.117.4.1 math::Quaternion gazebo::common::PoseKeyFrame::rotate [protected]

the rotation quaternion

10.117.4.2 math::Vector3 gazebo::common::PoseKeyFrame::translate [protected]

the translation vector

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

- **KeyFrame.hh**

10.118 gazebo::rendering::Projector Class Reference

Projects a material onto surface, light a light projector.

`#include <rendering/rendering.hh>`

Public Member Functions

- **Projector** (**VisualPtr** _parent)

  Constructor.
- virtual ∼**Projector** ()

  Destructor.
- **VisualPtr GetParent** ()

  Get the parent visual.
- void **Load** (**sdf::ElementPtr** _sdf)

  Load from an sdf pointer.
- void **Load** (const msgs::Projector &_msg)

  Load from a message.
- void **Load** (const std::string &_name, const **math::Pose** &_pose=**math::Pose**(0, 0, 0, 0, 0, 0), const std::string &_textureName="", double _nearClip=0.25, double _farClip=15.0, double _fov=M_PI ∗0.25)

  Load the projector.
- void **SetEnabled** (bool _enabled)

  Set whether the projector is enabled or disabled.
- void **SetTexture** (const std::string &_textureName)

  Load a texture into the projector.
- void **Toggle** ()

  Toggle the activation of the projector.

10.118.1 Detailed Description

Projects a material onto surface, light a light projector.

10.118.2 Constructor & Destructor Documentation

10.118.2.1 gazebo::rendering::Projector::Projector (VisualPtr _parent)

Constructor.

Parameters

in	*_parent*	Name of the parent visual.

10.118.2.2 virtual gazebo::rendering::Projector::~Projector ( ) [virtual]

Destructor.

10.118.3 Member Function Documentation

10.118.3.1 VisualPtr gazebo::rendering::Projector::GetParent ( )

Get the parent visual.

Returns

Pointer ot the parent visual.

10.118.3.2 void gazebo::rendering::Projector::Load (sdf::ElementPtr *_sdf*)

Load from an sdf pointer.

Parameters

in	*_sdf*	Pointer to the SDF element.

10.118.3.3 void gazebo::rendering::Projector::Load (const msgs::Projector & *_msg*)

Load from a message.

Parameters

in	*_msg*	Load from a message.

10.118.3.4 void gazebo::rendering::Projector::Load (const std::string & *_name*, const math::Pose & *_pose* = math::Pose(0, 0, 0, 0, 0, 0), const std::string & *_textureName* = "", double *_nearClip* = 0.25, double *_farClip* = 15.0, double *_fov* = M_PI *0.25)

Load the projector.

Parameters

in	*_name*	Name of the projector.
in	*_pos*	Pose of the projector.
in	*_textureName*	Name of the texture to project.
in	*_nearClip*	Near clip distance.
in	*_farClip*	Far clip distance.
in	*_fov*	Field of view.

10.118.3.5 void gazebo::rendering::Projector::SetEnabled (bool *_enabled*)

Set whether the projector is enabled or disabled.

Parameters

in	*_enabled*	True to enable the projector.

10.118.3.6 void gazebo::rendering::Projector::SetTexture (const std::string & *_textureName*)

Load a texture into the projector.

Parameters

in	*_textureName*	Name of the texture to project.

10.118.3.7 void gazebo::rendering::Projector::Toggle ( )

Toggle the activation of the projector.

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

- **Projector.hh**

10.119 gazebo::transport::Publication Class Reference

A publication for a topic.

`#include <transport/transport.hh>`

Public Member Functions

- **Publication** (const std::string &_topic, const std::string &_msgType)
  Constructor.
- virtual ~**Publication** ()
  Destructor.
- void **AddPublisher** (**PublisherPtr** _pub)
  Add a publisher.
- void **AddSubscription** (const **CallbackHelperPtr** _callback)
  Subscribe a callback to our topic.
- void **AddSubscription** (const **NodePtr** &_node)
  Subscribe a node to our topic.
- void **AddTransport** (const **PublicationTransportPtr** &_publink)
  Add a transport.
- unsigned int **GetCallbackCount** () const
  Get the number of callbacks.
- bool **GetLocallyAdvertised** () const
  Was the topic has been advertised from this process?
- std::string **GetMsgType** () const
  Get the type of message.
- unsigned int **GetNodeCount** () const
  Get the number of nodes.

- unsigned int **GetRemoteSubscriptionCount** ()

  Get the number of remote subscriptions.
- unsigned int **GetTransportCount** () const

  Get the number of transports.
- bool **HasTransport** (const std::string &_host, unsigned int _port)

  Does a given transport exist?
- void **LocalPublish** (const std::string &_data)

  Publish data to local subscribers (skip serialization)
- void **Publish** (**MessagePtr** _msg, boost::function< void(uint32_t)> _cb, uint32_t _id)

  Publish data to remote subscribers.
- void **RemoveSubscription** (const **NodePtr** &_node)

  Unsubscribe a node from our topic.
- void **RemoveSubscription** (const std::string &_host, unsigned int _port)

  Unsubscribe a a node by host/port from our topic.
- void **RemoveTransport** (const std::string &_host, unsigned int _port)

  Remove a transport.
- void **SetLocallyAdvertised** (bool _value)

  Set whether this topic has been advertised from this process.

10.119.1 Detailed Description

A publication for a topic.

This facilitates transport of messages

10.119.2 Constructor & Destructor Documentation

10.119.2.1 gazebo::transport::Publication::Publication (const std::string & *_topic*, const std::string & *_msgType*)

Constructor.

Parameters

in	*_topic*	The topic we're publishing
in	*_msgType*	The type of the topic we're publishing

10.119.2.2 virtual gazebo::transport::Publication::~Publication ( ) [virtual]

Destructor.

10.119.3 Member Function Documentation

10.119.3.1 void gazebo::transport::Publication::AddPublisher (PublisherPtr *_pub*)

Add a publisher.

Parameters

in,out	*_pub*	Pointer to publisher object to be added

10.119.3.2 void gazebo::transport::Publication::AddSubscription (const CallbackHelperPtr _callback)

Subscribe a callback to our topic.

Parameters

in	*_callback*	The callback

10.119.3.3 void gazebo::transport::Publication::AddSubscription (const NodePtr & _node)

Subscribe a node to our topic.

Parameters

in	*_node*	The node

10.119.3.4 void gazebo::transport::Publication::AddTransport (const PublicationTransportPtr & _publink)

Add a transport.

Parameters

in	*_publink*	Pointer to publication transport object to be added

10.119.3.5 unsigned int gazebo::transport::Publication::GetCallbackCount ( ) const

Get the number of callbacks.

Returns

The number of callbacks

10.119.3.6 bool gazebo::transport::Publication::GetLocallyAdvertised ( ) const

Was the topic has been advertised from this process?

Returns

true if the topic has been advertised from this process, false otherwise

10.119.3.7 std::string gazebo::transport::Publication::GetMsgType ( ) const

Get the type of message.

Returns

The type of message

10.119.3.8 unsigned int gazebo::transport::Publication::GetNodeCount ( ) const

Get the number of nodes.

Returns

The number of nodes

10.119.3.9 unsigned int gazebo::transport::Publication::GetRemoteSubscriptionCount ( )

Get the number of remote subscriptions.

Returns

The number of remote subscriptions

10.119.3.10 unsigned int gazebo::transport::Publication::GetTransportCount ( ) const

Get the number of transports.

Returns

The number of transports

10.119.3.11 bool gazebo::transport::Publication::HasTransport (const std::string & *_host,* unsigned int *_port*)

Does a given transport exist?

Parameters

in	*_host*	Hostname of the transport
in	*_port*	Port of the transport

Returns

true if the transport exists, false otherwise

10.119.3.12 void gazebo::transport::Publication::LocalPublish (const std::string & *_data*)

Publish data to local subscribers (skip serialization)

Parameters

in	*_data*	The data to be published

10.119.3.13 void gazebo::transport::Publication::Publish (MessagePtr _msg, boost::function< void(uint32_t)> _cb, uint32_t _id)

Publish data to remote subscribers.

Parameters

in	*_msg*	Message to be published
in	*_cb*	Callback to be invoked after publishing is completed

10.119.3.14 void gazebo::transport::Publication::RemoveSubscription (const NodePtr & _node)

Unsubscribe a node from our topic.

Parameters

in	*_node*	The node

10.119.3.15 void gazebo::transport::Publication::RemoveSubscription (const std::string & _host, unsigned int _port)

Unsubscribe a a node by host/port from our topic.

Parameters

in	*_host*	The node's hostname
in	*_port*	The node's port

10.119.3.16 void gazebo::transport::Publication::RemoveTransport (const std::string & _host, unsigned int _port)

Remove a transport.

Parameters

in	*_host*	The transport's hostname
in	*_port*	The transport's port

10.119.3.17 void gazebo::transport::Publication::SetLocallyAdvertised (bool _value)

Set whether this topic has been advertised from this process.

Parameters

in	*_value*	If true, the topic was locally advertise, otherwise it was not

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

- **Publication.hh**

10.120 gazebo::transport::PublicationTransport Class Reference

transport/transport.hh

```
#include <PublicationTransport.hh>
```

Public Member Functions

- **PublicationTransport** (const std::string &_topic, const std::string &_msgType)
  Constructor.
- virtual ~**PublicationTransport** ()
  Destructor.
- void **AddCallback** (const boost::function< void(const std::string &)> &_cb)
  Add a callback to the transport.
- void **Fini** ()
  Finalize the transport.
- const **ConnectionPtr GetConnection** () const
  Get the underlying connection.
- std::string **GetMsgType** () const
  Get the topic type.
- std::string **GetTopic** () const
  Get the topic name.
- void **Init** (const **ConnectionPtr** &_conn, bool _latched)
  Initialize the transport.

10.120.1 Detailed Description

transport/transport.hh

Reads data from a remote advertiser, and passes the data along to local subscribers

10.120.2 Constructor & Destructor Documentation

10.120.2.1 gazebo::transport::PublicationTransport::PublicationTransport (const std::string & *_topic*, const std::string & *_msgType*)

Constructor.

Parameters

in	*_topic*	Topic that we're publishing
in	*_topic*	Type of the topic that we're publishing

10.120.2.2 virtual gazebo::transport::PublicationTransport::~PublicationTransport ( ) [virtual]

Destructor.

10.120.3 Member Function Documentation

10.120.3.1 void gazebo::transport::PublicationTransport::AddCallback (const boost::function< void(const std::string &)> & _cb)

Add a callback to the transport.

Parameters

in	*_cb*	The callback to be added

10.120.3.2 void gazebo::transport::PublicationTransport::Fini ( )

Finalize the transport.

10.120.3.3 const ConnectionPtr gazebo::transport::PublicationTransport::GetConnection ( ) const

Get the underlying connection.

Returns

Pointer to the underlying connection

10.120.3.4 std::string gazebo::transport::PublicationTransport::GetMsgType ( ) const

Get the topic type.

Returns

The topic type

10.120.3.5 std::string gazebo::transport::PublicationTransport::GetTopic ( ) const

Get the topic name.

Returns

The topic name

10.120.3.6 void gazebo::transport::PublicationTransport::Init (const ConnectionPtr & _conn, bool _latched)

Initialize the transport.

Parameters

in	*_conn*	The underlying connection.
in	*_latched*	True to grab the last message sent on the topic.

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

- **PublicationTransport.hh**

10.121 gazebo::transport::Publisher Class Reference

A publisher of messages on a topic.

`#include <transport/transport.hh>`

Public Member Functions

- **Publisher** (const std::string &_topic, const std::string &_msgType, unsigned int _limit, bool _latch) **GAZEBO_DEPRECATED**(1.5)

  Deprecated.
- **Publisher** (const std::string &_topic, const std::string &_msgType, unsigned int _limit, double _hzRate)

  Constructor.
- virtual ∼**Publisher** ()

  Destructor.
- bool **GetLatching** () const **GAZEBO_DEPRECATED**(1.5)

  Deprecated.
- std::string **GetMsgType** () const

  Get the message type.
- unsigned int **GetOutgoingCount** () const

  Get the number of outgoing messages.
- std::string **GetPrevMsg** () const

  Get the previously published message.
- **MessagePtr GetPrevMsgPtr** () const

  Get the previously published message.
- std::string **GetTopic** () const

  Get the topic name.
- bool **HasConnections** () const

  Are there any connections?
- void **Publish** (const google::protobuf::Message &_message, bool _block=false)

  Publish a protobuf message on the topic.
- template<typename M >
  void **Publish** (M _message, bool _block=false)

  Publish an arbitrary message on the topic.
- void **SendMessage** ()

  Send latest message over the wire. For internal use only.
- void **SetNode** (**NodePtr** _node)

  Set our containing node.
- void **SetPublication** (**PublicationPtr** &_publication, int _i) **GAZEBO_DEPRECATED**(1.5)

  DEPRECATED in version 1.6.
- void **SetPublication** (**PublicationPtr** _publication)

  Set the publication object for a particular publication.
- void **WaitForConnection** () const

  Block until a connection has been established with this publisher.
- bool **WaitForConnection** (const **common::Time** &_timeout) const

  Block until a connection has been established with this publisher.

10.121.1 Detailed Description

A publisher of messages on a topic.

10.121.2 Constructor & Destructor Documentation

10.121.2.1 gazebo::transport::Publisher::Publisher (const std::string & *_topic,* const std::string & *_msgType,* unsigned int *_limit,* bool *_latch*)

Deprecated.

10.121.2.2 gazebo::transport::Publisher::Publisher (const std::string & *_topic,* const std::string & *_msgType,* unsigned int *_limit,* double *_hzRate*)

Constructor.

Parameters

in	*_topic*	Name of topic to be published
in	*_msgType*	Type of the message to be published
in	*_limit*	Maximum number of outgoing messages to queue
in	*_hz*	Update rate for the publisher. Units are 1.0/seconds.

10.121.2.3 virtual gazebo::transport::Publisher::~Publisher ( ) [virtual]

Destructor.

10.121.3 Member Function Documentation

10.121.3.1 bool gazebo::transport::Publisher::GetLatching ( ) const

Deprecated.

10.121.3.2 std::string gazebo::transport::Publisher::GetMsgType ( ) const

Get the message type.

Returns

The message type

10.121.3.3 unsigned int gazebo::transport::Publisher::GetOutgoingCount ( ) const

Get the number of outgoing messages.

Returns

The number of outgoing messages

10.121.3.4 std::string gazebo::transport::Publisher::GetPrevMsg ( ) const

Get the previously published message.

Returns

The previously published message, if any

10.121.3.5 MessagePtr gazebo::transport::Publisher::GetPrevMsgPtr ( ) const

Get the previously published message.

Returns

The previously published message, if any

10.121.3.6 std::string gazebo::transport::Publisher::GetTopic ( ) const

Get the topic name.

Returns

The topic name

10.121.3.7 bool gazebo::transport::Publisher::HasConnections ( ) const

Are there any connections?

Returns

true if there are any connections, false otherwise

10.121.3.8 void gazebo::transport::Publisher::Publish (const google::protobuf::Message & *_message*, bool *_block* = `false`) [inline]

Publish a protobuf message on the topic.

Parameters

in	*_message*	Message to be published
in	*_block*	Whether to block until the message is actually written out

10.121.3.9 template<typename M > void gazebo::transport::Publisher::Publish (M *_message*, bool *_block* = `false`) [inline]

Publish an arbitrary message on the topic.

Parameters

in	*_message*	Message to be published
in	*_block*	Whether to block until the message is actually written out

10.121.3.10 void gazebo::transport::Publisher::SendMessage ( )

Send latest message over the wire. For internal use only.

10.121.3.11 void gazebo::transport::Publisher::SetNode (NodePtr *_node*)

Set our containing node.

Parameters

in	*_node*	Pointer to a node. Should be the node that create this publisher.

10.121.3.12 void gazebo::transport::Publisher::SetPublication (PublicationPtr & *_publication,* int *_i*)

DEPRECATED in version 1.6.

See Also

SetPublication (p. 652)

10.121.3.13 void gazebo::transport::Publisher::SetPublication (PublicationPtr *_publication*)

Set the publication object for a particular publication.

Parameters

in	*_publication*	Pointer to the publication object to be set

10.121.3.14 void gazebo::transport::Publisher::WaitForConnection ( ) const

Block until a connection has been established with this publisher.

10.121.3.15 bool gazebo::transport::Publisher::WaitForConnection (const common::Time & *_timeout*) const

Block until a connection has been established with this publisher.

Parameters

in	*_timeout*	Maxiumum time to wait. Use a negative time value to wait forever.

Returns

True if a connection was established.

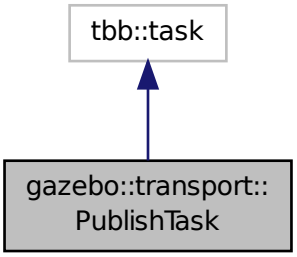
The documentation for this class was generated from the following file:

- **Publisher.hh**

10.122 gazebo::transport::PublishTask Class Reference

`#include <Node.hh>`

Inheritance diagram for gazebo::transport::PublishTask:



Public Member Functions

- **PublishTask** (**transport::PublisherPtr** _pub, const google::protobuf::Message &_message)

  Constructor.
- tbb::task * **execute** ()

  Overridden function from tbb::task that exectues the publish task.

10.122.1 Detailed Description

Task used by **Node::Publish** (p. 569) to publish on a one-time publisher

10.122.2 Constructor & Destructor Documentation

10.122.2.1 gazebo::transport::PublishTask::PublishTask (transport::PublisherPtr *_pub,* const google::protobuf::Message & *_message*) [inline]

Constructor.

Parameters

in	*_pub*	**Publisher** (p. 649) to publish the message on.
in	*_message*	Message to publish

10.122.3 Member Function Documentation

10.122.3.1 tbb::task* gazebo::transport::PublishTask::execute ( ) [inline]

Overridden function from tbb::task that exectues the publish task.

References NULL.

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

- **Node.hh**

10.123 gazebo::math::Quaternion Class Reference

A quaternion class.

`#include <math/gzmath.hh>`

Public Member Functions

- **Quaternion** ()

  Default Constructor.
- **Quaternion** (const double &_w, const double &_x, const double &_y, const double &_z)

  Constructor.
- **Quaternion** (const double &_roll, const double &_pitch, const double &_yaw)

  Constructor from Euler angles in radians.
- **Quaternion** (const **Vector3** &_axis, const double &_angle)

  Constructor from axis angle.
- **Quaternion** (const **Vector3** &_rpy)

  Constructor.
- **Quaternion** (const **Quaternion** &_qt)

  Copy constructor.
- ~**Quaternion** ()

  Destructor.
- void **Correct** ()

  Correct any nan.
- double **Dot** (const **Quaternion** &_q) const

  Dot product.
- void **GetAsAxis** (**Vector3** &_axis, double &_angle) const

  Return rotation as axis and angle.
- **Vector3 GetAsEuler** () const

  Return the rotation in Euler angles.
- **Matrix3 GetAsMatrix3** () const

  Get the quaternion as a 3x3 matrix.

- **Matrix4 GetAsMatrix4** () const

  Get the quaternion as a 4x4 matrix.
- **Quaternion GetExp** () const

  Return the exponent.
- **Quaternion GetInverse** () const

  Get the inverse of this quaternion.
- **Quaternion GetLog** () const

  Return the logarithm.
- double **GetPitch** ()

  Get the Euler pitch angle in radians.
- double **GetRoll** ()

  Get the Euler roll angle in radians.
- **Vector3 GetXAxis** () const

  Return the X axis.
- double **GetYaw** ()

  Get the Euler yaw angle in radians.
- **Vector3 GetYAxis** () const

  Return the Y axis.
- **Vector3 GetZAxis** () const

  Return the Z axis.
- void **Invert** ()

  Invert the quaternion.
- bool **IsFinite** () const

  See if a quatern is finite (e.g., not nan)
- void **Normalize** ()

  Normalize the quaternion.
- bool **operator!=** (const **Quaternion** &_qt) const

  Not equal to operator.
- **Quaternion operator*** (const **Quaternion** &_q) const

  Multiplication operator.
- **Quaternion operator*** (const double &_f) const

  Multiplication operator.
- **Vector3 operator*** (const **Vector3** &_v) const

  Vector3 *(p. 890) multiplication operator.*
- **Quaternion operator*=** (const **Quaternion** &qt)

  Multiplication operator.
- **Quaternion operator+** (const **Quaternion** &_qt) const

  Addition operator.
- **Quaternion operator+=** (const **Quaternion** &_qt)

  Addition operator.
- **Quaternion operator-** (const **Quaternion** &_qt) const

  Substraction operator.
- **Quaternion operator-** () const

  Unary minus operator.
- **Quaternion operator-=** (const **Quaternion** &_qt)

  Substraction operator.
- **Quaternion** & **operator=** (const **Quaternion** &_qt)

Equal operator.

- bool **operator==** (const **Quaternion** &_qt) const

  Equal to operator.
- **Vector3 RotateVector** (const **Vector3** &_vec) const

  Rotate a vector using the quaternion.
- **Vector3 RotateVectorReverse** (**Vector3** _vec) const

  Do the reverse rotation of a vector by this quaternion.
- void **Round** (int _precision)

  Round all values to _precision decimal places.
- void **Scale** (double _scale)

  Scale a Quaternionion.
- void **Set** (double _u, double _x, double _y, double _z)

  Set this quaternion from 4 floating numbers.
- void **SetFromAxis** (double _x, double _y, double _z, double _a)

  Set the quaternion from an axis and angle.
- void **SetFromAxis** (const **Vector3** &_axis, double _a)

  Set the quaternion from an axis and angle.
- void **SetFromEuler** (const **Vector3** &_vec)

  Set the quaternion from Euler angles.
- void **SetFromEuler** (double _roll, double _pitch, double _yaw)

  Set the quaternion from Euler angles.
- void **SetToIdentity** ()

  Set the quatern to the identity.

Static Public Member Functions

- static **Quaternion EulerToQuaternion** (const **Vector3** &_vec)

  Convert euler angles to quatern.
- static **Quaternion EulerToQuaternion** (double _x, double _y, double _z)

  Convert euler angles to quatern.
- static **Quaternion Slerp** (double _fT, const **Quaternion** &_rkP, const **Quaternion** &_rkQ, bool _shortestPath=false)

  Spherical linear interpolation between 2 quaternions, given the ends and an interpolation parameter between 0 and 1.
- static **Quaternion Squad** (double _fT, const **Quaternion** &_rkP, const **Quaternion** &_rkA, const **Quaternion** &_rkB, const **Quaternion** &_rkQ, bool _shortestPath=false)

  Spherical quadratic interpolation given the ends and an interpolation parameter between 0 and 1.

Public Attributes

- double **w**

  Attributes of the quaternion.
- double **x**

  Attributes of the quaternion.
- double **y**

  Attributes of the quaternion.
- double **z**

  Attributes of the quaternion.

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::math::Quaternion** &_q)

  Stream insertion operator.
- std::istream & **operator**>> (std::istream &_in, **gazebo::math::Quaternion** &_q)

  Stream extraction operator.

10.123.1 Detailed Description

A quaternion class.

10.123.2 Constructor & Destructor Documentation

10.123.2.1 gazebo::math::Quaternion::Quaternion ( )

Default Constructor.

Referenced by operator∗().

10.123.2.2 gazebo::math::Quaternion::Quaternion (const double & *_w,* const double & *_x,* const double & *_y,* const double & *_z*)

Constructor.

Parameters

in	*_w*	W param
in	*_x*	X param
in	*_y*	Y param
in	*_z*	Z param

10.123.2.3 gazebo::math::Quaternion::Quaternion (const double & *_roll,* const double & *_pitch,* const double & *_yaw*)

Constructor from Euler angles in radians.

Parameters

in	*_roll*	roll
in	*_pitch*	pitch
in	*_yaw*	yaw

10.123.2.4 gazebo::math::Quaternion::Quaternion (const Vector3 & *_axis,* const double & *_angle*)

Constructor from axis angle.

Parameters

in	*_axis*	the rotation axis
in	*_angle*	the rotation angle in radians

10.123.2.5 gazebo::math::Quaternion::Quaternion (const Vector3 & _rpy)

Constructor.

Parameters

in	*_rpy*	euler angles

10.123.2.6 gazebo::math::Quaternion::Quaternion (const Quaternion & _qt)

Copy constructor.

Parameters

qt	**Quaternion** (p. 654) to copy

10.123.2.7 gazebo::math::Quaternion::~Quaternion ( )

Destructor.

10.123.3 Member Function Documentation

10.123.3.1 void gazebo::math::Quaternion::Correct ( ) [inline]

Correct any nan.

References gazebo::math::equal(), w, x, y, and z.

Referenced by gazebo::math::Pose::Correct().

10.123.3.2 double gazebo::math::Quaternion::Dot (const Quaternion & _q) const

Dot product.

Parameters

in	*_q*	the other quaternion

Returns

the product

10.123.3.3 static Quaternion gazebo::math::Quaternion::EulerToQuaternion (const Vector3 & _vec) [static]

Convert euler angles to quatern.

Parameters

in		

10.123.3.4 static Quaternion gazebo::math::Quaternion::EulerToQuaternion (double _x, double _y, double _z) [static]

Convert euler angles to quatern.

Parameters

in	*_x*	rotation along x
in	*_y*	rotation along y
in	*_z*	rotation along z

10.123.3.5 void gazebo::math::Quaternion::GetAsAxis (Vector3 & _axis, double & _angle) const

Return rotation as axis and angle.

Parameters

in	*_axis*	rotation axis
in	*_angle*	ccw angle in radians

10.123.3.6 Vector3 gazebo::math::Quaternion::GetAsEuler ( ) const

Return the rotation in Euler angles.

Returns

This quaternion as an Euler vector

10.123.3.7 Matrix3 gazebo::math::Quaternion::GetAsMatrix3 ( ) const

Get the quaternion as a 3x3 matrix.

10.123.3.8 Matrix4 gazebo::math::Quaternion::GetAsMatrix4 ( ) const

Get the quaternion as a 4x4 matrix.

Returns

a 4x4 matrix

10.123.3.9 Quaternion gazebo::math::Quaternion::GetExp ( ) const

Return the exponent.

Returns

the exp

10.123.3.10 Quaternion gazebo::math::Quaternion::GetInverse ( ) const [inline]

Get the inverse of this quaternion.

Returns

Inverse quarenion

References gazebo::math::equal(), w, x, y, and z.

Referenced by gazebo::math::Pose::CoordPositionSub(), gazebo::math::Pose::CoordRotationSub(), and RotateVector().

10.123.3.11 Quaternion gazebo::math::Quaternion::GetLog ( ) const

Return the logarithm.

Returns

the log

10.123.3.12 double gazebo::math::Quaternion::GetPitch ( )

Get the Euler pitch angle in radians.

Returns

the pitch

10.123.3.13 double gazebo::math::Quaternion::GetRoll ( )

Get the Euler roll angle in radians.

Returns

the roll

10.123.3.14 Vector3 gazebo::math::Quaternion::GetXAxis ( ) const

Return the X axis.

Returns

the vector

10.123.3.15 double gazebo::math::Quaternion::GetYaw ( )

Get the Euler yaw angle in radians.

Returns

the yaw

10.123.3.16 Vector3 gazebo::math::Quaternion::GetYAxis ( ) const

Return the Y axis.

Returns

the vector

10.123.3.17 Vector3 gazebo::math::Quaternion::GetZAxis ( ) const

Return the Z axis.

Returns

the vector

10.123.3.18 void gazebo::math::Quaternion::Invert ( )

Invert the quaternion.

10.123.3.19 bool gazebo::math::Quaternion::IsFinite ( ) const

See if a quatern is finite (e.g., not nan)

Returns

True if quatern is finite

10.123.3.20 void gazebo::math::Quaternion::Normalize ( )

Normalize the quaternion.

Referenced by gazebo::math::Pose::CoordRotationSub().

10.123.3.21 bool gazebo::math::Quaternion::operator!= (const Quaternion & _qt) const

Not equal to operator.

Parameters

in	*_qt*	**Quaternion** (p. 654) for comparison

Returns

True if not equal

10.123.3.22 Quaternion gazebo::math::Quaternion::operator* (const Quaternion & _q) const [inline]

Multiplication operator.

Parameters

in	*_qt*	**Quaternion** (p. 654) for multiplication

Returns

This quaternion multiplied by the parameter

References Quaternion(), w, x, y, and z.

10.123.3.23 Quaternion gazebo::math::Quaternion::operator∗ (const double & _f) const

Multiplication operator.

Parameters

in	*_f*	factor

Returns

quaternion multiplied by _f

10.123.3.24 Vector3 gazebo::math::Quaternion::operator∗ (const Vector3 & _v) const

Vector3 (p. 890) multiplication operator.

Parameters

in	*_v*	vector to multiply

10.123.3.25 Quaternion gazebo::math::Quaternion::operator∗= (const Quaternion & *qt*)

Multiplication operator.

Parameters

in	*_qt*	**Quaternion** (p. 654) for multiplication

Returns

This quatern multiplied by the parameter

10.123.3.26 Quaternion gazebo::math::Quaternion::operator+ (const Quaternion & _qt) const

Addition operator.

Parameters

in	*_qt*	quaternion for addition

Returns

this quaternion + _qt

10.123.3.27 Quaternion gazebo::math::Quaternion::operator+= (const Quaternion & _qt)

Addition operator.

Parameters

in	*_qt*	quaternion for addition

Returns

this quaternion + qt

10.123.3.28 Quaternion gazebo::math::Quaternion::operator- (const Quaternion & _qt) const

Substraction operator.

Parameters

in	*_qt*	quaternion to substract

Returns

this quaternion - _qt

10.123.3.29 Quaternion gazebo::math::Quaternion::operator- ( ) const

Unary minus operator.

Returns

negates each component of the quaternion

10.123.3.30 Quaternion gazebo::math::Quaternion::operator-= (const Quaternion & _qt)

Substraction operator.

Parameters

in	*_qt*	**Quaternion** (p. 654) for substraction

Returns

This quatern - qt

10.123.3.31 Quaternion& gazebo::math::Quaternion::operator= (const Quaternion & _qt)

Equal operator.

Parameters

in	*_qt*	**Quaternion** (p. 654) to copy

10.123.3.32 bool gazebo::math::Quaternion::operator== (const Quaternion & _qt) const

Equal to operator.

Parameters

in	*_qt*	**Quaternion** (p. 654) for comparison

Returns

True if equal

10.123.3.33 Vector3 gazebo::math::Quaternion::RotateVector (const Vector3 & _vec) const [inline]

Rotate a vector using the quaternion.

Parameters

in	*_vec*	vector to rotate

Returns

the rotated vector

References GetInverse(), gazebo::math::Vector3::x, x, gazebo::math::Vector3::y, y, gazebo::math::Vector3::z, and z.

10.123.3.34 Vector3 gazebo::math::Quaternion::RotateVectorReverse (Vector3 _vec) const

Do the reverse rotation of a vector by this quaternion.

Parameters

in	*_vec*	the vector

Returns

the

10.123.3.35 void gazebo::math::Quaternion::Round (int _precision)

Round all values to _precision decimal places.

Parameters

in	*_precision*	the precision

10.123.3.36 void gazebo::math::Quaternion::Scale (double *_scale*)

Scale a Quaternionion.

Parameters

in	*_scale*	Amount to scale this rotation

10.123.3.37 void gazebo::math::Quaternion::Set (double *_u*, double *_x*, double *_y*, double *_z*)

Set this quaternion from 4 floating numbers.

Parameters

in	*_u*	u
in	*_x*	x
in	*_y*	y
in	*_z*	z

10.123.3.38 void gazebo::math::Quaternion::SetFromAxis (double *_x*, double *_y*, double *_z*, double *_a*)

Set the quaternion from an axis and angle.

Parameters

in	*_x*	X axis
in	*_y*	Y axis
in	*_z*	Z axis
in	*_a*	**Angle** (p. 121) in radians

10.123.3.39 void gazebo::math::Quaternion::SetFromAxis (const Vector3 & *_axis*, double *_a*)

Set the quaternion from an axis and angle.

Parameters

in	*_axis*	Axis
in	*_a*	**Angle** (p. 121) in radians

10.123.3.40 void gazebo::math::Quaternion::SetFromEuler (const Vector3 & *_vec*)

Set the quaternion from Euler angles.

The order of operations are roll, pitch, yaw.

Parameters

in	*vec*	Euler angle

10.123.3.41 void gazebo::math::Quaternion::SetFromEuler (double *_roll*, double *_pitch*, double *_yaw*)

Set the quaternion from Euler angles.

Parameters

in	*_roll*	Roll angle (radians).
in	*_pitch*	Roll angle (radians).
in	*_yaw*	Roll angle (radians).

10.123.3.42 void gazebo::math::Quaternion::SetToIdentity ( )

Set the quatern to the identity.

10.123.3.43 static Quaternion gazebo::math::Quaternion::Slerp (double *_fT*, const Quaternion & *_rkP*, const Quaternion & *_rkQ*, bool *_shortestPath* = `false`) [static]

Spherical linear interpolation between 2 quaternions, given the ends and an interpolation parameter between 0 and 1.

Parameters

in	*_ft*	the interpolation parameter
in	*_rkP*	the beginning quaternion
in	*_rkQ*	the end quaternion
in	*_shortestPath*	when true, the rotation may be inverted to get to minimize rotation

10.123.3.44 static Quaternion gazebo::math::Quaternion::Squad (double *_fT*, const Quaternion & *_rkP*, const Quaternion & *_rkA*, const Quaternion & *_rkB*, const Quaternion & *_rkQ*, bool *_shortestPath* = `false`) [static]

Spherical quadratic interpolation given the ends and an interpolation parameter between 0 and 1.

Parameters

in	*_ft*	the interpolation parameter
in	*_rkP*	the beginning quaternion
in	*_rkA*	first intermediate quaternion
in	*_rkB*	second intermediate quaternion
in	*_rkQ*	the end quaternion
in	*_shortestPath*	when true, the rotation may be inverted to get to minimize rotation

10.123.4 Friends And Related Function Documentation

10.123.4.1 std::ostream& operator<< (std::ostream & *_out*, const gazebo::math::Quaternion & *_q*) [friend]

Stream insertion operator.

Parameters

in	*_out*	output stream
in	*_q*	quaternion to output

Returns

the stream

10.123.4.2 std::istream& operator>> (std::istream & *_in,* gazebo::math::Quaternion & *_q*) [friend]

Stream extraction operator.

Parameters

in	*_in*	input stream
in	*_q*	**Quaternion** (p. 654) to read values into

Returns

The istream

10.123.5 Member Data Documentation

10.123.5.1 double gazebo::math::Quaternion::w

Attributes of the quaternion.

Referenced by Correct(), GetInverse(), and operator∗().

10.123.5.2 double gazebo::math::Quaternion::x

Attributes of the quaternion.

Referenced by gazebo::math::Pose::CoordPositionSub(), Correct(), GetInverse(), operator∗(), and RotateVector().

10.123.5.3 double gazebo::math::Quaternion::y

Attributes of the quaternion.

Referenced by gazebo::math::Pose::CoordPositionSub(), Correct(), GetInverse(), operator∗(), and RotateVector().

10.123.5.4 double gazebo::math::Quaternion::z

Attributes of the quaternion.

Referenced by gazebo::math::Pose::CoordPositionSub(), Correct(), GetInverse(), operator∗(), and RotateVector().

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

- **Quaternion.hh**

10.124 gazebo::math::Rand Class Reference

Random number generator class.

`#include <gzmath/gzmath.hh>`

Static Public Member Functions

- static double **GetDblNormal** (double _mean=0, double _sigma=1)

  Get a double from a normal distribution.
- static double **GetDblUniform** (double _min=0, double _max=1)

  Get a double from a uniform distribution.
- static int **GetIntNormal** (int _mean, int _sigma)

  Get a double from a normal distribution.
- static int **GetIntUniform** (int _min, int _max)

  Get a integer from a uniform distribution.
- static uint32_t **GetSeed** ()

  Get the seed value.
- static void **SetSeed** (uint32_t _seed)

  Set the seed value.

10.124.1 Detailed Description

Random number generator class.

10.124.2 Member Function Documentation

10.124.2.1 static double gazebo::math::Rand::GetDblNormal (double *_mean* = 0, double *_sigma* = 1) [static]

Get a double from a normal distribution.

Parameters

in	*_mean*	Mean value for the distribution
in	*_sigma*	Sigma value for the distribution

10.124.2.2 static double gazebo::math::Rand::GetDblUniform (double *_min* = 0, double *_max* = 1) [static]

Get a double from a uniform distribution.

Parameters

in	*_min*	Minimum bound for the random number
in	*_max*	Maximum bound for the random number

10.124.2.3 static int gazebo::math::Rand::GetIntNormal (int *_mean*, int *_sigma*) [static]

Get a double from a normal distribution.

Parameters

in	*_mean*	Mean value for the distribution
in	*_sigma*	Sigma value for the distribution

10.124.2.4 static int gazebo::math::Rand::GetIntUniform (int *_min*, int *_max*) [static]

Get a integer from a uniform distribution.

Parameters

in	*_min*	Minimum bound for the random number
in	*_max*	Maximum bound for the random number

10.124.2.5 static uint32_t gazebo::math::Rand::GetSeed ( ) [static]

Get the seed value.

Returns

The seed value used to initialize the random number generator.

10.124.2.6 static void gazebo::math::Rand::SetSeed (uint32_t *_seed*) [static]

Set the seed value.

Parameters

in	*_seed*	The seed used to initialize the randon number generator.

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

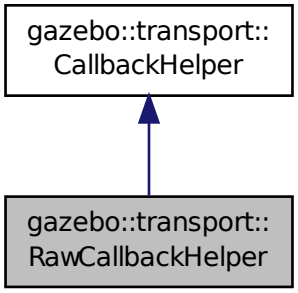
- **Rand.hh**

10.125 gazebo::transport::RawCallbackHelper Class Reference

Used to connect publishers to subscribers, where the subscriber wants the raw data from the publisher.

`#include <CallbackHelper.hh>`

Inheritance diagram for gazebo::transport::RawCallbackHelper:



Public Member Functions

- **RawCallbackHelper** (const boost::function< void(const std::string &)> &_cb, bool _latching=false)

  Constructor.
- std::string **GetMsgType** () const

  Get the typename of the message that is handled.
- virtual bool **HandleData** (const std::string &_newdata, boost::function< void(uint32_t)> _cb, uint32_t _id)

  Process new incoming data.
- virtual bool **HandleMessage** (**MessagePtr** _newMsg)

  Process new incoming message.
- virtual bool **IsLocal** () const

  Is the callback local?

Additional Inherited Members

10.125.1 Detailed Description

Used to connect publishers to subscribers, where the subscriber wants the raw data from the publisher.

Raw means that the data has not been converted into a protobuf message.

10.125.2 Constructor & Destructor Documentation

10.125.2.1 gazebo::transport::RawCallbackHelper::RawCallbackHelper (const boost::function< void(const std::string &)> & *_cb*, bool *_latching* = `false`) [inline]

Constructor.

Parameters

in	*_cb*	boost function to call on incoming messages
in	*_latching*	Set to true to make the callback helper latching.

10.125.3 Member Function Documentation

10.125.3.1 std::string gazebo::transport::RawCallbackHelper::GetMsgType ( ) const [inline], [virtual]

Get the typename of the message that is handled.

Returns

String representation of the message type

Reimplemented from **gazebo::transport::CallbackHelper** (p. 163).

10.125.3.2 virtual bool gazebo::transport::RawCallbackHelper::HandleData (const std::string & *_newdata,* boost::function< void(uint32_t)> *_cb,* uint32_t *_id*) [inline], [virtual]

Process new incoming data.

Parameters

in	*_newdata*	Incoming data to be processed

Returns

true if successfully processed; false otherwise

Parameters

in	*_cb*	If non-null, callback to be invoked which signals that transmission is complete.
in	*_id*	ID associated with the message data.

Implements **gazebo::transport::CallbackHelper** (p. 163).

10.125.3.3 virtual bool gazebo::transport::RawCallbackHelper::HandleMessage (MessagePtr *_newMsg*) [inline], [virtual]

Process new incoming message.

Parameters

in	*_newMsg*	Incoming message to be processed

Returns

true if successfully processed; false otherwise

Implements **gazebo::transport::CallbackHelper** (p. 163).

10.125.3.4 virtual bool gazebo::transport::RawCallbackHelper::IsLocal ( ) const [inline], [virtual]

Is the callback local?

Returns

true if the callback is local, false if the callback is tied to a remote connection

Implements **gazebo::transport::CallbackHelper** (p. 164).

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

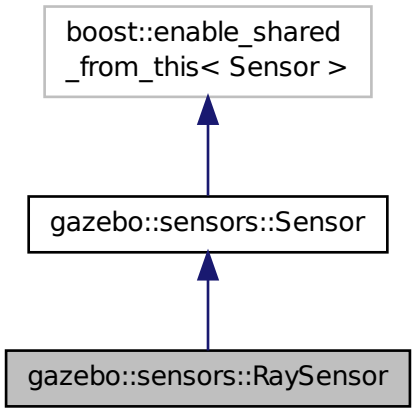
- **CallbackHelper.hh**

10.126 gazebo::sensors::RaySensor Class Reference

Sensor (p. 731) with one or more rays.

`#include <sensors/sensors.hh>`

Inheritance diagram for gazebo::sensors::RaySensor:



Public Member Functions

- **RaySensor** ()

  Constructor.
- virtual ~**RaySensor** ()

  Destructor.
- **math::Angle GetAngleMax** () const

  Get the maximum angle.
- **math::Angle GetAngleMin** () const

  Get the minimum angle.
- double **GetAngleResolution** () const

  Get the angle in radians between each range.
- int **GetFiducial** (int _index)

Get detected fiducial value for a ray.

- **physics::MultiRayShapePtr GetLaserShape** () const

  *Returns a pointer to the internal **physics::MultiRayShape** (p. 556).*

- double **GetRange** (int _index)

  Get detected range for a ray.

- int **GetRangeCount** () const

  Get the range count.

- double **GetRangeMax** () const

  Get the maximum range.

- double **GetRangeMin** () const

  Get the minimum range.

- double **GetRangeResolution** () const

  Get the range resolution.

- void **GetRanges** (std::vector< double > &_ranges)

  Get all the ranges.

- int **GetRayCount** () const

  Get the ray count.

- double **GetRetro** (int _index)

  Get detected retro (intensity) value for a ray.

- virtual std::string **GetTopic** () const

  Returns the topic name as set in SDF.

- **math::Angle GetVerticalAngleMax** () const

  Get the vertical scan line top angle.

- **math::Angle GetVerticalAngleMin** () const

  Get the vertical scan bottom angle.

- int **GetVerticalRangeCount** () const

  Get the vertical scan line count.

- int **GetVerticalRayCount** () const

  Get the vertical scan line count.

- virtual void **Init** ()

  Initialize the sensor.

- virtual bool **IsActive** ()

  Returns true if sensor generation is active.

- virtual void **Load** (const std::string &_worldName)

  Load the sensor with default parameters.

Protected Member Functions

- virtual void **Fini** ()

  Finalize the sensor.

- virtual void **UpdateImpl** (bool _force)

  This gets overwritten by derived sensor types.

Additional Inherited Members

10.126.1 Detailed Description

Sensor (p. 731) with one or more rays.

This sensor cast rays into the world, tests for intersections, and reports the range to the nearest object. It is used by ranging sensor models (e.g., sonars and scanning laser range finders).

10.126.2 Constructor & Destructor Documentation

10.126.2.1 gazebo::sensors::RaySensor::RaySensor ( )

Constructor.

10.126.2.2 virtual gazebo::sensors::RaySensor::~RaySensor ( ) [virtual]

Destructor.

10.126.3 Member Function Documentation

10.126.3.1 virtual void gazebo::sensors::RaySensor::Fini ( ) [protected], [virtual]

Finalize the sensor.

Reimplemented from **gazebo::sensors::Sensor** (p. 735).

10.126.3.2 math::Angle gazebo::sensors::RaySensor::GetAngleMax ( ) const

Get the maximum angle.

Returns

the maximum angle object

10.126.3.3 math::Angle gazebo::sensors::RaySensor::GetAngleMin ( ) const

Get the minimum angle.

Returns

The minimum angle object

10.126.3.4 double gazebo::sensors::RaySensor::GetAngleResolution ( ) const

Get the angle in radians between each range.

Returns

Resolution of the angle

10.126.3.5 int gazebo::sensors::RaySensor::GetFiducial (int _index)

Get detected fiducial value for a ray.

```
Warning: If you are accessing all the ray data in a loop
it's possible that the Ray will update in the middle of
your access loop. This means some data will come from one
scan, and some from another scan. You can solve this
problem by using SetActive(false) <your accessor loop>
SetActive(true).
```

Parameters

in	*_index*	Index value of specific ray

Returns

Fiducial value

10.126.3.6 physics::MultiRayShapePtr gazebo::sensors::RaySensor::GetLaserShape ( ) const [inline]

Returns a pointer to the internal **physics::MultiRayShape** (p. 556).

Returns

Pointer to ray shape

10.126.3.7 double gazebo::sensors::RaySensor::GetRange (int _index)

Get detected range for a ray.

```
Warning: If you are accessing all the ray data in a loop
it's possible that the Ray will update in the middle of
your access loop. This means some data will come from one
scan, and some from another scan. You can solve this
problem by using SetActive(false) <your accessor loop>
SetActive(true).
```

Parameters

in	*_index*	Index of specific ray

Returns

Returns DBL_MAX for no detection.

10.126.3.8 int gazebo::sensors::RaySensor::GetRangeCount ( ) const

Get the range count.

Returns

The number of ranges

10.126.3.9 double gazebo::sensors::RaySensor::GetRangeMax ( ) const

Get the maximum range.

Returns

The maximum range

10.126.3.10 double gazebo::sensors::RaySensor::GetRangeMin ( ) const

Get the minimum range.

Returns

The minimum range

10.126.3.11 double gazebo::sensors::RaySensor::GetRangeResolution ( ) const

Get the range resolution.

Returns

Resolution of the range

10.126.3.12 void gazebo::sensors::RaySensor::GetRanges (std::vector< double > & *_ranges*)

Get all the ranges.

Parameters

_ranges	A vector that will contain all the range data

10.126.3.13 int gazebo::sensors::RaySensor::GetRayCount ( ) const

Get the ray count.

Returns

The number of rays

10.126.3.14 double gazebo::sensors::RaySensor::GetRetro (int *_index*)

Get detected retro (intensity) value for a ray.

```
Warning: If you are accessing all the ray data in a loop
it's possible that the Ray will update in the middle of
your access loop. This means some data will come from one
scan, and some from another scan. You can solve this
problem by using SetActive(false) <your accessor loop>
SetActive(true).
```

Parameters

in	*_index*	Index of specific ray

Returns

Retro (intensity) value for ray

10.126.3.15 virtual std::string gazebo::sensors::RaySensor::GetTopic ( ) const [virtual]

Returns the topic name as set in SDF.

Returns

Topic name.

Reimplemented from **gazebo::sensors::Sensor** (p. 737).

10.126.3.16 math::Angle gazebo::sensors::RaySensor::GetVerticalAngleMax ( ) const

Get the vertical scan line top angle.

Returns

The Maximum angle of the scan block

10.126.3.17 math::Angle gazebo::sensors::RaySensor::GetVerticalAngleMin ( ) const

Get the vertical scan bottom angle.

Returns

The minimum angle of the scan block

10.126.3.18 int gazebo::sensors::RaySensor::GetVerticalRangeCount ( ) const

Get the vertical scan line count.

Returns

The number of scan lines vertically

10.126.3.19 int gazebo::sensors::RaySensor::GetVerticalRayCount ( ) const

Get the vertical scan line count.

Returns

The number of scan lines vertically

10.126.3.20 virtual void gazebo::sensors::RaySensor::Init ( ) [virtual]

Initialize the sensor.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.126.3.21 virtual bool gazebo::sensors::RaySensor::IsActive ( ) [virtual]

Returns true if sensor generation is active.

Returns

True if active, false if not.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.126.3.22 virtual void gazebo::sensors::RaySensor::Load (const std::string & *_worldName*) [virtual]

Load the sensor with default parameters.

Parameters

in	*_worldName*	Name of world to load from.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.126.3.23 virtual void gazebo::sensors::RaySensor::UpdateImpl (bool) [protected], [virtual]

This gets overwritten by derived sensor types.

```
This function is called during Sensor::Update.
And in turn, Sensor::Update is called by
SensorManager::Update
```

Parameters

in	*_force*	True if update is forced, false if not

Reimplemented from **gazebo::sensors::Sensor** (p. 739).

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

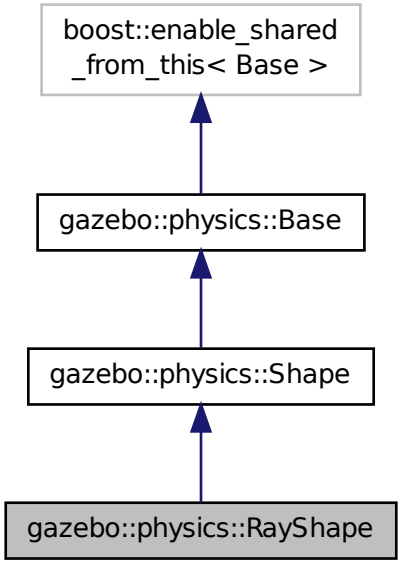
- **RaySensor.hh**

10.127 gazebo::physics::RayShape Class Reference

Base (p. 141) class for Ray collision geometry.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::RayShape:



Public Member Functions

- **RayShape** (**PhysicsEnginePtr** _physicsEngine)
  Constructor for a global ray.
- **RayShape** (**CollisionPtr** _parent)
  Constructor.
- virtual ~**RayShape** ()
  Destructor.
- void **FillMsg** (msgs::Geometry &_msg)
  Fill a message with data from this object.
- int **GetFiducial** () const
  Get the fiducial id detected by this ray.
- virtual void **GetGlobalPoints** (**math::Vector3** &_posA, **math::Vector3** &_posB)
  Get the global starting and ending points.
- virtual void **GetIntersection** (double &_dist, std::string &_entity)=0
  Get the nearest intersection.
- double **GetLength** () const
  Get the length of the ray.

- virtual void **GetRelativePoints** (**math::Vector3** &_posA, **math::Vector3** &_posB)

  Get the relative starting and ending points.
- float **GetRetro** () const

  Get the retro-reflectivness detected by this ray.
- virtual void **Init** ()

  In the ray.
- virtual void **ProcessMsg** (const msgs::Geometry &_msg)

  Update this shape from a message.
- void **SetFiducial** (int _fid)

  Set the fiducial id detected by this ray.
- virtual void **SetLength** (double _len)

  Set the length of the ray.
- virtual void **SetPoints** (const **math::Vector3** &_posStart, const **math::Vector3** &_posEnd)

  Set the ray based on starting and ending points relative to the body.
- void **SetRetro** (float _retro)

  Set the retro-reflectivness detected by this ray.
- virtual void **Update** ()=0

  Update the ray collision.

Protected Attributes

- int **contactFiducial**

  Fiducial ID value.
- double **contactLen**

  Length of the ray.
- double **contactRetro**

  Retro reflectance value.
- **math::Vector3 globalEndPos**

  End position of the ray in global cs.
- **math::Vector3 globalStartPos**

  Start position of the ray in global cs.
- **math::Vector3 relativeEndPos**

  End position of the ray, relative to the body.
- **math::Vector3 relativeStartPos**

  Start position of the ray, relative to the body.

Additional Inherited Members

10.127.1 Detailed Description

Base (p. 141) class for Ray collision geometry.

10.127.2 Constructor & Destructor Documentation

10.127.2.1 gazebo::physics::RayShape::RayShape (PhysicsEnginePtr _physicsEngine) [explicit]

Constructor for a global ray.

Parameters

in	*_physicsEngine*	Pointer to the physics engine.

10.127.2.2 gazebo::physics::RayShape::RayShape (CollisionPtr _parent) [explicit]

Constructor.

Parameters

in	*_parent*	**Collision** (p. 199) parent of the shape.

10.127.2.3 virtual gazebo::physics::RayShape::~RayShape ( ) [virtual]

Destructor.

10.127.3 Member Function Documentation

10.127.3.1 void gazebo::physics::RayShape::FillMsg (msgs::Geometry & _msg) [virtual]

Fill a message with data from this object.

Parameters

out	*_msg*	Message to fill. Implement this function.

Implements **gazebo::physics::Shape** (p. 756).

10.127.3.2 int gazebo::physics::RayShape::GetFiducial ( ) const

Get the fiducial id detected by this ray.

Returns

Fiducial id detected.

10.127.3.3 virtual void gazebo::physics::RayShape::GetGlobalPoints (math::Vector3 & _posA, math::Vector3 & _posB) [virtual]

Get the global starting and ending points.

Parameters

out	*_posA*	Returns the starting point.
out	*_posB*	Returns the ending point.

10.127.3.4 virtual void gazebo::physics::RayShape::GetIntersection (double & *_dist*, std::string & *_entity*) [pure virtual]

Get the nearest intersection.

Parameters

out	*_dist*	Distance to the intersection.
out	*_entity*	Name of the entity the ray intersected with.

10.127.3.5 double gazebo::physics::RayShape::GetLength ( ) const

Get the length of the ray.

Returns

The ray length.

10.127.3.6 virtual void gazebo::physics::RayShape::GetRelativePoints (math::Vector3 & *_posA*, math::Vector3 & *_posB*) [virtual]

Get the relative starting and ending points.

Parameters

in	*_posA*	Returns the starting point.
in	*_posB*	Returns the ending point.

10.127.3.7 float gazebo::physics::RayShape::GetRetro ( ) const

Get the retro-reflectivness detected by this ray.

Returns

Retro reflectance value.

10.127.3.8 virtual void gazebo::physics::RayShape::Init ( ) [virtual]

In the ray.

Implements **gazebo::physics::Shape** (p. 756).

10.127.3.9 virtual void gazebo::physics::RayShape::ProcessMsg (const msgs::Geometry & *_msg*) [virtual]

Update this shape from a message.

Parameters

in	*_msg*	Message to update from. Implement this function.

Implements **gazebo::physics::Shape** (p. 756).

10.127.3.10 void gazebo::physics::RayShape::SetFiducial (int *_fid*)

Set the fiducial id detected by this ray.

Parameters

in	*_fid*	Fiducial id detected by this ray.

10.127.3.11 virtual void gazebo::physics::RayShape::SetLength (double *_len*) [virtual]

Set the length of the ray.

Parameters

in	*_len*	Length of the array.

10.127.3.12 virtual void gazebo::physics::RayShape::SetPoints (const math::Vector3 & *_posStart*, const math::Vector3 & *_posEnd*) [virtual]

Set the ray based on starting and ending points relative to the body.

Parameters

in	*_posStart*	Start position, relative the body.
in	*_posEnd*	End position, relative to the body.

10.127.3.13 void gazebo::physics::RayShape::SetRetro (float *_retro*)

Set the retro-reflectivness detected by this ray.

Parameters

in	*_retro*	Retro reflectance value.

10.127.3.14 virtual void gazebo::physics::RayShape::Update ( ) [pure virtual]

Update the ray collision.

Reimplemented from **gazebo::physics::Base** (p. 152).

10.127.4 Member Data Documentation

10.127.4.1 int gazebo::physics::RayShape::contactFiducial [protected]

Fiducial ID value.

10.127.4.2 double gazebo::physics::RayShape::contactLen [protected]

Length of the ray.

10.127.4.3 double gazebo::physics::RayShape::contactRetro [protected]

Retro reflectance value.

10.127.4.4 math::Vector3 gazebo::physics::RayShape::globalEndPos [protected]

End position of the ray in global cs.

10.127.4.5 math::Vector3 gazebo::physics::RayShape::globalStartPos [protected]

Start position of the ray in global cs.

10.127.4.6 math::Vector3 gazebo::physics::RayShape::relativeEndPos [protected]

End position of the ray, relative to the body.

10.127.4.7 math::Vector3 gazebo::physics::RayShape::relativeStartPos [protected]

Start position of the ray, relative to the body.

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

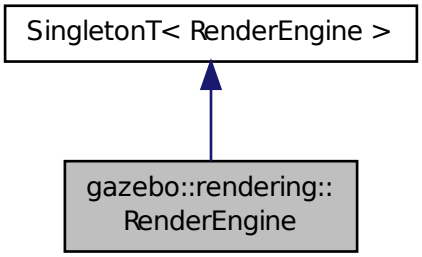
- **RayShape.hh**

10.128 gazebo::rendering::RenderEngine Class Reference

Adaptor to Ogre3d.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::RenderEngine:



Public Types

- enum **RenderPathType** {
  NONE = 0, **VERTEX** = 1, **FORWARD** = 2, **DEFERRED** = 3,
  RENDER_PATH_COUNT }

  The type of rendering path used by the rendering engine.

Public Member Functions

- void **AddResourcePath** (const std::string &_uri)

  *Add a new path for **Ogre** (p. 110) to search for resources.*
- **ScenePtr CreateScene** (const std::string &_name, bool _enableVisualizations)

  Create a scene.
- void **Fini** ()

  Tears down the rendering engine.
- **RenderPathType GetRenderPathType** () const

  Get the type of rendering path to use.
- **ScenePtr GetScene** (const std::string &_name="")

  Get a scene by name.
- **ScenePtr GetScene** (unsigned int _index)

  Get a scene by index.
- unsigned int **GetSceneCount** () const

  Get the number of scenes.
- **WindowManagerPtr GetWindowManager** () const

  Get a pointer to the window manager.
- void **Init** ()

  *Initialize **Ogre** (p. 110). Load must happen before Init.*
- void **Load** ()

  *Load the parameters for **Ogre** (p. 110). Load must happen before Init.*
- void **RemoveScene** (const std::string &_name)

  Remove a scene.

Public Attributes

- Ogre::Root * **root**

  Pointer to the root scene node.

Protected Attributes

- void * **dummyContext**

  GLX context used to render the scenes.Used for gui-less operation.
- void * **dummyDisplay**

  Pointer to the dummy display.Used for gui-less operation.
- uint64_t **dummyWindowId**

  ID for a dummy window. Used for gui-less operation.

Additional Inherited Members

10.128.1 Detailed Description

Adaptor to Ogre3d.

Provides the interface to load, initialize the rendering engine.

10.128.2 Member Enumeration Documentation

10.128.2.1 enum gazebo::rendering::RenderEngine::RenderPathType

The type of rendering path used by the rendering engine.

Enumerator:

NONE No rendering is done.

VERTEX Most basic rendering, with least fidelity.

FORWARD Utilizes the RTT shader system.

DEFERRED Utilizes deferred rendering. Best fidelity.

RENDER_PATH_COUNT Count of the rendering path enums.

10.128.3 Member Function Documentation

10.128.3.1 void gazebo::rendering::RenderEngine::AddResourcePath (const std::string & *_uri*)

Add a new path for **Ogre** (p. 110) to search for resources.

Parameters

in	*_uri*	URI of the path. The uri should be of the form `file://` or model://

10.128.3.2 ScenePtr gazebo::rendering::RenderEngine::CreateScene (const std::string & _name, bool _enableVisualizations)

Create a scene.

Parameters

in	*_name*	The name of the scene.
in	*_enable-Visualizations*	True enables visualization elements such as laser lines.

10.128.3.3 void gazebo::rendering::RenderEngine::Fini ( )

Tears down the rendering engine.

10.128.3.4 RenderPathType gazebo::rendering::RenderEngine::GetRenderPathType ( ) const

Get the type of rendering path to use.

This is automatically determined based on the computers capabilities

Returns

The RenderPathType

10.128.3.5 ScenePtr gazebo::rendering::RenderEngine::GetScene (const std::string & _name = " ")

Get a scene by name.

Parameters

in	*_name*	Name of the scene to retreive.

Returns

A pointer to the **Scene** (p. 707), or NULL if the scene doesn't exist.

10.128.3.6 ScenePtr gazebo::rendering::RenderEngine::GetScene (unsigned int _index)

Get a scene by index.

The index should be between 0 and **GetSceneCount()** (p. 688).

Parameters

in	*_index*	The index of the scene.

Returns

A pointer to a **Scene** (p. 707), or NULL if the index was invalid.

10.128.3.7 unsigned int gazebo::rendering::RenderEngine::GetSceneCount ( ) const

Get the number of scenes.

Returns

The number of scenes created by the **RenderEngine** (p. 684).

10.128.3.8 WindowManagerPtr gazebo::rendering::RenderEngine::GetWindowManager ( ) const

Get a pointer to the window manager.

Returns

Pointer to the window manager.

10.128.3.9 void gazebo::rendering::RenderEngine::Init ( )

Initialize **Ogre** (p. 110). Load must happen before Init.

10.128.3.10 void gazebo::rendering::RenderEngine::Load ( )

Load the parameters for **Ogre** (p. 110). Load must happen before Init.

10.128.3.11 void gazebo::rendering::RenderEngine::RemoveScene (const std::string & *_name*)

Remove a scene.

Parameters

in	*_name*	The name of the scene to remove.

10.128.4 Member Data Documentation

10.128.4.1 void* gazebo::rendering::RenderEngine::dummyContext [protected]

GLX context used to render the scenes.Used for gui-less operation.

10.128.4.2 void* gazebo::rendering::RenderEngine::dummyDisplay [protected]

Pointer to the dummy display.Used for gui-less operation.

10.128.4.3 uint64_t gazebo::rendering::RenderEngine::dummyWindowId [protected]

ID for a dummy window. Used for gui-less operation.

10.128.4.4 Ogre::Root* gazebo::rendering::RenderEngine::root

Pointer to the root scene node.

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

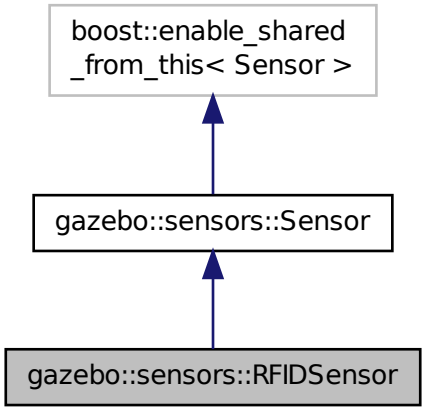
- **RenderEngine.hh**

10.129 gazebo::sensors::RFIDSensor Class Reference

Sensor (p. 731) class for RFID type of sensor.

`#include <sensors/sensors.hh>`

Inheritance diagram for gazebo::sensors::RFIDSensor:



Public Member Functions

- **RFIDSensor** ()
  Constructor.
- virtual ~**RFIDSensor** ()
  Destructor.
- void **AddTag** (**RFIDTag** *_tag)
- virtual void **Fini** ()
  Finalize the sensor.
- virtual void **Init** ()
  Initialize the sensor.
- virtual void **Load** (const std::string &_worldName, **sdf::ElementPtr** _sdf)
  Load the sensor with SDF parameters.
- virtual void **Load** (const std::string &_worldName)
  Load the sensor with default parameters.

Protected Member Functions

- virtual void **UpdateImpl** (bool _force)

  This gets overwritten by derived sensor types.

Additional Inherited Members

10.129.1 Detailed Description

Sensor (p. 731) class for RFID type of sensor.

10.129.2 Constructor & Destructor Documentation

10.129.2.1 gazebo::sensors::RFIDSensor::RFIDSensor ( )

Constructor.

10.129.2.2 virtual gazebo::sensors::RFIDSensor::~RFIDSensor ( ) [virtual]

Destructor.

10.129.3 Member Function Documentation

10.129.3.1 void gazebo::sensors::RFIDSensor::AddTag (RFIDTag * *_tag*)

10.129.3.2 virtual void gazebo::sensors::RFIDSensor::Fini ( ) [virtual]

Finalize the sensor.

Reimplemented from **gazebo::sensors::Sensor** (p. 735).

10.129.3.3 virtual void gazebo::sensors::RFIDSensor::Init ( ) [virtual]

Initialize the sensor.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.129.3.4 virtual void gazebo::sensors::RFIDSensor::Load (const std::string & *_worldName*, sdf::ElementPtr *_sdf*) [virtual]

Load the sensor with SDF parameters.

Parameters

in	*_sdf*	SDF **Sensor** (p. 731) parameters.
in	*_worldName*	Name of world to load from.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.129.3.5 virtual void gazebo::sensors::RFIDSensor::Load (const std::string & _worldName) [virtual]

Load the sensor with default parameters.

Parameters

in	*_worldName*	Name of world to load from.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.129.3.6 virtual void gazebo::sensors::RFIDSensor::UpdateImpl (bool) [protected], [virtual]

This gets overwritten by derived sensor types.

```
This function is called during Sensor::Update.
And in turn, Sensor::Update is called by
SensorManager::Update
```

Parameters

in	*_force*	True if update is forced, false if not

Reimplemented from **gazebo::sensors::Sensor** (p. 739).

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

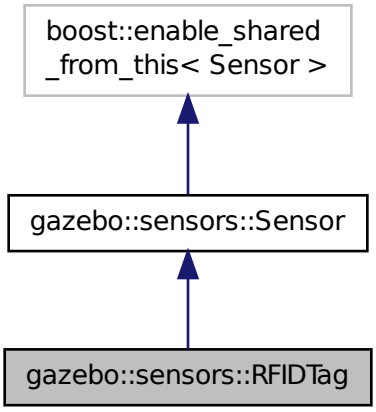
- **RFIDSensor.hh**

10.130 gazebo::sensors::RFIDTag Class Reference

RFIDTag (p. 691) to interact with RFIDTagSensors.

`#include <sensors/sensors.hh>`

Inheritance diagram for gazebo::sensors::RFIDTag:



Public Member Functions

- **RFIDTag** ()

  Constructor.
- virtual ~**RFIDTag** ()

  Destructor.
- virtual void **Fini** ()

  Finalize the sensor.
- **math::Pose GetTagPose** () const

  Returns pose of tag in world coordinate.
- virtual void **Init** ()

  Initialize the sensor.
- virtual void **Load** (const std::string &_worldName, **sdf::ElementPtr** &_sdf)
- virtual void **Load** (const std::string &_worldName)

  Load the sensor with default parameters.

Protected Member Functions

- virtual void **UpdateImpl** (bool _force)

  This gets overwritten by derived sensor types.

Additional Inherited Members

10.130.1 Detailed Description

RFIDTag (p. 691) to interact with RFIDTagSensors.

10.130.2 Constructor & Destructor Documentation

10.130.2.1 gazebo::sensors::RFIDTag::RFIDTag ( )

Constructor.

10.130.2.2 virtual gazebo::sensors::RFIDTag::~RFIDTag ( ) [virtual]

Destructor.

10.130.3 Member Function Documentation

10.130.3.1 virtual void gazebo::sensors::RFIDTag::Fini ( ) [virtual]

Finalize the sensor.

Reimplemented from **gazebo::sensors::Sensor** (p. 735).

10.130.3.2 math::Pose gazebo::sensors::RFIDTag::GetTagPose ( ) const [inline]

Returns pose of tag in world coordinate.

Returns

Pose of object.

10.130.3.3 virtual void gazebo::sensors::RFIDTag::Init ( ) [virtual]

Initialize the sensor.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.130.3.4 virtual void gazebo::sensors::RFIDTag::Load (const std::string & *_worldName,* sdf::ElementPtr & *_sdf*) [virtual]

10.130.3.5 virtual void gazebo::sensors::RFIDTag::Load (const std::string & *_worldName*) [virtual]

Load the sensor with default parameters.

Parameters

in	*_worldName*	Name of world to load from.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.130.3.6 virtual void gazebo::sensors::RFIDTag::UpdateImpl (bool) [protected], [virtual]

This gets overwritten by derived sensor types.

```
This function is called during Sensor::Update.
```

```
And in turn, Sensor::Update is called by
SensorManager::Update
```

Parameters

in	*_force*	True if update is forced, false if not

Reimplemented from **gazebo::sensors::Sensor** (p. 739).

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

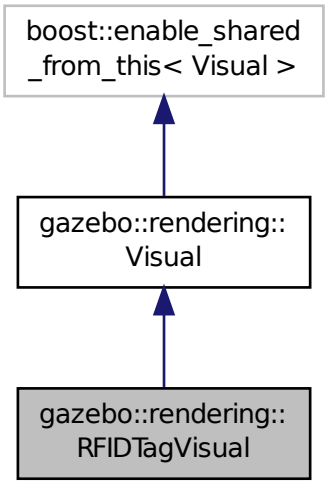
- **RFIDTag.hh**

10.131 gazebo::rendering::RFIDTagVisual Class Reference

Visualization for RFID tags sensor.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::RFIDTagVisual:



Public Member Functions

- **RFIDTagVisual** (const std::string &_name, **VisualPtr** _vis, const std::string &_topicName)

  Constructor.
- virtual ~**RFIDTagVisual** ()

  Destructor.

Additional Inherited Members

10.131.1 Detailed Description

Visualization for RFID tags sensor.

10.131.2 Constructor & Destructor Documentation

10.131.2.1 gazebo::rendering::RFIDTagVisual::RFIDTagVisual (const std::string & *_name,* VisualPtr *_vis,* const std::string & *_topicName*)

Constructor.

Parameters

in	*_name*	Name of the visual.
in	*_vis*	Parent visual.
in	*_topicName*	Name of the topic that publishes RFID data.

See Also

sensors::RFIDSensor (p. 689)

10.131.2.2 virtual gazebo::rendering::RFIDTagVisual::~RFIDTagVisual ( ) [virtual]

Destructor.

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

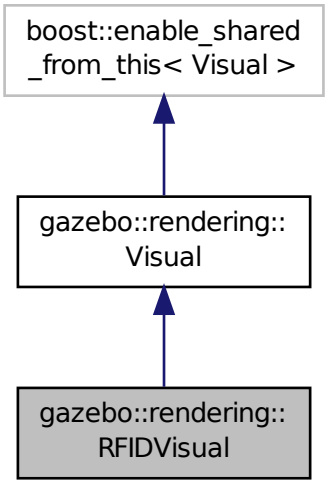
- **RFIDTagVisual.hh**

10.132 gazebo::rendering::RFIDVisual Class Reference

Visualization for RFID sensor.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::RFIDVisual:



Public Member Functions

- **RFIDVisual** (const std::string &_name, **VisualPtr** _vis, const std::string &_topicName)

  Constructor.
- virtual ~**RFIDVisual** ()

  Destructor.

Additional Inherited Members

10.132.1 Detailed Description

Visualization for RFID sensor.

10.132.2 Constructor & Destructor Documentation

10.132.2.1 gazebo::rendering::RFIDVisual::RFIDVisual (const std::string & *_name*, VisualPtr *_vis*, const std::string & *_topicName*)

Constructor.

Parameters

in	*_name*	Name of the **Visual** (p. 920).
in	*_vis*	Parent **Visual** (p. 920).
in	*_topicName*	Name of the topic which publishes RFID data.

10.132.2.2 virtual gazebo::rendering::RFIDVisual::~RFIDVisual ( ) [virtual]

Destructor.

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

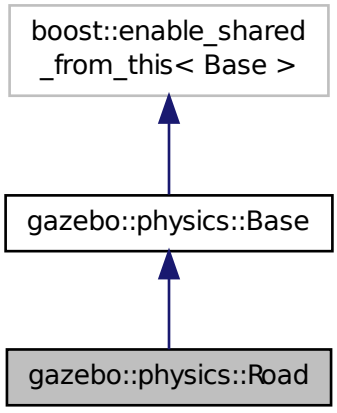
- **RFIDVisual.hh**

10.133 gazebo::physics::Road Class Reference

for building a **Road** (p. 697) from SDF

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::Road:



Public Member Functions

- **Road** (**BasePtr** _parent)

  Constructor.
- virtual ~**Road** ()

  Destructor.
- virtual void **Init** ()

  Initialize the road.
- void **Load** (**sdf::ElementPtr** _sdf)

  Load the road from SDF.

Additional Inherited Members

10.133.1 Detailed Description

for building a **Road** (p. 697) from SDF

10.133.2 Constructor & Destructor Documentation

10.133.2.1 gazebo::physics::Road::Road (BasePtr *_parent*) [explicit]

Constructor.

Parameters

in	*_parent*	Parent of this road object.

10.133.2.2 virtual gazebo::physics::Road::~Road ( ) [virtual]

Destructor.

10.133.3 Member Function Documentation

10.133.3.1 virtual void gazebo::physics::Road::Init ( ) [virtual]

Initialize the road.

Reimplemented from **gazebo::physics::Base** (p. 149).

10.133.3.2 void gazebo::physics::Road::Load (sdf::ElementPtr *_sdf*) [virtual]

Load the road from SDF.

Parameters

in	*_sdf*	SDF values to load from.

Reimplemented from **gazebo::physics::Base** (p. 149).

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

- **Road.hh**

10.134 Road Class Reference

Used to render a strip of road.

`#include <rendering/rendering.hh>`

10.134.1 Detailed Description

Used to render a strip of road.

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

- **Road2d.hh**

10.135 gazebo::rendering::Road2d Class Reference

`#include <Road2d.hh>`

Public Member Functions

- **Road2d** ()
  Constructor.
- virtual ~**Road2d** ()
  Destructor.
- void **Load** (**VisualPtr** _parent)
  Load the visual using a parent visual.

10.135.1 Constructor & Destructor Documentation

10.135.1.1 gazebo::rendering::Road2d::Road2d ( )

Constructor.

10.135.1.2 virtual gazebo::rendering::Road2d::~Road2d ( ) [virtual]

Destructor.

10.135.2 Member Function Documentation

10.135.2.1 void gazebo::rendering::Road2d::Load (VisualPtr *_parent*)

Load the visual using a parent visual.

Parameters

in	*_parent*	Pointer to the parent visual.

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

- **Road2d.hh**

10.136 gazebo::math::RotationSpline Class Reference

Spline (p. 793) for rotations.

`#include <math/gzmath.hh>`

Public Member Functions

- **RotationSpline** ()

  Constructor. Sets the autoCalc to true.
- ∼**RotationSpline** ()

  Destructor. Nothing is done.
- void **AddPoint** (const **Quaternion** &_p)

  Adds a control point to the end of the spline.
- void **Clear** ()

  Clears all the points in the spline.
- unsigned int **GetNumPoints** () const

  Gets the number of control points in the spline.
- const **Quaternion** & **GetPoint** (unsigned int _index) const

  Gets the detail of one of the control points of the spline.
- **Quaternion Interpolate** (double _t, bool _useShortestPath=true)

  Returns an interpolated point based on a parametric value over the whole series.
- **Quaternion Interpolate** (unsigned int _fromIndex, double _t, bool _useShortestPath=true)

  Interpolates a single segment of the spline given a parametric value.
- void **RecalcTangents** ()

  Recalculates the tangents associated with this spline.
- void **SetAutoCalculate** (bool _autoCalc)

  Tells the spline whether it should automatically calculate tangents on demand as points are added.
- void **UpdatePoint** (unsigned int _index, const **Quaternion** &_value)

  Updates a single point in the spline.

Protected Attributes

- bool **autoCalc**

  Automatic recalcultation of tangeants when control points are updated.
- std::vector< **Quaternion** > **points**

  the control points
- std::vector< **Quaternion** > **tangents**

  the tangents

10.136.1 Detailed Description

Spline (p. 793) for rotations.

10.136.2 Constructor & Destructor Documentation

10.136.2.1 gazebo::math::RotationSpline::RotationSpline ( )

Constructor. Sets the autoCalc to true.

10.136.2.2 gazebo::math::RotationSpline::∼RotationSpline ( )

Destructor. Nothing is done.

10.136.3 Member Function Documentation

10.136.3.1 void gazebo::math::RotationSpline::AddPoint (const Quaternion & _p)

Adds a control point to the end of the spline.

Parameters

in	*_p*	control point

10.136.3.2 void gazebo::math::RotationSpline::Clear ( )

Clears all the points in the spline.

10.136.3.3 unsigned int gazebo::math::RotationSpline::GetNumPoints ( ) const

Gets the number of control points in the spline.

Returns

the count

10.136.3.4 const Quaternion& gazebo::math::RotationSpline::GetPoint (unsigned int _index) const

Gets the detail of one of the control points of the spline.

Parameters

in	*_index*	the index of the control point.

Remarks

This point must already exist in the spline.

Returns

a quaternion (out of bound index result in assertion)

10.136.3.5 Quaternion gazebo::math::RotationSpline::Interpolate (double _t, bool _useShortestPath = true)

Returns an interpolated point based on a parametric value over the whole series.

Remarks

Given a t value between 0 and 1 representing the parametric distance along the whole length of the spline, this method returns an interpolated point.

Parameters

in	*_t*	Parametric value.
in	*_useShortestPath*	Defines if rotation should take the shortest possible path

Returns

the rotation

10.136.3.6 Quaternion gazebo::math::RotationSpline::Interpolate (unsigned int *_fromIndex,* double *_t,* bool *_useShortestPath =* `true`)

Interpolates a single segment of the spline given a parametric value.

Parameters

in	*_fromIndex*	The point index to treat as t = 0. _fromIndex + 1 is deemed to be t = 1
in	*_t*	Parametric value
in	*_useShortest-Path*	Defines if rotation should take the shortest possible path

Returns

the rotation

10.136.3.7 void gazebo::math::RotationSpline::RecalcTangents ( )

Recalculates the tangents associated with this spline.

Remarks

If you tell the spline not to update on demand by calling setAutoCalculate(false) then you must call this after completing your updates to the spline points.

10.136.3.8 void gazebo::math::RotationSpline::SetAutoCalculate (bool *_autoCalc*)

Tells the spline whether it should automatically calculate tangents on demand as points are added.

Remarks

The spline calculates tangents at each point automatically based on the input points. Normally it does this every time a point changes. However, if you have a lot of points to add in one go, you probably don't want to incur this overhead and would prefer to defer the calculation until you are finished setting all the points. You can do this by calling this method with a parameter of 'false'. Just remember to manually call the recalcTangents method when you are done.

Parameters

in	*_autoCalc*	If true, tangents are calculated for you whenever a point changes. If false, you must call reclacTangents to recalculate them when it best suits.

10.136.3.9 void gazebo::math::RotationSpline::UpdatePoint (unsigned int *_index,* const Quaternion & *_value*)

Updates a single point in the spline.

Remarks

This point must already exist in the spline.

Parameters

in	*_index*	index
in	*_value*	the new control point value

10.136.4 Member Data Documentation

10.136.4.1 bool gazebo::math::RotationSpline::autoCalc [protected]

Automatic recalcultation of tangeants when control points are updated.

10.136.4.2 std::vector<Quaternion> gazebo::math::RotationSpline::points [protected]

the control points

10.136.4.3 std::vector<Quaternion> gazebo::math::RotationSpline::tangents [protected]

the tangents

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

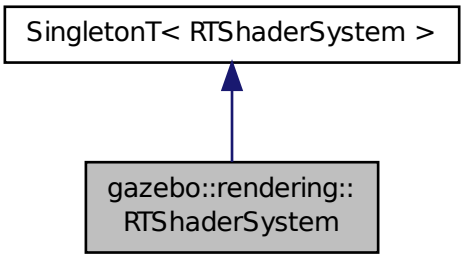
- **RotationSpline.hh**

10.137 gazebo::rendering::RTShaderSystem Class Reference

Implements **Ogre** (p. 110)'s Run-Time Shader system.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::RTShaderSystem:



Public Types

- enum **LightingModel** { **SSLM_PerVertexLighting**, **SSLM_PerPixelLighting**, **SSLM_NormalMapLighting-TangentSpace**, **SSLM_NormalMapLightingObjectSpace** }

Public Member Functions

- void **AddScene** (**ScenePtr** _scene)
  Add a scene manager.
- void **ApplyShadows** (**ScenePtr** _scene)
  Apply shadows to a scene.
- void **AttachEntity** (**Visual** ∗vis)
  Set an Ogre::Entity to use RT shaders.
- void **Clear** ()
  Clear the shader system.
- void **DetachEntity** (**Visual** ∗_vis)
  Remove and entity.
- void **Fini** ()
  Finalize the shader system.
- void **GenerateShaders** (**Visual** ∗_vis)
  Generate shaders for an entity.
- Ogre::PSSMShadowCameraSetup ∗ **GetPSSMShadowCameraSetup** () const
  *Get the **Ogre** (p. 110) PSSM Shadows camera setup.*
- void **Init** ()
  Init the run time shader system.
- void **RemoveScene** (**ScenePtr** _scene)
  Remove a scene.
- void **RemoveShadows** (**ScenePtr** _scene)
  Remove shadows from a scene.
- void **SetPerPixelLighting** (bool _set)
  Set the lighting model to per pixel or per vertex.
- void **UpdateShaders** ()
  Update the shaders. This should not be called frequently.

Static Public Member Functions

- static void **AttachViewport** (Ogre::Viewport ∗_viewport, **ScenePtr** _scene)
  Set a viewport to use shaders.
- static void **DetachViewport** (Ogre::Viewport ∗_viewport, **ScenePtr** _scene)
  Set a viewport to not use shaders.

Additional Inherited Members

10.137.1 Detailed Description

Implements **Ogre** (p. 110)'s Run-Time Shader system.

This class allows Gazebo to generate per-pixel shaders for every material at run-time.

10.137.2 Member Enumeration Documentation

10.137.2.1 enum gazebo::rendering::RTShaderSystem::LightingModel

The type of lighting.

Enumerator:

SSLM_PerVertexLighting Per-Vertex lighting: best performance.

SSLM_PerPixelLighting Per-Pixel lighting: best look.

SSLM_NormalMapLightingTangentSpace Normal Map lighting: lighting calculations have been stored in a light map (texture) using tangent space.

SSLM_NormalMapLightingObjectSpace Normal Map lighting: lighting calculations have been stored in a light map (texture) using object space.

10.137.3 Member Function Documentation

10.137.3.1 void gazebo::rendering::RTShaderSystem::AddScene (ScenePtr *_scene*)

Add a scene manager.

Parameters

in	*_scene*	The scene to process

10.137.3.2 void gazebo::rendering::RTShaderSystem::ApplyShadows (ScenePtr *_scene*)

Apply shadows to a scene.

Parameters

in	*_scene*	The scene to receive shadows.

10.137.3.3 void gazebo::rendering::RTShaderSystem::AttachEntity (Visual * *vis*)

Set an Ogre::Entity to use RT shaders.

Parameters

in	*_vis*	**Visual** (p. 920) that will use the **RTShaderSystem** (p. 703).

10.137.3.4 static void gazebo::rendering::RTShaderSystem::AttachViewport (Ogre::Viewport * *_viewport,* ScenePtr *_scene*) [static]

Set a viewport to use shaders.

Parameters

in	*_viewport*	The viewport to add.
in	*_scene*	The scene that the viewport uses.

10.137.3.5 void gazebo::rendering::RTShaderSystem::Clear ( )

Clear the shader system.

10.137.3.6 void gazebo::rendering::RTShaderSystem::DetachEntity (Visual * _vis)

Remove and entity.

Parameters

in	*_vis*	Remove this visual.

10.137.3.7 static void gazebo::rendering::RTShaderSystem::DetachViewport (Ogre::Viewport * _viewport, ScenePtr _scene) [static]

Set a viewport to not use shaders.

Parameters

in	*_viewport*	The viewport to remove.
in	*_scene*	The scene that the viewport uses.

10.137.3.8 void gazebo::rendering::RTShaderSystem::Fini ( )

Finalize the shader system.

10.137.3.9 void gazebo::rendering::RTShaderSystem::GenerateShaders (Visual * _vis)

Generate shaders for an entity.

Parameters

in	*_vis*	The visual to generate shaders for.

10.137.3.10 Ogre::PSSMShadowCameraSetup* gazebo::rendering::RTShaderSystem::GetPSSMShadowCameraSetup ( ) const

Get the **Ogre** (p. 110) PSSM Shadows camera setup.

Returns

The **Ogre** (p. 110) PSSM Shadows camera setup.

10.137.3.11 void gazebo::rendering::RTShaderSystem::Init ( )

Init the run time shader system.

10.137.3.12 void gazebo::rendering::RTShaderSystem::RemoveScene (ScenePtr _scene)

Remove a scene.

Parameters

in	*The*	scene to remove

10.137.3.13 void gazebo::rendering::RTShaderSystem::RemoveShadows (ScenePtr _scene)

Remove shadows from a scene.

Parameters

in	*_scene*	The scene to remove shadows from.

10.137.3.14 void gazebo::rendering::RTShaderSystem::SetPerPixelLighting (bool _set)

Set the lighting model to per pixel or per vertex.

Parameters

in	*_set*	True means to use per-pixel shaders.

10.137.3.15 void gazebo::rendering::RTShaderSystem::UpdateShaders ( )

Update the shaders. This should not be called frequently.

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

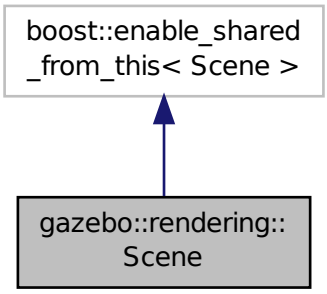
- **RTShaderSystem.hh**

10.138 gazebo::rendering::Scene Class Reference

Representation of an entire scene graph.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::Scene:



Public Types

- enum **SkyXMode** { **GZ_SKYX_ALL** = 0x0FFFFFFF, **GZ_SKYX_CLOUDS** = 0x0000001, **GZ_SKYX_MOON** = 0x0000002, **GZ_SKYX_NONE** = 0 }

Public Member Functions

- **Scene** (const std::string &_name, bool _enableVisualizations=false)

  Constructor.
- virtual ~**Scene** ()

  Destructor.
- void **AddVisual** (**VisualPtr** _vis)

  Add a visual to the scene.
- void **Clear** ()

  *Clear **rendering::Scene** (p. 707).*
- **VisualPtr CloneVisual** (const std::string &_visualName, const std::string &_newName)

  Clone a visual.
- **CameraPtr CreateCamera** (const std::string &_name, bool _autoRender=true)

  Create a camera.
- **DepthCameraPtr CreateDepthCamera** (const std::string &_name, bool _autoRender=true)

  Create depth camera.
- **GpuLaserPtr CreateGpuLaser** (const std::string &_name, bool _autoRender=true)

  Create laser that generates data from rendering.
- void **CreateGrid** (uint32_t _cellCount, float _cellLength, float _lineWidth, const **common::Color** &_color)

  Create a square grid of cells.
- **UserCameraPtr CreateUserCamera** (const std::string &_name)

  Create a user camera.
- void **DrawLine** (const **math::Vector3** &_start, const **math::Vector3** &_end, const std::string &_name)

  Draw a named line.

- **common::Color GetAmbientColor** () const

  Get the ambient color.
- **common::Color GetBackgroundColor** () const

  Get the background color.
- **CameraPtr GetCamera** (uint32_t _index) const

  Get a camera based on an index.
- **CameraPtr GetCamera** (const std::string &_name) const

  Get a camera by name.
- uint32_t **GetCameraCount** () const

  Get the number of cameras in this scene.
- bool **GetFirstContact** (**CameraPtr** _camera, const **math::Vector2i** &_mousePos, **math::Vector3** &_position)

  Get the world pos of a the first contact at a pixel location.
- **Grid** ∗ **GetGrid** (uint32_t _index) const

  Get a grid based on an index.
- uint32_t **GetGridCount** () const

  Get the number of grids.
- double **GetHeightBelowPoint** (const **math::Vector3** &_pt)

  Get the Z-value of the first object below the given point.
- **Heightmap** ∗ **GetHeightmap** () const

  Get a pointer to the heightmap.
- uint32_t **GetId** () const

  Get the scene ID.
- std::string **GetIdString** () const

  Get the scene Id as a string.
- bool **GetInitialized** () const

  *Return true if the **Scene** (p. 707) has been initialized.*
- **LightPtr GetLight** (const std::string &_name) const

  Get a light by name.
- **LightPtr GetLight** (uint32_t _index) const

  Get a light based on an index.
- uint32_t **GetLightCount** () const

  Get the count of the lights.
- Ogre::SceneManager ∗ **GetManager** () const

  Get the OGRE scene manager.
- **VisualPtr GetModelVisualAt** (**CameraPtr** _camera, const **math::Vector2i** &_mousePos)

  Get a model's visual at a mouse position.
- std::string **GetName** () const

  Get the name of the scene.
- **VisualPtr GetSelectedVisual** () const

  Get the currently selected visual.
- bool **GetShadowsEnabled** () const

  Get whether shadows are on or off.
- bool **GetShowClouds** () const

  Get whether or not clouds are displayed.
- **common::Time GetSimTime** () const

  Get the scene simulation time.
- **UserCameraPtr GetUserCamera** (uint32_t _index) const

Get a user camera by index.

- uint32_t **GetUserCameraCount** () const

  Get the number of user cameras in this scene.

- **VisualPtr GetVisual** (const std::string &_name) const

  Get a visual by name.

- **VisualPtr GetVisualAt** (**CameraPtr** _camera, const **math::Vector2i** &_mousePos, std::string &_mod)

  Get an entity at a pixel location using a camera.

- **VisualPtr GetVisualAt** (**CameraPtr** _camera, const **math::Vector2i** &_mousePos)

  Get a visual at a mouse position.

- **VisualPtr GetVisualBelow** (const std::string &_visualName)

  Get the closest visual below a given visual.

- void **GetVisualsBelowPoint** (const **math::Vector3** &_pt, std::vector< **VisualPtr** > &_visuals)

  Get a visual directly below a point.

- **VisualPtr GetWorldVisual** () const

  Get the top level world visual.

- void **Init** ()

  *Init **rendering::Scene*** (p. 707).

- void **Load** (**sdf::ElementPtr** _scene)

  Load the scene from a set of parameters.

- void **Load** ()

  Load the scene with default parameters.

- void **PreRender** ()

  Process all received messages.

- void **PrintSceneGraph** ()

  Print the scene graph to std_out.

- void **RemoveCamera** (const std::string &_name)

  Remove a camera from the scene.

- void **RemoveVisual** (**VisualPtr** _vis)

  Remove a visual from the scene.

- void **SelectVisual** (const std::string &_name, const std::string &_mode)

  Select a visual by name.

- void **SetAmbientColor** (const **common::Color** &_color)

  Set the ambient color.

- void **SetBackgroundColor** (const **common::Color** &_color)

  Set the background color.

- void **SetFog** (const std::string &_type, const **common::Color** &_color, double _density, double _start, double _end)

  Set the fog parameters.

- void **SetGrid** (bool _enabled)

  Set the grid on or off.

- void **SetShadowsEnabled** (bool _value)

  Set whether shadows are on or off.

- void **SetSkyXMode** (unsigned int _mode)

  *Set **SkyX*** (p. 113) *mode to enable/disable skyx components such as clouds and moon.*

- void **SetTransparent** (bool _show)

  Enable or disable transparency for all visuals.

- void **SetVisible** (const std::string &_name, bool _visible)

Hide or show a visual.

- void **SetWireframe** (bool _show)

  Enable or disable wireframe for all visuals.
- void **ShowClouds** (bool _show)

  Display clouds in the sky.
- void **ShowCollisions** (bool _show)

  Enable or disable collision visualization.
- void **ShowCOMs** (bool _show)

  Enable or disable center of mass visualization.
- void **ShowContacts** (bool _show)

  Enable or disable contact visualization.
- void **ShowJoints** (bool _show)

  Enable or disable joint visualization.
- void **SnapVisualToNearestBelow** (const std::string &_visualName)

  Move the visual to be ontop of the nearest visual below it.
- std::string **StripSceneName** (const std::string &_name) const

  Remove the name of scene from a string.

Public Attributes

- SkyX::SkyX * **skyx**

  Pointer to the sky.

10.138.1 Detailed Description

Representation of an entire scene graph.

Maintains all the Visuals, Lights, and Cameras for a World.

10.138.2 Member Enumeration Documentation

10.138.2.1 enum gazebo::rendering::Scene::SkyXMode

Enumerator:

GZ_SKYX_ALL

GZ_SKYX_CLOUDS

GZ_SKYX_MOON

GZ_SKYX_NONE

10.138.3 Constructor & Destructor Documentation

10.138.3.1 gazebo::rendering::Scene::Scene (const std::string & *_name*, bool *_enableVisualizations* = `false`)

Constructor.

Parameters

in	*_name*	Name of the scene.
in	*_enable-Visualizations*	True to enable visualizations, this should be set to true for user interfaces, and false for sensor generation.

10.138.3.2 virtual gazebo::rendering::Scene::~Scene ( ) [virtual]

Destructor.

10.138.4 Member Function Documentation

10.138.4.1 void gazebo::rendering::Scene::AddVisual (VisualPtr _vis)

Add a visual to the scene.

Parameters

in	*_vis*	**Visual** (p. 920) to add.

10.138.4.2 void gazebo::rendering::Scene::Clear ( )

Clear **rendering::Scene** (p. 707).

10.138.4.3 VisualPtr gazebo::rendering::Scene::CloneVisual (const std::string & *_visualName,* const std::string & *_newName*)

Clone a visual.

Parameters

in	*_visualName*	Name of the visual to clone.
in	*_newName*	New name of the visual.

Returns

Pointer to the cloned visual.

10.138.4.4 CameraPtr gazebo::rendering::Scene::CreateCamera (const std::string & *_name,* bool *_autoRender* = true)

Create a camera.

Parameters

in	*_name*	Name of the new camera.
in	*_autoRender*	True to allow Gazebo to automatically render the camera. This should almost always be true.

Returns

Pointer to the new camera.

10.138.4.5 DepthCameraPtr gazebo::rendering::Scene::CreateDepthCamera (const std::string & *_name,* bool *_autoRender* = true)

Create depth camera.

Parameters

in	*_name*	Name of the new camera.
in	*_autoRender*	True to allow Gazebo to automatically render the camera. This should almost always be true.

Returns

Pointer to the new camera.

10.138.4.6 GpuLaserPtr gazebo::rendering::Scene::CreateGpuLaser (const std::string & *_name*, bool *_autoRender* = `true`)

Create laser that generates data from rendering.

Parameters

in	*_name*	Name of the new laser.
in	*_autoRender*	True to allow Gazebo to automatically render the camera. This should almost always be true.

Returns

Pointer to the new laser.

10.138.4.7 void gazebo::rendering::Scene::CreateGrid (uint32_t *_cellCount*, float *_cellLength*, float *_lineWidth*, const common::Color & *_color*)

Create a square grid of cells.

Parameters

in	*_cellCount*	Number of grid cells in one direction.
in	*_cellLength*	Length of one grid cell.
in	*_lineWidth*	Width of the grid lines.
in	*_color*	Color of the grid lines.

10.138.4.8 UserCameraPtr gazebo::rendering::Scene::CreateUserCamera (const std::string & *_name*)

Create a user camera.

A user camera is one design for use with a GUI.

Parameters

in	*_name*	Name of the **UserCamera** (p. 864).

Returns

A pointer to the new **UserCamera** (p. 864).

10.138.4.9 void gazebo::rendering::Scene::DrawLine (const math::Vector3 & _start, const math::Vector3 & _end, const std::string & _name)

Draw a named line.

Parameters

in	*_start*	Start position of the line.
in	*_end*	End position of the line.
in	*_name*	Name of the line.

10.138.4.10 common::Color gazebo::rendering::Scene::GetAmbientColor ( ) const

Get the ambient color.

Returns

The scene's ambient color.

10.138.4.11 common::Color gazebo::rendering::Scene::GetBackgroundColor ( ) const

Get the background color.

Returns

The background color.

10.138.4.12 CameraPtr gazebo::rendering::Scene::GetCamera (uint32_t _index) const

Get a camera based on an index.

Index must be between 0 and **Scene::GetCameraCount** (p. 715).

Parameters

in	*_index*	Index of the camera to get.

Returns

Pointer to the camera. Or NULL if the index is invalid.

10.138.4.13 CameraPtr gazebo::rendering::Scene::GetCamera (const std::string & _name) const

Get a camera by name.

Parameters

in	*_name*	Name of the camera.

Returns

Pointer to the camera. Or NULL if the name is invalid.

10.138.4.14 uint32_t gazebo::rendering::Scene::GetCameraCount ( ) const

Get the number of cameras in this scene.

Returns

Number of lasers.

10.138.4.15 bool gazebo::rendering::Scene::GetFirstContact (CameraPtr *_camera,* const math::Vector2i & *_mousePos,* math::Vector3 & *_position*)

Get the world pos of a the first contact at a pixel location.

Parameters

in	*_camera*	Pointer to the camera.
in	*_mousePos*	2D position of the mouse in pixels.
out	*_position*	3D position of the first contact point.

Returns

True if a valid object was hit by the raycast.

10.138.4.16 Grid* gazebo::rendering::Scene::GetGrid (uint32_t *_index*) const

Get a grid based on an index.

Index must be between 0 and **Scene::GetGridCount** (p. 715).

Parameters

in	*_index*	Index of the grid.

10.138.4.17 uint32_t gazebo::rendering::Scene::GetGridCount ( ) const

Get the number of grids.

Returns

The number of grids.

10.138.4.18 double gazebo::rendering::Scene::GetHeightBelowPoint (const math::Vector3 & *_pt*)

Get the Z-value of the first object below the given point.

Parameters

in	*_pt*	Position to search below for a visual.

Returns

The Z-value of the nearest visual below the point. Zero is returned if no visual is found.

10.138.4.19 Heightmap* gazebo::rendering::Scene::GetHeightmap ( ) const

Get a pointer to the heightmap.

Returns

Pointer to the heightmap, NULL if no heightmap.

10.138.4.20 uint32_t gazebo::rendering::Scene::GetId ( ) const

Get the scene ID.

Returns

The ID of the scene.

10.138.4.21 std::string gazebo::rendering::Scene::GetIdString ( ) const

Get the scene Id as a string.

Returns

The ID as a string.

10.138.4.22 bool gazebo::rendering::Scene::GetInitialized ( ) const

Return true if the **Scene** (p. 707) has been initialized.

10.138.4.23 LightPtr gazebo::rendering::Scene::GetLight (const std::string & *_name*) const

Get a light by name.

Parameters

in	*_name*	Name of the light to get.

Returns

Pointer to the light, or NULL if the light was not found.

10.138.4.24 LightPtr gazebo::rendering::Scene::GetLight (uint32_t *_index*) const

Get a light based on an index.

The index must be between 0 and **Scene::GetLightCount** (p. 717).

Parameters

in	*_index*	Index of the light.

Returns

Pointer to the **Light** (p. 433) or NULL if index was invalid.

10.138.4.25 uint32_t gazebo::rendering::Scene::GetLightCount ( ) const

Get the count of the lights.

Returns

The number of lights.

10.138.4.26 Ogre::SceneManager* gazebo::rendering::Scene::GetManager ( ) const

Get the OGRE scene manager.

Returns

Pointer to the **Ogre** (p. 110) SceneManager.

10.138.4.27 VisualPtr gazebo::rendering::Scene::GetModelVisualAt (CameraPtr *_camera,* const math::Vector2i & *_mousePos*)

Get a model's visual at a mouse position.

Parameters

in	*_camera*	Pointer to the camera used to project the mouse position.
in	*_mousePos*	The 2d position of the mouse in pixels.

Returns

Pointer to the visual, NULL if none found.

10.138.4.28 std::string gazebo::rendering::Scene::GetName ( ) const

Get the name of the scene.

Returns

Name of the scene.

10.138.4.29 VisualPtr gazebo::rendering::Scene::GetSelectedVisual ( ) const

Get the currently selected visual.

Returns

Pointer to the currently selected visual, or NULL if nothing is selected.

10.138.4.30 bool gazebo::rendering::Scene::GetShadowsEnabled ( ) const

Get whether shadows are on or off.

Returns

True if shadows are enabled.

10.138.4.31 bool gazebo::rendering::Scene::GetShowClouds ( ) const

Get whether or not clouds are displayed.

Returns

True if clouds are displayed.

10.138.4.32 common::Time gazebo::rendering::Scene::GetSimTime ( ) const

Get the scene simulation time.

Note this is different from World::GetSimTime() because there is a lag between the time new poses are sent out by World and when they are received and applied by the **Scene** (p. 707).

Returns

The current simulation time in **Scene** (p. 707)

10.138.4.33 UserCameraPtr gazebo::rendering::Scene::GetUserCamera (uint32_t *_index*) const

Get a user camera by index.

The index value must be between 0 and **Scene::GetUserCameraCount** (p. 719).

Parameters

in	*_index*	Index of the **UserCamera** (p. 864) to get.

Returns

Pointer to the **UserCamera** (p. 864), or NULL if the index was invalid.

10.138.4.34 uint32_t gazebo::rendering::Scene::GetUserCameraCount ( ) const

Get the number of user cameras in this scene.

Returns

The number of user cameras.

10.138.4.35 VisualPtr gazebo::rendering::Scene::GetVisual (const std::string & *_name*) const

Get a visual by name.

10.138.4.36 VisualPtr gazebo::rendering::Scene::GetVisualAt (CameraPtr *_camera,* const math::Vector2i & *_mousePos,* std::string & *_mod*)

Get an entity at a pixel location using a camera.

Used for mouse picking.

Parameters

in	*_camera*	The ogre camera, used to do mouse picking
in	*_mousePos*	The position of the mouse in screen coordinates
out	*_mod*	Used for object manipulation

Returns

The selected entity, or NULL

10.138.4.37 VisualPtr gazebo::rendering::Scene::GetVisualAt (CameraPtr *_camera,* const math::Vector2i & *_mousePos*)

Get a visual at a mouse position.

Parameters

in	*_camera*	Pointer to the camera used to project the mouse position.
in	*_mousePos*	The 2d position of the mouse in pixels.

Returns

Pointer to the visual, NULL if none found.

10.138.4.38 VisualPtr gazebo::rendering::Scene::GetVisualBelow (const std::string & *_visualName*)

Get the closest visual below a given visual.

Parameters

in	*_visualName*	Name of the visual to search below.

Returns

Pointer to the visual below, or NULL if no visual.

10.138.4.39 void gazebo::rendering::Scene::GetVisualsBelowPoint (const math::Vector3 & *_pt*, std::vector< VisualPtr > & *_visuals*)

Get a visual directly below a point.

Parameters

in	*_pt*	3D point to get the visual below.
out	*_visuals*	The visuals below the point order in proximity.

10.138.4.40 VisualPtr gazebo::rendering::Scene::GetWorldVisual ( ) const

Get the top level world visual.

Returns

Pointer to the world visual.

10.138.4.41 void gazebo::rendering::Scene::Init ( )

Init **rendering::Scene** (p. 707).

10.138.4.42 void gazebo::rendering::Scene::Load (sdf::ElementPtr *_scene*)

Load the scene from a set of parameters.

Parameters

in	*_scene*	SDF scene element to load.

10.138.4.43 void gazebo::rendering::Scene::Load ( )

Load the scene with default parameters.

10.138.4.44 void gazebo::rendering::Scene::PreRender ( )

Process all received messages.

10.138.4.45 void gazebo::rendering::Scene::PrintSceneGraph ( )

Print the scene graph to std_out.

10.138.4.46 void gazebo::rendering::Scene::RemoveCamera (const std::string & *_name*)

Remove a camera from the scene.

Parameters

in	*_name*	Name of the camera.

10.138.4.47 void gazebo::rendering::Scene::RemoveVisual (VisualPtr *_vis*)

Remove a visual from the scene.

Parameters

in	*_vis*	**Visual** (p. 920) to remove.

10.138.4.48 void gazebo::rendering::Scene::SelectVisual (const std::string & *_name,* const std::string & *_mode*)

Select a visual by name.

Parameters

in	*_name*	Name of the visual to select.
in	*_mode*	Selection mode (normal, or move).

10.138.4.49 void gazebo::rendering::Scene::SetAmbientColor (const common::Color & *_color*)

Set the ambient color.

Parameters

in	*_color*	The ambient color to use.

10.138.4.50 void gazebo::rendering::Scene::SetBackgroundColor (const common::Color & *_color*)

Set the background color.

Parameters

in	*_color*	The background color.

10.138.4.51 void gazebo::rendering::Scene::SetFog (const std::string & *_type,* const common::Color & *_color,* double *_density,* double *_start,* double *_end*)

Set the fog parameters.

Parameters

in	*_type*	Type of fog: "linear", "exp", or "exp2".
in	*_color*	Color of the fog.
in	*_density*	Fog density.
in	*_start*	Distance from camera to start the fog.
in	*_end*	Distance from camera at which the fog is at max density.

10.138.4.52 void gazebo::rendering::Scene::SetGrid (bool *_enabled*)

Set the grid on or off.

Parameters

in	*_enabled*	Set to true to turn on the grid

10.138.4.53 void gazebo::rendering::Scene::SetShadowsEnabled (bool *_value*)

Set whether shadows are on or off.

Parameters

in	*_value*	True to enable shadows, False to disable

10.138.4.54 void gazebo::rendering::Scene::SetSkyXMode (unsigned int *_mode*)

Set **SkyX** (p. 113) mode to enable/disable skyx components such as clouds and moon.

Parameters

in	*_mode*	**SkyX** (p. 113) mode bitmask.

See Also

Scene::SkyXMode (p. 711)

10.138.4.55 void gazebo::rendering::Scene::SetTransparent (bool *_show*)

Enable or disable transparency for all visuals.

Parameters

in	*_show*	True to enable transparency for all visuals.

10.138.4.56 void gazebo::rendering::Scene::SetVisible (const std::string & _name, bool _visible)

Hide or show a visual.

Parameters

in	*_name*	Name of the visual to change.
in	*_visible*	True to make visual visible, False to make it invisible.

10.138.4.57 void gazebo::rendering::Scene::SetWireframe (bool _show)

Enable or disable wireframe for all visuals.

Parameters

in	*_show*	True to enable wireframe for all visuals.

10.138.4.58 void gazebo::rendering::Scene::ShowClouds (bool _show)

Display clouds in the sky.

Parameters

in	*_show*	True to display clouds.

10.138.4.59 void gazebo::rendering::Scene::ShowCollisions (bool _show)

Enable or disable collision visualization.

Parameters

in	*_show*	True to enable collision visualization.

10.138.4.60 void gazebo::rendering::Scene::ShowCOMs (bool _show)

Enable or disable center of mass visualization.

Parameters

in	*_show*	True to enable center of mass visualization.

10.138.4.61 void gazebo::rendering::Scene::ShowContacts (bool _show)

Enable or disable contact visualization.

Parameters

in	*_show*	True to enable contact visualization.

10.138.4.62 void gazebo::rendering::Scene::ShowJoints (bool *_show*)

Enable or disable joint visualization.

Parameters

in	*_show*	True to enable joint visualization.

10.138.4.63 void gazebo::rendering::Scene::SnapVisualToNearestBelow (const std::string & *_visualName*)

Move the visual to be ontop of the nearest visual below it.

Parameters

in	*_visualName*	Name of the visual to move.

10.138.4.64 std::string gazebo::rendering::Scene::StripSceneName (const std::string & *_name*) const

Remove the name of scene from a string.

Parameters

in	*_name*	Name to string the scene name from.

Returns

The stripped name.

10.138.5 Member Data Documentation

10.138.5.1 SkyX::SkyX* gazebo::rendering::Scene::skyx

Pointer to the sky.

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

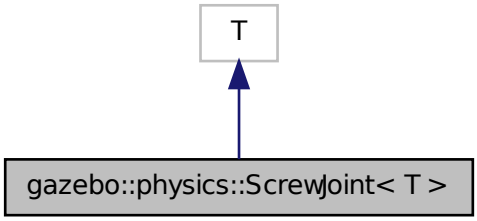
- **Scene.hh**

10.139 gazebo::physics::ScrewJoint< T > Class Template Reference

A screw joint, which has both prismatic and rotational DOFs.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::ScrewJoint< T >:



Public Member Functions

- **ScrewJoint** (**BasePtr** _parent)
  Constructor.
- virtual ~**ScrewJoint** ()
  Destructor.
- virtual **math::Vector3 GetAnchor** (int _index) const
  Get the anchor.
- virtual unsigned int **GetAngleCount** () const
- virtual double **GetThreadPitch** (unsigned int _index)=0
  Get screw joint thread pitch.
- virtual void **Load** (**sdf::ElementPtr** _sdf)
  *Load a **ScrewJoint** (p. 724).*
- virtual void **SetAnchor** (int _index, const **math::Vector3** &_anchor)
  Set the anchor.
- virtual void **SetThreadPitch** (int _index, double _threadPitch)=0
  Set screw joint thread pitch.

Protected Attributes

- **math::Vector3 fakeAnchor**
  The anchor value is not used internally.
- double **threadPitch**
  Pitch of the thread.

10.139.1 Detailed Description

template<class T>class gazebo::physics::ScrewJoint< T >

A screw joint, which has both prismatic and rotational DOFs.

10.139.2 Constructor & Destructor Documentation

10.139.2.1 template<class T > gazebo::physics::ScrewJoint< T >::ScrewJoint (BasePtr _parent) [inline], [explicit]

Constructor.

Parameters

in	*_parent*	Parent of the joint.

References gazebo::physics::Base::SCREW_JOINT.

10.139.2.2 template<class T > virtual gazebo::physics::ScrewJoint< T >::~ScrewJoint ( ) [inline], [virtual]

Destructor.

10.139.3 Member Function Documentation

10.139.3.1 template<class T > math::Vector3 gazebo::physics::ScrewJoint< T >::GetAnchor (int _index) const [virtual]

Get the anchor.

Parameters

in	*_index*	Index of the axis. Not Used.

Returns

Anchor for the joint.

10.139.3.2 template<class T > virtual unsigned int gazebo::physics::ScrewJoint< T >::GetAngleCount ( ) const [inline], [virtual]

10.139.3.3 template<class T > virtual double gazebo::physics::ScrewJoint< T >::GetThreadPitch (unsigned int _index) [pure virtual]

Get screw joint thread pitch.

This must be implemented in a child class

Parameters

in	*_index*	Index of the axis.

Returns

_threadPitch Thread pitch value.

10.139.3.4 template<class T > virtual void gazebo::physics::ScrewJoint< T >::Load (sdf::ElementPtr _sdf) [inline], [virtual]

Load a **ScrewJoint** (p. 724).

Parameters

in	*_sdf*	SDF value to load from

References gzerr, and gazebo::physics::ScrewJoint< T >::threadPitch.

10.139.3.5 template<class T > void gazebo::physics::ScrewJoint< T >::SetAnchor (int _index, const math::Vector3 & _anchor) [virtual]

Set the anchor.

Parameters

in	*_index*	Index of the axis. Not Used.
in	*_anchor*	Anchor value for the joint.

10.139.3.6 template<class T > virtual void gazebo::physics::ScrewJoint< T >::SetThreadPitch (int _index, double _threadPitch) [pure virtual]

Set screw joint thread pitch.

This must be implemented in a child class

Parameters

in	*_index*	Index of the axis.
in	*_threadPitch*	Thread pitch value.

10.139.4 Member Data Documentation

10.139.4.1 template<class T > math::Vector3 gazebo::physics::ScrewJoint< T >::fakeAnchor [protected]

The anchor value is not used internally.

10.139.4.2 template<class T > double gazebo::physics::ScrewJoint< T >::threadPitch [protected]

Pitch of the thread.

Referenced by gazebo::physics::ScrewJoint< T >::Load().

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

- **ScrewJoint.hh**

10.140 sdf::SDF Class Reference

Base **SDF** (p. 728) class.

`#include <SDF.hh>`

Public Member Functions

- **SDF** () **GAZEBO_DEPRECATED**(1.6)
- void **PrintDescription** () **GAZEBO_DEPRECATED**(1.6)
- void **PrintDoc** () **GAZEBO_DEPRECATED**(1.6)
- void **PrintValues** () **GAZEBO_DEPRECATED**(1.6)
- void **PrintWiki** () **GAZEBO_DEPRECATED**(1.6)
- void **SetFromString** (const std::string &_sdfData) **GAZEBO_DEPRECATED**(1.6)

  *Set **SDF** (p. 728) values from a string.*
- std::string **ToString** () const **GAZEBO_DEPRECATED**(1.6)
- void **Write** (const std::string &_filename) **GAZEBO_DEPRECATED**(1.6)

Public Attributes

- **ElementPtr root**

Static Public Attributes

- static std::string **version**

10.140.1 Detailed Description

Base **SDF** (p. 728) class.

10.140.2 Constructor & Destructor Documentation

10.140.2.1 sdf::SDF::SDF ( )

10.140.3 Member Function Documentation

10.140.3.1 void sdf::SDF::PrintDescription ( )

10.140.3.2 void sdf::SDF::PrintDoc ( )

10.140.3.3 void sdf::SDF::PrintValues ( )

10.140.3.4 void sdf::SDF::PrintWiki ( )

10.140.3.5 void sdf::SDF::SetFromString (const std::string & *_sdfData*)

Set **SDF** (p. 728) values from a string.

10.140.3.6 std::string sdf::SDF::ToString ( ) const

10.140.3.7 void sdf::SDF::Write (const std::string & *_filename*)

10.140.4 Member Data Documentation

10.140.4.1 ElementPtr sdf::SDF::root

10.140.4.2 std::string sdf::SDF::version [static]

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

- **SDF.hh**

10.141 gazebo::rendering::SelectionObj Class Reference

A graphical selection object.

`#include <rendering/rendering.hh>`

Public Member Functions

- **SelectionObj** (**Scene** *_scene)

  Constructor.
- virtual ~**SelectionObj** ()

  Destructor.
- void **Attach** (**VisualPtr** _visual)

  Set the position of the node.
- void **Clear** ()

  *Clear the **rendering::SelectionObj** (p. 729) object.*
- std::string **GetVisualName** () const

  Get the name of the visual the selection obj is attached to.
- void **Init** ()

  *Initialize the **rendering::SelectionObj** (p. 729) object.*
- bool **IsActive** () const

  Return true if the user is move the selection obj.
- void **SetActive** (bool _active)

  Set true if the user is moving the selection obj.
- void **SetHighlight** (const std::string &_mod)

  Highlight the selection object based on a modifier.

10.141.1 Detailed Description

A graphical selection object.

Used to draw a visual around a selected object.

10.141.2 Constructor & Destructor Documentation

10.141.2.1 gazebo::rendering::SelectionObj::SelectionObj (Scene * _scene)

Constructor.

Parameters

in	*_scene*	**Scene** (p. 707) to use.

10.141.2.2 virtual gazebo::rendering::SelectionObj::~SelectionObj ( ) [virtual]

Destructor.

10.141.3 Member Function Documentation

10.141.3.1 void gazebo::rendering::SelectionObj::Attach (VisualPtr _visual)

Set the position of the node.

Parameters

in	*This*	draws the selection object around the passed in visual.

10.141.3.2 void gazebo::rendering::SelectionObj::Clear ( )

Clear the **rendering::SelectionObj** (p. 729) object.

10.141.3.3 std::string gazebo::rendering::SelectionObj::GetVisualName ( ) const

Get the name of the visual the selection obj is attached to.

Returns

Name of the selected visual.

10.141.3.4 void gazebo::rendering::SelectionObj::Init ( )

Initialize the **rendering::SelectionObj** (p. 729) object.

10.141.3.5 bool gazebo::rendering::SelectionObj::IsActive ( ) const

Return true if the user is move the selection obj.

Returns

True if something is selected.

10.141.3.6 void gazebo::rendering::SelectionObj::SetActive (bool *_active*)

Set true if the user is moving the selection obj.

Parameters

in	*_active*	True if the user is interacting with the selection object.

10.141.3.7 void gazebo::rendering::SelectionObj::SetHighlight (const std::string & *_mod*)

Highlight the selection object based on a modifier.

Parameters

in	*_mod*	Modifier used when highlighting the selection object.

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

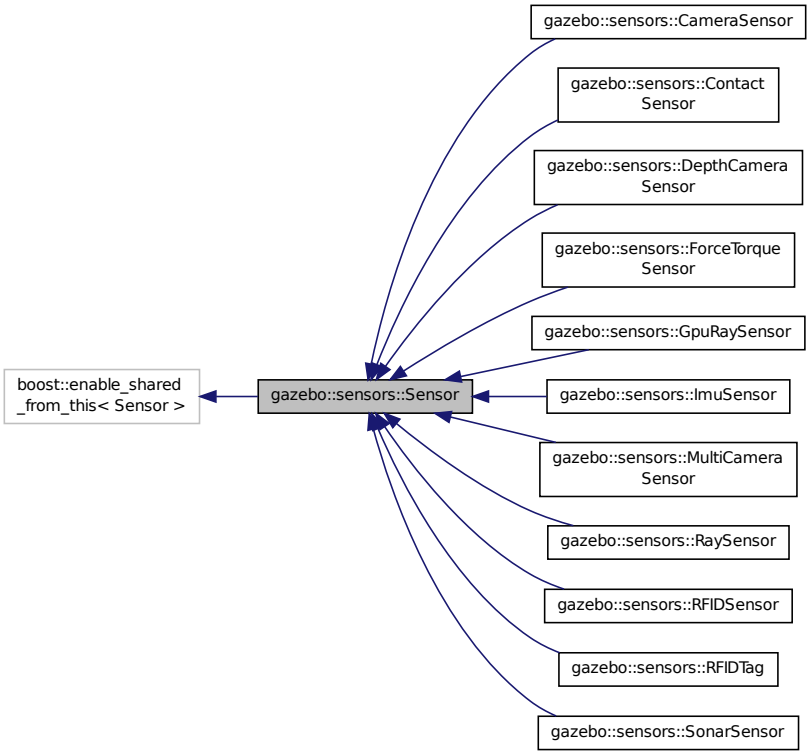
- **SelectionObj.hh**

10.142 gazebo::sensors::Sensor Class Reference

Base class for sensors.

`#include <sensors/sensors.hh>`

Inheritance diagram for gazebo::sensors::Sensor:



Public Member Functions

- **Sensor** (**SensorCategory** _cat)

  Constructor.
- virtual ~**Sensor** ()

  Destructor.
- template<typename T >
  event::ConnectionPtr ConnectUpdated (T _subscriber)

  Connect a signal that is triggered when the sensor is updated.
- void **DisconnectUpdated** (**event::ConnectionPtr** &_c)

  Disconnect from a the updated signal.
- void **FillMsg** (msgs::Sensor &_msg)

  fills a msgs::Sensor message.
- virtual void **Fini** ()

  Finalize the sensor.
- **SensorCategory GetCategory** () const

  Get the category of the sensor.
- **common::Time GetLastMeasurementTime** ()

  Return last measurement time.
- **common::Time GetLastUpdateTime** ()

Return last update time.

- std::string **GetName** () const

  Get name.

- std::string **GetParentName** () const

  Returns the name of the sensor parent.

- virtual **math::Pose GetPose** () const

  Get the current pose.

- std::string **GetScopedName** () const

  Get fully scoped name of the sensor.

- virtual std::string **GetTopic** () const

  Returns the topic name as set in SDF.

- std::string **GetType** () const

  Get sensor type.

- double **GetUpdateRate** ()

  Get the update rate of the sensor.

- bool **GetVisualize** () const

  Return true if user requests the sensor to be visualized via tag: <visualize>true</visualize> in SDF.

- std::string **GetWorldName** () const

  Returns the name of the world the sensor is in.

- virtual void **Init** ()

  Initialize the sensor.

- virtual bool **IsActive** ()

  Returns true if sensor generation is active.

- virtual void **Load** (const std::string &_worldName, **sdf::ElementPtr** _sdf)

  Load the sensor with SDF parameters.

- virtual void **Load** (const std::string &_worldName)

  Load the sensor with default parameters.

- void **ResetLastUpdateTime** ()

  Reset the lastUpdateTime to zero.

- virtual void **SetActive** (bool _value)

  Set whether the sensor is active or not.

- virtual void **SetParent** (const std::string &_name)

  Set the parent of the sensor.

- void **SetUpdateRate** (double _hz)

  Set the update rate of the sensor.

- void **Update** (bool _force)

  Update the sensor.

Protected Member Functions

- virtual void **UpdateImpl** (bool)

  This gets overwritten by derived sensor types.

Protected Attributes

- bool **active**

  True if sensor generation is active.
- std::vector< **event::ConnectionPtr** > **connections**

  All event connections.
- **common::Time lastMeasurementTime**

  Stores last time that a sensor measurement was generated; this value must be updated within each sensor's UpdateImpl.
- **common::Time lastUpdateTime**

  Time of the last update.
- **transport::NodePtr node**

  Node for communication.
- std::string **parentName**

  Name of the parent.
- std::vector< **SensorPluginPtr** > **plugins**

  All the plugins for the sensor.
- **math::Pose pose**

  Pose of the sensor.
- **transport::SubscriberPtr poseSub**

  Subscribe to pose updates.
- **sdf::ElementPtr sdf**

  Pointer the the SDF element for the sensor.
- **common::Time updatePeriod**

  *Desired time between updates, set indirectly by **Sensor::SetUpdateRate** (p. 739).*
- **gazebo::physics::WorldPtr world**

  Pointer to the world.

10.142.1 Detailed Description

Base class for sensors.

10.142.2 Constructor & Destructor Documentation

10.142.2.1 gazebo::sensors::Sensor::Sensor (SensorCategory *_cat*) [explicit]

Constructor.

Parameters

in	*_class*	

10.142.2.2 virtual gazebo::sensors::Sensor::~Sensor ( ) [virtual]

Destructor.

10.142.3 Member Function Documentation

10.142.3.1 template<typename T > event::ConnectionPtr gazebo::sensors::Sensor::ConnectUpdated (T _subscriber) [inline]

Connect a signal that is triggered when the sensor is updated.

Parameters

in	*_subscriber*	Callback that receives the signal.

Returns

A pointer to the connection. This must be kept in scope.

See Also

Sensor::DisconnectUpdated (p. 735)

References gazebo::event::EventT< T >::Connect().

10.142.3.2 void gazebo::sensors::Sensor::DisconnectUpdated (event::ConnectionPtr & _c) [inline]

Disconnect from a the updated signal.

Parameters

in	*_c*	The connection to disconnect

See Also

Sensor::ConnectUpdated (p. 735)

References gazebo::event::EventT< T >::Disconnect().

10.142.3.3 void gazebo::sensors::Sensor::FillMsg (msgs::Sensor & _msg)

fills a msgs::Sensor message.

Parameters

out	*_msg*	Message to fill.

10.142.3.4 virtual void gazebo::sensors::Sensor::Fini ( ) [virtual]

Finalize the sensor.

Reimplemented in **gazebo::sensors::MultiCameraSensor** (p. 553), **gazebo::sensors::ForceTorqueSensor** (p. 330), **gazebo::sensors::CameraSensor** (p. 194), **gazebo::sensors::GpuRaySensor** (p. 349), **gazebo::sensors::ContactSensor** (p. 250), **gazebo::sensors::DepthCameraSensor** (p. 266), **gazebo::sensors::RFIDSensor** (p. 690), **gazebo::sensors::RaySensor** (p. 674), **gazebo::sensors::RFIDTag** (p. 693), **gazebo::sensors::SonarSensor** (p. 787), and **gazebo::sensors::ImuSensor** (p. 386).

10.142.3.5 SensorCategory gazebo::sensors::Sensor::GetCategory ( ) const

Get the category of the sensor.

Returns

The category of the sensor.

See Also

SensorCategory (p. 106)

10.142.3.6 common::Time gazebo::sensors::Sensor::GetLastMeasurementTime ( )

Return last measurement time.

Returns

Time of last measurement.

10.142.3.7 common::Time gazebo::sensors::Sensor::GetLastUpdateTime ( )

Return last update time.

Returns

Time of last update.

10.142.3.8 std::string gazebo::sensors::Sensor::GetName ( ) const

Get name.

Returns

Name of sensor.

10.142.3.9 std::string gazebo::sensors::Sensor::GetParentName ( ) const

Returns the name of the sensor parent.

The parent name is set by **Sensor::SetParent** (p. 739).

Returns

Name of Parent.

10.142.3.10 virtual math::Pose gazebo::sensors::Sensor::GetPose ( ) const [virtual]

Get the current pose.

Returns

Current pose of the sensor.

10.142.3.11 std::string gazebo::sensors::Sensor::GetScopedName ( ) const

Get fully scoped name of the sensor.

Returns

world_name::parent_name::sensor_name.

10.142.3.12 virtual std::string gazebo::sensors::Sensor::GetTopic ( ) const [virtual]

Returns the topic name as set in SDF.

Returns

Topic name.

Reimplemented in **gazebo::sensors::GpuRaySensor** (p. 353), **gazebo::sensors::RaySensor** (p. 677), **gazebo::sensors::CameraSensor** (p. 195), **gazebo::sensors::SonarSensor** (p. 788), **gazebo::sensors::MultiCameraSensor** (p. 555), and **gazebo::sensors::ForceTorqueSensor** (p. 330).

10.142.3.13 std::string gazebo::sensors::Sensor::GetType ( ) const

Get sensor type.

Returns

Type of sensor.

10.142.3.14 double gazebo::sensors::Sensor::GetUpdateRate ( )

Get the update rate of the sensor.

Returns

_hz update rate of sensor. Returns 0 if unthrottled.

10.142.3.15 bool gazebo::sensors::Sensor::GetVisualize ( ) const

Return true if user requests the sensor to be visualized via tag: <visualize>true</visualize> in SDF.

Returns

True if visualized, false if not.

10.142.3.16 std::string gazebo::sensors::Sensor::GetWorldName ( ) const

Returns the name of the world the sensor is in.

Returns

Name of the world.

10.142.3.17 virtual void gazebo::sensors::Sensor::Init ( ) [virtual]

Initialize the sensor.

Reimplemented in **gazebo::sensors::GpuRaySensor** (p. 354), **gazebo::sensors::ContactSensor** (p. 251), **gazebo::sensors::CameraSensor** (p. 195), **gazebo::sensors::DepthCameraSensor** (p. 267), **gazebo::sensors::RFIDSensor** (p. 690), **gazebo::sensors::RaySensor** (p. 678), **gazebo::sensors::RFIDTag** (p. 693), **gazebo::sensors::SonarSensor** (p. 788), **gazebo::sensors::MultiCameraSensor** (p. 555), **gazebo::sensors::ImuSensor** (p. 387), and **gazebo::sensors::ForceTorqueSensor** (p. 331).

10.142.3.18 virtual bool gazebo::sensors::Sensor::IsActive ( ) [virtual]

Returns true if sensor generation is active.

Returns

True if active, false if not.

Reimplemented in **gazebo::sensors::GpuRaySensor** (p. 354), **gazebo::sensors::RaySensor** (p. 678), **gazebo::sensors::ContactSensor** (p. 251), **gazebo::sensors::CameraSensor** (p. 195), **gazebo::sensors::MultiCameraSensor** (p. 555), **gazebo::sensors::SonarSensor** (p. 788), **gazebo::sensors::ImuSensor** (p. 387), and **gazebo::sensors::ForceTorqueSensor** (p. 331).

10.142.3.19 virtual void gazebo::sensors::Sensor::Load (const std::string & *_worldName*, sdf::ElementPtr *_sdf*) [virtual]

Load the sensor with SDF parameters.

Parameters

in	*_sdf*	SDF **Sensor** (p. 731) parameters.
in	*_worldName*	Name of world to load from.

Reimplemented in **gazebo::sensors::ContactSensor** (p. 252), **gazebo::sensors::CameraSensor** (p. 195), **gazebo::sensors::RFIDSensor** (p. 690), and **gazebo::sensors::ImuSensor** (p. 387).

10.142.3.20 virtual void gazebo::sensors::Sensor::Load (const std::string & *_worldName*) [virtual]

Load the sensor with default parameters.

Parameters

in	*_worldName*	Name of world to load from.

Reimplemented in **gazebo::sensors::GpuRaySensor** (p. 354), **gazebo::sensors::ContactSensor** (p. 252), **gazebo::sensors::CameraSensor** (p. 196), **gazebo::sensors::DepthCameraSensor** (p. 267), **gazebo::sensors::RFIDSensor** (p. 691), **gazebo::sensors::RaySensor** (p. 678), **gazebo::sensors::RFIDTag** (p. 693), **gazebo::sensors::SonarSensor** (p. 788), **gazebo::sensors::MultiCameraSensor** (p. 556), **gazebo::sensors::ImuSensor** (p. 387), and **gazebo::sensors::ForceTorqueSensor** (p. 331).

10.142.3.21 void gazebo::sensors::Sensor::ResetLastUpdateTime ( )

Reset the lastUpdateTime to zero.

10.142.3.22 virtual void gazebo::sensors::Sensor::SetActive (bool *_value*) [virtual]

Set whether the sensor is active or not.

Parameters

in	*_value*	True if active, false if not.

Reimplemented in **gazebo::sensors::DepthCameraSensor** (p. 267).

10.142.3.23 virtual void gazebo::sensors::Sensor::SetParent (const std::string & *_name*) [virtual]

Set the parent of the sensor.

Parameters

in	*_name*	Name of the parent.

Reimplemented in **gazebo::sensors::CameraSensor** (p. 196), and **gazebo::sensors::DepthCameraSensor** (p. 268).

10.142.3.24 void gazebo::sensors::Sensor::SetUpdateRate (double *_hz*)

Set the update rate of the sensor.

Parameters

in	*_hz*	update rate of sensor.

10.142.3.25 void gazebo::sensors::Sensor::Update (bool *_force*)

Update the sensor.

Parameters

in	*_force*	True to force update, false otherwise.

10.142.3.26 virtual void gazebo::sensors::Sensor::UpdateImpl (bool) [inline], [protected], [virtual]

This gets overwritten by derived sensor types.

```
This function is called during Sensor::Update.
And in turn, Sensor::Update is called by
SensorManager::Update
```

Parameters

in	*_force*	True if update is forced, false if not

Reimplemented in **gazebo::sensors::MultiCameraSensor** (p. 556), **gazebo::sensors::ForceTorqueSensor** (p. 331), **gazebo::sensors::CameraSensor** (p. 196), **gazebo::sensors::GpuRaySensor** (p. 355), **gazebo::sensors::ContactSensor** (p. 252), **gazebo::sensors::DepthCameraSensor** (p. 268), **gazebo::sensors::RFIDSensor** (p. 691), **gazebo::sensors::RaySensor** (p. 678), **gazebo::sensors::RFIDTag** (p. 693), **gazebo::sensors::SonarSensor** (p. 788), and **gazebo::sensors::ImuSensor** (p. 388).

10.142.4 Member Data Documentation

10.142.4.1 bool gazebo::sensors::Sensor::active [protected]

True if sensor generation is active.

10.142.4.2 std::vector<event::ConnectionPtr> gazebo::sensors::Sensor::connections [protected]

All event connections.

10.142.4.3 common::Time gazebo::sensors::Sensor::lastMeasurementTime [protected]

Stores last time that a sensor measurement was generated; this value must be updated within each sensor's UpdateImpl.

10.142.4.4 common::Time gazebo::sensors::Sensor::lastUpdateTime [protected]

Time of the last update.

10.142.4.5 transport::NodePtr gazebo::sensors::Sensor::node [protected]

Node for communication.

10.142.4.6 std::string gazebo::sensors::Sensor::parentName [protected]

Name of the parent.

10.142.4.7 std::vector<SensorPluginPtr> gazebo::sensors::Sensor::plugins [protected]

All the plugins for the sensor.

10.142.4.8 math::Pose gazebo::sensors::Sensor::pose [protected]

Pose of the sensor.

10.142.4.9 transport::SubscriberPtr gazebo::sensors::Sensor::poseSub [protected]

Subscribe to pose updates.

10.142.4.10 sdf::ElementPtr gazebo::sensors::Sensor::sdf [protected]

Pointer the the SDF element for the sensor.

10.142.4.11 common::Time gazebo::sensors::Sensor::updatePeriod [protected]

Desired time between updates, set indirectly by **Sensor::SetUpdateRate** (p. 739).

10.142.4.12 gazebo::physics::WorldPtr gazebo::sensors::Sensor::world [protected]

Pointer to the world.

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

- **Sensor.hh**

10.143 SensorFactor Class Reference

The sensor factory; the class is just for namespacing purposes.

`#include <sensors/sensors.hh>`

10.143.1 Detailed Description

The sensor factory; the class is just for namespacing purposes.

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

- **SensorFactory.hh**

10.144 gazebo::sensors::SensorFactory Class Reference

`#include <SensorFactory.hh>`

Static Public Member Functions

- static void **GetSensorTypes** (std::vector< std::string > &_types)

  Get all the sensor types.
- static **SensorPtr NewSensor** (const std::string &_className)

  Create a new instance of a sensor.
- static void **RegisterAll** ()

  Register all known sensors.
- static void **RegisterSensor** (const std::string &_className, **SensorFactoryFn** _factoryfn)

  Register a sensor class (called by sensor registration function).

10.144.1 Member Function Documentation

10.144.1.1 static void gazebo::sensors::SensorFactory::GetSensorTypes (std::vector< std::string > & *_types*) [static]

Get all the sensor types.

Parameters

_types	Vector of strings of the sensor types, populated by function

10.144.1.2 static SensorPtr gazebo::sensors::SensorFactory::NewSensor (const std::string & *_className*) [static]

Create a new instance of a sensor.

Used by the world when reading the world file.

Parameters

in	*_className*	Name of sensor class

Returns

Pointer to **Sensor** (p. 731)

10.144.1.3 static void gazebo::sensors::SensorFactory::RegisterAll ( ) [static]

Register all known sensors.

- **sensors::CameraSensor** (p. 192)
- **sensors::DepthCameraSensor** (p. 265)
- **sensors::GpuRaySensor** (p. 345)
- **sensors::RaySensor** (p. 672)
- **sensors::ContactSensor** (p. 248)
- **sensors::RFIDSensor** (p. 689)
- **sensors::RFIDTag** (p. 691)

10.144.1.4 static void gazebo::sensors::SensorFactory::RegisterSensor (const std::string & *_className*, SensorFactoryFn *_factoryfn*) [static]

Register a sensor class (called by sensor registration function).

Parameters

in	*_className*	Name of class of sensor to register.
in	*_factoryfn*	Function handle for registration.

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

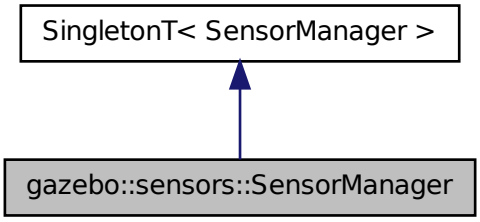
- **SensorFactory.hh**

10.145 gazebo::sensors::SensorManager Class Reference

Class to manage and update all sensors.

`#include <sensors/sensors.hh>`

Inheritance diagram for gazebo::sensors::SensorManager:



Public Member Functions

- std::string **CreateSensor** (**sdf::ElementPtr** _elem, const std::string &_worldName, const std::string &_parentName)

  Add a sensor from an SDF element.
- void **Fini** ()

  Finalize all the sensors.
- **SensorPtr GetSensor** (const std::string &_name) const

  Get a sensor.
- **Sensor_V GetSensors** () const

  Get all the sensors.
- void **GetSensorTypes** (std::vector< std::string > &_types) const

  Get all the sensor types.
- void **Init** ()

  Init all the sensors.
- void **RemoveSensor** (const std::string &_name)

  Remove a sensor.
- void **RemoveSensors** ()

  Remove all sensors.
- void **ResetLastUpdateTimes** ()

  Reset last update times in all sensors.
- void **Run** () **GAZEBO_DEPRECATED**(1.5)

Deprecated.

- void **RunThreads** ()

  Run sensor updates in separate threads.

- bool **SensorsInitialized** ()

  True if SensorManager::initSensors queue is empty i.e.

- void **Stop** ()

  Stop the run thread.

- void **Update** (bool _force=false)

  Update all the sensors.

Additional Inherited Members

10.145.1 Detailed Description

Class to manage and update all sensors.

10.145.2 Member Function Documentation

10.145.2.1 std::string gazebo::sensors::SensorManager::CreateSensor (sdf::ElementPtr *_elem,* const std::string & *_worldName,* const std::string & *_parentName*)

Add a sensor from an SDF element.

This function will also Load and Init the sensor.

Parameters

in	*_elem*	The SDF element that describes the sensor
in	*_worldName*	Name of the world in which to create the sensor
in	*_parentName*	The name of the parent link which the sensor is attached to.

Returns

The name of the sensor

10.145.2.2 void gazebo::sensors::SensorManager::Fini ( )

Finalize all the sensors.

10.145.2.3 SensorPtr gazebo::sensors::SensorManager::GetSensor (const std::string & *_name*) const

Get a sensor.

Parameters

in	*_name*	The name of a sensor to find.

Returns

A pointer to the sensor. NULL if not found.

10.145.2.4 Sensor_V gazebo::sensors::SensorManager::GetSensors ( ) const

Get all the sensors.

Returns

Vector of all the sensors.

10.145.2.5 void gazebo::sensors::SensorManager::GetSensorTypes (std::vector< std::string > & *_types*) const

Get all the sensor types.

Parameters

out	*All*	the sensor types.

10.145.2.6 void gazebo::sensors::SensorManager::Init ( )

Init all the sensors.

10.145.2.7 void gazebo::sensors::SensorManager::RemoveSensor (const std::string & *_name*)

Remove a sensor.

Parameters

in	*_name*	The name of the sensor to remove.

10.145.2.8 void gazebo::sensors::SensorManager::RemoveSensors ( )

Remove all sensors.

10.145.2.9 void gazebo::sensors::SensorManager::ResetLastUpdateTimes ( )

Reset last update times in all sensors.

10.145.2.10 void gazebo::sensors::SensorManager::Run ( )

Deprecated.

See Also

RunThreads (p. 746)

10.145.2.11 void gazebo::sensors::SensorManager::RunThreads ( )

Run sensor updates in separate threads.

This will only run non-image based sensor updates.

10.145.2.12 bool gazebo::sensors::SensorManager::SensorsInitialized ( )

True if SensorManager::initSensors queue is empty i.e.

all sensors managed by **SensorManager** (p. 743) have been initialized

10.145.2.13 void gazebo::sensors::SensorManager::Stop ( )

Stop the run thread.

10.145.2.14 void gazebo::sensors::SensorManager::Update (bool *_force* = `false`)

Update all the sensors.

Checks to see if any sensor need to be initialized first, then updates all sensors once.

Parameters

in	*_force*	True force update, false if not

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

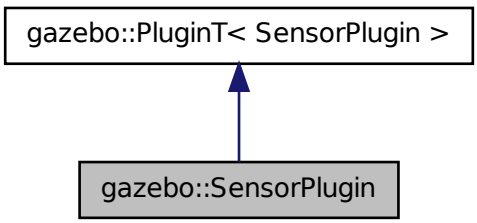
- **SensorManager.hh**

10.146 gazebo::SensorPlugin Class Reference

A plugin with access to physics::Sensor.

`#include <common/common.hh>`

Inheritance diagram for gazebo::SensorPlugin:



Public Member Functions

- **SensorPlugin** ()

  Constructor.
- virtual ∼**SensorPlugin** ()

  Destructor.
- virtual void **Init** ()

  Override this method for custom plugin initialization behavior.
- virtual void **Load** (**sensors::SensorPtr** _sensor, **sdf::ElementPtr** _sdf)=0

  Load function.
- virtual void **Reset** ()

  Override this method for custom plugin reset behavior.

Additional Inherited Members

10.146.1 Detailed Description

A plugin with access to physics::Sensor.

See `reference`.

10.146.2 Constructor & Destructor Documentation

10.146.2.1 gazebo::SensorPlugin::SensorPlugin ( ) [inline]

Constructor.

References gazebo::SENSOR_PLUGIN, and gazebo::PluginT< SensorPlugin >::type.

10.146.2.2 virtual gazebo::SensorPlugin::∼SensorPlugin ( ) [inline], [virtual]

Destructor.

10.146.3 Member Function Documentation

10.146.3.1 virtual void gazebo::SensorPlugin::Init ( ) [inline], [virtual]

Override this method for custom plugin initialization behavior.

10.146.3.2 virtual void gazebo::SensorPlugin::Load (sensors::SensorPtr *_sensor,* sdf::ElementPtr *_sdf*) [pure virtual]

Load function.

Called when a Plugin is first created, and after the World has been loaded. This function should not be blocking.

Parameters

in	*_sensor*	Pointer the Sensor.
in	*_sdf*	Pointer the the SDF element of the plugin.

10.146.3.3 virtual void gazebo::SensorPlugin::Reset ( ) [inline], [virtual]

Override this method for custom plugin reset behavior.

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

- **common/Plugin.hh**

10.147 gazebo::Server Class Reference

`#include <Server.hh>`

Public Member Functions

- **Server** ()
- virtual ∼**Server** ()
- void **Fini** ()
- bool **GetInitialized** () const
- void **Init** ()
- bool **LoadFile** (const std::string &_filename="worlds/empty.world", const std::string &_physics="")

  Load a world file and optionally override physics engine type.
- bool **LoadString** (const std::string &_sdfString)
- bool **ParseArgs** (int argc, char ∗∗argv)
- void **PrintUsage** ()
- void **Run** ()
- void **SetParams** (const **common::StrStr_M** &params)
- void **Stop** ()

Public Attributes

- int **systemPluginsArgc**
- char ∗∗ **systemPluginsArgv**

10.147.1 Constructor & Destructor Documentation

10.147.1.1 gazebo::Server::Server ( )

10.147.1.2 virtual gazebo::Server::∼Server ( ) [virtual]

10.147.2 Member Function Documentation

10.147.2.1 void gazebo::Server::Fini ( )

10.147.2.2 bool gazebo::Server::GetInitialized ( ) const

10.147.2.3 void gazebo::Server::Init ( )

10.147.2.4 bool gazebo::Server::LoadFile (const std::string & *_filename* = "worlds/empty.world", const std::string & *_physics* = "")

Load a world file and optionally override physics engine type.

Parameters

in	*_filename*	Name of the world file to load.
in	*_physics*	Type of physics engine to use (ode|bullet).

10.147.2.5 bool gazebo::Server::LoadString (const std::string & *_sdfString*)

10.147.2.6 bool gazebo::Server::ParseArgs (int *argc*, char ** *argv*)

10.147.2.7 void gazebo::Server::PrintUsage ( )

10.147.2.8 void gazebo::Server::Run ( )

10.147.2.9 void gazebo::Server::SetParams (const common::StrStr_M & *params*)

10.147.2.10 void gazebo::Server::Stop ( )

10.147.3 Member Data Documentation

10.147.3.1 int gazebo::Server::systemPluginsArgc

10.147.3.2 char** gazebo::Server::systemPluginsArgv

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

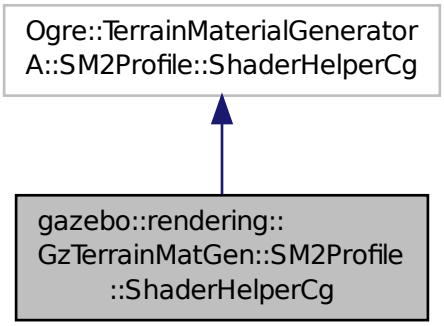
- **Server.hh**

10.148 gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperCg Class Reference

Keeping the CG shader for reference.

`#include <Heightmap.hh>`

Inheritance diagram for gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperCg:



Public Member Functions

- virtual Ogre::HighLevelGpuProgramPtr **generateFragmentProgram** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType _tt)
- virtual Ogre::HighLevelGpuProgramPtr **generateVertexProgram** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType _tt)

Protected Member Functions

- virtual void **defaultVpParams** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType _tt, const Ogre::HighLevelGpuProgramPtr &_prog)
- virtual void **generateVertexProgramSource** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType _tt, Ogre::StringUtil::StrStreamType &_outStream)
- virtual void **generateVpDynamicShadows** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType _tt, Ogre::StringUtil::StrStreamType &_outStream)
- virtual unsigned int **generateVpDynamicShadowsParams** (unsigned int _texCoordStart, const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType _tt, Ogre::StringUtil::StrStreamType &_outStream)
- virtual void **generateVpFooter** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType _tt, Ogre::StringUtil::StrStreamType &_outStream)
- virtual void **generateVpHeader** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType _tt, Ogre::StringUtil::StrStreamType &_outStream)

10.148.1 Detailed Description

Keeping the CG shader for reference.

Utility class to help with generating shaders for Cg / HLSL.

10.148.2 Member Function Documentation

10.148.2.1 virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperCg::defaultVpParams (const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* const Ogre::HighLevelGpuProgramPtr & *_prog*) [protected], [virtual]

10.148.2.2 virtual Ogre::HighLevelGpuProgramPtr gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperCg::generateFragmentProgram (const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt*) [virtual]

10.148.2.3 virtual Ogre::HighLevelGpuProgramPtr gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperCg::generateVertexProgram (const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt*) [virtual]

10.148.2.4 virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperCg::generateVertexProgramSource (const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* Ogre::StringUtil::StrStreamType & *_outStream*) [protected], [virtual]

10.148.2.5 virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperCg::generateVpDynamicShadows (const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* Ogre::StringUtil::StrStreamType & *_outStream*) [protected], [virtual]

10.148.2.6 virtual unsigned int gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperCg::generateVpDynamicShadowsParams (unsigned int *_texCoordStart,* const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* Ogre::StringUtil::StrStreamType & *_outStream*) [protected], [virtual]

10.148.2.7 virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperCg::generateVpFooter (const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* Ogre::StringUtil::StrStreamType & *_outStream*) [protected], [virtual]

10.148.2.8 virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperCg::generateVpHeader (const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* Ogre::StringUtil::StrStreamType & *_outStream*) [protected], [virtual]

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

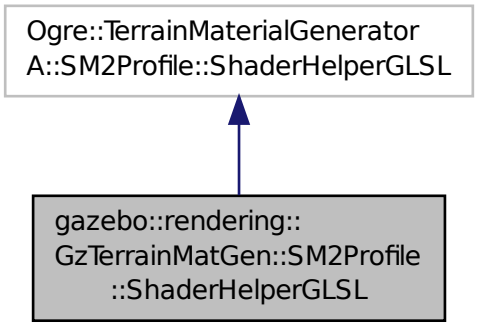
- **Heightmap.hh**

10.149 gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL Class Reference

Utility class to help with generating shaders for GLSL.

`#include <Heightmap.hh>`

Inheritance diagram for gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL:



Public Member Functions

- virtual Ogre::HighLevelGpuProgramPtr **generateFragmentProgram** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType _tt)
- virtual Ogre::HighLevelGpuProgramPtr **generateVertexProgram** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType _tt)
- virtual void **updateParams** (const **SM2Profile** ∗_prof, const Ogre::MaterialPtr &_mat, const Ogre::Terrain ∗_terrain, bool _compositeMap)

Protected Member Functions

- virtual void **defaultVpParams** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType _tt, const Ogre::HighLevelGpuProgramPtr &_prog)
- void **generateFpDynamicShadows** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType _tt, Ogre::StringUtil::StrStreamType &_outStream)
- virtual void **generateFpDynamicShadowsHelpers** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType tt, Ogre::StringUtil::StrStreamType &_outStream)
- virtual void **generateFpDynamicShadowsParams** (Ogre::uint ∗_texCoord, Ogre::uint ∗_sampler, const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType _tt, Ogre::StringUtil::StrStreamType &_outStream)
- virtual void **generateFpFooter** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType tt, Ogre::StringUtil::StrStreamType &_outStream)
- virtual void **generateFpHeader** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType tt, Ogre::StringUtil::StrStreamType &_outStream)
- virtual void **generateFpLayer** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType tt, Ogre::uint _layer, Ogre::StringUtil::StrStreamType &_outStream)
- virtual void **generateFragmentProgramSource** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType _tt, Ogre::StringUtil::StrStreamType &_outStream)
- virtual void **generateVertexProgramSource** (const **SM2Profile** ∗_prof, const Ogre::Terrain ∗_terrain, TechniqueType _tt, Ogre::StringUtil::StrStreamType &_outStream)

- virtual void **generateVpDynamicShadows** (const **SM2Profile** *_prof, const Ogre::Terrain *_terrain, TechniqueType _tt, Ogre::StringUtil::StrStreamType &_outStream)
- virtual unsigned int **generateVpDynamicShadowsParams** (unsigned int _texCoordStart, const **SM2Profile** *_prof, const Ogre::Terrain *_terrain, TechniqueType _tt, Ogre::StringUtil::StrStreamType &_outStream)
- virtual void **generateVpFooter** (const **SM2Profile** *_prof, const Ogre::Terrain *_terrain, TechniqueType _tt, Ogre::StringUtil::StrStreamType &_outStream)
- virtual void **generateVpHeader** (const **SM2Profile** *_prof, const Ogre::Terrain *_terrain, TechniqueType _tt, Ogre::StringUtil::StrStreamType &_outStream)
- virtual void **updateVpParams** (const **SM2Profile** *_prof, const Ogre::Terrain *_terrain, TechniqueType _tt, const Ogre::GpuProgramParametersSharedPtr &_params)

10.149.1 Detailed Description

Utility class to help with generating shaders for GLSL.

10.149.2 Member Function Documentation

10.149.2.1 **virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::defaultVpParams (const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* const Ogre::HighLevelGpuProgramPtr & *_prog*)** [protected], [virtual]

10.149.2.2 **void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::generateFpDynamicShadows (const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* Ogre::StringUtil::StrStreamType & *_outStream*)** [protected]

10.149.2.3 **virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::generateFpDynamicShadowsHelpers (const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *tt,* Ogre::StringUtil::StrStreamType & *_outStream*)** [protected], [virtual]

10.149.2.4 **virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::generateFpDynamicShadowsParams (Ogre::uint * *_texCoord,* Ogre::uint * *_sampler,* const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* Ogre::StringUtil::StrStreamType & *_outStream*)** [protected], [virtual]

10.149.2.5 **virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::generateFpFooter (const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *tt,* Ogre::StringUtil::StrStreamType & *_outStream*)** [protected], [virtual]

10.149.2.6 **virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::generateFpHeader (const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *tt,* Ogre::StringUtil::StrStreamType & *_outStream*)** [protected], [virtual]

10.149.2.7 **virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::generateFpLayer (const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *tt,* Ogre::uint *_layer,* Ogre::StringUtil::StrStreamType & *_outStream*)** [protected], [virtual]

10.149.2.8 **virtual Ogre::HighLevelGpuProgramPtr gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::generateFragmentProgram (const SM2Profile * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt*)** [virtual]

10.149.2.9 virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::generateFragmentProgramSource (const **SM2Profile** * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* Ogre::StringUtil::StrStreamType & *_outStream*) [protected], [virtual]

10.149.2.10 virtual Ogre::HighLevelGpuProgramPtr gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL-::generateVertexProgram (const **SM2Profile** * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt*) [virtual]

10.149.2.11 virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::generateVertexProgramSource (const **SM2Profile** * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* Ogre::StringUtil::StrStreamType & *_outStream*) [protected], [virtual]

10.149.2.12 virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::generateVpDynamicShadows (const **SM2Profile** * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* Ogre::StringUtil::StrStreamType & *_outStream*) [protected], [virtual]

10.149.2.13 virtual unsigned int gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::generateVpDynamic-ShadowsParams (unsigned int *_texCoordStart,* const **SM2Profile** * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* Ogre::StringUtil::StrStreamType & *_outStream*) [protected], [virtual]

10.149.2.14 virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::generateVpFooter (const **SM2Profile** * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* Ogre::StringUtil::StrStreamType & *_outStream*) [protected], [virtual]

10.149.2.15 virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::generateVpHeader (const **SM2Profile** * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* Ogre::StringUtil::StrStreamType & *_outStream*) [protected], [virtual]

10.149.2.16 virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::updateParams (const **SM2Profile** * *_prof,* const Ogre::MaterialPtr & *_mat,* const Ogre::Terrain * *_terrain,* bool *_compositeMap*) [virtual]

10.149.2.17 virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL::updateVpParams (const **SM2Profile** * *_prof,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt,* const Ogre::GpuProgramParametersSharedPtr & *_params*) [protected], [virtual]

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

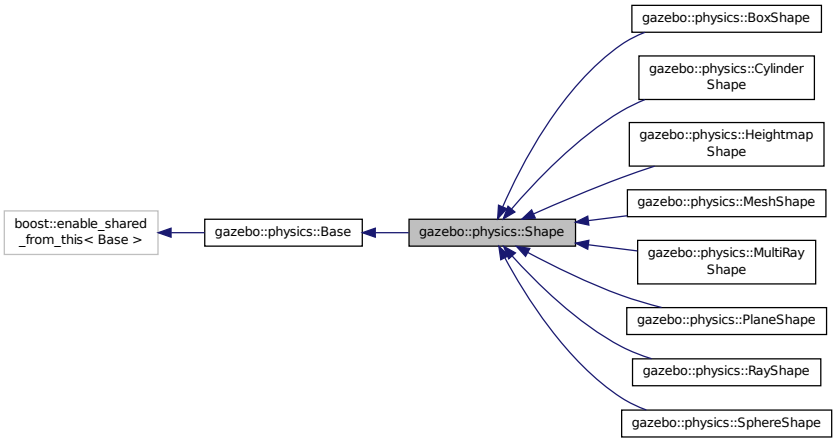
- **Heightmap.hh**

10.150 gazebo::physics::Shape Class Reference

Base (p. 141) class for all shapes.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::Shape:



Public Member Functions

- **Shape** (**CollisionPtr** _parent)

  Constructor.
- virtual ~**Shape** ()

  Destructor.
- virtual void **FillMsg** (msgs::Geometry &_msg)=0

  Fill in the values for a geometry message.
- virtual void **Init** ()=0

  Initialize the shape.
- virtual void **ProcessMsg** (const msgs::Geometry &_msg)=0

  Process a geometry message.

Protected Attributes

- **CollisionPtr collisionParent**

  This shape's collision parent.

Additional Inherited Members

10.150.1 Detailed Description

Base (p. 141) class for all shapes.

10.150.2 Constructor & Destructor Documentation

10.150.2.1 gazebo::physics::Shape::Shape (CollisionPtr *_parent*) [explicit]

Constructor.

Parameters

in	*_parent*	Parent of the shape.

10.150.2.2 virtual gazebo::physics::Shape::~Shape ( ) [virtual]

Destructor.

10.150.3 Member Function Documentation

10.150.3.1 virtual void gazebo::physics::Shape::FillMsg (msgs::Geometry & *_msg*) [pure virtual]

Fill in the values for a geometry message.

Parameters

out	*_msg*	The geometry message to fill.

Implemented in **gazebo::physics::MultiRayShape** (p. 560), **gazebo::physics::RayShape** (p. 681), **gazebo::physics::MeshShape** (p. 514), **gazebo::physics::HeightmapShape** (p. 373), **gazebo::physics::PlaneShape** (p. 621), **gazebo::physics::CylinderShape** (p. 259), **gazebo::physics::SphereShape** (p. 792), and **gazebo::physics::BoxShape** (p. 159).

10.150.3.2 virtual void gazebo::physics::Shape::Init ( ) [pure virtual]

Initialize the shape.

Reimplemented from **gazebo::physics::Base** (p. 149).

Implemented in **gazebo::physics::RayShape** (p. 682), **gazebo::physics::HeightmapShape** (p. 374), **gazebo::physics::MeshShape** (p. 515), **gazebo::physics::MultiRayShape** (p. 563), **gazebo::physics::PlaneShape** (p. 622), **gazebo::physics::SphereShape** (p. 792), **gazebo::physics::BoxShape** (p. 159), and **gazebo::physics::CylinderShape** (p. 260).

10.150.3.3 virtual void gazebo::physics::Shape::ProcessMsg (const msgs::Geometry & *_msg*) [pure virtual]

Process a geometry message.

Parameters

in	*_msg*	The message to set values from.

Implemented in **gazebo::physics::MultiRayShape** (p. 563), **gazebo::physics::RayShape** (p. 682), **gazebo::physics::MeshShape** (p. 515), **gazebo::physics::HeightmapShape** (p. 375), **gazebo::physics::PlaneShape** (p. 622), **gazebo::physics::CylinderShape** (p. 260), **gazebo::physics::SphereShape** (p. 792), and **gazebo::physics::BoxShape** (p. 159).

10.150.4 Member Data Documentation

10.150.4.1 CollisionPtr gazebo::physics::Shape::collisionParent [protected]

This shape's collision parent.

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

- **Shape.hh**

10.151 gazebo::sensors::SimTimeEvent Class Reference

`#include <SensorManager.hh>`

Public Attributes

- boost::condition_variable * **condition**

  The condition to notify.
- **common::Time time**

  The time at which to trigger the condition.

10.151.1 Detailed Description

A simulation time event

10.151.2 Member Data Documentation

10.151.2.1 boost::condition_variable* gazebo::sensors::SimTimeEvent::condition

The condition to notify.

10.151.2.2 common::Time gazebo::sensors::SimTimeEvent::time

The time at which to trigger the condition.

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

- **SensorManager.hh**

10.152 gazebo::sensors::SimTimeEventHandler Class Reference

Monitors simulation time, and notifies conditions when a specified time has been reached.

`#include <SensorManager.hh>`

Public Member Functions

- **SimTimeEventHandler** ()

  Constructor.

- virtual ∼**SimTimeEventHandler** ()

  Destructor.
- void **AddRelativeEvent** (const **common::Time** &_time, boost::condition_variable ∗_var)

  Add a new event to the handler.

10.152.1 Detailed Description

Monitors simulation time, and notifies conditions when a specified time has been reached.

10.152.2 Constructor & Destructor Documentation

10.152.2.1 gazebo::sensors::SimTimeEventHandler::SimTimeEventHandler ( )

Constructor.

10.152.2.2 virtual gazebo::sensors::SimTimeEventHandler::∼SimTimeEventHandler ( ) [virtual]

Destructor.

10.152.3 Member Function Documentation

10.152.3.1 void gazebo::sensors::SimTimeEventHandler::AddRelativeEvent (const common::Time & *_time*, boost::condition_variable ∗ *_var*)

Add a new event to the handler.

Parameters

in	*_time*	Time of the new event. The current sim time will be add to this time.
in	*_var*	Condition to notify when the time has been reached.

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

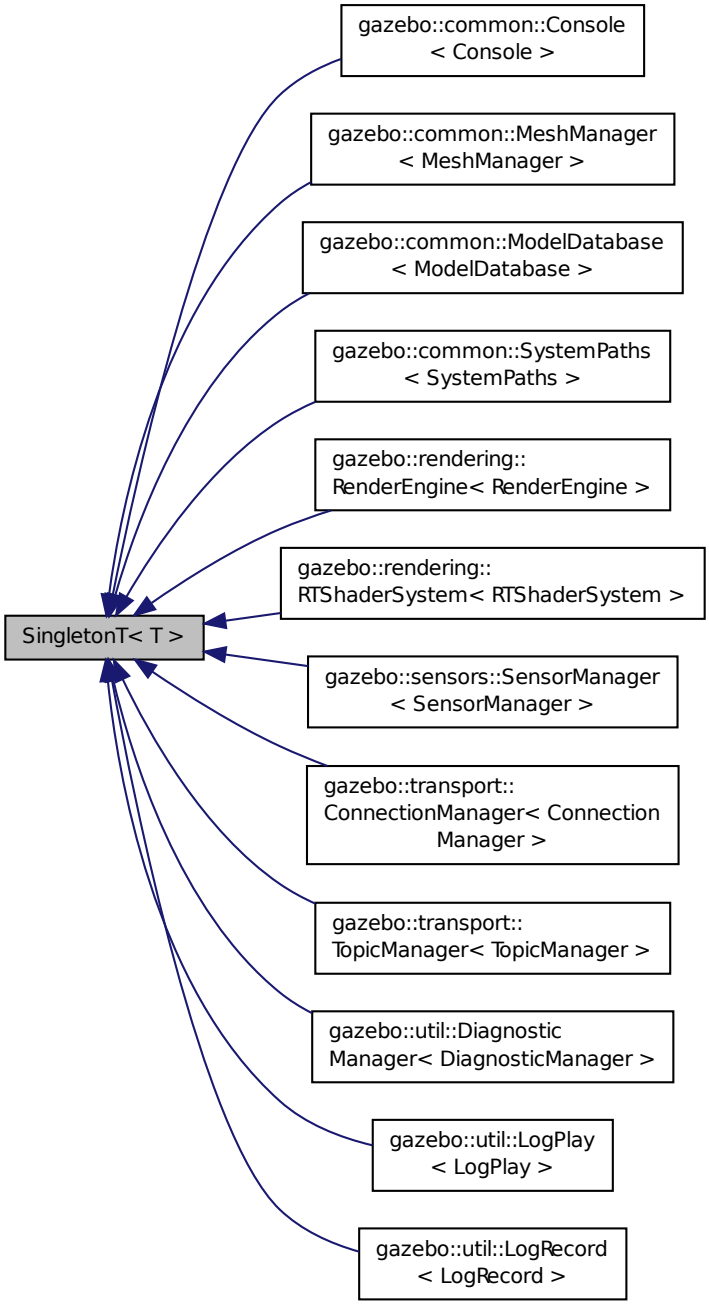
- **SensorManager.hh**

10.153 SingletonT< T > Class Template Reference

Singleton template class.

`#include <common/common.hh>`

Inheritance diagram for SingletonT< T >:



Static Public Member Functions

- static T * **Instance** ()

Get an instance of the singleton.

Protected Member Functions

- **SingletonT** ()
  Constructor.
- virtual ~**SingletonT** ()
  Destructor.

10.153.1 Detailed Description

template<class T>class SingletonT< T >

Singleton template class.

10.153.2 Constructor & Destructor Documentation

10.153.2.1 template<class T> SingletonT< T >::SingletonT ( ) [inline], [protected]

Constructor.

10.153.2.2 template<class T> virtual SingletonT< T >::~SingletonT ( ) [inline], [protected], [virtual]

Destructor.

10.153.3 Member Function Documentation

10.153.3.1 template<class T> static T* SingletonT< T >::Instance ( ) [inline], [static]

Get an instance of the singleton.

Referenced by gazebo::transport::TopicManager::Advertise(), gazebo::transport::Node::Advertise(), gazebo::PluginT< ModelPlugin >::Create(), and gazebo::transport::Node::Subscribe().

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

- **SingletonT.hh**

10.154 gazebo::common::Skeleton Class Reference

A skeleton.

`#include <common/common.hh>`

Public Member Functions

- **Skeleton** ()
  Constructor.

- **Skeleton** (**SkeletonNode** ∗_root)

  Constructor.
- virtual ∼**Skeleton** ()

  Destructor.
- void **AddAnimation** (**SkeletonAnimation** ∗_anim)

  Add an animation.
- void **AddVertNodeWeight** (unsigned int _vertex, std::string _node, double _weight)

  Add a new weight to a node (bone)
- **SkeletonAnimation** ∗ **GetAnimation** (const unsigned int _i)

  Find animation.
- **math::Matrix4 GetBindShapeTransform** ()

  Return bind pose skeletal transform.
- **SkeletonNode** ∗ **GetNodeByHandle** (unsigned int _handle)

  Find or create node with handle.
- **SkeletonNode** ∗ **GetNodeById** (std::string _id)

  Find node by index.
- **SkeletonNode** ∗ **GetNodeByName** (std::string _name)

  Find a node.
- **NodeMap GetNodes** ()

  Get a copy or the node dictionary.
- unsigned int **GetNumAnimations** ()

  Returns the number of animations.
- unsigned int **GetNumJoints** ()

  Returns the number of joints.
- unsigned int **GetNumNodes** ()

  Returns the node count.
- unsigned int **GetNumVertNodeWeights** (unsigned int _vertex)

  Returns the number of bone weights for a vertex.
- **SkeletonNode** ∗ **GetRootNode** ()

  Return the root.
- std::pair< std::string, double > **GetVertNodeWeight** (unsigned int _v, unsigned int _i)

  Weight of a bone for a vertex.
- void **PrintTransforms** ()

  Outputs the transforms to std::err stream.
- void **Scale** (double _scale)

  Scale all nodes, transforms and animation data.
- void **SetBindShapeTransform** (**math::Matrix4** _trans)

  Set the bind pose skeletal transform.
- void **SetNumVertAttached** (unsigned int _vertices)

  Resizes the raw node weight array.
- void **SetRootNode** (**SkeletonNode** ∗_node)

  Change the root node.

Protected Member Functions

- void **BuildNodeMap** ()

  Initializes the hande numbers for each node in the map using breadth first traversal.

Protected Attributes

- std::vector< **SkeletonAnimation** * > **anims**

  the array of animations
- **math::Matrix4 bindShapeTransform**

  the bind pose skeletal transform
- **NodeMap nodes**

  The dictionary of nodes, indexed by name.
- **RawNodeWeights rawNW**

  the node weight table
- **SkeletonNode** * **root**

  the root node

10.154.1 Detailed Description

A skeleton.

10.154.2 Constructor & Destructor Documentation

10.154.2.1 gazebo::common::Skeleton::Skeleton ( )

Constructor.

10.154.2.2 gazebo::common::Skeleton::Skeleton (**SkeletonNode** * *_root*)

Constructor.

Parameters

in	*_root*	node

10.154.2.3 virtual gazebo::common::Skeleton::~Skeleton ( ) [virtual]

Destructor.

10.154.3 Member Function Documentation

10.154.3.1 void gazebo::common::Skeleton::AddAnimation (**SkeletonAnimation** * *_anim*)

Add an animation.

The skeleton does not take ownership of the animation

Parameters

in	*_anim*	the animation to add

10.154.3.2 void gazebo::common::Skeleton::AddVertNodeWeight (unsigned int _vertex, std::string _node, double _weight)

Add a new weight to a node (bone)

Parameters

in	*_vertex*	index of the vertex
in	*_node*	name of the bone
in	*_weight*	the new weight (range 0 to 1)

10.154.3.3 void gazebo::common::Skeleton::BuildNodeMap ( ) [protected]

Initializes the hande numbers for each node in the map using breadth first traversal.

10.154.3.4 SkeletonAnimation* gazebo::common::Skeleton::GetAnimation (const unsigned int _i)

Find animation.

Parameters

in	*_i*	the animation index

Returns

the animation, or NULL if _i is out of bounds

10.154.3.5 math::Matrix4 gazebo::common::Skeleton::GetBindShapeTransform ( )

Return bind pose skeletal transform.

Returns

a matrix

10.154.3.6 SkeletonNode* gazebo::common::Skeleton::GetNodeByHandle (unsigned int _handle)

Find or create node with handle.

Parameters

in	*_handle*	

Returns

the node. A new node is created if it didn't exist

10.154.3.7 SkeletonNode* gazebo::common::Skeleton::GetNodeById (std::string _id)

Find node by index.

Parameters

in	*_id*	the index

Returns

the node, or NULL if not found

10.154.3.8 SkeletonNode* gazebo::common::Skeleton::GetNodeByName (std::string *_name*)

Find a node.

Parameters

in	*_name*	the name of the node to look for

Returns

the node, or NULL if not found

10.154.3.9 NodeMap gazebo::common::Skeleton::GetNodes ( )

Get a copy or the node dictionary.

10.154.3.10 unsigned int gazebo::common::Skeleton::GetNumAnimations ( )

Returns the number of animations.

Returns

the count

10.154.3.11 unsigned int gazebo::common::Skeleton::GetNumJoints ( )

Returns the number of joints.

Returns

the count

10.154.3.12 unsigned int gazebo::common::Skeleton::GetNumNodes ( )

Returns the node count.

Returns

the count

10.154.3.13 unsigned int gazebo::common::Skeleton::GetNumVertNodeWeights (unsigned int _vertex)

Returns the number of bone weights for a vertex.

Parameters

in	*_vertex*	the index of the vertex

Returns

the count

10.154.3.14 SkeletonNode* gazebo::common::Skeleton::GetRootNode ( )

Return the root.

Returns

the root

10.154.3.15 std::pair<std::string, double> gazebo::common::Skeleton::GetVertNodeWeight (unsigned int _v, unsigned int _i)

Weight of a bone for a vertex.

Parameters

in	*_v*	the index of the vertex
in	*_i*	the index of the weight for that vertex

Returns

a pair containing the name of the node and the weight

10.154.3.16 void gazebo::common::Skeleton::PrintTransforms ( )

Outputs the transforms to std::err stream.

10.154.3.17 void gazebo::common::Skeleton::Scale (double _scale)

Scale all nodes, transforms and animation data.

Parameters

in	*the*	scaling factor

10.154.3.18 void gazebo::common::Skeleton::SetBindShapeTransform (math::Matrix4 _trans)

Set the bind pose skeletal transform.

Parameters

in	*_trans*	the transform

10.154.3.19 void gazebo::common::Skeleton::SetNumVertAttached (unsigned int *_vertices*)

Resizes the raw node weight array.

Parameters

in	*_vertices*	the new size

10.154.3.20 void gazebo::common::Skeleton::SetRootNode (SkeletonNode * *_node*)

Change the root node.

Parameters

in	*_node*	the new node

10.154.4 Member Data Documentation

10.154.4.1 std::vector<SkeletonAnimation*> gazebo::common::Skeleton::anims [protected]

the array of animations

10.154.4.2 math::Matrix4 gazebo::common::Skeleton::bindShapeTransform [protected]

the bind pose skeletal transform

10.154.4.3 NodeMap gazebo::common::Skeleton::nodes [protected]

The dictionary of nodes, indexed by name.

10.154.4.4 RawNodeWeights gazebo::common::Skeleton::rawNW [protected]

the node weight table

10.154.4.5 SkeletonNode* gazebo::common::Skeleton::root [protected]

the root node

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

- **Skeleton.hh**

10.155 gazebo::common::SkeletonAnimation Class Reference

Skeleton (p. 760) animation.

`#include <SkeletonAnimation.hh>`

Public Member Functions

- **SkeletonAnimation** (const std::string &_name)

  The Constructor.
- ∼**SkeletonAnimation** ()

  The destructor.
- void **AddKeyFrame** (const std::string &_node, const double _time, const **math::Matrix4** _mat)

  Adds or replaces a named key frame at a specific time.
- void **AddKeyFrame** (const std::string &_node, const double _time, const **math::Pose** _pose)

  Adds or replaces a named key frame at a specific time.
- double **GetLength** () const

  Returns the duration of the animations.
- std::string **GetName** () const

  Returns the name.
- unsigned int **GetNodeCount** () const

  Returns the number of animation nodes.
- **math::Matrix4 GetNodePoseAt** (const std::string &_node, const double _time, const bool _loop=true)

  Returns the key frame transformation for a named animation at a specific time if a node does not exist at that time (with tolerance of 1e-6 sec), the transformation is interpolated.
- std::map< std::string, **math::Matrix4** > **GetPoseAt** (const double _time, const bool _loop=true) const

  Returns a dictionary of transformations indexed by name at a specific time if a node does not exist at that specific time (with tolerance of 1e-6 sec), the transformation is interpolated.
- std::map< std::string, **math::Matrix4** > **GetPoseAtX** (const double _x, const std::string &_node, const bool _loop=true) const

  Returns a dictionary of transformations indexed by name where a named node transformation's translational value along the X axis is equal to _x.
- bool **HasNode** (const std::string &_node) const

  Looks for a node with a specific name in the animations.
- void **Scale** (const double _scale)

  Scales every animation in the animations list.
- void **SetName** (const std::string &_name)

  Changes the name.

Protected Attributes

- std::map< std::string, **NodeAnimation** ∗ > **animations**

  a dictionary of node animations
- double **length**

  the duration of the longest animation
- std::string **name**

  the node name

10.155.1 Detailed Description

Skeleton (p. 760) animation.

10.155.2 Constructor & Destructor Documentation

10.155.2.1 gazebo::common::SkeletonAnimation::SkeletonAnimation (const std::string & *_name*)

The Constructor.

Parameters

in	*_name*	the name of the animation

10.155.2.2 gazebo::common::SkeletonAnimation::~SkeletonAnimation ( )

The destructor.

Clears the list without destroying the animations

10.155.3 Member Function Documentation

10.155.3.1 void gazebo::common::SkeletonAnimation::AddKeyFrame (const std::string & *_node*, const double *_time*, const math::Matrix4 *_mat*)

Adds or replaces a named key frame at a specific time.

Parameters

in	*_node*	the name of the new or existing node
in	*_time*	the time
in	*_mat*	the key frame transformation

10.155.3.2 void gazebo::common::SkeletonAnimation::AddKeyFrame (const std::string & *_node*, const double *_time*, const math::Pose *_pose*)

Adds or replaces a named key frame at a specific time.

Parameters

in	*_node*	the name of the new or existing node
in	*_time*	the time
in	*_pose*	the key frame transformation as a **math::Pose** (p. 626)

10.155.3.3 double gazebo::common::SkeletonAnimation::GetLength ( ) const

Returns the duration of the animations.

Returns

the duration in seconds

10.155.3.4 std::string gazebo::common::SkeletonAnimation::GetName ( ) const

Returns the name.

Returns

the name

10.155.3.5 unsigned int gazebo::common::SkeletonAnimation::GetNodeCount ( ) const

Returns the number of animation nodes.

Returns

the count

10.155.3.6 math::Matrix4 gazebo::common::SkeletonAnimation::GetNodePoseAt (const std::string & *_node*, const double *_time*, const bool *_loop* = `true`)

Returns the key frame transformation for a named animation at a specific time if a node does not exist at that time (with tolerance of 1e-6 sec), the transformation is interpolated.

Parameters

in	*_node*	the name of the animation node
in	*_time*	the time
in	*_loop*	when true, the time is divided by the duration (see GetLength)

Returns

the transformation

10.155.3.7 std::map<std::string, math::Matrix4> gazebo::common::SkeletonAnimation::GetPoseAt (const double *_time*, const bool *_loop* = `true`) const

Returns a dictionary of transformations indexed by name at a specific time if a node does not exist at that specific time (with tolerance of 1e-6 sec), the transformation is interpolated.

Parameters

in	*_time*	the time
in	*_loop*	when true, the time is divided by the duration (see GetLength)

Returns

the transformation for every node

10.155.3.8 std::map<std::string, math::Matrix4> gazebo::common::SkeletonAnimation::GetPoseAtX (const double *_x,* const std::string & *_node,* const bool *_loop* = `true`) const

Returns a dictionary of transformations indexed by name where a named node transformation's translational value along the X axis is equal to _x.

Parameters

in	*_x*	the value along x. You must ensure that _x is within a valid range.
in	*_node*	the name of the animation node
in	*_loop*	when true, the time is divided by the duration (see GetLength)

10.155.3.9 bool gazebo::common::SkeletonAnimation::HasNode (const std::string & *_node*) const

Looks for a node with a specific name in the animations.

Parameters

in	*_node*	the name of the node

Returns

true if the node exits

10.155.3.10 void gazebo::common::SkeletonAnimation::Scale (const double *_scale*)

Scales every animation in the animations list.

Parameters

in	*_scale*	the scaling factor

10.155.3.11 void gazebo::common::SkeletonAnimation::SetName (const std::string & *_name*)

Changes the name.

Parameters

in	*_name*	the new name

10.155.4 Member Data Documentation

10.155.4.1 std::map<std::string, NodeAnimation*> gazebo::common::SkeletonAnimation::animations [protected]

a dictionary of node animations

10.155.4.2 double gazebo::common::SkeletonAnimation::length [protected]

the duration of the longest animation

10.155.4.3 std::string gazebo::common::SkeletonAnimation::name [protected]

the node name

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

- **SkeletonAnimation.hh**

10.156 gazebo::common::SkeletonNode Class Reference

A skeleton node.

`#include <common/common.hh>`

Public Types

- enum **SkeletonNodeType** { **NODE**, **JOINT** }
  enumeration of node types

Public Member Functions

- **SkeletonNode** (**SkeletonNode** *_parent)
  Constructor.
- **SkeletonNode** (**SkeletonNode** *_parent, std::string _name, std::string _id, **SkeletonNodeType** _type=**JOINT**)
  Constructor.
- virtual ~**SkeletonNode** ()
  Destructor.
- void **AddChild** (**SkeletonNode** *_child)
  Add a new child.
- void **AddRawTransform** (**NodeTransform** _t)
  Add a raw transform.
- **SkeletonNode** * **GetChild** (unsigned int _index)
  Find a child by index.
- **SkeletonNode** * **GetChildById** (std::string _id)
  Get child by string id.
- **SkeletonNode** * **GetChildByName** (std::string _name)
  Get child by name.
- unsigned int **GetChildCount** ()
  Returns the children count.
- unsigned int **GetHandle** ()
  Get the handle index.
- std::string **GetId** ()
  Returns the index.

- **math::Matrix4 GetInverseBindTransform** ()

  Retrieve the inverse of the bind pose skeletal transform.
- **math::Matrix4 GetModelTransform** ()

  Retrieve the model transform.
- std::string **GetName** ()

  Returns the name.
- unsigned int **GetNumRawTrans** ()

  Return the raw transformations count.
- **SkeletonNode** ∗ **GetParent** ()

  Returns the parent node.
- **NodeTransform GetRawTransform** (unsigned int _i)

  Find a raw transformation.
- std::vector< **NodeTransform** > **GetRawTransforms** ()

  Retrieve the raw transformations.
- **math::Matrix4 GetTransform** ()

  Get transform relative to parent.
- std::vector< **NodeTransform** > **GetTransforms** ()

  Returns a copy of the array of transformations.
- bool **IsJoint** ()

  Is a joint query.
- bool **IsRootNode** ()

  Queries wether a node has no parent parent.
- void **Reset** (bool _resetChildren)

  Reset the transformation to the initial transformation.
- void **SetHandle** (unsigned int _h)

  Assign a handle number.
- void **SetId** (std::string _id)

  Change the id string.
- void **SetInitialTransform** (**math::Matrix4** _tras)

  Sets the initial transformation.
- void **SetInverseBindTransform** (**math::Matrix4** _invBM)

  Assign the inverse of the bind pose skeletal transform.
- void **SetModelTransform** (**math::Matrix4** _trans, bool _updateChildren=true)

  Set the model transformation.
- void **SetName** (std::string _name)

  Change the name.
- void **SetParent** (**SkeletonNode** ∗_parent)

  Set the parent node.
- void **SetTransform** (**math::Matrix4** _trans, bool _updateChildren=true)

  Set a transformation.
- void **SetType** (**SkeletonNodeType** _type)

  Change the skeleton node type.
- void **UpdateChildrenTransforms** ()

  Apply model transformations in order for each node in the tree.

Protected Attributes

- std::vector< **SkeletonNode** * > **children**

  the children nodes
- unsigned int **handle**

  handle index number
- std::string **id**

  a string identifier
- **math::Matrix4 initialTransform**

  the initial transformation
- **math::Matrix4 invBindTransform**

  the inverse of the bind pose skeletal transform
- **math::Matrix4 modelTransform**

  the model transformation
- std::string **name**

  the name of the skeletal node
- **SkeletonNode** * **parent**

  the parent node
- std::vector< **NodeTransform** > **rawTransforms**

  the raw transformation
- **math::Matrix4 transform**

  the transform
- **SkeletonNodeType type**

  the type fo node

10.156.1 Detailed Description

A skeleton node.

10.156.2 Member Enumeration Documentation

10.156.2.1 enum gazebo::common::SkeletonNode::SkeletonNodeType

enumeration of node types

Enumerator:

NODE

JOINT

10.156.3 Constructor & Destructor Documentation

10.156.3.1 gazebo::common::SkeletonNode::SkeletonNode (SkeletonNode * *_parent*)

Constructor.

Parameters

in	*_parent*	The parent node

10.156.3.2 gazebo::common::SkeletonNode::SkeletonNode (SkeletonNode * _parent, std::string _name, std::string _id, SkeletonNodeType _type = JOINT)

Constructor.

Parameters

in	*_parent*	the parent node
in	*_name*	name of node
in	*_id*	Id of node
in	*_type*	The type of this node

10.156.3.3 virtual gazebo::common::SkeletonNode::~SkeletonNode ( ) [virtual]

Destructor.

10.156.4 Member Function Documentation

10.156.4.1 void gazebo::common::SkeletonNode::AddChild (SkeletonNode * _child)

Add a new child.

Parameters

in	*_child*	a child

10.156.4.2 void gazebo::common::SkeletonNode::AddRawTransform (NodeTransform _t)

Add a raw transform.

Parameters

in	*_t*	the transform

10.156.4.3 SkeletonNode* gazebo::common::SkeletonNode::GetChild (unsigned int _index)

Find a child by index.

Parameters

in	*_index*	the index

Returns

the child skeleton. NO BOUNDS CHECKING

10.156.4.4 SkeletonNode* gazebo::common::SkeletonNode::GetChildById (std::string _id)

Get child by string id.

Parameters

in	*_id*	the string id

Returns

the child skeleton or NULL if not found

10.156.4.5 SkeletonNode* gazebo::common::SkeletonNode::GetChildByName (std::string *_name*)

Get child by name.

Parameters

in	*_name*	the name of the child skeleton

Returns

the skeleton, or NULL if not found

10.156.4.6 unsigned int gazebo::common::SkeletonNode::GetChildCount ( )

Returns the children count.

Returns

the count

10.156.4.7 unsigned int gazebo::common::SkeletonNode::GetHandle ( )

Get the handle index.

Returns

the handle index

10.156.4.8 std::string gazebo::common::SkeletonNode::GetId ( )

Returns the index.

Returns

the id string

10.156.4.9 math::Matrix4 gazebo::common::SkeletonNode::GetInverseBindTransform ( )

Retrieve the inverse of the bind pose skeletal transform.

Returns

the transform

10.156.4.10 math::Matrix4 gazebo::common::SkeletonNode::GetModelTransform ( )

Retrieve the model transform.

Returns

the transform

10.156.4.11 std::string gazebo::common::SkeletonNode::GetName ( )

Returns the name.

Returns

the name

10.156.4.12 unsigned int gazebo::common::SkeletonNode::GetNumRawTrans ( )

Return the raw transformations count.

Returns

the count

10.156.4.13 SkeletonNode* gazebo::common::SkeletonNode::GetParent ( )

Returns the parent node.

Returns

the parent

10.156.4.14 NodeTransform gazebo::common::SkeletonNode::GetRawTransform (unsigned int _i)

Find a raw transformation.

Parameters

in	_i	the index of the transformation

Returns

the node transform. NO BOUNDS CHECKING PERFORMED

10.156.4.15 std::vector<NodeTransform> gazebo::common::SkeletonNode::GetRawTransforms ( )

Retrieve the raw transformations.

Returns

an array of transformations

10.156.4.16 math::Matrix4 gazebo::common::SkeletonNode::GetTransform ( )

Get transform relative to parent.

10.156.4.17 std::vector<NodeTransform> gazebo::common::SkeletonNode::GetTransforms ( )

Returns a copy of the array of transformations.

Returns

the array of transform (These are the same as the raw trans)

10.156.4.18 bool gazebo::common::SkeletonNode::IsJoint ( )

Is a joint query.

Returns

true if the skeleton type is a joint, false otherwise

10.156.4.19 bool gazebo::common::SkeletonNode::IsRootNode ( )

Queries wether a node has no parent parent.

Returns

true if the node has no parent, fasle otherwise

10.156.4.20 void gazebo::common::SkeletonNode::Reset (bool *_resetChildren*)

Reset the transformation to the initial transformation.

Parameters

in	*_resetChildren*	when true, performs the operation for every node in the tree

10.156.4.21 void gazebo::common::SkeletonNode::SetHandle (unsigned int *_h*)

Assign a handle number.

Parameters

in	*_h*	the handle

10.156.4.22 void gazebo::common::SkeletonNode::SetId (std::string _id)

Change the id string.

Parameters

in	*_id*	the new id string

10.156.4.23 void gazebo::common::SkeletonNode::SetInitialTransform (math::Matrix4 _tras)

Sets the initial transformation.

Parameters

in	*_tras*	the transfromation matrix

10.156.4.24 void gazebo::common::SkeletonNode::SetInverseBindTransform (math::Matrix4 _invBM)

Assign the inverse of the bind pose skeletal transform.

Parameters

in	*_invBM*	the transform

10.156.4.25 void gazebo::common::SkeletonNode::SetModelTransform (math::Matrix4 _trans, bool _updateChildren = true)

Set the model transformation.

Parameters

in	*_trans*	the transformation
in	*_updateChildren*	when true the UpdateChildrenTransforms operation is performed

10.156.4.26 void gazebo::common::SkeletonNode::SetName (std::string _name)

Change the name.

Parameters

in	*_name*	the new name

10.156.4.27 void gazebo::common::SkeletonNode::SetParent (SkeletonNode * _parent)

Set the parent node.

Parameters

in	*_parent*	the new parent

10.156.4.28 void gazebo::common::SkeletonNode::SetTransform (math::Matrix4 _trans, bool _updateChildren = true)

Set a transformation.

Parameters

in	*_trans*	the transformation
in	*_updateChildren*	when true the UpdateChildrenTransforms operation is performed

10.156.4.29 void gazebo::common::SkeletonNode::SetType (SkeletonNodeType _type)

Change the skeleton node type.

Parameters

in	*_type*	the new type

10.156.4.30 void gazebo::common::SkeletonNode::UpdateChildrenTransforms ( )

Apply model transformations in order for each node in the tree.

10.156.5 Member Data Documentation

10.156.5.1 std::vector<SkeletonNode*> gazebo::common::SkeletonNode::children [protected]

the children nodes

10.156.5.2 unsigned int gazebo::common::SkeletonNode::handle [protected]

handle index number

10.156.5.3 std::string gazebo::common::SkeletonNode::id [protected]

a string identifier

10.156.5.4 math::Matrix4 gazebo::common::SkeletonNode::initialTransform [protected]

the initial transformation

10.156.5.5 math::Matrix4 gazebo::common::SkeletonNode::invBindTransform [protected]

the inverse of the bind pose skeletal transform

10.156.5.6 math::Matrix4 gazebo::common::SkeletonNode::modelTransform [protected]

the model transformation

10.156.5.7 std::string gazebo::common::SkeletonNode::name [protected]

the name of the skeletal node

10.156.5.8 SkeletonNode* gazebo::common::SkeletonNode::parent [protected]

the parent node

10.156.5.9 std::vector<NodeTransform> gazebo::common::SkeletonNode::rawTransforms [protected]

the raw transformation

10.156.5.10 math::Matrix4 gazebo::common::SkeletonNode::transform [protected]

the transform

10.156.5.11 SkeletonNodeType gazebo::common::SkeletonNode::type [protected]

the type fo node

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

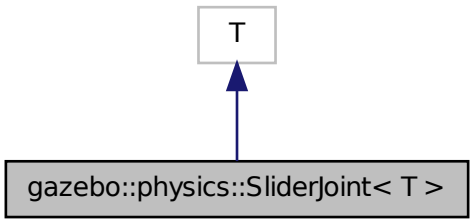
- **Skeleton.hh**

10.157 gazebo::physics::SliderJoint< T > Class Template Reference

A slider joint.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::SliderJoint< T >:



Public Member Functions

- **SliderJoint** (**BasePtr** _parent)

Constructor.

- virtual ~**SliderJoint** ()

  Destructor.
- virtual **math::Vector3 GetAnchor** (int _index) const

  Get the anchor.
- virtual unsigned int **GetAngleCount** () const
- virtual void **Load** (**sdf::ElementPtr** _sdf)

  *Load a **SliderJoint** (p. 780).*
- virtual void **SetAnchor** (int _index, const **math::Vector3** &_anchor)

  Set the anchor.

Protected Attributes

- **math::Vector3 fakeAnchor**

  The anchor value is not used internally.

10.157.1 Detailed Description

template<class T>class gazebo::physics::SliderJoint< T >

A slider joint.

10.157.2 Constructor & Destructor Documentation

10.157.2.1 template<class T > gazebo::physics::SliderJoint< T >::SliderJoint (BasePtr *_parent*) [inline], [explicit]

Constructor.

Parameters

in	*_parent*	Parent of the joint.

References gazebo::physics::Base::SLIDER_JOINT.

10.157.2.2 template<class T > virtual gazebo::physics::SliderJoint< T >::~SliderJoint ( ) [inline], [virtual]

Destructor.

10.157.3 Member Function Documentation

10.157.3.1 template<class T > math::Vector3 gazebo::physics::SliderJoint< T >::GetAnchor (int *_index*) const [virtual]

Get the anchor.

Parameters

in	*_index*	Index of the axis. Not used.

Returns

Anchor for the joint.

10.157.3.2 template<class T > virtual unsigned int gazebo::physics::SliderJoint< T >::GetAngleCount ( ) const [inline], [virtual]

10.157.3.3 template<class T > virtual void gazebo::physics::SliderJoint< T >::Load (sdf::ElementPtr *_sdf*) [inline], [virtual]

Load a **SliderJoint** (p. 780).

Parameters

in	*_sdf*	SDF values to load from

10.157.3.4 template<class T > void gazebo::physics::SliderJoint< T >::SetAnchor (int *_index*, const math::Vector3 & *_anchor*) [virtual]

Set the anchor.

Parameters

in	*_index*	Index of the axis. Not used.
in	*_anchor*	Anchor for the axis.

10.157.4 Member Data Documentation

10.157.4.1 template<class T > math::Vector3 gazebo::physics::SliderJoint< T >::fakeAnchor [protected]

The anchor value is not used internally.

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

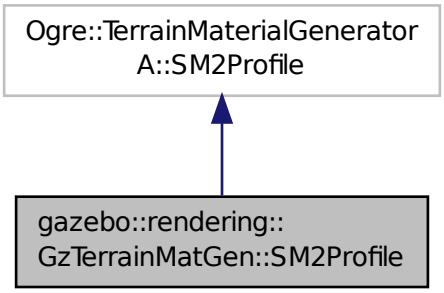
- **SliderJoint.hh**

10.158 gazebo::rendering::GzTerrainMatGen::SM2Profile Class Reference

Shader model 2 profile target.

`#include <Heightmap.hh>`

Inheritance diagram for gazebo::rendering::GzTerrainMatGen::SM2Profile:



Classes

- class **ShaderHelperCg**

  Keeping the CG shader for reference.
- class **ShaderHelperGLSL**

  Utility class to help with generating shaders for GLSL.

Public Member Functions

- **SM2Profile** (Ogre::TerrainMaterialGenerator ∗_parent, const Ogre::String &_name, const Ogre::String &_desc)

  Constructor.
- virtual ∼**SM2Profile** ()

  Destructor.
- Ogre::MaterialPtr **generate** (const Ogre::Terrain ∗_terrain)
- Ogre::MaterialPtr **generateForCompositeMap** (const Ogre::Terrain ∗_terrain)
- void **UpdateParams** (const Ogre::MaterialPtr &_mat, const Ogre::Terrain ∗_terrain)
- void **UpdateParamsForCompositeMap** (const Ogre::MaterialPtr &_mat, const Ogre::Terrain ∗_terrain)

Protected Member Functions

- virtual void **addTechnique** (const Ogre::MaterialPtr &_mat, const Ogre::Terrain ∗_terrain, TechniqueType _tt)

10.158.1 Detailed Description

Shader model 2 profile target.

10.158.2 Constructor & Destructor Documentation

10.158.2.1 gazebo::rendering::GzTerrainMatGen::SM2Profile::SM2Profile (Ogre::TerrainMaterialGenerator * *_parent,* const Ogre::String & *_name,* const Ogre::String & *_desc*)

Constructor.

10.158.2.2 virtual gazebo::rendering::GzTerrainMatGen::SM2Profile::~SM2Profile ( ) [virtual]

Destructor.

10.158.3 Member Function Documentation

10.158.3.1 virtual void gazebo::rendering::GzTerrainMatGen::SM2Profile::addTechnique (const Ogre::MaterialPtr & *_mat,* const Ogre::Terrain * *_terrain,* TechniqueType *_tt*) [protected], [virtual]

10.158.3.2 Ogre::MaterialPtr gazebo::rendering::GzTerrainMatGen::SM2Profile::generate (const Ogre::Terrain * *_terrain*)

10.158.3.3 Ogre::MaterialPtr gazebo::rendering::GzTerrainMatGen::SM2Profile::generateForCompositeMap (const Ogre::Terrain * *_terrain*)

10.158.3.4 void gazebo::rendering::GzTerrainMatGen::SM2Profile::UpdateParams (const Ogre::MaterialPtr & *_mat,* const Ogre::Terrain * *_terrain*)

10.158.3.5 void gazebo::rendering::GzTerrainMatGen::SM2Profile::UpdateParamsForCompositeMap (const Ogre::MaterialPtr & *_mat,* const Ogre::Terrain * *_terrain*)

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

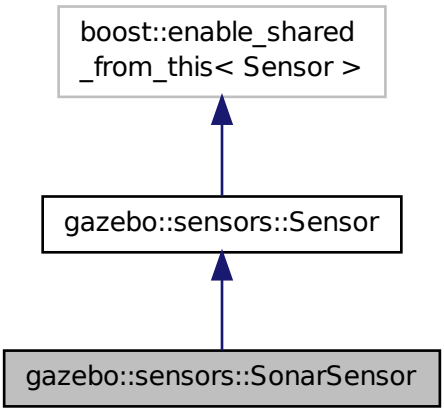
- **Heightmap.hh**

10.159 gazebo::sensors::SonarSensor Class Reference

Sensor (p. 731) with sonar cone.

`#include <sensors/sensors.hh>`

Inheritance diagram for gazebo::sensors::SonarSensor:



Public Member Functions

- **SonarSensor** ()

  Constructor.
- virtual ~**SonarSensor** ()

  Destructor.
- template<typename T >
  event::ConnectionPtr ConnectUpdate (T _subscriber)

  Connect a to the new update signal.
- void **DisconnectUpdate** (**event::ConnectionPtr** &_conn)

  Disconnect from the update signal.
- double **GetRadius** () const

  Get the radius of the sonar cone at maximum range.
- double **GetRange** ()

  Get detected range for a sonar.
- double **GetRangeMax** () const

  Get the minimum range of the sonar.
- double **GetRangeMin** () const

  Get the minimum range of the sonar.
- virtual std::string **GetTopic** () const

  Returns the topic name as set in SDF.
- virtual void **Init** ()

  Initialize the sensor.
- virtual bool **IsActive** ()

  Returns true if sensor generation is active.
- virtual void **Load** (const std::string &_worldName)

  Load the sensor with default parameters.

Protected Member Functions

- virtual void **Fini** ()

  Finalize the sensor.
- virtual void **UpdateImpl** (bool _force)

  This gets overwritten by derived sensor types.

Protected Attributes

- **event::EventT**< void(msgs::SonarStamped)> **update**

  Update event.

10.159.1 Detailed Description

Sensor (p. 731) with sonar cone.

This sensor uses a cone .

10.159.2 Constructor & Destructor Documentation

10.159.2.1 gazebo::sensors::SonarSensor::SonarSensor ( )

Constructor.

10.159.2.2 virtual gazebo::sensors::SonarSensor::~SonarSensor ( ) [virtual]

Destructor.

10.159.3 Member Function Documentation

10.159.3.1 template<typename T > event::ConnectionPtr gazebo::sensors::SonarSensor::ConnectUpdate (T _subscriber) [inline]

Connect a to the new update signal.

Parameters

in	*_subscriber*	Callback function.

Returns

The connection, which must be kept in scope.

References gazebo::event::EventT< T >::Connect(), and update.

10.159.3.2 void gazebo::sensors::SonarSensor::DisconnectUpdate (event::ConnectionPtr & _conn) [inline]

Disconnect from the update signal.

Parameters

in	*_conn*	Connection to remove.

References gazebo::event::EventT< T >::Disconnect(), and update.

10.159.3.3 virtual void gazebo::sensors::SonarSensor::Fini ( ) [protected], [virtual]

Finalize the sensor.

Reimplemented from **gazebo::sensors::Sensor** (p. 735).

10.159.3.4 double gazebo::sensors::SonarSensor::GetRadius ( ) const

Get the radius of the sonar cone at maximum range.

Returns

The radisu of the sonar cone at max range.

10.159.3.5 double gazebo::sensors::SonarSensor::GetRange ( )

Get detected range for a sonar.

```
Warning: If you are accessing all the ray data in a loop
it's possible that the Ray will update in the middle of
your access loop. This means some data will come from one
scan, and some from another scan. You can solve this
problem by using SetActive(false) <your accessor loop>
SetActive(true).
```

Returns

Returns DBL_MAX for no detection.

10.159.3.6 double gazebo::sensors::SonarSensor::GetRangeMax ( ) const

Get the minimum range of the sonar.

Returns

The sonar's maximum range.

10.159.3.7 double gazebo::sensors::SonarSensor::GetRangeMin ( ) const

Get the minimum range of the sonar.

Returns

The sonar's minimum range.

10.159.3.8 virtual std::string gazebo::sensors::SonarSensor::GetTopic ( ) const [virtual]

Returns the topic name as set in SDF.

Returns

Topic name.

Reimplemented from **gazebo::sensors::Sensor** (p. 737).

10.159.3.9 virtual void gazebo::sensors::SonarSensor::Init ( ) [virtual]

Initialize the sensor.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.159.3.10 virtual bool gazebo::sensors::SonarSensor::IsActive ( ) [virtual]

Returns true if sensor generation is active.

Returns

True if active, false if not.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.159.3.11 virtual void gazebo::sensors::SonarSensor::Load (const std::string & *_worldName*) [virtual]

Load the sensor with default parameters.

Parameters

in	*_worldName*	Name of world to load from.

Reimplemented from **gazebo::sensors::Sensor** (p. 738).

10.159.3.12 virtual void gazebo::sensors::SonarSensor::UpdateImpl (bool) [protected], [virtual]

This gets overwritten by derived sensor types.

```
This function is called during Sensor::Update.
And in turn, Sensor::Update is called by
SensorManager::Update
```

Parameters

in	*_force*	True if update is forced, false if not

Reimplemented from **gazebo::sensors::Sensor** (p. 739).

10.159.4 Member Data Documentation

10.159.4.1 event::EventT<void(msgs::SonarStamped)> gazebo::sensors::SonarSensor::update [protected]

Update event.

Referenced by ConnectUpdate(), and DisconnectUpdate().

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

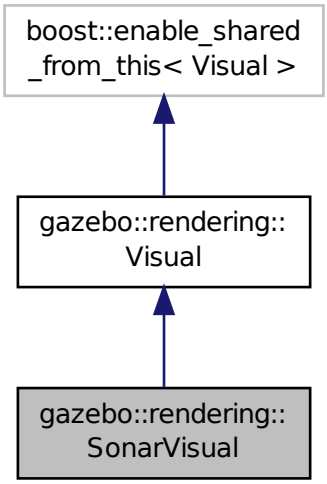
- **SonarSensor.hh**

10.160 gazebo::rendering::SonarVisual Class Reference

Visualization for sonar data.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::SonarVisual:



Public Member Functions

- **SonarVisual** (const std::string &_name, **VisualPtr** _vis, const std::string &_topicName)

  Constructor.
- virtual ~**SonarVisual** ()

  Destructor.
- virtual void **Load** ()

  Load the visual with default parameters.

Additional Inherited Members

10.160.1 Detailed Description

Visualization for sonar data.

10.160.2 Constructor & Destructor Documentation

10.160.2.1 gazebo::rendering::SonarVisual::SonarVisual (const std::string & *_name,* VisualPtr *_vis,* const std::string & *_topicName*)

Constructor.

Parameters

in	*_name*	Name of the visual.
in	*_vis*	Pointer to the parent **Visual** (p. 920).
in	*_topicName*	Name of the topic that has sonar data.

10.160.2.2 virtual gazebo::rendering::SonarVisual::~SonarVisual ( ) [virtual]

Destructor.

10.160.3 Member Function Documentation

10.160.3.1 virtual void gazebo::rendering::SonarVisual::Load ( ) [virtual]

Load the visual with default parameters.

Reimplemented from **gazebo::rendering::Visual** (p. 933).

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

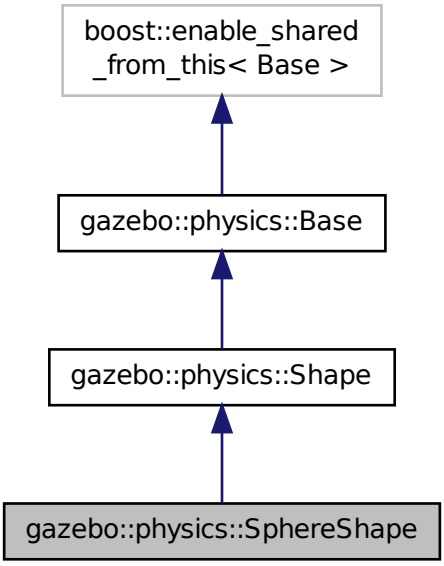
- **SonarVisual.hh**

10.161 gazebo::physics::SphereShape Class Reference

Sphere collision shape.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::SphereShape:



Public Member Functions

- **SphereShape** (**CollisionPtr** _parent)
  Constructor.
- virtual ~**SphereShape** ()
  Destructor.
- virtual void **FillMsg** (msgs::Geometry &_msg)
  Fill in the values for a geomertry message.
- double **GetRadius** () const
  Get the sphere's radius.
- virtual void **Init** ()
  Initialize the sphere.
- virtual void **ProcessMsg** (const msgs::Geometry &_msg)
  Process a geometry message.
- virtual void **SetRadius** (double _radius)
  Set the size.

Additional Inherited Members

10.161.1 Detailed Description

Sphere collision shape.

10.161.2 Constructor & Destructor Documentation

10.161.2.1 gazebo::physics::SphereShape::SphereShape (CollisionPtr _parent) [explicit]

Constructor.

Parameters

in	*_parent*	Parent collision object.

10.161.2.2 virtual gazebo::physics::SphereShape::~SphereShape ( ) [virtual]

Destructor.

10.161.3 Member Function Documentation

10.161.3.1 virtual void gazebo::physics::SphereShape::FillMsg (msgs::Geometry & _msg) [virtual]

Fill in the values for a geomertry message.

Parameters

out	*_msg*	The geometry message to fill.

Implements **gazebo::physics::Shape** (p. 756).

10.161.3.2 double gazebo::physics::SphereShape::GetRadius ( ) const

Get the sphere's radius.

Returns

Radius of the sphere.

10.161.3.3 virtual void gazebo::physics::SphereShape::Init ( ) [virtual]

Initialize the sphere.

Implements **gazebo::physics::Shape** (p. 756).

10.161.3.4 virtual void gazebo::physics::SphereShape::ProcessMsg (const msgs::Geometry & _msg) [virtual]

Process a geometry message.

Parameters

in	*_msg*	The message to set values from.

Implements **gazebo::physics::Shape** (p. 756).

10.161.3.5 virtual void gazebo::physics::SphereShape::SetRadius (double *_radius*) [virtual]

Set the size.

Parameters

in	*_radius*	Radius of the sphere.

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

- **SphereShape.hh**

10.162 gazebo::math::Spline Class Reference

Splines.

`#include <math/gzmath.hh>`

Public Member Functions

- **Spline** ()

  constructor
- ~**Spline** ()

  destructor
- void **AddPoint** (const **Vector3** &_pt)

  Adds a control point to the end of the spline.
- void **Clear** ()

  Clears all the points in the spline.
- **Vector3 GetPoint** (unsigned int _index) const

  Gets the detail of one of the control points of the spline.
- unsigned int **GetPointCount** () const

  Gets the number of control points in the spline.
- **Vector3 GetTangent** (unsigned int _index) const

  Get the tangent value for a point.
- double **GetTension** () const

  Get the tension value.
- **Vector3 Interpolate** (double _t) const

  Returns an interpolated point based on a parametric value over the whole series.
- **Vector3 Interpolate** (unsigned int _fromIndex, double _t) const

  Interpolates a single segment of the spline given a parametric value.
- void **RecalcTangents** ()

  Recalculates the tangents associated with this spline.
- void **SetAutoCalculate** (bool _autoCalc)

  Tells the spline whether it should automatically calculate tangents on demand as points are added.
- void **SetTension** (double _t)

  Set the tension parameter.
- void **UpdatePoint** (unsigned int _index, const **Vector3** &_value)

  Updates a single point in the spline.

Protected Attributes

- bool **autoCalc**

  when true, the tangents are recalculated when the control point change
- **Matrix4 coeffs**

  Matrix of coefficients.
- std::vector< **Vector3** > **points**

  control points
- std::vector< **Vector3** > **tangents**

  tangents
- double **tension**

  Tension of 0 = Catmull-Rom spline, otherwise a Cardinal spline.

10.162.1 Detailed Description

Splines.

10.162.2 Constructor & Destructor Documentation

10.162.2.1 gazebo::math::Spline::Spline ( )

constructor

10.162.2.2 gazebo::math::Spline::~Spline ( )

destructor

10.162.3 Member Function Documentation

10.162.3.1 void gazebo::math::Spline::AddPoint (const Vector3 & *_pt*)

Adds a control point to the end of the spline.

Parameters

in	*_pt*	point to add

10.162.3.2 void gazebo::math::Spline::Clear ( )

Clears all the points in the spline.

10.162.3.3 Vector3 gazebo::math::Spline::GetPoint (unsigned int *_index*) const

Gets the detail of one of the control points of the spline.

Parameters

in	*_index*	the control point index

Returns

the control point, or [0,0,0] and a message on the error stream

10.162.3.4 unsigned int gazebo::math::Spline::GetPointCount ( ) const

Gets the number of control points in the spline.

Returns

the count

10.162.3.5 Vector3 gazebo::math::Spline::GetTangent (unsigned int *_index*) const

Get the tangent value for a point.

Parameters

in	*_index*	the control point index

10.162.3.6 double gazebo::math::Spline::GetTension ( ) const

Get the tension value.

Returns

The value of the tension, which is between 0.0 and 1.0

10.162.3.7 Vector3 gazebo::math::Spline::Interpolate (double *_t*) const

Returns an interpolated point based on a parametric value over the whole series.

Parameters

in	*_t*	parameter (range 0 to 1)

10.162.3.8 Vector3 gazebo::math::Spline::Interpolate (unsigned int *_fromIndex,* double *_t*) const

Interpolates a single segment of the spline given a parametric value.

Parameters

in	*_fromIndex*	The point index to treat as t = 0. fromIndex + 1 is deemed to be t = 1
in	*_t*	Parametric value

10.162.3.9 void gazebo::math::Spline::RecalcTangents ( )

Recalculates the tangents associated with this spline.

Remarks

If you tell the spline not to update on demand by calling setAutoCalculate(false) then you must call this after completing your updates to the spline points.

10.162.3.10 void gazebo::math::Spline::SetAutoCalculate (bool *_autoCalc*)

Tells the spline whether it should automatically calculate tangents on demand as points are added.

Remarks

The spline calculates tangents at each point automatically based on the input points. Normally it does this every time a point changes. However, if you have a lot of points to add in one go, you probably don't want to incur this overhead and would prefer to defer the calculation until you are finished setting all the points. You can do this by calling this method with a parameter of 'false'. Just remember to manually call the recalcTangents method when you are done.

Parameters

in	*_autoCalc*	If true, tangents are calculated for you whenever a point changes. If false, you must call reclacTangents to recalculate them when it best suits.

10.162.3.11 void gazebo::math::Spline::SetTension (double *_t*)

Set the tension parameter.

A value of 0 = Catmull-Rom spline.

Parameters

in	*_t*	Tension value between 0.0 and 1.0

10.162.3.12 void gazebo::math::Spline::UpdatePoint (unsigned int *_index*, const Vector3 & *_value*)

Updates a single point in the spline.

Remarks

an error to the error stream is printed when the index is out of bounds

Parameters

in	*_index*	the control point index
in	*_value*	the new position

10.162.4 Member Data Documentation

10.162.4.1 bool gazebo::math::Spline::autoCalc [protected]

when true, the tangents are recalculated when the control point change

10.162.4.2 Matrix4 gazebo::math::Spline::coeffs [protected]

Matrix of coefficients.

10.162.4.3 std::vector<Vector3> gazebo::math::Spline::points [protected]

control points

10.162.4.4 std::vector<Vector3> gazebo::math::Spline::tangents [protected]

tangents

10.162.4.5 double gazebo::math::Spline::tension [protected]

Tension of 0 = Catmull-Rom spline, otherwise a Cardinal spline.

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

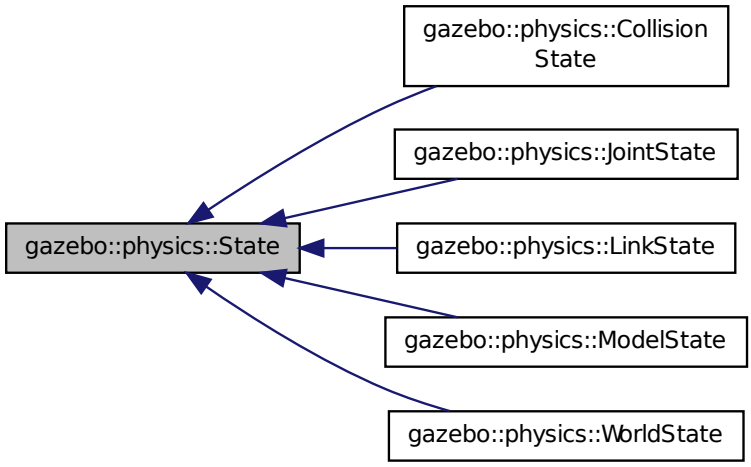
- **Spline.hh**

10.163 gazebo::physics::State Class Reference

State (p. 797) of an entity.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::State:



Public Member Functions

- **State** ()

  Default constructor.
- **State** (const std::string &_name, const **common::Time** &_realTime, const **common::Time** &_simTime)

  Constructor.
- virtual ~**State** ()

  Destructor.
- std::string **GetName** () const

  *Get the name associated with this **State** (p. 797).*
- **common::Time GetRealTime** () const

  Get the real time when this state was generated.
- **common::Time GetSimTime** () const

  Get the sim time when this state was generated.
- **common::Time GetWallTime** () const

  Get the wall time when this state was generated.
- virtual void **Load** (const **sdf::ElementPtr** _elem)

  Load state from SDF element.
- **State operator-** (const **State** &_state) const

  Subtraction operator.
- **State** & **operator=** (const **State** &_state)

  Assignment operator.
- void **SetName** (const std::string &_name)

  *Set the name associated with this **State** (p. 797).*
- virtual void **SetRealTime** (const **common::Time** &_time)

  Set the real time when this state was generated.

- virtual void **SetSimTime** (const **common::Time** &_time)

  Set the sim time when this state was generated.
- virtual void **SetWallTime** (const **common::Time** &_time)

  Set the wall time when this state was generated.

Protected Attributes

- std::string **name**

  *Name associated with this **State** (p. 797).*
- **common::Time realTime**
- **common::Time simTime**
- **common::Time wallTime**

  Times for the state data.

10.163.1 Detailed Description

State (p. 797) of an entity.

This is the base class for all **State** (p. 797) information.

10.163.2 Constructor & Destructor Documentation

10.163.2.1 gazebo::physics::State::State ( )

Default constructor.

10.163.2.2 gazebo::physics::State::State (const std::string & *_name,* const **common::Time** & *_realTime,* const **common::Time** & *_simTime*)

Constructor.

Construct a **State** (p. 797) object using some basic information.

Parameters

_name	Name associated with the **State** (p. 797) information. This is typically the name of an **Entity** (p. 288). _realTime Clock time since simulation started.
_simTime	Simulation time associated with this **State** (p. 797) info.

10.163.2.3 virtual gazebo::physics::State::~State ( ) [virtual]

Destructor.

10.163.3 Member Function Documentation

10.163.3.1 std::string gazebo::physics::State::GetName ( ) const

Get the name associated with this **State** (p. 797).

Returns

Name associated with this state information. Typically a name of an **Entity** (p. 288).

10.163.3.2 common::Time gazebo::physics::State::GetRealTime ( ) const

Get the real time when this state was generated.

Returns

Clock time since simulation was stated.

10.163.3.3 common::Time gazebo::physics::State::GetSimTime ( ) const

Get the sim time when this state was generated.

Returns

Simulation time when the data was recorded.

10.163.3.4 common::Time gazebo::physics::State::GetWallTime ( ) const

Get the wall time when this state was generated.

Returns

The absolute clock time when the **State** (p. 797) data was recorded.

10.163.3.5 virtual void gazebo::physics::State::Load (const sdf::ElementPtr *_elem*) [virtual]

Load state from SDF element.

Populates the **State** (p. 797) information from data stored in an SDF::Element

Parameters

_elem	Pointer to the SDF::Element

Reimplemented in **gazebo::physics::ModelState** (p. 540), **gazebo::physics::LinkState** (p. 464), **gazebo::physics::WorldState** (p. 962), **gazebo::physics::JointState** (p. 424), and **gazebo::physics::CollisionState** (p. 212).

10.163.3.6 State gazebo::physics::State::operator- (const State & *_state*) const

Subtraction operator.

Parameters

in	*_pt*	A state to substract.

Returns

The resulting state.

10.163.3.7 State& gazebo::physics::State::operator= (const State & *_state*)

Assignment operator.

Parameters

in	*_state*	**State** (p. 797) value

Returns

this

10.163.3.8 void gazebo::physics::State::SetName (const std::string & *_name*)

Set the name associated with this **State** (p. 797).

Parameters

in	*_name*	Name associated with this state information. Typically the name of an **Entity** (p. 288).

10.163.3.9 virtual void gazebo::physics::State::SetRealTime (const common::Time & *_time*) [virtual]

Set the real time when this state was generated.

Parameters

in	*_time*	Clock time since simulation was stated.

Reimplemented in **gazebo::physics::ModelState** (p. 540), **gazebo::physics::LinkState** (p. 465), and **gazebo::physics::WorldState** (p. 963).

10.163.3.10 virtual void gazebo::physics::State::SetSimTime (const common::Time & *_time*) [virtual]

Set the sim time when this state was generated.

Parameters

in	*_time*	Simulation time when the data was recorded.

Reimplemented in **gazebo::physics::ModelState** (p. 541), **gazebo::physics::LinkState** (p. 465), and **gazebo::physics::WorldState** (p. 963).

10.163.3.11 virtual void gazebo::physics::State::SetWallTime (const common::Time & *_time*) [virtual]

Set the wall time when this state was generated.

Parameters

in	*_time*	The absolute clock time when the **State** (p. 797) data was recorded.

Reimplemented in **gazebo::physics::ModelState** (p. 541), **gazebo::physics::LinkState** (p. 466), and **gazebo::physics::WorldState** (p. 964).

10.163.4 Member Data Documentation

10.163.4.1 std::string gazebo::physics::State::name [protected]

Name associated with this **State** (p. 797).

10.163.4.2 common::Time gazebo::physics::State::realTime [protected]

10.163.4.3 common::Time gazebo::physics::State::simTime [protected]

10.163.4.4 common::Time gazebo::physics::State::wallTime [protected]

Times for the state data.

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

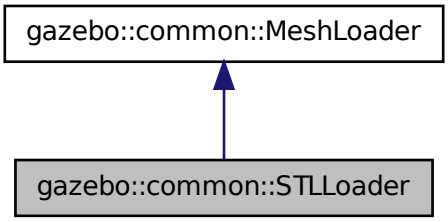
- **State.hh**

10.164 gazebo::common::STLLoader Class Reference

Class used to load STL mesh files.

`#include <common/common.hh>`

Inheritance diagram for gazebo::common::STLLoader:



Public Member Functions

- **STLLoader** ()

Constructor.

- virtual ∼**STLLoader** ()

  Destructor.

- virtual **Mesh** ∗ **Load** (const std::string &_filename)

  Creates a new mesh and loads the data from a file.

10.164.1 Detailed Description

Class used to load STL mesh files.

10.164.2 Constructor & Destructor Documentation

10.164.2.1 gazebo::common::STLLoader::STLLoader ( )

Constructor.

10.164.2.2 virtual gazebo::common::STLLoader::∼STLLoader ( ) [virtual]

Destructor.

10.164.3 Member Function Documentation

10.164.3.1 virtual Mesh∗ gazebo::common::STLLoader::Load (const std::string & *_filename*) [virtual]

Creates a new mesh and loads the data from a file.

Parameters

in	*_filename*	the mesh file

Implements **gazebo::common::MeshLoader** (p. 507).

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

- **STLLoader.hh**

10.165 gazebo::common::SubMesh Class Reference

A child mesh.

`#include <Mesh.hh>`

Public Types

- enum **PrimitiveType** {
  POINTS, **LINES**, **LINESTRIPS**, **TRIANGLES**,
  TRIFANS, **TRISTRIPS** }

  An enumeration of the geometric mesh primitives.

Public Member Functions

- **SubMesh** ()

  Constructor.
- **SubMesh** (const **SubMesh** *_mesh)

  Copy Constructor.
- virtual ~**SubMesh** ()

  Destructor.
- void **AddIndex** (unsigned int _i)

  Add an index to the mesh.
- void **AddNodeAssignment** (unsigned int _vertex, unsigned int _node, float _weight)

  Add a vertex - skeleton node assignment.
- void **AddNormal** (const **math::Vector3** &_n)

  Add a normal to the mesh.
- void **AddNormal** (double _x, double _y, double _z)

  Add a normal to the mesh.
- void **AddTexCoord** (double _u, double _v)

  Add a texture coord to the mesh.
- void **AddVertex** (const **math::Vector3** &_v)

  Add a vertex to the mesh.
- void **AddVertex** (double _x, double _y, double _z)

  Add a vertex to the mesh.
- void **Center** (const **math::Vector3** &_center=**math::Vector3::Zero**)

  Move the center of the submesh to the given coordinate.
- void **CopyNormals** (const std::vector< **math::Vector3** > &_norms)

  Copy normals from a vector.
- void **CopyVertices** (const std::vector< **math::Vector3** > &_verts)

  Copy vertices from a vector.
- void **FillArrays** (float **_vertArr, int **_indArr) const

  Put all the data into flat arrays.
- void **GenSphericalTexCoord** (const **math::Vector3** &_center)

  Generate texture coordinates using spherical projection from center.
- unsigned int **GetIndex** (unsigned int _i) const

  Get an index.
- unsigned int **GetIndexCount** () const

  Return the number of indicies.
- unsigned int **GetMaterialIndex** () const

  Get the material index.
- **math::Vector3 GetMax** () const

  Get the maximun X, Y, Z values.
- unsigned int **GetMaxIndex** () const

  Get the highest index value.
- **math::Vector3 GetMin** () const

  Get the minimum X, Y, Z values.
- std::string **GetName** () const

  Get the name of this mesh.
- **NodeAssignment GetNodeAssignment** (unsigned int _i) const

Get a vertex - skeleton node assignment.

- unsigned int **GetNodeAssignmentsCount** () const

  Return the number of vertex - skeleton node assignments.
- **math::Vector3 GetNormal** (unsigned int _i) const

  Get a normal.
- unsigned int **GetNormalCount** () const

  Return the number of normals.
- **PrimitiveType GetPrimitiveType** () const

  Get the primitive type.
- **math::Vector2d GetTexCoord** (unsigned int _i) const

  Get a tex coord.
- unsigned int **GetTexCoordCount** () const

  Return the number of texture coordinates.
- **math::Vector3 GetVertex** (unsigned int _i) const

  Get a vertex.
- unsigned int **GetVertexCount** () const

  Return the number of vertices.
- unsigned int **GetVertexIndex** (const **math::Vector3** &_v) const

  Get the index of the vertex.
- bool **HasVertex** (const **math::Vector3** &_v) const

  Return true if this submesh has the vertex.
- void **RecalculateNormals** ()

  Recalculate all the normals.
- void **Scale** (double _factor)

  Scale all vertices by _factor.
- void **SetIndexCount** (unsigned int _count)

  Resize the index array.
- void **SetMaterialIndex** (unsigned int _index)

  Set the material index.
- void **SetName** (const std::string &_n)

  Set the name of this mesh.
- void **SetNormal** (unsigned int _i, const **math::Vector3** &_n)

  Set a normal.
- void **SetNormalCount** (unsigned int _count)

  Resize the normal array.
- void **SetPrimitiveType** (**PrimitiveType** _type)

  Set the primitive type.
- void **SetScale** (const **math::Vector3** &_factor)

  Scale all vertices by the _factor vector.
- void **SetSubMeshCenter** (**math::Vector3** _center)

  Reset mesh center to geometric center.
- void **SetTexCoord** (unsigned int _i, const **math::Vector2d** &_t)

  Set a tex coord.
- void **SetTexCoordCount** (unsigned int _count)

  Resize the texture coordinate array.
- void **SetVertex** (unsigned int _i, const **math::Vector3** &_v)

  Set a vertex.

- void **SetVertexCount** (unsigned int _count)

  Resize the vertex array.
- void **Translate** (const **math::Vector3** &_vec)

  Move all vertices by _vec.

10.165.1 Detailed Description

A child mesh.

10.165.2 Member Enumeration Documentation

10.165.2.1 enum gazebo::common::SubMesh::PrimitiveType

An enumeration of the geometric mesh primitives.

Enumerator:

POINTS

LINES

LINESTRIPS

TRIANGLES

TRIFANS

TRISTRIPS

10.165.3 Constructor & Destructor Documentation

10.165.3.1 gazebo::common::SubMesh::SubMesh ( )

Constructor.

10.165.3.2 gazebo::common::SubMesh::SubMesh (const SubMesh * *_mesh*)

Copy Constructor.

10.165.3.3 virtual gazebo::common::SubMesh::~SubMesh ( ) [virtual]

Destructor.

10.165.4 Member Function Documentation

10.165.4.1 void gazebo::common::SubMesh::AddIndex (unsigned int *_i*)

Add an index to the mesh.

Parameters

in	*_i*	the new vertex index

10.165.4.2 void gazebo::common::SubMesh::AddNodeAssignment (unsigned int _vertex, unsigned int _node, float _weight)

Add a vertex - skeleton node assignment.

Parameters

in	*_vertex*	the vertex index
in	*_node*	the node index
in	*_weight*	the weight (between 0 and 1)

10.165.4.3 void gazebo::common::SubMesh::AddNormal (const math::Vector3 & _n)

Add a normal to the mesh.

Parameters

in	*_n*	the normal

10.165.4.4 void gazebo::common::SubMesh::AddNormal (double _x, double _y, double _z)

Add a normal to the mesh.

Parameters

in	*_x*	position along x
in	*_y*	position along y
in	*_z*	position along z

10.165.4.5 void gazebo::common::SubMesh::AddTexCoord (double _u, double _v)

Add a texture coord to the mesh.

Parameters

in	*_u*	position along u
in	*_v*	position along v

10.165.4.6 void gazebo::common::SubMesh::AddVertex (const math::Vector3 & _v)

Add a vertex to the mesh.

Parameters

in	*_v*	the new position

10.165.4.7 void gazebo::common::SubMesh::AddVertex (double _x, double _y, double _z)

Add a vertex to the mesh.

Parameters

in	*_x*	position along x
in	*_y*	position along y
in	*_z*	position along z

10.165.4.8 void gazebo::common::SubMesh::Center (const math::Vector3 & *_center* = math::Vector3::Zero)

Move the center of the submesh to the given coordinate.

This will move all the vertices.

Parameters

in	*_center*	Location of the mesh center.

10.165.4.9 void gazebo::common::SubMesh::CopyNormals (const std::vector< math::Vector3 > & *_norms*)

Copy normals from a vector.

Parameters

in	*_norms*	to copy from

10.165.4.10 void gazebo::common::SubMesh::CopyVertices (const std::vector< math::Vector3 > & *_verts*)

Copy vertices from a vector.

Parameters

in	*_verts*	the vertices to copy from

10.165.4.11 void gazebo::common::SubMesh::FillArrays (float ** *_vertArr*, int ** *_indArr*) const

Put all the data into flat arrays.

Parameters

in	*_verArr*	
in	*_indArr*	

10.165.4.12 void gazebo::common::SubMesh::GenSphericalTexCoord (const math::Vector3 & *_center*)

Generate texture coordinates using spherical projection from center.

Parameters

in	*_center*	

10.165.4.13 unsigned int gazebo::common::SubMesh::GetIndex (unsigned int _i) const

Get an index.

Parameters

in	_i	

10.165.4.14 unsigned int gazebo::common::SubMesh::GetIndexCount ( ) const

Return the number of indicies.

10.165.4.15 unsigned int gazebo::common::SubMesh::GetMaterialIndex ( ) const

Get the material index.

10.165.4.16 math::Vector3 gazebo::common::SubMesh::GetMax ( ) const

Get the maximun X, Y, Z values.

Returns

10.165.4.17 unsigned int gazebo::common::SubMesh::GetMaxIndex ( ) const

Get the highest index value.

10.165.4.18 math::Vector3 gazebo::common::SubMesh::GetMin ( ) const

Get the minimum X, Y, Z values.

Returns

10.165.4.19 std::string gazebo::common::SubMesh::GetName ( ) const

Get the name of this mesh.

Returns

the name

10.165.4.20 NodeAssignment gazebo::common::SubMesh::GetNodeAssignment (unsigned int _i) const

Get a vertex - skeleton node assignment.

Parameters

in	_i	the index of the assignment

10.165.4.21 unsigned int gazebo::common::SubMesh::GetNodeAssignmentsCount ( ) const

Return the number of vertex - skeleton node assignments.

10.165.4.22 math::Vector3 gazebo::common::SubMesh::GetNormal (unsigned int _i) const

Get a normal.

Parameters

in	_i	the normal index

Returns

the orientation of the normal, or throws an exception

10.165.4.23 unsigned int gazebo::common::SubMesh::GetNormalCount ( ) const

Return the number of normals.

10.165.4.24 PrimitiveType gazebo::common::SubMesh::GetPrimitiveType ( ) const

Get the primitive type.

Returns

the primitive type

10.165.4.25 math::Vector2d gazebo::common::SubMesh::GetTexCoord (unsigned int _i) const

Get a tex coord.

Parameters

in	_i	the texture index

Returns

the texture coordinates

10.165.4.26 unsigned int gazebo::common::SubMesh::GetTexCoordCount ( ) const

Return the number of texture coordinates.

10.165.4.27 math::Vector3 gazebo::common::SubMesh::GetVertex (unsigned int _i) const

Get a vertex.

Parameters

in	*_i*	the vertex index

Returns

the position or throws an exception

10.165.4.28 unsigned int gazebo::common::SubMesh::GetVertexCount ( ) const

Return the number of vertices.

10.165.4.29 unsigned int gazebo::common::SubMesh::GetVertexIndex (const math::Vector3 & _v) const

Get the index of the vertex.

Parameters

in	*_v*	

10.165.4.30 bool gazebo::common::SubMesh::HasVertex (const math::Vector3 & _v) const

Return true if this submesh has the vertex.

Parameters

in	*_v*	

10.165.4.31 void gazebo::common::SubMesh::RecalculateNormals ( )

Recalculate all the normals.

10.165.4.32 void gazebo::common::SubMesh::Scale (double _factor)

Scale all vertices by _factor.

Parameters

in	*_factor*	Scaling factor

10.165.4.33 void gazebo::common::SubMesh::SetIndexCount (unsigned int _count)

Resize the index array.

Parameters

in	*_count*	the new size of the array

10.165.4.34 void gazebo::common::SubMesh::SetMaterialIndex (unsigned int _index)

Set the material index.

Relates to the parent mesh material list

Parameters

in	*_index*	

10.165.4.35 void gazebo::common::SubMesh::SetName (const std::string & _n)

Set the name of this mesh.

Parameters

in	*_n*	the name to set

10.165.4.36 void gazebo::common::SubMesh::SetNormal (unsigned int _i, const math::Vector3 & _n)

Set a normal.

Parameters

in	*_i*	the normal index
in	*_n*	the normal direction

10.165.4.37 void gazebo::common::SubMesh::SetNormalCount (unsigned int _count)

Resize the normal array.

Parameters

in	*_count*	the new size of the array

10.165.4.38 void gazebo::common::SubMesh::SetPrimitiveType (PrimitiveType _type)

Set the primitive type.

Parameters

in	*_type*	the type

10.165.4.39 void gazebo::common::SubMesh::SetScale (const math::Vector3 & *_factor*)

Scale all vertices by the _factor vector.

Parameters

in	*_factor*	Scaling vector

10.165.4.40 void gazebo::common::SubMesh::SetSubMeshCenter (math::Vector3 *_center*)

Reset mesh center to geometric center.

Parameters

in	*_center*	

10.165.4.41 void gazebo::common::SubMesh::SetTexCoord (unsigned int *_i,* const math::Vector2d & *_t*)

Set a tex coord.

Parameters

in	*_i*	
in	*_t*	

10.165.4.42 void gazebo::common::SubMesh::SetTexCoordCount (unsigned int *_count*)

Resize the texture coordinate array.

Parameters

in	*_count*	

10.165.4.43 void gazebo::common::SubMesh::SetVertex (unsigned int *_i,* const math::Vector3 & *_v*)

Set a vertex.

Parameters

in	*_i*	the index
in	*_v*	the position

10.165.4.44 void gazebo::common::SubMesh::SetVertexCount (unsigned int _count)

Resize the vertex array.

Parameters

in	*_count*	the new size of the array

10.165.4.45 void gazebo::common::SubMesh::Translate (const math::Vector3 & _vec)

Move all vertices by _vec.

Parameters

in	*_vec*	Amount to translate vertices.

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

- **Mesh.hh**

10.166 gazebo::transport::SubscribeOptions Class Reference

Options for a subscription.

`#include <transport/transport.hh>`

Public Member Functions

- **SubscribeOptions** ()
  Constructor.
- bool **GetLatching** () const
  Are we latching?
- std::string **GetMsgType** () const
  Get the type of the topic we're subscribed to.
- **NodePtr GetNode** () const
  Get the node we're subscribed to.
- std::string **GetTopic** () const
  Get the topic we're subscribed to.
- template<class M >
  void **Init** (const std::string &_topic, **NodePtr** _node, bool _latching)
  Initialize the options.
- void **Init** (const std::string &_topic, **NodePtr** _node, bool _latching)
  Initialize the options.

10.166.1 Detailed Description

Options for a subscription.

10.166.2 Constructor & Destructor Documentation

10.166.2.1 gazebo::transport::SubscribeOptions::SubscribeOptions ( ) [inline]

Constructor.

10.166.3 Member Function Documentation

10.166.3.1 bool gazebo::transport::SubscribeOptions::GetLatching ( ) const [inline]

Are we latching?

Returns

true if we're latching the latest message, false otherwise

10.166.3.2 std::string gazebo::transport::SubscribeOptions::GetMsgType ( ) const [inline]

Get the type of the topic we're subscribed to.

Returns

The type of the topic we're subscribed to

10.166.3.3 NodePtr gazebo::transport::SubscribeOptions::GetNode ( ) const [inline]

Get the node we're subscribed to.

Returns

The associated node

10.166.3.4 std::string gazebo::transport::SubscribeOptions::GetTopic ( ) const [inline]

Get the topic we're subscribed to.

Returns

The topic we're subscribed to

10.166.3.5 template<class M > void gazebo::transport::SubscribeOptions::Init (const std::string & *_topic,* NodePtr *_node,* bool *_latching*) [inline]

Initialize the options.

Parameters

in	*_topic*	Topic we're subscribing to
in,out	*_node*	The associated node
in	*_latching*	If true, latch the latest message; if false, don't latch

References gzthrow, and NULL.

Referenced by gazebo::transport::Node::Subscribe().

10.166.3.6 void gazebo::transport::SubscribeOptions::Init (const std::string & *_topic,* NodePtr *_node,* bool *_latching*) [inline]

Initialize the options.

This version of init is only used when creating subscribers of raw data.

Parameters

in	*_topic*	Topic we're subscribing to
in,out	*_node*	The associated node
in	*_latching*	If true, latch the latest message; if false, don't latch

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

- **SubscribeOptions.hh**

10.167 gazebo::transport::Subscriber Class Reference

A subscriber to a topic.

`#include <transport/transport.hh>`

Public Member Functions

- **Subscriber** (const std::string &_topic, **NodePtr** _node)
  Constructor.
- virtual ~**Subscriber** ()
  Destructor.
- unsigned int **GetCallbackId** () const
- std::string **GetTopic** () const
  Get the topic name.
- void **SetCallbackId** (unsigned int _id)
- void **Unsubscribe** () const
  Unsubscribe from the topic.

10.167.1 Detailed Description

A subscriber to a topic.

10.167.2 Constructor & Destructor Documentation

10.167.2.1 gazebo::transport::Subscriber::Subscriber (const std::string & *_topic,* NodePtr *_node*)

Constructor.

Parameters

in	*_topic*	The topic we're subscribing to
in	*_node*	The associated node

10.167.2.2 virtual gazebo::transport::Subscriber::~Subscriber ( ) [virtual]

Destructor.

10.167.3 Member Function Documentation

10.167.3.1 unsigned int gazebo::transport::Subscriber::GetCallbackId ( ) const

10.167.3.2 std::string gazebo::transport::Subscriber::GetTopic ( ) const

Get the topic name.

Returns

The topic name

10.167.3.3 void gazebo::transport::Subscriber::SetCallbackId (unsigned int *_id*)

10.167.3.4 void gazebo::transport::Subscriber::Unsubscribe ( ) const

Unsubscribe from the topic.

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

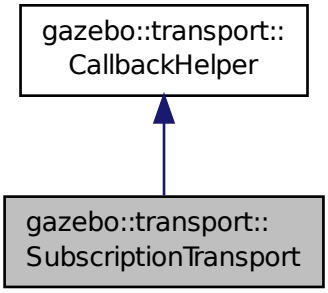
- **Subscriber.hh**

10.168 gazebo::transport::SubscriptionTransport Class Reference

transport/transport.hh

```
#include <SubscriptionTransport.hh>
```

Inheritance diagram for gazebo::transport::SubscriptionTransport:



Public Member Functions

- **SubscriptionTransport** ()

  Constructor.
- virtual ∼**SubscriptionTransport** ()

  Destructor.
- const **ConnectionPtr** & **GetConnection** () const

  Get the connection we're using.
- virtual bool **HandleData** (const std::string &_newdata, boost::function< void(uint32_t)> _cb, uint32_t _id)

  Output a message to a connection.
- virtual bool **HandleMessage** (**MessagePtr** _newMsg)

  Process new incoming message.
- void **Init** (**ConnectionPtr** _conn, bool _latching)

  Initialize the publication link.
- virtual bool **IsLocal** () const

  Is the callback local?

Additional Inherited Members

10.168.1 Detailed Description

transport/transport.hh

Handles sending data over the wire to remote subscribers

10.168.2 Constructor & Destructor Documentation

10.168.2.1 gazebo::transport::SubscriptionTransport::SubscriptionTransport ( )

Constructor.

10.168.2.2 virtual gazebo::transport::SubscriptionTransport::~SubscriptionTransport ( ) [virtual]

Destructor.

10.168.3 Member Function Documentation

10.168.3.1 const ConnectionPtr& gazebo::transport::SubscriptionTransport::GetConnection ( ) const

Get the connection we're using.

Returns

Pointer to the connection we're using

10.168.3.2 virtual bool gazebo::transport::SubscriptionTransport::HandleData (const std::string & *_newdata,* boost::function< void(uint32_t)> *_cb,* uint32_t *_id*) [virtual]

Output a message to a connection.

Parameters

in	*_newdata*	The message to be handled

Returns

true if the message was handled successfully, false otherwise

Parameters

in	*_cb*	If non-null, callback to be invoked after transmission is complete.
in	*_id*	ID associated with the message data.

Implements **gazebo::transport::CallbackHelper** (p. 163).

10.168.3.3 virtual bool gazebo::transport::SubscriptionTransport::HandleMessage (MessagePtr *_newMsg*) [virtual]

Process new incoming message.

Parameters

in	*_newMsg*	Incoming message to be processed

Returns

true if successfully processed; false otherwise

Implements **gazebo::transport::CallbackHelper** (p. 163).

10.168.3.4 void gazebo::transport::SubscriptionTransport::Init (ConnectionPtr *_conn,* bool *_latching*)

Initialize the publication link.

Parameters

in	*_conn*	The connection to use
in	*_latching*	If true, latch the latest message; if false, don't latch

10.168.3.5 virtual bool gazebo::transport::SubscriptionTransport::IsLocal ( ) const [virtual]

Is the callback local?

Returns

true if the callback is local, false if the callback is tied to a remote connection

Implements **gazebo::transport::CallbackHelper** (p. 164).

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

- **SubscriptionTransport.hh**

10.169 gazebo::physics::SurfaceParams Class Reference

SurfaceParams (p. 820) defines various Surface contact parameters.

`#include <physics/physics.hh>`

Public Member Functions

- **SurfaceParams** ()
  Constructor.
- virtual ~**SurfaceParams** ()
  Destructor.
- void **FillMsg** (msgs::Surface &_msg)
  Fill in a surface message.
- virtual void **Load** (**sdf::ElementPtr** _sdf)
  Load the contact params.
- virtual void **ProcessMsg** (const msgs::Surface &_msg)

Public Attributes

- double **bounce**
  bounce restitution coefficient [0,1], with 0 being inelastic, and 1 being perfectly elastic.
- double **bounceThreshold**
  minimum contact velocity for bounce to take effect, otherwise the collision is treated as an inelastic collision.
- double **cfm**
  Constraint Force Mixing parameter.
- bool **collideWithoutContact**
  Allow collision checking without generating a contact joint.
- unsigned int **collideWithoutContactBitmask**
  Custom collision filtering used when collideWithoutContact is true.

- double **erp**

  Error Reduction Parameter.
- **math::Vector3 fdir1**

  *Primary friction direction for dry friction coefficient (**SurfaceParams::mu1** (p. 824)) of the friction pyramid.*
- double **kd**

  *spring damping constant equivalents of a contact as a function of **SurfaceParams::cfm** (p. 822) and **SurfaceParams::erp** (p. 822).*
- double **kp**

  *spring constant equivalents of a contact as a function of **SurfaceParams::cfm** (p. 822) and **SurfaceParams::erp** (p. 822).*
- double **maxVel**

  Maximum interpenetration error correction velocity.
- double **minDepth**

  Minimum depth before ERP takes effect.
- double **mu1**

  Dry friction coefficient in the primary friction direction as defined by the friction pyramid.
- double **mu2**

  Dry friction coefficient in the second friction direction as defined by the friction pyramid.
- double **slip1**

  Artificial contact slip in the primary friction direction.
- double **slip2**

  Artificial contact slip in the secondary friction dirction.

10.169.1 Detailed Description

SurfaceParams (p. 820) defines various Surface contact parameters.

These parameters defines the properties of a **physics::Contact** (p. 241) constraint.

10.169.2 Constructor & Destructor Documentation

10.169.2.1 gazebo::physics::SurfaceParams::SurfaceParams ( )

Constructor.

10.169.2.2 virtual gazebo::physics::SurfaceParams::~SurfaceParams ( ) [virtual]

Destructor.

10.169.3 Member Function Documentation

10.169.3.1 void gazebo::physics::SurfaceParams::FillMsg (msgs::Surface & *_msg*)

Fill in a surface message.

Parameters

in	*_msg*	Message to fill with this object's values.

10.169.3.2 virtual void gazebo::physics::SurfaceParams::Load (sdf::ElementPtr _sdf) [virtual]

Load the contact params.

Parameters

in	_sdf	SDF values to load from.

10.169.3.3 virtual void gazebo::physics::SurfaceParams::ProcessMsg (const msgs::Surface & _msg) [virtual]

10.169.4 Member Data Documentation

10.169.4.1 double gazebo::physics::SurfaceParams::bounce

bounce restitution coefficient [0,1], with 0 being inelastic, and 1 being perfectly elastic.

See Also

http://www.ode.org/ode-latest-userguide.html#sec_7_3_7

10.169.4.2 double gazebo::physics::SurfaceParams::bounceThreshold

minimum contact velocity for bounce to take effect, otherwise the collision is treated as an inelastic collision.

See Also

http://www.ode.org/ode-latest-userguide.html#sec_7_3_7

10.169.4.3 double gazebo::physics::SurfaceParams::cfm

Constraint Force Mixing parameter.

See for example http://www.ode.org/ode-latest-userguide.html#sec_3_8_0 for more details.

10.169.4.4 bool gazebo::physics::SurfaceParams::collideWithoutContact

Allow collision checking without generating a contact joint.

10.169.4.5 unsigned int gazebo::physics::SurfaceParams::collideWithoutContactBitmask

Custom collision filtering used when collideWithoutContact is true.

10.169.4.6 double gazebo::physics::SurfaceParams::erp

Error Reduction Parameter.

See Also

See for example http://www.ode.org/ode-latest-userguide.html#sec_3_8_0 for more details.

10.169.4.7 math::Vector3 gazebo::physics::SurfaceParams::fdir1

Primary friction direction for dry friction coefficient (**SurfaceParams::mu1** (p. 824)) of the friction pyramid.

```
If undefined, a vector consstrained to be perpendicular
to the contact normal in the global y-z plane is used.
```

See Also

http://www.ode.org/ode-latest-userguide.html#sec_7_3_7

10.169.4.8 double gazebo::physics::SurfaceParams::kd

spring damping constant equivalents of a contact as a function of **SurfaceParams::cfm** (p. 822) and **SurfaceParams::erp** (p. 822).

See Also

See for example http://www.ode.org/ode-latest-userguide.html#sec_3_8_2 for more details.

10.169.4.9 double gazebo::physics::SurfaceParams::kp

spring constant equivalents of a contact as a function of **SurfaceParams::cfm** (p. 822) and **SurfaceParams::erp** (p. 822).

See Also

See for example http://www.ode.org/ode-latest-userguide.html#sec_3_8_2 for more details.

10.169.4.10 double gazebo::physics::SurfaceParams::maxVel

Maximum interpenetration error correction velocity.

If set to 0, two objects interpenetrating each other will not be pushed apart.

See Also

See dWroldSetContactMaxCorrectingVel (http://www.ode.org/ode-latest-userguide.html#sec_5_2_0)

10.169.4.11 double gazebo::physics::SurfaceParams::minDepth

Minimum depth before ERP takes effect.

See Also

See dWorldSetContactSurfaceLayer (http://www.ode.org/ode-latest-userguide.html#sec_5_2_0)

10.169.4.12 double gazebo::physics::SurfaceParams::mu1

Dry friction coefficient in the primary friction direction as defined by the friction pyramid.

This is fdir1 if defined, otherwise, a vector consstrained to be perpendicular to the contact normal in the global y-z plane is used.

See Also

http://www.ode.org/ode-latest-userguide.html#sec_7_3_7

10.169.4.13 double gazebo::physics::SurfaceParams::mu2

Dry friction coefficient in the second friction direction as defined by the friction pyramid.

This is fdir1 if defined, otherwise, a vector consstrained to be perpendicular to the contact normal in the global y-z plane is used.

See Also

http://www.ode.org/ode-latest-userguide.html#sec_7_3_7

10.169.4.14 double gazebo::physics::SurfaceParams::slip1

Artificial contact slip in the primary friction direction.

See Also

See dContactSlip1 in http://www.ode.org/ode-latest-userguide.html#sec_7_3_7

10.169.4.15 double gazebo::physics::SurfaceParams::slip2

Artificial contact slip in the secondary friction dirction.

See Also

See dContactSlip2 in http://www.ode.org/ode-latest-userguide.html#sec_7_3_7

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

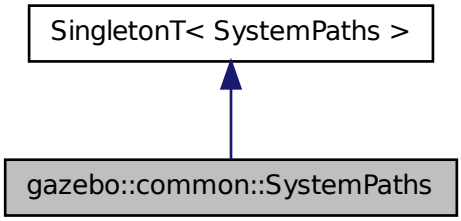
- **SurfaceParams.hh**

10.170 gazebo::common::SystemPaths Class Reference

Functions to handle getting system paths, keeps track of:

`#include <common/common.hh>`

Inheritance diagram for gazebo::common::SystemPaths:



Public Member Functions

- void **AddGazeboPaths** (const std::string &_path)

  Add colon delimited paths to Gazebo install.
- void **AddModelPaths** (const std::string &_path)

  Add colon delimited paths to modelPaths.
- void **AddOgrePaths** (const std::string &_path)

  Add colon delimited paths to ogre install.
- void **AddPluginPaths** (const std::string &_path)

  Add colon delimited paths to plugins.
- void **AddSearchPathSuffix** (const std::string &_suffix)

  add _suffix to the list of path search suffixes
- void **ClearGazeboPaths** ()

  clear out SystemPaths::gazeboPaths
- void **ClearModelPaths** ()

  clear out SystemPaths::modelPaths
- void **ClearOgrePaths** ()

  clear out SystemPaths::ogrePaths
- void **ClearPluginPaths** ()

  clear out SystemPaths::pluginPaths
- std::string **FindFile** (const std::string &_filename, bool _searchLocalPath=true)

  Find a file in the gazebo paths.
- std::string **FindFileURI** (const std::string &_uri)

  Find a file or path using a URI.
- const std::list< std::string > & **GetGazeboPaths** ()

  Get the gazebo install paths.
- std::string **GetLogPath** () const

  Get the log path.
- const std::list< std::string > & **GetModelPaths** ()

  Get the model paths.
- const std::list< std::string > & **GetOgrePaths** ()

Get the ogre install paths.

- const std::list< std::string > & **GetPluginPaths** ()

  Get the plugin paths.

- std::string **GetWorldPathExtension** ()

  Returns the world path extension.

Public Attributes

- bool **gazeboPathsFromEnv**

  *if true, call UpdateGazeboPaths() within **GetGazeboPaths()** (p. 828)*

- bool **modelPathsFromEnv**

  *if true, call UpdateGazeboPaths() within **GetGazeboPaths()** (p. 828)*

- bool **ogrePathsFromEnv**

  *if true, call UpdateOgrePaths() within **GetOgrePaths()** (p. 828)*

- bool **pluginPathsFromEnv**

  *if true, call UpdatePluginPaths() within **GetPluginPaths()** (p. 829)*

Additional Inherited Members

10.170.1 Detailed Description

Functions to handle getting system paths, keeps track of:

- SystemPaths::gazeboPaths - media paths containing worlds, models, sdf descriptions, material scripts, textures.
- SystemPaths::ogrePaths - ogre library paths. Should point to **Ogre** (p. 110) RenderSystem_GL.so et. al.
- SystemPaths::pluginPaths - plugin library paths for common::WorldPlugin

10.170.2 Member Function Documentation

10.170.2.1 void gazebo::common::SystemPaths::AddGazeboPaths (const std::string & *_path*)

Add colon delimited paths to Gazebo install.

Parameters

in	*_path*	the directory to add

10.170.2.2 void gazebo::common::SystemPaths::AddModelPaths (const std::string & *_path*)

Add colon delimited paths to modelPaths.

Parameters

in	*_path*	the directory to add

10.170.2.3 void gazebo::common::SystemPaths::AddOgrePaths (const std::string & _path)

Add colon delimited paths to ogre install.

Parameters

in	*_path*	the directory to add

10.170.2.4 void gazebo::common::SystemPaths::AddPluginPaths (const std::string & _path)

Add colon delimited paths to plugins.

Parameters

in	*_path*	the directory to add

10.170.2.5 void gazebo::common::SystemPaths::AddSearchPathSuffix (const std::string & _suffix)

add _suffix to the list of path search suffixes

Parameters

in	*_suffix*	The suffix to add

10.170.2.6 void gazebo::common::SystemPaths::ClearGazeboPaths ( )

clear out SystemPaths::gazeboPaths

10.170.2.7 void gazebo::common::SystemPaths::ClearModelPaths ( )

clear out SystemPaths::modelPaths

10.170.2.8 void gazebo::common::SystemPaths::ClearOgrePaths ( )

clear out SystemPaths::ogrePaths

10.170.2.9 void gazebo::common::SystemPaths::ClearPluginPaths ( )

clear out SystemPaths::pluginPaths

10.170.2.10 std::string gazebo::common::SystemPaths::FindFile (const std::string & _filename, bool _searchLocalPath = true)

Find a file in the gazebo paths.

Parameters

in	*_filename*	Name of the file to find.
in	*_searchLocal-Path*	True to search in the current working directory.

Returns

Returns full path name to file

10.170.2.11 std::string gazebo::common::SystemPaths::FindFileURI (const std::string & *_uri*)

Find a file or path using a URI.

Parameters

in	*_uri*	the uniform resource identifier

Returns

Returns full path name to file

10.170.2.12 const std::list<std::string>& gazebo::common::SystemPaths::GetGazeboPaths ( )

Get the gazebo install paths.

Returns

a list of paths

10.170.2.13 std::string gazebo::common::SystemPaths::GetLogPath ( ) const

Get the log path.

Returns

the path

10.170.2.14 const std::list<std::string>& gazebo::common::SystemPaths::GetModelPaths ( )

Get the model paths.

Returns

a list of paths

10.170.2.15 const std::list<std::string>& gazebo::common::SystemPaths::GetOgrePaths ( )

Get the ogre install paths.

Returns

a list of paths

10.170.2.16 const std::list<std::string>& gazebo::common::SystemPaths::GetPluginPaths ( )

Get the plugin paths.

Returns

a list of paths

10.170.2.17 std::string gazebo::common::SystemPaths::GetWorldPathExtension ( )

Returns the world path extension.

Returns

Right now, it just returns "/worlds"

10.170.3 Member Data Documentation

10.170.3.1 bool gazebo::common::SystemPaths::gazeboPathsFromEnv

if true, call UpdateGazeboPaths() within **GetGazeboPaths()** (p. 828)

10.170.3.2 bool gazebo::common::SystemPaths::modelPathsFromEnv

if true, call UpdateGazeboPaths() within **GetGazeboPaths()** (p. 828)

10.170.3.3 bool gazebo::common::SystemPaths::ogrePathsFromEnv

if true, call UpdateOgrePaths() within **GetOgrePaths()** (p. 828)

10.170.3.4 bool gazebo::common::SystemPaths::pluginPathsFromEnv

if true, call UpdatePluginPaths() within **GetPluginPaths()** (p. 829)

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

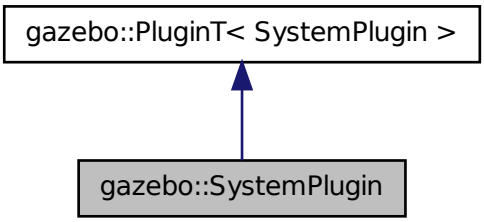
- **SystemPaths.hh**

10.171 gazebo::SystemPlugin Class Reference

A plugin loaded within the gzserver on startup.

`#include <Plugin.hh>`

Inheritance diagram for gazebo::SystemPlugin:



Public Member Functions

- **SystemPlugin** ()
  Constructor.
- virtual ∼**SystemPlugin** ()
  Destructor.
- virtual void **Init** ()
  Initialize the plugin.
- virtual void **Load** (int _argc=0, char ∗∗_argv=**NULL**)=0
  Load function.
- virtual void **Reset** ()
  Override this method for custom plugin reset behavior.

Additional Inherited Members

10.171.1 Detailed Description

A plugin loaded within the gzserver on startup.

See `reference`.

Todo how to make doxygen reference to the file gazebo.cc::g_plugins?

10.171.2 Constructor & Destructor Documentation

10.171.2.1 gazebo::SystemPlugin::SystemPlugin ( ) [inline]

Constructor.

References gazebo::SYSTEM_PLUGIN, and gazebo::PluginT< SystemPlugin >::type.

10.171.2.2 virtual gazebo::SystemPlugin::∼SystemPlugin ( ) [inline], [virtual]

Destructor.

10.171.3 Member Function Documentation

10.171.3.1 virtual void gazebo::SystemPlugin::Init ( ) [inline], [virtual]

Initialize the plugin.

Called after Gazebo has been loaded. Must not block.

10.171.3.2 virtual void gazebo::SystemPlugin::Load (int *_argc* = 0, char ** *_argv* = NULL) [pure virtual]

Load function.

Called before Gazebo is loaded. Must not block.

Parameters

_argc	Number of command line arguments.
_argv	Array of command line arguments.

10.171.3.3 virtual void gazebo::SystemPlugin::Reset ( ) [inline], [virtual]

Override this method for custom plugin reset behavior.

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

- **common/Plugin.hh**

10.172 gazebo::common::Time Class Reference

A **Time** (p. 831) class, can be used to hold wall- or sim-time.

`#include <common/common.hh>`

Public Member Functions

- **Time** ()
  Constructors.
- **Time** (const **Time** &_time)
  Copy constructor.
- **Time** (const struct timeval &_tv)
  Constructor.
- **Time** (const struct timespec &_tv)
  Constructor.
- **Time** (int32_t _sec, int32_t _nsec)
  Constructor.
- **Time** (double _time)
  Constuctor.
- virtual ~**Time** ()
  Destructor.

- double **Double** () const

  Get the time as a double.
- float **Float** () const

  Get the time as a float.
- bool **operator!=** (const struct timeval &_tv) const

  Equal to operator.
- bool **operator!=** (const struct timespec &_tv) const

  Equal to operator.
- bool **operator!=** (const **Time** &_time) const

  Equal to operator.
- bool **operator!=** (double _time) const

  Equal to operator.
- **Time operator**∗ (const struct timeval &_tv) const

  Multiplication operator.
- **Time operator**∗ (const struct timespec &_tv) const

  Multiplication operator.
- **Time operator**∗ (const **Time** &_time) const

  Multiplication operators.
- const **Time** & **operator**∗**=** (const struct timeval &_tv)

  Multiplication assignment operator.
- const **Time** & **operator**∗**=** (const struct timespec &_tv)

  Multiplication assignment operator.
- const **Time** & **operator**∗**=** (const **Time** &_time)

  Multiplication operators.
- **Time operator+** (const struct timeval &_tv) const

  Addition operators.
- **Time operator+** (const struct timespec &_tv) const

  Addition operators.
- **Time operator+** (const **Time** &_time) const

  Addition operators.
- const **Time** & **operator+=** (const struct timeval &_tv)

  Addition assignment operator.
- const **Time** & **operator+=** (const struct timespec &_tv)

  Addition assignment operator.
- const **Time** & **operator+=** (const **Time** &_time)

  Addition assignemtn operator.
- **Time operator-** (const struct timeval &_tv) const

  Subtraction operator.
- **Time operator-** (const struct timespec &_tv) const

  Subtraction operator.
- **Time operator-** (const **Time** &_time) const

  Subtraction operator.
- const **Time** & **operator-=** (const struct timeval &_tv)

  Subtraction assignment operator.
- const **Time** & **operator-=** (const struct timespec &_tv)

  Subtraction assignment operator.
- const **Time** & **operator-=** (const **Time** &_time)

Subtraction assignment operator.

- **Time operator**/ (const struct timeval &_tv) const

  Division operator.
- **Time operator**/ (const struct timespec &_tv) const

  Division operator.
- **Time operator**/ (const **Time** &_time) const

  Division operator.
- const **Time** & **operator/=** (const struct timeval &_tv)

  Division assignment operator.
- const **Time** & **operator/=** (const struct timespec &_tv)

  Division assignment operator.
- const **Time** & **operator/=** (const **Time** &time)

  Division assignment operator.
- bool **operator**< (const struct timeval &_tv) const

  Less than operator.
- bool **operator**< (const struct timespec &_tv) const

  Less than operator.
- bool **operator**< (const **Time** &_time) const

  Less than operator.
- bool **operator**< (double _time) const

  Less than operator.
- bool **operator**<**=** (const struct timeval &_tv) const

  Less than or equal to operator.
- bool **operator**<**=** (const struct timespec &_tv) const

  Less than or equal to operator.
- bool **operator**<**=** (const **Time** &_time) const

  Less than or equal to operator.
- bool **operator**<**=** (double _time) const

  Less than or equal to operator.
- **Time** & **operator=** (const struct timeval &_tv)

  Assignment operator.
- **Time** & **operator=** (const struct timespec &_tv)

  Assignment operator.
- **Time** & **operator=** (const **Time** &_time)

  Assignment operator.
- bool **operator==** (const struct timeval &_tv) const

  Equal to operator.
- bool **operator==** (const struct timespec &_tv) const

  Equal to operator.
- bool **operator==** (const **Time** &_time) const

  Equal to operator.
- bool **operator==** (double _time) const

  Equal to operator.
- bool **operator**> (const struct timeval &_tv) const

  Greater than operator.
- bool **operator**> (const struct timespec &_tv) const

  Greater than operator.

- bool **operator**> (const **Time** &_time) const

  Greater than operator.
- bool **operator**> (double _time) const

  Greater than operator.
- bool **operator**>= (const struct timeval &_tv) const

  Greater than or equal operator.
- bool **operator**>= (const struct timespec &_tv) const

  Greater than or equal operator.
- bool **operator**>= (const **Time** &_time) const

  Greater than or equal operator.
- bool **operator**>= (double _time) const

  Greater than or equal operator.
- void **Set** (int32_t _sec, int32_t _nsec)

  Set to sec and nsec.
- void **Set** (double _seconds)

  Set to seconds.
- void **SetToWallTime** ()

  Set the time to the wall time.

Static Public Member Functions

- static const **Time** & **GetWallTime** ()

  Get the wall time.
- static const std::string & **GetWallTimeAsISOString** ()

  Get the wall time as an ISO string: YYYY-MM-DDTHH:MM:SS.
- static double **MicToNano** (double _ms)

  Convert microseconds to nanoseconds.
- static double **MilToNano** (double _ms)

  Convert milliseconds to nanoseconds.
- static **Time MSleep** (unsigned int _ms)

  Millisecond sleep.
- static **Time NSleep** (unsigned int _ns)

  Nano sleep.
- static **Time NSleep** (**Time** _time) **GAZEBO_DEPRECATED**(1.5)

  Nano sleep.
- static double **SecToNano** (double _sec)

  Convert seconds to nanoseconds.
- static **Time Sleep** (const **common::Time** &_time)

  Sleep for the specified time.

Public Attributes

- int32_t **nsec**

  Nanoseconds.
- int32_t **sec**

  Seconds.

Static Public Attributes

- static const **Time Zero**

  A static zero time variable set to common::Time(0, 0).

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::common::Time** &_time)

  Stream insertion operator.
- std::istream & **operator**>> (std::istream &_in, **gazebo::common::Time** &_time)

  Stream extraction operator.

10.172.1 Detailed Description

A **Time** (p. 831) class, can be used to hold wall- or sim-time.

stored as sec and nano-sec.

10.172.2 Constructor & Destructor Documentation

10.172.2.1 gazebo::common::Time::Time ( )

Constructors.

10.172.2.2 gazebo::common::Time::Time (const Time & *_time*)

Copy constructor.

Parameters

in	*time*	**Time** (p. 831) to copy

10.172.2.3 gazebo::common::Time::Time (const struct timeval & *_tv*)

Constructor.

Parameters

in	*_tv*	**Time** (p. 831) to initialize to

10.172.2.4 gazebo::common::Time::Time (const struct timespec & *_tv*)

Constructor.

Parameters

in	*_tv*	**Time** (p. 831) to initialize to

10.172.2.5 gazebo::common::Time::Time (int32_t _sec, int32_t _nsec)

Constructor.

Parameters

in	*_sec*	Seconds
in	*_nsec*	Nanoseconds

10.172.2.6 gazebo::common::Time::Time (double _time)

Constuctor.

Parameters

in	*_time*	**Time** (p. 831) in double format sec.nsec

10.172.2.7 virtual gazebo::common::Time::~Time ( ) [virtual]

Destructor.

10.172.3 Member Function Documentation

10.172.3.1 double gazebo::common::Time::Double ( ) const

Get the time as a double.

Returns

Time (p. 831) as a double in seconds

10.172.3.2 float gazebo::common::Time::Float ( ) const

Get the time as a float.

Returns

Time (p. 831) as a float in seconds

10.172.3.3 static const Time& gazebo::common::Time::GetWallTime ( ) [static]

Get the wall time.

Returns

the current time

10.172.3.4 static const std::string& gazebo::common::Time::GetWallTimeAsISOString ( ) [static]

Get the wall time as an ISO string: YYYY-MM-DDTHH:MM:SS.

Returns

The current wall time as an ISO string.

10.172.3.5 static double gazebo::common::Time::MicToNano (double _ms) [inline], [static]

Convert microseconds to nanoseconds.

Parameters

_ms	microseconds

Returns

nanoseconds

10.172.3.6 static double gazebo::common::Time::MilToNano (double _ms) [inline], [static]

Convert milliseconds to nanoseconds.

Parameters

in	*_ms*	milliseconds

Returns

nanoseconds

10.172.3.7 static Time gazebo::common::Time::MSleep (unsigned int _ms) [static]

Millisecond sleep.

Parameters

in	*_ms*	milliseconds

Returns

Time (p. 831) actually slept

10.172.3.8 static Time gazebo::common::Time::NSleep (unsigned int _ns) [static]

Nano sleep.

Parameters

in	*_ns*	nanoseconds

Returns

Time (p. 831) actually slept

10.172.3.9 static Time gazebo::common::Time::NSleep (Time *_time*) [static]

Nano sleep.

Parameters

in	*_time*	is a **Time** (p. 831)

Returns

Time (p. 831) actually slept

10.172.3.10 bool gazebo::common::Time::operator!= (const struct timeval & *_tv*) const

Equal to operator.

Parameters

in	*_tv*	the time to compare to

Returns

true if values are the same, false otherwise

10.172.3.11 bool gazebo::common::Time::operator!= (const struct timespec & *_tv*) const

Equal to operator.

Parameters

in	*_tv*	the time to compare to

Returns

true if values are the same, false otherwise

10.172.3.12 bool gazebo::common::Time::operator!= (const Time & *_time*) const

Equal to operator.

Parameters

in	*_time*	the time to compare to

Returns

true if values are the same, false otherwise

10.172.3.13 bool gazebo::common::Time::operator!= (double *_time*) const

Equal to operator.

Parameters

in	*_time*	the time to compare to

Returns

true if values are the same, false otherwise

10.172.3.14 Time gazebo::common::Time::operator* (const struct timeval & *_tv*) const

Multiplication operator.

Parameters

in	*_tv*	The scaling duration

Returns

Time (p. 831) instance

10.172.3.15 Time gazebo::common::Time::operator* (const struct timespec & *_tv*) const

Multiplication operator.

Parameters

in	*_tv*	the scaling duration

Returns

Time (p. 831) instance

10.172.3.16 Time gazebo::common::Time::operator* (const Time & *_time*) const

Multiplication operators.

Parameters

in	*_time*	the scaling factor

Returns

a scaled **Time** (p. 831) instance

10.172.3.17 const Time& gazebo::common::Time::operator*= (const struct timeval & *_tv*)

Multiplication assignment operator.

Parameters

in	*_tv*	the scaling duration

Returns

a reference to this instance

10.172.3.18 const Time& gazebo::common::Time::operator*= (const struct timespec & *_tv*)

Multiplication assignment operator.

Parameters

in	*_tv*	the scaling duration

Returns

a reference to this instance

10.172.3.19 const Time& gazebo::common::Time::operator*= (const Time & *_time*)

Multiplication operators.

Parameters

in	*_time*	scale factor

Returns

a scaled **Time** (p. 831) instance

10.172.3.20 Time gazebo::common::Time::operator+ (const struct timeval & *_tv*) const

Addition operators.

Parameters

in	*_tv*	the time to add

Returns

a **Time** (p. 831) instance

10.172.3.21 Time gazebo::common::Time::operator+ (const struct timespec & _tv) const

Addition operators.

Parameters

in	*_tv*	the time to add

Returns

a **Time** (p. 831) instance

10.172.3.22 Time gazebo::common::Time::operator+ (const Time & _time) const

Addition operators.

Parameters

in	*_time*	The time to add

Returns

a **Time** (p. 831) instance

10.172.3.23 const Time& gazebo::common::Time::operator+= (const struct timeval & _tv)

Addition assignment operator.

Parameters

in	*_tv*	the time to add

Returns

a reference to this instance

10.172.3.24 const Time& gazebo::common::Time::operator+= (const struct timespec & _tv)

Addition assignment operator.

Parameters

in	*_tv*	the time to add

Returns

a reference to this instance

10.172.3.25 const Time& gazebo::common::Time::operator+= (const Time & *_time*)

Addition assignemtn operator.

Parameters

in	*_time*	The time to add

Returns

a **Time** (p. 831) instance

10.172.3.26 Time gazebo::common::Time::operator- (const struct timeval & *_tv*) const

Subtraction operator.

Parameters

in	*_tv*	The time to subtract

Returns

a **Time** (p. 831) instance

10.172.3.27 Time gazebo::common::Time::operator- (const struct timespec & *_tv*) const

Subtraction operator.

Parameters

in	*_tv*	The time to subtract

Returns

a **Time** (p. 831) instance

10.172.3.28 Time gazebo::common::Time::operator- (const Time & *_time*) const

Subtraction operator.

Parameters

in	*_time*	The time to subtract

Returns

a **Time** (p. 831) instance

10.172.3.29 const Time& gazebo::common::Time::operator-= (const struct timeval & _tv)

Subtraction assignment operator.

Parameters

in	*_tv*	The time to subtract

Returns

a **Time** (p. 831) instance

10.172.3.30 const Time& gazebo::common::Time::operator-= (const struct timespec & _tv)

Subtraction assignment operator.

Parameters

in	*_tv*	The time to subtract

Returns

a **Time** (p. 831) instance

10.172.3.31 const Time& gazebo::common::Time::operator-= (const Time & _time)

Subtraction assignment operator.

Parameters

in	*_time*	The time to subtract

Returns

a reference to this instance

10.172.3.32 Time gazebo::common::Time::operator/ (const struct timeval & _tv) const

Division operator.

Parameters

in	*_tv*	a timeval divisor

Returns

a **Time** (p. 831) instance

10.172.3.33 Time gazebo::common::Time::operator/ (const struct timespec & _tv) const

Division operator.

Parameters

in	*_tv*	a timespec divisor

Returns

a **Time** (p. 831) instance

10.172.3.34 Time gazebo::common::Time::operator/ (const Time & _time) const

Division operator.

Parameters

in	*_time*	the divisor

Returns

a **Time** (p. 831) instance

10.172.3.35 const Time& gazebo::common::Time::operator/= (const struct timeval & _tv)

Division assignment operator.

Parameters

in	*_tv*	a divisor

Returns

a **Time** (p. 831) instance

10.172.3.36 const Time& gazebo::common::Time::operator/= (const struct timespec & _tv)

Division assignment operator.

Parameters

in	*_tv*	a divisor

Returns

a **Time** (p. 831) instance

10.172.3.37 const Time& gazebo::common::Time::operator/= (const Time & *time*)

Division assignment operator.

Parameters

in	*time*	the divisor

Returns

a **Time** (p. 831) instance

10.172.3.38 bool gazebo::common::Time::operator< (const struct timeval & *_tv*) const

Less than operator.

Parameters

in	*_tv*	the time to compare with

Returns

true if tv is shorter than this, false otherwise

10.172.3.39 bool gazebo::common::Time::operator< (const struct timespec & *_tv*) const

Less than operator.

Parameters

in	*_tv*	the time to compare with

Returns

true if tv is shorter than this, false otherwise

10.172.3.40 bool gazebo::common::Time::operator< (const Time & *_time*) const

Less than operator.

Parameters

in	*_time*	the time to compare with

Returns

true if time is shorter than this, false otherwise

10.172.3.41 bool gazebo::common::Time::operator< (double *_time*) const

Less than operator.

Parameters

in	*_time*	the time to compare with

Returns

true if time is shorter than this, false otherwise

10.172.3.42 bool gazebo::common::Time::operator<= (const struct timeval & *_tv*) const

Less than or equal to operator.

Parameters

in	*_tv*	the time to compare with

Returns

true if tv is shorter than or equal to this, false otherwise

10.172.3.43 bool gazebo::common::Time::operator<= (const struct timespec & *_tv*) const

Less than or equal to operator.

Parameters

in	*_tv*	the time to compare with

Returns

true if tv is shorter than or equal to this, false otherwise

10.172.3.44 bool gazebo::common::Time::operator<= (const Time & *_time*) const

Less than or equal to operator.

Parameters

in	*_time*	the time to compare with

Returns

true if time is shorter than or equal to this, false otherwise

10.172.3.45 bool gazebo::common::Time::operator<= (double *_time*) const

Less than or equal to operator.

Parameters

in	*_time*	the time to compare with

Returns

true if time is shorter than or equal to this, false otherwise

10.172.3.46 Time& gazebo::common::Time::operator= (const struct timeval & *_tv*)

Assignment operator.

Parameters

in	*_tv*	the new time

Returns

a reference to this instance

10.172.3.47 Time& gazebo::common::Time::operator= (const struct timespec & *_tv*)

Assignment operator.

Parameters

in	*_tv*	the new time

Returns

a reference to this instance

10.172.3.48 Time& gazebo::common::Time::operator= (const Time & *_time*)

Assignment operator.

Parameters

in	*_time*	the new time

Returns

a reference to this instance

10.172.3.49 bool gazebo::common::Time::operator== (const struct timeval & *_tv*) const

Equal to operator.

Parameters

in	*_tv*	the time to compare to

Returns

true if values are the same, false otherwise

10.172.3.50 bool gazebo::common::Time::operator== (const struct timespec & *_tv*) const

Equal to operator.

Parameters

in	*_tv*	the time to compare to

Returns

true if values are the same, false otherwise

10.172.3.51 bool gazebo::common::Time::operator== (const Time & *_time*) const

Equal to operator.

Parameters

in	*_time*	the time to compare to

Returns

true if values are the same, false otherwise

10.172.3.52 bool gazebo::common::Time::operator== (double *_time*) const

Equal to operator.

Parameters

in	*_time*	the time to compare to

Returns

true if values are the same, false otherwise

10.172.3.53 bool gazebo::common::Time::operator> (const struct timeval & *_tv*) const

Greater than operator.

Parameters

in	*_tv*	the time to compare with

Returns

true if time is greater than this, false otherwise

10.172.3.54 bool gazebo::common::Time::operator> (const struct timespec & *_tv*) const

Greater than operator.

Parameters

in	*_tv*	the time to compare with

Returns

true if time is greater than this, false otherwise

10.172.3.55 bool gazebo::common::Time::operator> (const Time & *_time*) const

Greater than operator.

Parameters

in	*_time*	the time to compare with

Returns

true if time is greater than this, false otherwise

10.172.3.56 bool gazebo::common::Time::operator> (double *_time*) const

Greater than operator.

Parameters

in	*_time*	the time to compare with

Returns

true if time is greater than this, false otherwise

10.172.3.57 bool gazebo::common::Time::operator>= (const struct timeval & *_tv*) const

Greater than or equal operator.

Parameters

in	*_tv*	the time to compare with

Returns

true if tv is greater than or equal to this, false otherwise

10.172.3.58 bool gazebo::common::Time::operator>= (const struct timespec & *_tv*) const

Greater than or equal operator.

Parameters

in	*_tv*	the time to compare with

Returns

true if tv is greater than or equal to this, false otherwise

10.172.3.59 bool gazebo::common::Time::operator>= (const Time & *_time*) const

Greater than or equal operator.

Parameters

in	*_time*	the time to compare with

Returns

true if time is greater than or equal to this, false otherwise

10.172.3.60 bool gazebo::common::Time::operator>= (double *_time*) const

Greater than or equal operator.

Parameters

in	*_time*	the time to compare with

Returns

true if time is greater than or equal to this, false otherwise

10.172.3.61 static double gazebo::common::Time::SecToNano (double *_sec*) [inline],[static]

Convert seconds to nanoseconds.

Parameters

in	*_sec*	duration in seconds

Returns

nanoseconds

10.172.3.62 void gazebo::common::Time::Set (int32_t *_sec*, int32_t *_nsec*)

Set to sec and nsec.

Parameters

in	*_sec*	Seconds
in	*_nsec*	Nanoseconds

10.172.3.63 void gazebo::common::Time::Set (double *_seconds*)

Set to seconds.

Parameters

in	*_seconds*	Number of seconds

10.172.3.64 void gazebo::common::Time::SetToWallTime ( )

Set the time to the wall time.

10.172.3.65 static Time gazebo::common::Time::Sleep (const common::Time & *_time*) [static]

Sleep for the specified time.

Parameters

in	*_time*	Sleep time

Returns

Time (p. 831) actually slept

10.172.4 Friends And Related Function Documentation

10.172.4.1 std::ostream& operator<< (std::ostream & *_out,* const gazebo::common::Time & *_time*) [friend]

Stream insertion operator.

Parameters

in	*_out*	the output stream
in	*_time*	time to write to the stream

Returns

the output stream

10.172.4.2 std::istream& operator>> (std::istream & *_in,* gazebo::common::Time & *_time*) [friend]

Stream extraction operator.

Parameters

in	*_in*	the input stream
in	*_time*	time to read from to the stream

Returns

the input stream

10.172.5 Member Data Documentation

10.172.5.1 int32_t gazebo::common::Time::nsec

Nanoseconds.

10.172.5.2 int32_t gazebo::common::Time::sec

Seconds.

10.172.5.3 const Time gazebo::common::Time::Zero [static]

A static zero time variable set to common::Time(0, 0).

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

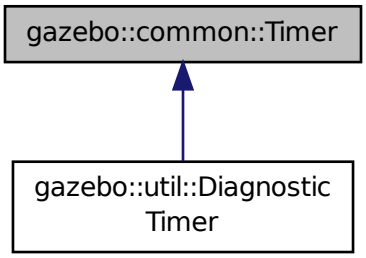
- **Time.hh**

10.173 gazebo::common::Timer Class Reference

A timer class, used to time things in real world walltime.

`#include <common/common.hh>`

Inheritance diagram for gazebo::common::Timer:



Public Member Functions

- **Timer** ()

  Constructor.
- virtual ~**Timer** ()

  Destructor.
- **Time GetElapsed** () const

  Get the elapsed time.
- bool **GetRunning** () const

  Returns true if the timer is running.
- virtual void **Start** ()

  Start the timer.
- virtual void **Stop** ()

  Stop the timer.

Friends

- std::ostream & **operator**<< (std::ostream &out, const **gazebo::common::Timer** &t)

  Stream operator friendly.

10.173.1 Detailed Description

A timer class, used to time things in real world walltime.

10.173.2 Constructor & Destructor Documentation

10.173.2.1 gazebo::common::Timer::Timer ( )

Constructor.

10.173.2.2 virtual gazebo::common::Timer::~Timer ( ) [virtual]

Destructor.

10.173.3 Member Function Documentation

10.173.3.1 Time gazebo::common::Timer::GetElapsed ( ) const

Get the elapsed time.

Returns

The time

10.173.3.2 bool gazebo::common::Timer::GetRunning ( ) const

Returns true if the timer is running.

Returns

Tue if the timer has been started and not stopped.

10.173.3.3 virtual void gazebo::common::Timer::Start ( ) [virtual]

Start the timer.

Reimplemented in **gazebo::util::DiagnosticTimer** (p. 273).

10.173.3.4 virtual void gazebo::common::Timer::Stop ( ) [virtual]

Stop the timer.

Reimplemented in **gazebo::util::DiagnosticTimer** (p. 273).

10.173.4 Friends And Related Function Documentation

10.173.4.1 std::ostream& operator<< (std::ostream & *out,* const gazebo::common::Timer & *t*) [friend]

Stream operator friendly.

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

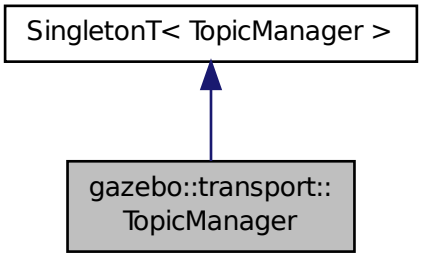
- **Timer.hh**

10.174 gazebo::transport::TopicManager Class Reference

Manages topics and their subscriptions.

`#include <transport/transport.hh>`

Inheritance diagram for gazebo::transport::TopicManager:



Public Types

- typedef std::map< std::string,
  std::list< **NodePtr** > > **SubNodeMap**

  *A map of string->list of **Node** (p. 564) pointers.*

Public Member Functions

- void **AddNode** (**NodePtr** _node)

  Add a node to the manager.
- void **AddNodeToProcess** (**NodePtr** _ptr)

  Add a node to the list of nodes that requires processing.
- template<typename M >
  PublisherPtr Advertise (const std::string &_topic, unsigned int _queueLimit, double _hzRate)

  Advertise on a topic.
- void **ClearBuffers** ()

  Clear all buffers.
- void **ConnectPubToSub** (const std::string &_topic, const **SubscriptionTransportPtr** _sublink)

  ***Connection** (p. 227) a local **Publisher** (p. 649) to a remote **Subscriber** (p. 816).*
- void **ConnectSubscribers** (const std::string &_topic)

  Connect all subscribers on a topic to known publishers.
- void **ConnectSubToPub** (const msgs::Publish &_pub)

  *Connect a local **Subscriber** (p. 816) to a remote **Publisher** (p. 649).*
- void **DisconnectPubFromSub** (const std::string &_topic, const std::string &_host, unsigned int _port)

  Disconnect a local publisher from a remote subscriber.
- void **DisconnectSubFromPub** (const std::string &_topic, const std::string &_host, unsigned int _port)

Disconnect all local subscribers from a remote publisher.

- **PublicationPtr FindPublication** (const std::string &_topic)

  Find a publication object by topic.
- void **Fini** ()

  Finalize the manager.
- std::map< std::string, std::list< std::string > > **GetAdvertisedTopics** () const **GAZEBO_DEPRECATED**(1.5)

  Get a list of all the topics.
- void **GetTopicNamespaces** (std::list< std::string > &_namespaces)

  Get all the topic namespaces.
- void **Init** ()

  Initialize the manager.
- bool **IsAdvertised** (const std::string &_topic)

  Has the topic been advertised?
- void **PauseIncoming** (bool _pause)

  Pause or unpause processing of incoming messages.
- void **ProcessNodes** (bool _onlyOut=false)

  Process all nodes under management.
- void **Publish** (const std::string &_topic, **MessagePtr** _message, boost::function< void(uint32_t)> _cb, uint32_t _id)

  Send a message.
- void **RegisterTopicNamespace** (const std::string &_name)

  Register a new topic namespace.
- void **RemoveNode** (unsigned int _id)

  Remove a node by its id.
- **SubscriberPtr Subscribe** (const **SubscribeOptions** &_options)

  Subscribe to a topic.
- void **Unadvertise** (const std::string &_topic)

  Unadvertise a topic.
- void **Unsubscribe** (const std::string &_topic, const **NodePtr** &_sub)

  Unsubscribe from a topic.
- **PublicationPtr UpdatePublications** (const std::string &_topic, const std::string &_msgType)

  Update our list of advertised topics.

Additional Inherited Members

10.174.1 Detailed Description

Manages topics and their subscriptions.

10.174.2 Member Typedef Documentation

10.174.2.1 typedef std::map<std::string, std::list<NodePtr> > gazebo::transport::TopicManager::SubNodeMap

A map of string->list of **Node** (p. 564) pointers.

10.174.3 Member Function Documentation

10.174.3.1 void gazebo::transport::TopicManager::AddNode (NodePtr _node)

Add a node to the manager.

Parameters

in,out	*_node*	The node to be added

10.174.3.2 void gazebo::transport::TopicManager::AddNodeToProcess (NodePtr _ptr)

Add a node to the list of nodes that requires processing.

Parameters

in	*_ptr*	**Node** (p. 564) to process.

10.174.3.3 template<typename M > PublisherPtr gazebo::transport::TopicManager::Advertise (const std::string & _topic, unsigned int _queueLimit, double _hzRate) [inline]

Advertise on a topic.

Parameters

in	*_topic*	The name of the topic
in	*_queueLimit*	The maximum number of outgoing messages to queue
in	*_hz*	Update rate for the publisher. Units are 1.0/seconds.

Returns

Pointer to the newly created **Publisher** (p. 649)

References FindPublication(), GZ_ASSERT, gzthrow, SingletonT< T >::Instance(), NULL, and UpdatePublications().

10.174.3.4 void gazebo::transport::TopicManager::ClearBuffers ( )

Clear all buffers.

10.174.3.5 void gazebo::transport::TopicManager::ConnectPubToSub (const std::string & _topic, const SubscriptionTransportPtr _sublink)

Connection (p. 227) a local **Publisher** (p. 649) to a remote **Subscriber** (p. 816).

Parameters

in	*_topic*	The topic to use
in	*_sublink*	The subscription transport object to use

10.174.3.6 void gazebo::transport::TopicManager::ConnectSubscribers (const std::string & _topic)

Connect all subscribers on a topic to known publishers.

Parameters

in	*_topic*	The topic to be connected

10.174.3.7 void gazebo::transport::TopicManager::ConnectSubToPub (const msgs::Publish & _pub)

Connect a local **Subscriber** (p. 816) to a remote **Publisher** (p. 649).

Parameters

in	*_pub*	The publish object to use

10.174.3.8 void gazebo::transport::TopicManager::DisconnectPubFromSub (const std::string & _topic, const std::string & _host, unsigned int _port)

Disconnect a local publisher from a remote subscriber.

Parameters

in	*_topic*	The topic to be disconnected
in	*_host*	The host to be disconnected
in	*_port*	The port to be disconnected

10.174.3.9 void gazebo::transport::TopicManager::DisconnectSubFromPub (const std::string & _topic, const std::string & _host, unsigned int _port)

Disconnect all local subscribers from a remote publisher.

Parameters

in	*_topic*	The topic to be disconnected
in	*_host*	The host to be disconnected
in	*_port*	The port to be disconnected

10.174.3.10 PublicationPtr gazebo::transport::TopicManager::FindPublication (const std::string & _topic)

Find a publication object by topic.

Parameters

in	*_topic*	The topic to search for

Returns

Pointer to the publication object, if found (can be null)

Referenced by Advertise().

10.174.3.11 void gazebo::transport::TopicManager::Fini ( )

Finalize the manager.

10.174.3.12 std::map<std::string, std::list<std::string> > gazebo::transport::TopicManager::GetAdvertisedTopics ( ) const

Get a list of all the topics.

Returns

A map where keys are message types, and values are a list of topic names.

See Also

transport::GetAdvertisedTopics

10.174.3.13 void gazebo::transport::TopicManager::GetTopicNamespaces (std::list< std::string > & *_namespaces*)

Get all the topic namespaces.

Parameters

out	*_namespaces*	The list of namespaces will be written here

10.174.3.14 void gazebo::transport::TopicManager::Init ( )

Initialize the manager.

10.174.3.15 bool gazebo::transport::TopicManager::IsAdvertised (const std::string & *_topic*)

Has the topic been advertised?

Parameters

in	*_topic*	The name of the topic to check

Returns

true if the topic has been advertised, false otherwise

10.174.3.16 void gazebo::transport::TopicManager::PauseIncoming (bool *_pause*)

Pause or unpause processing of incoming messages.

Parameters

in	*_pause*	If true pause processing; otherwse unpause

10.174.3.17 void gazebo::transport::TopicManager::ProcessNodes (bool *_onlyOut* = `false`)

Process all nodes under management.

Parameters

in	*_onlyOut*	True means only outbound messages on nodes will be sent. False means nodes process both outbound and inbound messages

10.174.3.18 void gazebo::transport::TopicManager::Publish (const std::string & *_topic,* **MessagePtr** *_message,* boost::function< void(uint32_t)> *_cb,* uint32_t *_id*)

Send a message.

Use a **Publisher** (p. 649) instead of calling this function directly.

Parameters

in	*_topic*	Name of the topic
in	*_message*	The message to send.
in	*_cb*	Callback, used when the publish is completed.
in	*_id*	ID associated with the message.

10.174.3.19 void gazebo::transport::TopicManager::RegisterTopicNamespace (const std::string & *_name*)

Register a new topic namespace.

Parameters

in	*_name*	The name of the new namespace

10.174.3.20 void gazebo::transport::TopicManager::RemoveNode (unsigned int *_id*)

Remove a node by its id.

Parameters

in	*_id*	The ID of the node to be removed

10.174.3.21 SubscriberPtr gazebo::transport::TopicManager::Subscribe (const SubscribeOptions & *_options*)

Subscribe to a topic.

Parameters

in	*_options*	The options to use for the subscription

Returns

Pointer to the newly created subscriber

10.174.3.22 void gazebo::transport::TopicManager::Unadvertise (const std::string & *_topic*)

Unadvertise a topic.

Parameters

in	*_topic*	The topic to be unadvertised

10.174.3.23 void gazebo::transport::TopicManager::Unsubscribe (const std::string & *_topic,* const **NodePtr** & *_sub*)

Unsubscribe from a topic.

Use a **Subscriber** (p. 816) rather than calling this function directly

Parameters

in	*_topic*	The topic to unsubscribe from
in	*_sub*	The node to unsubscribe

10.174.3.24 **PublicationPtr** gazebo::transport::TopicManager::UpdatePublications (const std::string & *_topic,* const std::string & *_msgType*)

Update our list of advertised topics.

Parameters

in	*_topic*	The topic to be updated
in	*_msgType*	The type of the topic to be updated

Returns

True if the provided params define a new publisher, false otherwise

Referenced by Advertise().

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

- **TopicManager.hh**

10.175 gazebo::physics::TrajectoryInfo Struct Reference

`#include <Actor.hh>`

Public Attributes

- double **duration**
- double **endTime**
- unsigned int **id**
- double **startTime**
- bool **translated**
- std::string **type**

10.175.1 Member Data Documentation

10.175.1.1 double gazebo::physics::TrajectoryInfo::duration

10.175.1.2 double gazebo::physics::TrajectoryInfo::endTime

10.175.1.3 unsigned int gazebo::physics::TrajectoryInfo::id

10.175.1.4 double gazebo::physics::TrajectoryInfo::startTime

10.175.1.5 bool gazebo::physics::TrajectoryInfo::translated

10.175.1.6 std::string gazebo::physics::TrajectoryInfo::type

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

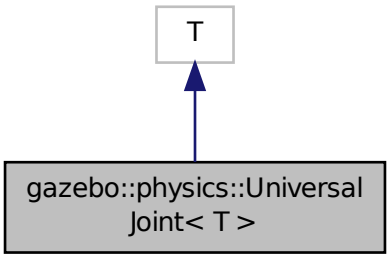
- **Actor.hh**

10.176 gazebo::physics::UniversalJoint< T > Class Template Reference

A universal joint.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::UniversalJoint< T >:



Public Member Functions

- **UniversalJoint** (**BasePtr** _parent)

  Constructor.
- virtual ~**UniversalJoint** ()

  Destuctor.
- virtual unsigned int **GetAngleCount** () const

- virtual void **Load** (**sdf::ElementPtr** _sdf)

  *Load a **UniversalJoint** (p. 862).*

10.176.1 Detailed Description

template< class T > class gazebo::physics::UniversalJoint< T >

A universal joint.

10.176.2 Constructor & Destructor Documentation

10.176.2.1 template< class T > gazebo::physics::UniversalJoint< T >::UniversalJoint (BasePtr _parent) [inline], [explicit]

Constructor.

Parameters

in	*_parent*	Parent link of the univeral joint.

References gazebo::physics::Base::UNIVERSAL_JOINT.

10.176.2.2 template< class T > virtual gazebo::physics::UniversalJoint< T >::~UniversalJoint ( ) [inline], [virtual]

Destuctor.

10.176.3 Member Function Documentation

10.176.3.1 template< class T > virtual unsigned int gazebo::physics::UniversalJoint< T >::GetAngleCount ( ) const [inline], [virtual]

10.176.3.2 template< class T > virtual void gazebo::physics::UniversalJoint< T >::Load (sdf::ElementPtr _sdf) [inline], [virtual]

Load a **UniversalJoint** (p. 862).

Parameters

in	*_sdf*	SDF values to load from.

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

- **UniversalJoint.hh**

10.177 gazebo::common::UpdateInfo Class Reference

Information for use in an update event.

`#include <common/common.hh>`

Public Attributes

- **common::Time realTime**

  Current real time.
- **common::Time simTime**

  Current simulation time.
- std::string **worldName**

  Name of the world.

10.177.1 Detailed Description

Information for use in an update event.

10.177.2 Member Data Documentation

10.177.2.1 common::Time gazebo::common::UpdateInfo::realTime

Current real time.

10.177.2.2 common::Time gazebo::common::UpdateInfo::simTime

Current simulation time.

10.177.2.3 std::string gazebo::common::UpdateInfo::worldName

Name of the world.

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

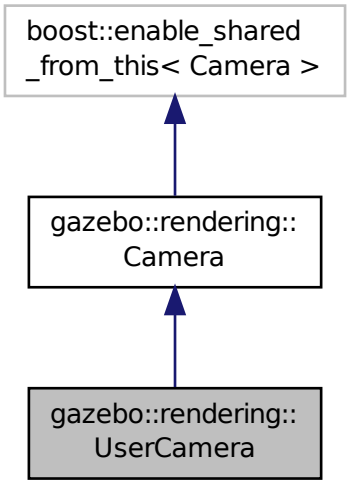
- **UpdateInfo.hh**

10.178 gazebo::rendering::UserCamera Class Reference

A camera used for user visualization of a scene.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::UserCamera:



Public Member Functions

- **UserCamera** (const std::string &_name, **ScenePtr** _scene)

  Constructor.
- virtual ~**UserCamera** ()

  Destructor.
- void **EnableViewController** (bool _value) const

  Set whether the view controller is enabled.
- void **Fini** ()

  Finialize.
- float **GetAvgFPS** () const

  Get the average frames per second.
- **GUIOverlay** * **GetGUIOverlay** ()

  Get the GUI overlay.
- virtual unsigned int **GetImageHeight** () const

  Get the height of the image.
- virtual unsigned int **GetImageWidth** () const

  Get the width of the image.
- float **GetTriangleCount** () const

  Get the triangle count.
- std::string **GetViewControllerTypeString** ()

  Get current view controller type.
- **VisualPtr GetVisual** (const **math::Vector2i** &_mousePos, std::string &_mod)

  Get an entity at a pixel location using a camera.

- **VisualPtr GetVisual** (const **math::Vector2i** &_mousePos) const

  Get a visual at a mouse position.
- void **HandleKeyPressEvent** (const std::string &_key)

  Handle a key press.
- void **HandleKeyReleaseEvent** (const std::string &_key)

  Handle a key release.
- void **HandleMouseEvent** (const **common::MouseEvent** &_evt)

  Handle a mouse event.
- void **Init** ()

  Initialize.
- void **Load** (**sdf::ElementPtr** _sdf)

  Load the user camera.
- void **Load** ()

  Generic load function.
- virtual bool **MoveToPosition** (const **math::Pose** &_pose, double _time)

  Move the camera to a position (this is an animated motion).
- void **MoveToVisual** (**VisualPtr** _visual)

  Move the camera to focus on a visual.
- void **MoveToVisual** (const std::string &_visualName)

  Move the camera to focus on a visual.
- virtual void **PostRender** ()

  Post render.
- void **Resize** (unsigned int _w, unsigned int _h)

  Resize the camera.
- void **SetFocalPoint** (const **math::Vector3** &_pt)

  Set the point the camera should orbit around.
- virtual void **SetRenderTarget** (Ogre::RenderTarget ∗_target)

  Set to true to enable rendering.
- void **SetViewController** (const std::string &_type)

  Set view controller.
- void **SetViewController** (const std::string &_type, const **math::Vector3** &_pos)

  Set view controller.
- void **SetViewportDimensions** (float _x, float _y, float _w, float _h)

  Set the dimensions of the viewport.
- virtual void **SetWorldPose** (const **math::Pose** &_pose)

  Set the pose in the world coordinate frame.
- virtual void **Update** ()

  Render the camera.

Protected Member Functions

- virtual void **AnimationComplete** ()

  Internal function used to indicate that an animation has completed.
- virtual bool **AttachToVisualImpl** (**VisualPtr** _visual, bool _inheritOrientation, double _minDist=0, double _maxDist=0)

  Set the camera to be attached to a visual.
- virtual bool **TrackVisualImpl** (**VisualPtr** _visual)

  Set the camera to track a scene node.

Additional Inherited Members

10.178.1 Detailed Description

A camera used for user visualization of a scene.

10.178.2 Constructor & Destructor Documentation

10.178.2.1 gazebo::rendering::UserCamera::UserCamera (const std::string & *_name,* ScenePtr *_scene*)

Constructor.

Parameters

in	*_name*	Name of the camera.
in	*_scene*	**Scene** (p. 707) to put the camera in.

10.178.2.2 virtual gazebo::rendering::UserCamera::~UserCamera ( ) [virtual]

Destructor.

10.178.3 Member Function Documentation

10.178.3.1 virtual void gazebo::rendering::UserCamera::AnimationComplete ( ) [protected], [virtual]

Internal function used to indicate that an animation has completed.

Reimplemented from **gazebo::rendering::Camera** (p. 173).

10.178.3.2 virtual bool gazebo::rendering::UserCamera::AttachToVisualImpl (VisualPtr *_visual,* bool *_inheritOrientation,* double *_minDist* = 0, double *_maxDist* = 0) [protected], [virtual]

Set the camera to be attached to a visual.

This causes the camera to move in relation to the specified visual.

Parameters

in	*_visual*	The visual to attach to.
in	*_inherit-Orientation*	True if the camera should also rotate when the visual rotates.
in	*_minDist*	Minimum distance the camera can get to the visual.
in	*_maxDist*	Maximum distance the camera can get from the visual.

Returns

True if successfully attach to the visual.

Reimplemented from **gazebo::rendering::Camera** (p. 174).

10.178.3.3 void gazebo::rendering::UserCamera::EnableViewController (bool *_value*) const

Set whether the view controller is enabled.

The view controller is used to handle user camera movements.

Parameters

in	*_value*	True to enable viewcontroller, False to disable.

10.178.3.4 void gazebo::rendering::UserCamera::Fini ( ) [virtual]

Finialize.

Reimplemented from **gazebo::rendering::Camera** (p. 175).

10.178.3.5 float gazebo::rendering::UserCamera::GetAvgFPS ( ) const

Get the average frames per second.

Returns

The average rendering frames per second

10.178.3.6 GUIOverlay* gazebo::rendering::UserCamera::GetGUIOverlay ( )

Get the GUI overlay.

An overlay allows you to draw 2D elements on the viewport.

Returns

Pointer to the **GUIOverlay** (p. 361).

10.178.3.7 virtual unsigned int gazebo::rendering::UserCamera::GetImageHeight ( ) const [virtual]

Get the height of the image.

Returns

Image height

Reimplemented from **gazebo::rendering::Camera** (p. 177).

10.178.3.8 virtual unsigned int gazebo::rendering::UserCamera::GetImageWidth ( ) const [virtual]

Get the width of the image.

Returns

Image width

Reimplemented from **gazebo::rendering::Camera** (p. 178).

10.178.3.9 float gazebo::rendering::UserCamera::GetTriangleCount ( ) const

Get the triangle count.

Returns

The number of triangles currently being rendered.

10.178.3.10 std::string gazebo::rendering::UserCamera::GetViewControllerTypeString ( )

Get current view controller type.

Returns

Type of the current view controller: "orbit", "fps"

10.178.3.11 VisualPtr gazebo::rendering::UserCamera::GetVisual (const math::Vector2i & _mousePos, std::string & _mod)

Get an entity at a pixel location using a camera.

Used for mouse picking.

Parameters

in	*_mousePos*	The position of the mouse in screen coordinates
out	*_mod*	Used for object manipulation

Returns

The selected entity, or NULL

10.178.3.12 VisualPtr gazebo::rendering::UserCamera::GetVisual (const math::Vector2i & _mousePos) const

Get a visual at a mouse position.

Parameters

in	*_mousePos*	2D position of the mouse in pixels.

10.178.3.13 void gazebo::rendering::UserCamera::HandleKeyPressEvent (const std::string & _key)

Handle a key press.

Parameters

in	*_key*	The key pressed.

10.178.3.14 void gazebo::rendering::UserCamera::HandleKeyReleaseEvent (const std::string & _key)

Handle a key release.

Parameters

in	*_key*	The key released.

10.178.3.15 void gazebo::rendering::UserCamera::HandleMouseEvent (const common::MouseEvent & _evt)

Handle a mouse event.

Parameters

in	*_evt*	The mouse event.

10.178.3.16 void gazebo::rendering::UserCamera::Init ( ) [virtual]

Initialize.

Reimplemented from **gazebo::rendering::Camera** (p. 182).

10.178.3.17 void gazebo::rendering::UserCamera::Load (sdf::ElementPtr _sdf) [virtual]

Load the user camera.

Parameters

in	*_sdf*	Parameters for the camera.

Reimplemented from **gazebo::rendering::Camera** (p. 183).

10.178.3.18 void gazebo::rendering::UserCamera::Load ( ) [virtual]

Generic load function.

Reimplemented from **gazebo::rendering::Camera** (p. 183).

10.178.3.19 virtual bool gazebo::rendering::UserCamera::MoveToPosition (const math::Pose & _pose, double _time) [virtual]

Move the camera to a position (this is an animated motion).

See Also

Camera::MoveToPositions (p. 184)

Parameters

in	*_pose*	End position of the camera
in	*_time*	Duration of the camera's movement

Reimplemented from **gazebo::rendering::Camera** (p. 184).

10.178.3.20 void gazebo::rendering::UserCamera::MoveToVisual (VisualPtr _visual)

Move the camera to focus on a visual.

Parameters

in	*_visual*	**Visual** (p. 920) to move the camera to.

10.178.3.21 void gazebo::rendering::UserCamera::MoveToVisual (const std::string & _visualName)

Move the camera to focus on a visual.

Parameters

in	*_visualName*	Name of the visual to move the camera to.

10.178.3.22 virtual void gazebo::rendering::UserCamera::PostRender ( ) [virtual]

Post render.

Reimplemented from **gazebo::rendering::Camera** (p. 184).

10.178.3.23 void gazebo::rendering::UserCamera::Resize (unsigned int _w, unsigned int _h)

Resize the camera.

Parameters

in	*_w*	Width of the camera image.
in	*_h*	Height of the camera image.

10.178.3.24 void gazebo::rendering::UserCamera::SetFocalPoint (const math::Vector3 & _pt)

Set the point the camera should orbit around.

Parameters

in	*_pt*	The focal point

10.178.3.25 virtual void gazebo::rendering::UserCamera::SetRenderTarget (Ogre::RenderTarget * _target) [virtual]

Set to true to enable rendering.

Use this only if you really know what you're doing.

Parameters

in	*_target*	The new rendering target.

Reimplemented from **gazebo::rendering::Camera** (p. 187).

10.178.3.26 void gazebo::rendering::UserCamera::SetViewController (const std::string & *_type*)

Set view controller.

Parameters

in	*_type*	The type of view controller: "orbit", "fps"

10.178.3.27 void gazebo::rendering::UserCamera::SetViewController (const std::string & *_type,* const **math::Vector3** & *_pos*)

Set view controller.

Parameters

in	*_type*	The type of view controller: "orbit", "fps"
in	*_pos*	The initial pose of the camera.

10.178.3.28 void gazebo::rendering::UserCamera::SetViewportDimensions (float *_x,* float *_y,* float *_w,* float *_h*)

Set the dimensions of the viewport.

Parameters

in	*_x*	X position of the viewport.
in	*_y*	Y position of the viewport.
in	*_w*	Width of the viewport.
in	*_h*	Height of the viewport.

10.178.3.29 virtual void gazebo::rendering::UserCamera::SetWorldPose (const **math::Pose** & *_pose*) [virtual]

Set the pose in the world coordinate frame.

Parameters

in	*_pose*	New pose of the camera.

Reimplemented from **gazebo::rendering::Camera** (p. 188).

10.178.3.30 virtual bool gazebo::rendering::UserCamera::TrackVisualImpl (**VisualPtr** *_visual*) [protected], [virtual]

Set the camera to track a scene node.

Tracking just causes the camera to rotate to follow the visual.

Parameters

in	*_visual*	**Visual** (p. 920) to track.

Returns

True if the camera is now tracking the visual.

Reimplemented from **gazebo::rendering::Camera** (p. 189).

10.178.3.31 virtual void gazebo::rendering::UserCamera::Update ( ) [virtual]

Render the camera.

Reimplemented from **gazebo::rendering::Camera** (p. 190).

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

- **UserCamera.hh**

10.179 gazebo::math::Vector2d Class Reference

Generic double x, y vector.

`#include <Vector2d.hh>`

Public Member Functions

- **Vector2d** ()
  Constructor.
- **Vector2d** (const double &_x, const double &_y)
  Constructor.
- **Vector2d** (const **Vector2d** &_v)
  Copy constructor.
- virtual ∼**Vector2d** ()
  Destructor.
- **Vector2d Cross** (const **Vector2d** &_v) const
  Return the cross product of this vector and _v.
- double **Distance** (const **Vector2d** &_pt) const
  Calc distance to the given point.
- bool **IsFinite** () const
  See if a point is finite (e.g., not nan)
- void **Normalize** ()
  Normalize the vector length.
- bool **operator!=** (const **Vector2d** &_v) const
  Not equal to operator.
- const **Vector2d operator**∗ (const **Vector2d** &_v) const
  Multiplication operators.
- const **Vector2d operator**∗ (double _v) const

Multiplication operators.

- const **Vector2d** & **operator∗=** (const **Vector2d** &_v)

  Multiplication assignment operator.
- const **Vector2d** & **operator∗=** (double _v)

  Multiplication assignment operator.
- **Vector2d operator+** (const **Vector2d** &_v) const

  Addition operator.
- const **Vector2d** & **operator+=** (const **Vector2d** &_v)

  Addition assignment operator.
- **Vector2d operator-** (const **Vector2d** &_v) const

  Subtraction operator.
- const **Vector2d** & **operator-=** (const **Vector2d** &_v)

  Subtraction assignment operator.
- const **Vector2d operator**/ (const **Vector2d** &_v) const

  Division operator.
- const **Vector2d operator**/ (double _v) const

  Division operator.
- const **Vector2d** & **operator**/**=** (const **Vector2d** &_v)

  Division operator.
- const **Vector2d** & **operator**/**=** (double _v)

  Division operator.
- **Vector2d** & **operator=** (const **Vector2d** &_v)

  Assignment operator.
- const **Vector2d** & **operator=** (double _v)

  Assignment operator.
- bool **operator==** (const **Vector2d** &_v) const

  Equal to operator.
- double **operator[]** (unsigned int _index) const

  Array subscript operator.
- void **Set** (double _x, double _y)

  Set the contents of the vector.

Public Attributes

- double **x**

  x data
- double **y**

  y data

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::math::Vector2d** &_pt)

  Stream extraction operator.
- std::istream & **operator**>> (std::istream &_in, **gazebo::math::Vector2d** &_pt)

  Stream extraction operator.

10.179.1 Detailed Description

Generic double x, y vector.

10.179.2 Constructor & Destructor Documentation

10.179.2.1 gazebo::math::Vector2d::Vector2d ( )

Constructor.

10.179.2.2 gazebo::math::Vector2d::Vector2d (const double & _x, const double & _y)

Constructor.

Parameters

in	_x	value along x
in	_y	value along y

10.179.2.3 gazebo::math::Vector2d::Vector2d (const Vector2d & _v)

Copy constructor.

Parameters

in	_v	the value

10.179.2.4 virtual gazebo::math::Vector2d::~Vector2d ( ) [virtual]

Destructor.

10.179.3 Member Function Documentation

10.179.3.1 Vector2d gazebo::math::Vector2d::Cross (const Vector2d & _v) const

Return the cross product of this vector and _v.

Parameters

in	_v	the vector

Returns

the cross product

10.179.3.2 double gazebo::math::Vector2d::Distance (const Vector2d & _pt) const

Calc distance to the given point.

Parameters

in	*_pt*	The point to measure to

Returns

the distance

10.179.3.3 bool gazebo::math::Vector2d::IsFinite ( ) const

See if a point is finite (e.g., not nan)

Returns

true if finite, false otherwise

10.179.3.4 void gazebo::math::Vector2d::Normalize ( )

Normalize the vector length.

10.179.3.5 bool gazebo::math::Vector2d::operator!= (const **Vector2d** & *_v*) const

Not equal to operator.

Returns

true if elements are of diffent values (tolerence 1e-6)

10.179.3.6 const **Vector2d** gazebo::math::Vector2d::operator* (const **Vector2d** & *_v*) const

Multiplication operators.

Parameters

in	*_v*	the vector

Returns

the result

10.179.3.7 const **Vector2d** gazebo::math::Vector2d::operator* (double *_v*) const

Multiplication operators.

Parameters

in	*_v*	the scaling factor

Returns

a scaled vector

10.179.3.8 const Vector2d& gazebo::math::Vector2d::operator*= (const Vector2d & _v)

Multiplication assignment operator.

Remarks

this is an element wise multiplication

Parameters

in	_v	the vector

Returns

this

10.179.3.9 const Vector2d& gazebo::math::Vector2d::operator*= (double _v)

Multiplication assignment operator.

Parameters

in	_v	the scaling factor

Returns

a scaled vector

10.179.3.10 Vector2d gazebo::math::Vector2d::operator+ (const Vector2d & _v) const

Addition operator.

Parameters

in	_v	vector to add

Returns

sum vector

10.179.3.11 const Vector2d& gazebo::math::Vector2d::operator+= (const Vector2d & _v)

Addition assignment operator.

Parameters

in	_v	the vector to add

10.179.3.12 Vector2d gazebo::math::Vector2d::operator- (const Vector2d & _v) const

Subtraction operator.

Parameters

in	_v	the vector to substract

Returns

the subtracted vector

10.179.3.13 const Vector2d& gazebo::math::Vector2d::operator-= (const Vector2d & _v)

Subtraction assignment operator.

Parameters

in	_v	the vector to substract

Returns

this

10.179.3.14 const Vector2d gazebo::math::Vector2d::operator/ (const Vector2d & _v) const

Division operator.

Remarks

this is an element wise division

Parameters

in	_v	a vector

Returns

a result

10.179.3.15 const Vector2d gazebo::math::Vector2d::operator/ (double _v) const

Division operator.

Parameters

in	_v	the value

Returns

a vector

10.179.3.16 const Vector2d& gazebo::math::Vector2d::operator/= (const Vector2d & _v)

Division operator.

Remarks

this is an element wise division

Parameters

in	_v	a vector

Returns

this

10.179.3.17 const Vector2d& gazebo::math::Vector2d::operator/= (double _v)

Division operator.

Parameters

in	_v	the divisor

Returns

a vector

10.179.3.18 Vector2d& gazebo::math::Vector2d::operator= (const Vector2d & _v)

Assignment operator.

Parameters

in	_v	a value for x and y element

Returns

this

10.179.3.19 const Vector2d& gazebo::math::Vector2d::operator= (double _v)

Assignment operator.

Parameters

in	_v	the value for x and y element

Returns

this

10.179.3.20 bool gazebo::math::Vector2d::operator== (const Vector2d & _v) const

Equal to operator.

Parameters

in	_v	the vector to compare to

Returns

true if the elements of the 2 vectors are equal within a tolerence (1e-6)

10.179.3.21 double gazebo::math::Vector2d::operator[ ] (unsigned int _index) const

Array subscript operator.

Parameters

in	_index	the index

Returns

the value, or 0 if _index is out of bounds

10.179.3.22 void gazebo::math::Vector2d::Set (double _x, double _y)

Set the contents of the vector.

Parameters

in	_x	value along x
in	_y	value along y

10.179.4 Friends And Related Function Documentation

10.179.4.1 std::ostream& operator<< (std::ostream & _out, const gazebo::math::Vector2d & _pt) [friend]

Stream extraction operator.

Parameters

in	*_out*	output stream
in	*_pt*	**Vector2d** (p. 873) to output

Returns

The stream

10.179.4.2 std::istream& operator>> (std::istream & _in, gazebo::math::Vector2d & _pt) [friend]

Stream extraction operator.

Parameters

in	*_in*	input stream
in	*_pt*	**Vector3** (p. 890) to read values into

Returns

The stream

10.179.5 Member Data Documentation

10.179.5.1 double gazebo::math::Vector2d::x

x data

10.179.5.2 double gazebo::math::Vector2d::y

y data

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

- **Vector2d.hh**

10.180 gazebo::math::Vector2i Class Reference

Generic integer x, y vector.

`#include <math/gzmath.hh>`

Public Member Functions

- **Vector2i** ()

Constructor.

- **Vector2i** (const int &_x, const int &_y)

  Constructor.
- **Vector2i** (const **Vector2i** &_pt)

  Copy onstructor.
- virtual ~**Vector2i** ()

  Destructor.
- **Vector2i Cross** (const **Vector2i** &_pt) const

  Return the cross product of this vector and _pt.
- int **Distance** (const **Vector2i** &_pt) const

  Calc distance to the given point.
- bool **IsFinite** () const

  See if a point is finite (e.g., not nan)
- void **Normalize** ()

  Normalize the vector length.
- bool **operator!=** (const **Vector2i** &_v) const

  Equality operators.
- const **Vector2i operator**∗ (const **Vector2i** &_v) const

  Multiplication operator.
- const **Vector2i operator**∗ (int _v) const

  Multiplication operator.
- const **Vector2i** & **operator**∗**=** (const **Vector2i** &_v)

  Multiplication operators.
- const **Vector2i** & **operator**∗**=** (int _v)

  Multiplication operator.
- **Vector2i operator+** (const **Vector2i** &_v) const

  Addition operator.
- const **Vector2i** & **operator+=** (const **Vector2i** &_v)

  Addition assignment operator.
- **Vector2i operator-** (const **Vector2i** &_v) const

  Subtraction operator.
- const **Vector2i** & **operator-=** (const **Vector2i** &_v)

  Subtraction operators.
- const **Vector2i operator**/ (const **Vector2i** &_v) const

  Division operator.
- const **Vector2i operator**/ (int _v) const

  Division operator.
- const **Vector2i** & **operator**/**=** (const **Vector2i** &_v)

  Division operator.
- const **Vector2i** & **operator**/**=** (int _v)

  Division operator.
- **Vector2i** & **operator=** (const **Vector2i** &_v)

  Assignment operator.
- const **Vector2i** & **operator=** (int _value)

  Assignment operator.
- bool **operator==** (const **Vector2i** &_v) const

  Equality operator.

- int **operator[]** (unsigned int _index) const

  Array subscript operator.
- void **Set** (int _x, int _y)

  Set the contents of the vector.

Public Attributes

- int **x**

  x data
- int **y**

  y data

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::math::Vector2i** &_pt)

  Stream insertion operator.
- std::istream & **operator**>> (std::istream &_in, **gazebo::math::Vector2i** &_pt)

  Stream extraction operator.

10.180.1 Detailed Description

Generic integer x, y vector.

10.180.2 Constructor & Destructor Documentation

10.180.2.1 gazebo::math::Vector2i::Vector2i ( )

Constructor.

10.180.2.2 gazebo::math::Vector2i::Vector2i (const int & _x, const int & _y)

Constructor.

Parameters

in	*_x*	value along x
in	*_y*	value along y

10.180.2.3 gazebo::math::Vector2i::Vector2i (const Vector2i & _pt)

Copy onstructor.

Parameters

in	*_pt*	a point

10.180.2.4 virtual gazebo::math::Vector2i::~Vector2i ( ) [virtual]

Destructor.

10.180.3 Member Function Documentation

10.180.3.1 Vector2i gazebo::math::Vector2i::Cross (const Vector2i & _pt) const

Return the cross product of this vector and _pt.

Parameters

in	*_pt*	the other vector

Returns

the product

10.180.3.2 int gazebo::math::Vector2i::Distance (const Vector2i & _pt) const

Calc distance to the given point.

Parameters

in	*_pt*	a point

Returns

the distance

10.180.3.3 bool gazebo::math::Vector2i::IsFinite ( ) const

See if a point is finite (e.g., not nan)

Returns

the result

10.180.3.4 void gazebo::math::Vector2i::Normalize ( )

Normalize the vector length.

10.180.3.5 bool gazebo::math::Vector2i::operator!= (const Vector2i & _v) const

Equality operators.

Parameters

_v	the vector to compare with

Returns

true if component have different values, false otherwise

10.180.3.6 const Vector2i gazebo::math::Vector2i::operator∗ (const Vector2i & _v) const

Multiplication operator.

Remarks

this is an element wise multiplication

Parameters

in	_v	the vector

Returns

the result

10.180.3.7 const Vector2i gazebo::math::Vector2i::operator∗ (int _v) const

Multiplication operator.

Parameters

in	_v	the scaling factor

Returns

the result

10.180.3.8 const Vector2i& gazebo::math::Vector2i::operator∗= (const Vector2i & _v)

Multiplication operators.

Remarks

this is an element wise multiplication

Parameters

in	_v	the vector

Returns

this

10.180.3.9 const Vector2i& gazebo::math::Vector2i::operator*= (int _v)

Multiplication operator.

Parameters

in	_v	scaling factor

Returns

this

10.180.3.10 Vector2i gazebo::math::Vector2i::operator+ (const Vector2i & _v) const

Addition operator.

Parameters

in	_v	the vector to add

Returns

the sum vector

10.180.3.11 const Vector2i& gazebo::math::Vector2i::operator+= (const Vector2i & _v)

Addition assignment operator.

Parameters

in	_v	the vector to add

Returns

this

10.180.3.12 Vector2i gazebo::math::Vector2i::operator- (const Vector2i & _v) const

Subtraction operator.

Parameters

in	_v	the vector to substract

Returns

the result vector

10.180.3.13 const Vector2i& gazebo::math::Vector2i::operator-= (const Vector2i & _v)

Subtraction operators.

Parameters

in	_v	the vector to substract

Returns

this

10.180.3.14 const Vector2i gazebo::math::Vector2i::operator/ (const Vector2i & _v) const

Division operator.

Remarks

this is an element wise division.

Parameters

in	_v	the vector to divide

Returns

the result

10.180.3.15 const Vector2i gazebo::math::Vector2i::operator/ (int _v) const

Division operator.

Remarks

this is an element wise division.

Parameters

in	_v	the vector to divide

Returns

the result

10.180.3.16 const Vector2i& gazebo::math::Vector2i::operator/= (const Vector2i & _v)

Division operator.

Remarks

this is an element wise division.

Parameters

in	*_v*	the vector to divide

Returns

this

10.180.3.17 const Vector2i& gazebo::math::Vector2i::operator/= (int _v)

Division operator.

Remarks

this is an element wise division.

Parameters

in	*_v*	the vector to divide

Returns

this

10.180.3.18 Vector2i& gazebo::math::Vector2i::operator= (const Vector2i & _v)

Assignment operator.

Parameters

in	*_v*	the value

Returns

this

10.180.3.19 const Vector2i& gazebo::math::Vector2i::operator= (int _value)

Assignment operator.

Parameters

in	*_value*	the value for x and y

Returns

this

10.180.3.20 bool gazebo::math::Vector2i::operator== (const Vector2i & _v) const

Equality operator.

Parameters

_v	the vector to compare with

Returns

true if component have the same values, false otherwise

10.180.3.21 int gazebo::math::Vector2i::operator[] (unsigned int _index) const

Array subscript operator.

Parameters

in	*_index*	the array index

10.180.3.22 void gazebo::math::Vector2i::Set (int _x, int _y)

Set the contents of the vector.

Parameters

in	*_x*	value along x
in	*_y*	value along y

10.180.4 Friends And Related Function Documentation

10.180.4.1 std::ostream& operator<< (std::ostream & _out, const gazebo::math::Vector2i & _pt) [friend]

Stream insertion operator.

Parameters

in	*_out*	output stream
in	*pt*	**Vector2i** (p. 881) to output

Returns

the stream

10.180.4.2 std::istream& operator>> (std::istream & _in, gazebo::math::Vector2i & _pt) [friend]

Stream extraction operator.

Parameters

in	*_in*	input stream
in	*_pt*	**Vector3** (p. 890) to read values into

Returns

The stream

10.180.5 Member Data Documentation

10.180.5.1 int gazebo::math::Vector2i::x

x data

10.180.5.2 int gazebo::math::Vector2i::y

y data

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

- **Vector2i.hh**

10.181 gazebo::math::Vector3 Class Reference

The **Vector3** (p. 890) class represents the generic vector containing 3 elements.

`#include <math/gzmath.hh>`

Public Member Functions

- **Vector3** ()
  Constructor.
- **Vector3** (const double &_x, const double &_y, const double &_z)
  Constructor.
- **Vector3** (const **Vector3** &_v)
  Copy constructor.
- virtual ~**Vector3** ()
  Destructor.
- void **Correct** ()
  Corrects any nan values.
- **Vector3 Cross** (const **Vector3** &_pt) const
  Return the cross product of this vector and pt.
- double **Distance** (const **Vector3** &_pt) const
  Calc distance to the given point.

- double **Distance** (double _x, double _y, double _z) const

  Calc distance to the given point.
- double **Dot** (const **Vector3** &_pt) const

  Return the dot product of this vector and pt.
- bool **Equal** (const **Vector3** &_v) const

  Equality test.
- **Vector3 GetAbs** () const

  Get the absolute value of the vector.
- double **GetDistToLine** (const **Vector3** &_pt1, const **Vector3** &_pt2)

  Get distance to a line.
- double **GetLength** () const

  Returns the length (magnitude) of the vector \ return the length.
- double **GetMax** () const

  Get the maximum value in the vector.
- double **GetMin** () const

  Get the minimum value in the vector.
- **Vector3 GetPerpendicular** () const

  Return a vector that is perpendicular to this one.
- **Vector3 GetRounded** () const

  Get a rounded version of this vector.
- double **GetSquaredLength** () const

  Return the square of the length (magnitude) of the vector.
- double **GetSum** () const

  Return the sum of the values.
- bool **IsFinite** () const

  See if a point is finite (e.g., not nan)
- **Vector3 Normalize** ()

  Normalize the vector length.
- bool **operator!=** (const **Vector3** &_v) const

  Not equal to operator.
- **Vector3 operator*** (const **Vector3** &_p) const

  Multiplication operator.
- **Vector3 operator*** (double _v) const

  Multiplication operators.
- const **Vector3** & **operator*=** (const **Vector3** &_v)

  Multiplication operators.
- const **Vector3** & **operator*=** (double _v)

  Multiplication operator.
- **Vector3 operator+** (const **Vector3** &_v) const

  Addition operator.
- const **Vector3** & **operator+=** (const **Vector3** &_v)

  Addition assignment operator.
- **Vector3 operator-** () const

  Negation operator.
- **Vector3 operator-** (const **Vector3** &_pt) const

  Subtraction operators.
- const **Vector3** & **operator-=** (const **Vector3** &_pt)

Subtraction operators.

- const **Vector3 operator**/ (const **Vector3** &_pt) const

  Division operator.

- const **Vector3 operator**/ (double _v) const

  Division operator.

- const **Vector3** & **operator/=** (const **Vector3** &_pt)

  Division assignment operator.

- const **Vector3** & **operator/=** (double _v)

  Division operator.

- **Vector3** & **operator=** (const **Vector3** &_v)

  Assignment operator.

- **Vector3** & **operator=** (double _value)

  Assignment operator.

- bool **operator==** (const **Vector3** &_pt) const

  Equal to operator.

- double **operator[]** (unsigned int index) const

  [] operator

- **Vector3 Round** ()

  Round to near whole number, return the result.

- void **Round** (int _precision)

  Round all values to _precision decimal places.

- void **Set** (double _x=0, double _y=0, double _z=0)

  Set the contents of the vector.

- void **SetToMax** (const **Vector3** &_v)

  Set this vector's components to the maximum of itself and the passed in vector.

- void **SetToMin** (const **Vector3** &_v)

  Set this vector's components to the minimum of itself and the passed in vector.

Static Public Member Functions

- static **Vector3 GetNormal** (const **Vector3** &_v1, const **Vector3** &_v2, const **Vector3** &_v3)

  Get a normal vector to a triangle.

Public Attributes

- double **x**

  X location.

- double **y**

  Y location.

- double **z**

  Z location.

Static Public Attributes

- static const **Vector3 One**

  math::Vector3(1, 1, 1)
- static const **Vector3 UnitX**

  math::Vector3(1, 0, 0)
- static const **Vector3 UnitY**

  math::Vector3(0, 1, 0)
- static const **Vector3 UnitZ**

  math::Vector3(0, 0, 1)
- static const **Vector3 Zero**

  math::Vector3(0, 0, 0)

Friends

- **Vector3 operator**∗ (double _s, const **Vector3** &_v)

  Multiplication operators.
- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::math::Vector3** &_pt)

  Stream insertion operator.
- std::istream & **operator**>> (std::istream &_in, **gazebo::math::Vector3** &_pt)

  Stream extraction operator.

10.181.1 Detailed Description

The **Vector3** (p. 890) class represents the generic vector containing 3 elements.

Since it's commonly used to keep coordinate system related information, its elements are labeled by x, y, z.

10.181.2 Constructor & Destructor Documentation

10.181.2.1 gazebo::math::Vector3::Vector3 ( )

Constructor.

Referenced by operator-().

10.181.2.2 gazebo::math::Vector3::Vector3 (const double & _x, const double & _y, const double & _z)

Constructor.

Parameters

in	*_x*	value along x
in	*_y*	value along y
in	*_z*	value along z

10.181.2.3 gazebo::math::Vector3::Vector3 (const Vector3 & _v)

Copy constructor.

Parameters

in	_v	a vector

10.181.2.4 virtual gazebo::math::Vector3::~Vector3 ( ) [virtual]

Destructor.

10.181.3 Member Function Documentation

10.181.3.1 void gazebo::math::Vector3::Correct ( ) [inline]

Corrects any nan values.

References x, y, and z.

Referenced by gazebo::math::Pose::Correct().

10.181.3.2 Vector3 gazebo::math::Vector3::Cross (const Vector3 & _pt) const

Return the cross product of this vector and pt.

Returns

the product

10.181.3.3 double gazebo::math::Vector3::Distance (const Vector3 & _pt) const

Calc distance to the given point.

Parameters

in	_pt	the point

Returns

the distance

10.181.3.4 double gazebo::math::Vector3::Distance (double _x, double _y, double _z) const

Calc distance to the given point.

Parameters

in	_x	value along x
in	_y	value along y
in	_z	value along z

Returns

the distance

10.181.3.5 double gazebo::math::Vector3::Dot (const Vector3 & _pt) const

Return the dot product of this vector and pt.

Returns

the product

10.181.3.6 bool gazebo::math::Vector3::Equal (const Vector3 & _v) const

Equality test.

Remarks

This is equivalent to the == operator

Parameters

in	_v	the other vector

Returns

true if the 2 vectors have the same values, false otherwise

10.181.3.7 Vector3 gazebo::math::Vector3::GetAbs ( ) const

Get the absolute value of the vector.

Returns

a vector with positive elements

10.181.3.8 double gazebo::math::Vector3::GetDistToLine (const Vector3 & _pt1, const Vector3 & _pt2)

Get distance to a line.

Parameters

in	_pt1	first point on the line
in	_pt2	second point on the line

Returns

the minimum distance from this point to the line

10.181.3.9 double gazebo::math::Vector3::GetLength ( ) const

Returns the length (magnitude) of the vector \ return the length.

10.181.3.10 double gazebo::math::Vector3::GetMax ( ) const

Get the maximum value in the vector.

Returns

the maximum element

10.181.3.11 double gazebo::math::Vector3::GetMin ( ) const

Get the minimum value in the vector.

Returns

the minimum element

10.181.3.12 static Vector3 gazebo::math::Vector3::GetNormal (const Vector3 & *_v1*, const Vector3 & *_v2*, const Vector3 & *_v3*) [static]

Get a normal vector to a triangle.

Parameters

in	*_v1*	first vertex of the triangle
in	*_v2*	second vertex
in	*_v3*	third vertex

Returns

the normal

10.181.3.13 Vector3 gazebo::math::Vector3::GetPerpendicular ( ) const

Return a vector that is perpendicular to this one.

Returns

an orthogonal vector

10.181.3.14 Vector3 gazebo::math::Vector3::GetRounded ( ) const

Get a rounded version of this vector.

Returns

a rounded vector

10.181.3.15 double gazebo::math::Vector3::GetSquaredLength ( ) const

Return the square of the length (magnitude) of the vector.

Returns

the squared length

10.181.3.16 double gazebo::math::Vector3::GetSum ( ) const

Return the sum of the values.

Returns

the sum

10.181.3.17 bool gazebo::math::Vector3::IsFinite ( ) const

See if a point is finite (e.g., not nan)

10.181.3.18 Vector3 gazebo::math::Vector3::Normalize ( )

Normalize the vector length.

Returns

unit length vector

10.181.3.19 bool gazebo::math::Vector3::operator!= (const Vector3 & _v) const

Not equal to operator.

Parameters

in	_v	The vector to compare against

Returns

true if each component is equal withing a default tolerence (1e-6), false otherwise

10.181.3.20 Vector3 gazebo::math::Vector3::operator* (const Vector3 & _p) const

Multiplication operator.

Remarks

this is an element wise multiplication, not a cross product

Parameters

in	_v	

10.181.3.21 Vector3 gazebo::math::Vector3::operator∗ (double _v) const

Multiplication operators.

Parameters

in	_v	the scaling factor

Returns

a scaled vector

10.181.3.22 const Vector3& gazebo::math::Vector3::operator∗= (const Vector3 & _v)

Multiplication operators.

Remarks

this is an element wise multiplication, not a cross product

Parameters

in	_v	a vector

Returns

this

10.181.3.23 const Vector3& gazebo::math::Vector3::operator∗= (double _v)

Multiplication operator.

Parameters

in	_v	scaling factor

Returns

this

10.181.3.24 Vector3 gazebo::math::Vector3::operator+ (const Vector3 & _v) const

Addition operator.

Parameters

in	*_v*	vector to add

Returns

the sum vector

10.181.3.25 const Vector3& gazebo::math::Vector3::operator+= (const Vector3 & _v)

Addition assignment operator.

Parameters

in	*_v*	vector to add

10.181.3.26 Vector3 gazebo::math::Vector3::operator- ( ) const [inline]

Negation operator.

Returns

negative of this vector

References Vector3(), x, y, and z.

10.181.3.27 Vector3 gazebo::math::Vector3::operator- (const Vector3 & _pt) const [inline]

Subtraction operators.

Parameters

in	*_pt*	a vector to substract

Returns

a vector

References Vector3(), x, y, and z.

10.181.3.28 const Vector3& gazebo::math::Vector3::operator-= (const Vector3 & _pt)

Subtraction operators.

Parameters

in	*_pt*	subtrahend

10.181.3.29 const Vector3 gazebo::math::Vector3::operator/ (const Vector3 & _pt) const

Division operator.

[in] _pt the vector divisor

Remarks

this is an element wise division

Returns

a vector

10.181.3.30 const Vector3 gazebo::math::Vector3::operator/ (double _v) const

Division operator.

Remarks

this is an element wise division

Returns

a vector

10.181.3.31 const Vector3& gazebo::math::Vector3::operator/= (const Vector3 & _pt)

Division assignment operator.

[in] _pt the vector divisor

Remarks

this is an element wise division

Returns

a vector

10.181.3.32 const Vector3& gazebo::math::Vector3::operator/= (double _v)

Division operator.

Remarks

this is an element wise division

Returns

this

10.181.3.33 Vector3& gazebo::math::Vector3::operator= (const Vector3 & _v)

Assignment operator.

Parameters

in	_v	a new value

Returns

this

10.181.3.34 Vector3& gazebo::math::Vector3::operator= (double _value)

Assignment operator.

Parameters

in	_value	assigned to all elements

Returns

this

10.181.3.35 bool gazebo::math::Vector3::operator== (const Vector3 & _pt) const

Equal to operator.

Parameters

in	_pt	The vector to compare against

Returns

true if each component is equal withing a default tolerence (1e-6), false otherwise

10.181.3.36 double gazebo::math::Vector3::operator[] (unsigned int index) const

[] operator

10.181.3.37 Vector3 gazebo::math::Vector3::Round ( )

Round to near whole number, return the result.

Returns

the result

10.181.3.38 void gazebo::math::Vector3::Round (int *_precision*)

Round all values to _precision decimal places.

Parameters

in	*_precision*	the decimal places

10.181.3.39 void gazebo::math::Vector3::Set (double *_x* = 0, double *_y* = 0, double *_z* = 0) [inline]

Set the contents of the vector.

Parameters

in	*_x*	value along x
in	*_y*	value along y
in	*_z*	value aling z

References x, y, and z.

10.181.3.40 void gazebo::math::Vector3::SetToMax (const Vector3 & *_v*)

Set this vector's components to the maximum of itself and the passed in vector.

Parameters

in	*_v*	the maximum clamping vector

10.181.3.41 void gazebo::math::Vector3::SetToMin (const Vector3 & *_v*)

Set this vector's components to the minimum of itself and the passed in vector.

Parameters

in	*_v*	the minimum clamping vector

10.181.4 Friends And Related Function Documentation

10.181.4.1 Vector3 operator* (double *_s*, const Vector3 & *_v*) [friend]

Multiplication operators.

Parameters

in	*_s*	the scaling factor
in	*_v*	input vector

Returns

a scaled vector

10.181.4.2 std::ostream& operator<< (std::ostream & _out, const gazebo::math::Vector3 & _pt) [friend]

Stream insertion operator.

Parameters

_out	output stream
_pt	**Vector3** (p. 890) to output

Returns

the stream

10.181.4.3 std::istream& operator>> (std::istream & _in, gazebo::math::Vector3 & _pt) [friend]

Stream extraction operator.

Parameters

_in	input stream
_pt	vector3 to read values into

Returns

the stream

10.181.5 Member Data Documentation

10.181.5.1 const Vector3 gazebo::math::Vector3::One [static]

math::Vector3(1, 1, 1)

10.181.5.2 const Vector3 gazebo::math::Vector3::UnitX [static]

math::Vector3(1, 0, 0)

10.181.5.3 const Vector3 gazebo::math::Vector3::UnitY [static]

math::Vector3(0, 1, 0)

10.181.5.4 const Vector3 gazebo::math::Vector3::UnitZ [static]

math::Vector3(0, 0, 1)

10.181.5.5 double gazebo::math::Vector3::x

X location.

Referenced by gazebo::math::Pose::CoordPositionSub(), Correct(), operator-(), gazebo::math::Quaternion::RotateVector(), and Set().

10.181.5.6 double gazebo::math::Vector3::y

Y location.

Referenced by gazebo::math::Pose::CoordPositionSub(), Correct(), operator-(), gazebo::math::Quaternion::RotateVector(), and Set().

10.181.5.7 double gazebo::math::Vector3::z

Z location.

Referenced by gazebo::math::Pose::CoordPositionSub(), Correct(), operator-(), gazebo::math::Quaternion::RotateVector(), and Set().

10.181.5.8 const Vector3 gazebo::math::Vector3::Zero [static]

math::Vector3(0, 0, 0)

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

- **Vector3.hh**

10.182 gazebo::math::Vector4 Class Reference

double Generic x, y, z, w vector

`#include <math/gzmath.hh>`

Public Member Functions

- **Vector4** ()

  Constructor.
- **Vector4** (const double &_x, const double &_y, const double &_z, const double &_w)

  Constructor with component values.
- **Vector4** (const **Vector4** &_v)

  Copy constructor.
- virtual ~**Vector4** ()

  Destructor.
- double **Distance** (const **Vector4** &_pt) const

  Calc distance to the given point.
- double **GetLength** () const

  Returns the length (magnitude) of the vector.
- double **GetSquaredLength** () const

  Return the square of the length (magnitude) of the vector.
- bool **IsFinite** () const

  See if a point is finite (e.g., not nan)
- void **Normalize** ()

  Normalize the vector length.
- bool **operator!=** (const **Vector4** &_pt) const

Not equal to operator.

- const **Vector4 operator**∗ (const **Vector4** &_pt) const

  Multiplication operator.
- const **Vector4 operator**∗ (const **Matrix4** &_m) const

  Matrix multiplication operator.
- const **Vector4 operator**∗ (double _v) const

  Multiplication operators.
- const **Vector4** & **operator**∗**=** (const **Vector4** &_pt)

  Multiplication assignment operator.
- const **Vector4** & **operator**∗**=** (double _v)

  Multiplication assignment operator.
- **Vector4 operator+** (const **Vector4** &_v) const

  Addition operator.
- const **Vector4** & **operator+=** (const **Vector4** &_v)

  Addition operator.
- **Vector4 operator-** (const **Vector4** &_v) const

  Subtraction operator.
- const **Vector4** & **operator-=** (const **Vector4** &_v)

  Subtraction assigment operators.
- const **Vector4 operator**/ (const **Vector4** &_v) const

  Division assignment operator.
- const **Vector4 operator**/ (double _v) const

  Division assignment operator.
- const **Vector4** & **operator**/**=** (const **Vector4** &_v)

  Division assignment operator.
- const **Vector4** & **operator**/**=** (double _v)

  Division operator.
- **Vector4** & **operator=** (const **Vector4** &_v)

  Assignment operator.
- **Vector4** & **operator=** (double _value)

  Assignment operator.
- bool **operator==** (const **Vector4** &_pt) const

  Equal to operator.
- double **operator[]** (unsigned int _index) const

  Array subscript operator.
- void **Set** (double _x=0, double _y=0, double _z=0, double _w=0)

  Set the contents of the vector.

Public Attributes

- double **w**

  W value.
- double **x**

  X value.
- double **y**

  Y value.
- double **z**

  Z value.

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::math::Vector4** &_pt)

  Stream insertion operator.
- std::istream & **operator**>> (std::istream &_in, **gazebo::math::Vector4** &_pt)

  Stream extraction operator.

10.182.1 Detailed Description

double Generic x, y, z, w vector

10.182.2 Constructor & Destructor Documentation

10.182.2.1 gazebo::math::Vector4::Vector4 ( )

Constructor.

10.182.2.2 gazebo::math::Vector4::Vector4 (const double & *_x*, const double & *_y*, const double & *_z*, const double & *_w*)

Constructor with component values.

Parameters

in	*_x*	value along x axis
in	*_y*	value along y axis
in	*_z*	value along z axis
in	*_w*	value along w axis

10.182.2.3 gazebo::math::Vector4::Vector4 (const **Vector4** & *_v*)

Copy constructor.

Parameters

in	*_v*	vector

10.182.2.4 virtual gazebo::math::Vector4::~Vector4 ( ) [virtual]

Destructor.

10.182.3 Member Function Documentation

10.182.3.1 double gazebo::math::Vector4::Distance (const **Vector4** & *_pt*) const

Calc distance to the given point.

Parameters

in	*_pt*	the point

Returns

the distance

10.182.3.2 double gazebo::math::Vector4::GetLength ( ) const

Returns the length (magnitude) of the vector.

10.182.3.3 double gazebo::math::Vector4::GetSquaredLength ( ) const

Return the square of the length (magnitude) of the vector.

Returns

the length

10.182.3.4 bool gazebo::math::Vector4::IsFinite ( ) const

See if a point is finite (e.g., not nan)

Returns

true if finite, false otherwise

10.182.3.5 void gazebo::math::Vector4::Normalize ( )

Normalize the vector length.

10.182.3.6 bool gazebo::math::Vector4::operator!= (const Vector4 & *_pt*) const

Not equal to operator.

Parameters

in	*_pt*	the other vector

Returns

true if each component is equal withing a default tolerence (1e-6), false otherwise

10.182.3.7 const Vector4 gazebo::math::Vector4::operator* (const Vector4 & *_pt*) const

Multiplication operator.

Remarks

Performs element wise multiplication, which has limited use.

Parameters

in	*_pt*	another vector

Returns

result vector

10.182.3.8 const Vector4 gazebo::math::Vector4::operator∗ (const Matrix4 & *_m*) const

Matrix multiplication operator.

Parameters

in	*_m*	matrix

Returns

the vector multiplied by _m

10.182.3.9 const Vector4 gazebo::math::Vector4::operator∗ (double *_v*) const

Multiplication operators.

Parameters

in	*_v*	scaling factor

Returns

a scaled vector

10.182.3.10 const Vector4& gazebo::math::Vector4::operator∗= (const Vector4 & *_pt*)

Multiplication assignment operator.

Remarks

Performs element wise multiplication, which has limited use.

Parameters

in	*_pt*	a vector

Returns

this

10.182.3.11 const Vector4& gazebo::math::Vector4::operator*= (double _v)

Multiplication assignment operator.

Parameters

in	_v	scaling factor

Returns

this

10.182.3.12 Vector4 gazebo::math::Vector4::operator+ (const Vector4 & _v) const

Addition operator.

Parameters

in	_v	the vector to add

Returns

a sum vector

10.182.3.13 const Vector4& gazebo::math::Vector4::operator+= (const Vector4 & _v)

Addition operator.

Parameters

in	_v	the vector to add

Returns

this vector

10.182.3.14 Vector4 gazebo::math::Vector4::operator- (const Vector4 & _v) const

Subtraction operator.

Parameters

in	_v	the vector to substract

Returns

a vector

10.182.3.15 const Vector4& gazebo::math::Vector4::operator-= (const Vector4 & _v)

Subtraction assigment operators.

Parameters

in	_v	the vector to substract

Returns

this vector

10.182.3.16 const Vector4 gazebo::math::Vector4::operator/ (const Vector4 & _v) const

Division assignment operator.

Remarks

Performs element wise division, which has limited use.

Parameters

in	_v	the vector to perform element wise division with

Returns

a result vector

10.182.3.17 const Vector4 gazebo::math::Vector4::operator/ (double _v) const

Division assignment operator.

Remarks

Performs element wise division, which has limited use.

Parameters

in	_pt	another vector

Returns

a result vector

10.182.3.18 const Vector4& gazebo::math::Vector4::operator/= (const Vector4 & _v)

Division assignment operator.

Remarks

Performs element wise division, which has limited use.

Parameters

in	_v	the vector to perform element wise division with

Returns

this

10.182.3.19 const Vector4& gazebo::math::Vector4::operator/= (double _v)

Division operator.

Parameters

in	_v	scaling factor

Returns

a vector

10.182.3.20 Vector4& gazebo::math::Vector4::operator= (const Vector4 & _v)

Assignment operator.

Parameters

in	_v	the vector

Returns

a reference to this vector

10.182.3.21 Vector4& gazebo::math::Vector4::operator= (double _value)

Assignment operator.

Parameters

in	_value	

10.182.3.22 bool gazebo::math::Vector4::operator== (const Vector4 & _pt) const

Equal to operator.

Parameters

in	*_pt*	the other vector

Returns

true if each component is equal withing a default tolerence (1e-6), false otherwise

10.182.3.23 double gazebo::math::Vector4::operator[] (unsigned int _index) const

Array subscript operator.

Parameters

in	*_index*	

10.182.3.24 void gazebo::math::Vector4::Set (double _x = 0, double _y = 0, double _z = 0, double _w = 0)

Set the contents of the vector.

Parameters

in	*_x*	value along x axis
in	*_y*	value along y axis
in	*_z*	value along z axis
in	*_w*	value along w axis

10.182.4 Friends And Related Function Documentation

10.182.4.1 std::ostream& operator<< (std::ostream & _out, const gazebo::math::Vector4 & _pt) [friend]

Stream insertion operator.

Parameters

in	*_out*	output stream
in	*_pt*	**Vector4** (p. 904) to output

Returns

The stream

10.182.4.2 std::istream& operator>> (std::istream & _in, gazebo::math::Vector4 & _pt) [friend]

Stream extraction operator.

Parameters

in	*_in*	input stream
in	*_pt*	**Vector4** (p. 904) to read values into

Returns

the stream

10.182.5 Member Data Documentation

10.182.5.1 double gazebo::math::Vector4::w

W value.

10.182.5.2 double gazebo::math::Vector4::x

X value.

10.182.5.3 double gazebo::math::Vector4::y

Y value.

10.182.5.4 double gazebo::math::Vector4::z

Z value.

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

- **Vector4.hh**

10.183 gazebo::common::Video Class Reference

Handle video encoding and decoding using libavcodec.

`#include <common/common.hh>`

Public Member Functions

- **Video** ()

  Constructor.
- virtual ∼**Video** ()

  Destructor.
- int **GetHeight** () const

  Get the height of the video in pixels.
- bool **GetNextFrame** (unsigned char ∗∗_buffer)

  Get the next frame of the video.
- int **GetWidth** () const

Get the width of the video in pixels.

- bool **Load** (const std::string &_filename)

  Load a video file.

10.183.1 Detailed Description

Handle video encoding and decoding using libavcodec.

10.183.2 Constructor & Destructor Documentation

10.183.2.1 gazebo::common::Video::Video ( )

Constructor.

10.183.2.2 virtual gazebo::common::Video::~Video ( ) [virtual]

Destructor.

10.183.3 Member Function Documentation

10.183.3.1 int gazebo::common::Video::GetHeight ( ) const

Get the height of the video in pixels.

Returns

the height

10.183.3.2 bool gazebo::common::Video::GetNextFrame (unsigned char ** *_buffer*)

Get the next frame of the video.

Parameters

out	*_img*	**Image** (p. 379) in which the frame is stored

Returns

false if HAVE_FFMPEG is not defined, true otherwise

10.183.3.3 int gazebo::common::Video::GetWidth ( ) const

Get the width of the video in pixels.

Returns

the width

10.183.3.4 bool gazebo::common::Video::Load (const std::string & *_filename*)

Load a video file.

Parameters

in	*_filename*	Full path of the video file

Returns

false if HAVE_FFMPEG is not defined or if a video stream can't be found

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

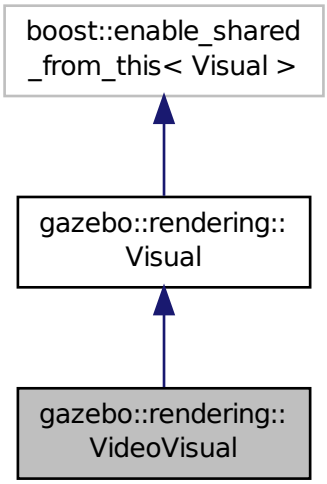
- **Video.hh**

10.184 gazebo::rendering::VideoVisual Class Reference

A visual element that displays a video as a texture.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::VideoVisual:



Public Member Functions

- **VideoVisual** (const std::string &_name, **VisualPtr** _parent)

  Constructor.
- virtual ~**VideoVisual** ()

  Destructor.

Additional Inherited Members

10.184.1 Detailed Description

A visual element that displays a video as a texture.

10.184.2 Constructor & Destructor Documentation

10.184.2.1 gazebo::rendering::VideoVisual::VideoVisual (const std::string & *_name,* VisualPtr *_parent*)

Constructor.

Parameters

in	*_name*	Name of the video visual.
in	*_parent*	Parent of the video visual.

10.184.2.2 virtual gazebo::rendering::VideoVisual::~VideoVisual ( ) [virtual]

Destructor.

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

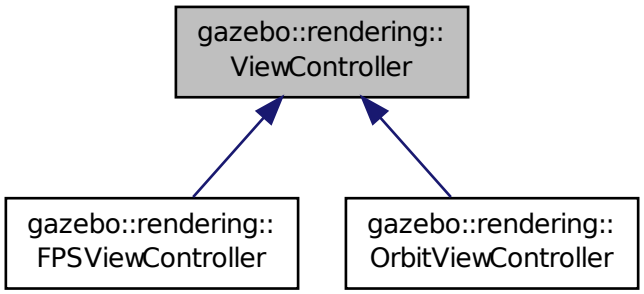
- **VideoVisual.hh**

10.185 gazebo::rendering::ViewController Class Reference

Base class for view controllers.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::ViewController:



Public Member Functions

- **ViewController** (**UserCameraPtr** _camera)

  Constructor.
- virtual ~**ViewController** ()

  Destructor.
- std::string **GetTypeString** () const

  Get the type of view controller.
- virtual void **HandleKeyPressEvent** (const std::string &_key)=0

  Handle a key press event.
- virtual void **HandleKeyReleaseEvent** (const std::string &_key)=0

  Handle a key release event.
- virtual void **HandleMouseEvent** (const **common::MouseEvent** &_event)=0

  Handle a mouse event.
- virtual void **Init** ()=0

  Initialize the view controller.
- virtual void **Init** (const **math::Vector3** &_focalPoint)

  Initialize with a focus point.
- void **SetEnabled** (bool _value)

  Set whether the controller is enabled.
- virtual void **Update** ()=0

  *Update the controller, which should update the position of the **Camera*** (p. *166*).

Protected Attributes

- **UserCameraPtr camera**

  Pointer to the camera to control.
- bool **enabled**

  True if enabled.
- std::string **typeString**

  Type of view controller.

10.185.1 Detailed Description

Base class for view controllers.

10.185.2 Constructor & Destructor Documentation

10.185.2.1 gazebo::rendering::ViewController::ViewController (UserCameraPtr *_camera*)

Constructor.

Parameters

in	*_camera*	The user camera to controll.

10.185.2.2 virtual gazebo::rendering::ViewController::~ViewController ( ) [virtual]

Destructor.

10.185.3 Member Function Documentation

10.185.3.1 std::string gazebo::rendering::ViewController::GetTypeString ( ) const

Get the type of view controller.

Returns

The view controller type string.

10.185.3.2 virtual void gazebo::rendering::ViewController::HandleKeyPressEvent (const std::string & *_key*) [pure virtual]

Handle a key press event.

Parameters

in	*_key*	The key that was pressed.

Implemented in **gazebo::rendering::OrbitViewController** (p. 586), and **gazebo::rendering::FPSViewController** (p. 333).

10.185.3.3 virtual void gazebo::rendering::ViewController::HandleKeyReleaseEvent (const std::string & *_key*) [pure virtual]

Handle a key release event.

Parameters

in	*_key*	The key that was released.

Implemented in **gazebo::rendering::OrbitViewController** (p. 587), and **gazebo::rendering::FPSViewController** (p. 333).

10.185.3.4 virtual void gazebo::rendering::ViewController::HandleMouseEvent (const common::MouseEvent & *_event*) [pure virtual]

Handle a mouse event.

Parameters

in	*_event*	The mouse position.

Implemented in **gazebo::rendering::OrbitViewController** (p. 587), and **gazebo::rendering::FPSViewController** (p. 334).

10.185.3.5 virtual void gazebo::rendering::ViewController::Init ( ) [pure virtual]

Initialize the view controller.

Implemented in **gazebo::rendering::OrbitViewController** (p. 587), and **gazebo::rendering::FPSViewController** (p. 334).

10.185.3.6 virtual void gazebo::rendering::ViewController::Init (const math::Vector3 & *_focalPoint*) [virtual]

Initialize with a focus point.

Parameters

in	*_focalPoint*	The point to look at.

Reimplemented in **gazebo::rendering::OrbitViewController** (p. 587).

10.185.3.7 void gazebo::rendering::ViewController::SetEnabled (bool *_value*)

Set whether the controller is enabled.

Parameters

in	*_value*	True if the controller is enabled.

10.185.3.8 virtual void gazebo::rendering::ViewController::Update ( ) [pure virtual]

Update the controller, which should update the position of the **Camera** (p. 166).

Implemented in **gazebo::rendering::OrbitViewController** (p. 588), and **gazebo::rendering::FPSViewController** (p. 334).

10.185.4 Member Data Documentation

10.185.4.1 UserCameraPtr gazebo::rendering::ViewController::camera [protected]

Pointer to the camera to control.

10.185.4.2 bool gazebo::rendering::ViewController::enabled [protected]

True if enabled.

10.185.4.3 std::string gazebo::rendering::ViewController::typeString [protected]

Type of view controller.

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

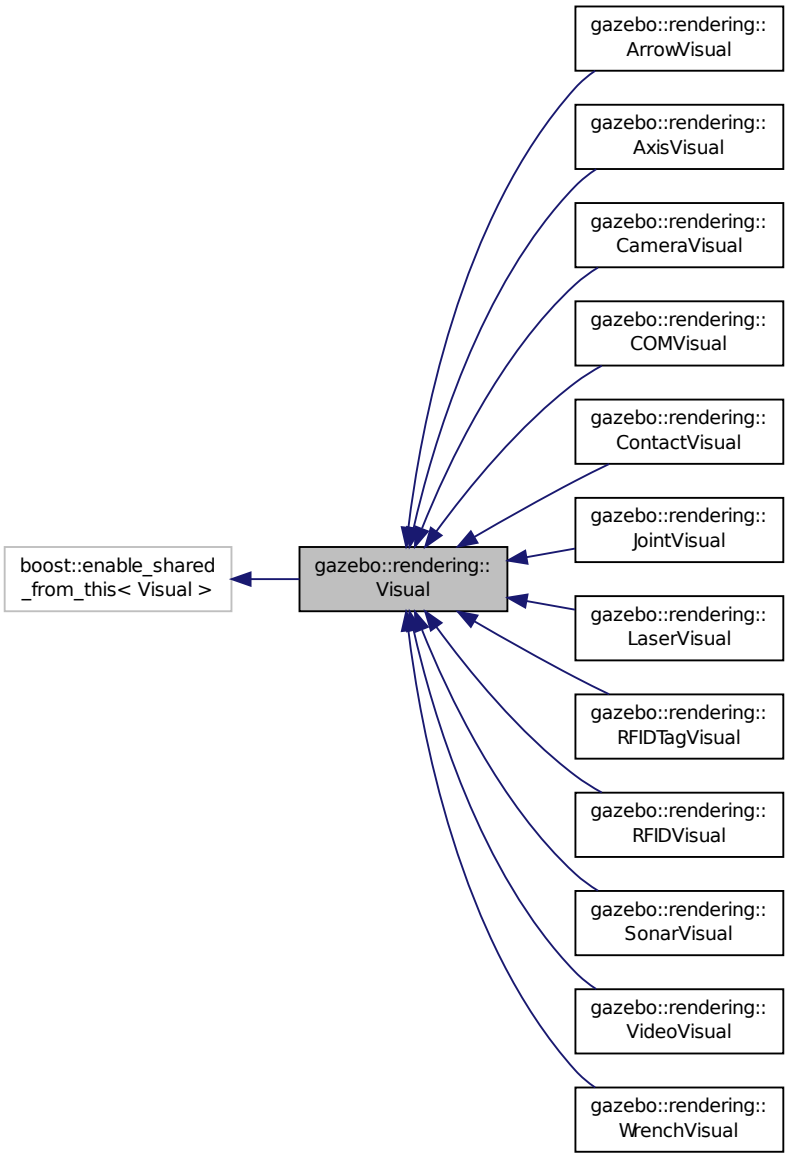
- **ViewController.hh**

10.186 gazebo::rendering::Visual Class Reference

A renderable object.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::Visual:



Public Member Functions

- **Visual** (const std::string &_name, **VisualPtr** _parent, bool _useRTShader=true)

Constructor.

- **Visual** (const std::string &_name, **ScenePtr** _scene, bool _useRTShader=true)

  Constructor.
- virtual ~**Visual** ()

  Destructor.
- void **AttachAxes** ()

  Attach visualization axes.
- void **AttachLineVertex** (**DynamicLines** *_line, unsigned int _index)

  Attach a vertex of a line to the position of the visual.
- Ogre::MovableObject * **AttachMesh** (const std::string &_meshName, const std::string &_subMesh="", bool _centerSubmesh=false, const std::string &_objName="")

  Attach a mesh to this visual by name.
- void **AttachObject** (Ogre::MovableObject *_obj)

  Attach a renerable object to the visual.
- void **AttachVisual** (**VisualPtr** _vis)

  Attach a visual to this visual.
- void **ClearParent** ()

  Clear parents.
- **VisualPtr Clone** (const std::string &_name, **VisualPtr** _newParent)

  Clone the visual with a new name.
- **DynamicLines** * **CreateDynamicLine** (**RenderOpType** _type=**RENDERING_LINE_STRIP**)

  Add a line to the visual.
- void **DeleteDynamicLine** (**DynamicLines** *_line)

  Delete a dynamic line.
- void **DetachObjects** ()

  Detach all objects.
- void **DetachVisual** (**VisualPtr** _vis)

  Detach a visual.
- void **DetachVisual** (const std::string &_name)

  Detach a visual.
- void **DisableTrackVisual** ()

  Disable tracking of a visual.
- void **EnableTrackVisual** (**VisualPtr** _vis)

  Set one visual to track/follow another.
- void **Fini** ()

  Helper for the destructor.
- unsigned int **GetAttachedObjectCount** () const

  Return the number of attached movable objects.
- **math::Box GetBoundingBox** () const

  Get the bounding box for the visual.
- **VisualPtr GetChild** (unsigned int _index)

  Get an attached visual based on an index.
- unsigned int **GetChildCount** ()

  Get the number of attached visuals.
- std::string **GetMaterialName** () const

  Get the name of the material.
- std::string **GetMeshName** () const

The name of the mesh set in the visual's SDF.

- std::string **GetName** () const

  Get the name of the visual.
- std::string **GetNormalMap** () const

  Get the normal map.
- **VisualPtr GetParent** () const

  Get the parent visual, if one exists.
- **math::Pose GetPose** () const

  Get the pose of the visual.
- **math::Vector3 GetPosition** () const

  Get the position of the visual.
- **VisualPtr GetRootVisual** ()

  Get the root visual.
- **math::Quaternion GetRotation** () const

  Get the rotation of the visual.
- **math::Vector3 GetScale** ()

  Get the scale.
- **ScenePtr GetScene** () const

  Get current.
- Ogre::SceneNode * **GetSceneNode** () const

  Return the scene Node of this visual entity.
- std::string **GetShaderType** () const

  Get the shader type.
- std::string **GetSubMeshName** () const

  Get the name of the sub mesh set in the visual's SDF.
- float **GetTransparency** ()

  Get the transparency.
- uint32_t **GetVisibilityFlags** ()

  Get visibility flags for this visual and all children.
- bool **GetVisible** () const

  Get whether the visual is visible.
- **math::Pose GetWorldPose** () const

  Get the global pose of the node.
- bool **HasAttachedObject** (const std::string &_name)

  Returns true if an object with _name is attached.
- void **Init** ()

  Helper for the contructor.
- void **InsertMesh** (const std::string &_meshName, const std::string &_subMesh="", bool _centerSubmesh=false)

  *Insert a mesh into **Ogre** (p. 110).*
- bool **IsPlane** () const

  Return true if the visual is a plane.
- bool **IsStatic** () const

  Return true if the visual is a static geometry.
- void **Load** (**sdf::ElementPtr** _sdf)

  Load the visual with a set of parameters.
- virtual void **Load** ()

  Load the visual with default parameters.

- void **LoadFromMsg** (ConstVisualPtr &_msg)

  Load from a message.
- void **LoadPlugin** (const std::string &_filename, const std::string &_name, **sdf::ElementPtr** _sdf)

  Load a plugin.
- void **MakeStatic** ()

  Make the visual objects static renderables.
- void **MoveToPosition** (const **math::Pose** &_pose, double _time)

  Move to a pose and over a given time.
- void **MoveToPositions** (const std::vector< **math::Pose** > &_pts, double _time, boost::function< void()> _onComplete=**NULL**)

  Move to a series of pose and over a given time.
- void **RemovePlugin** (const std::string &_name)

  Remove a running plugin.
- void **SetAmbient** (const **common::Color** &_color)

  Set the ambient color of the visual.
- void **SetCastShadows** (bool _shadows)

  Set whether the visual should cast shadows.
- void **SetDiffuse** (const **common::Color** &_color)

  Set the diffuse color of the visual.
- virtual void **SetEmissive** (const **common::Color** &_color)

  Set the emissive value.
- void **SetHighlighted** (bool _highlighted)

  Set the visual to be visually highlighted.
- void **SetMaterial** (const std::string &_materialName, bool _unique=true)

  Set the material.
- void **SetName** (const std::string &_name)

  Set the name of the visual.
- void **SetNormalMap** (const std::string &_nmap)

  Set the normal map.
- void **SetPose** (const **math::Pose** &_pose)

  Set the pose of the visual.
- void **SetPosition** (const **math::Vector3** &_pos)

  Set the position of the visual.
- void **SetRibbonTrail** (bool _value, const **common::Color** &_initialColor, const **common::Color** &_changeColor)

  True on or off a ribbon trail.
- void **SetRotation** (const **math::Quaternion** &_rot)

  Set the rotation of the visual.
- void **SetScale** (const **math::Vector3** &_scale)

  Set the scale.
- void **SetScene** (**ScenePtr** _scene)

  Set current scene.
- void **SetShaderType** (const std::string &_type)

  Set the shader type for the visual's material.
- void **SetSkeletonPose** (const msgs::PoseAnimation &_pose)

  Set animation skeleton pose.
- void **SetSpecular** (const **common::Color** &_color)

Set the specular color of the visual.

- void **SetTransparency** (float _trans)

  Set the transparency.

- void **SetVisibilityFlags** (uint32_t _flags)

  Set visibility flags for this visual and all children.

- void **SetVisible** (bool _visible, bool _cascade=true)

  Set whether the visual is visible.

- void **SetWireframe** (bool _show)

  Enable or disable wireframe for this visual.

- void **SetWorldPose** (const **math::Pose** _pose)

  Set the world pose of the visual.

- void **SetWorldPosition** (const **math::Vector3** &_pos)

  Set the world linear position of the visual.

- void **SetWorldRotation** (const **math::Quaternion** &_rot)

  Set the world orientation of the visual.

- void **ShowBoundingBox** ()

  Display the bounding box visual.

- void **ShowCollision** (bool _show)

  Display the collision visuals.

- void **ShowCOM** (bool _show)

  Display Center of Mass visuals.

- void **ShowJoints** (bool _show)

  Display joint visuals.

- void **ShowSkeleton** (bool _show)

  Display the skeleton visuals.

- void **ToggleVisible** ()

  Toggle whether this visual is visible.

- void **Update** ()

  Update the visual.

- void **UpdateFromMsg** (ConstVisualPtr &_msg)

  Update a visual based on a message.

Static Public Member Functions

- static void **InsertMesh** (const **common::Mesh** *_mesh, const std::string &_subMesh="", bool _centerSubmesh=false)

  *Insert a mesh into **Ogre** (p. 110).*

Protected Attributes

- **VisualPtr parent**

  Parent visual.

- **ScenePtr scene**

  Pointer to the visual's scene.

- Ogre::SceneNode * **sceneNode**

  *Pointer to the visual's scene node in **Ogre** (p. 110).*

10.186.1 Detailed Description

A renderable object.

10.186.2 Constructor & Destructor Documentation

10.186.2.1 gazebo::rendering::Visual::Visual (const std::string & *_name*, VisualPtr *_parent*, bool *_useRTShader* = true)

Constructor.

Parameters

in	*_name*	Name of the visual.
in	*_parent*	Parent of the visual.
in	*_useRTShader*	True if the visual should use the real-time shader system (RTShader).

10.186.2.2 gazebo::rendering::Visual::Visual (const std::string & *_name*, ScenePtr *_scene*, bool *_useRTShader* = true)

Constructor.

Parameters

in	*_name*	Name of the visual.
in	*_scene*	**Scene** (p. 707) containing the visual.
in	*_useRTShader*	True if the visual should use the real-time shader system (RTShader).

10.186.2.3 virtual gazebo::rendering::Visual::~Visual ( ) [virtual]

Destructor.

10.186.3 Member Function Documentation

10.186.3.1 void gazebo::rendering::Visual::AttachAxes ( )

Attach visualization axes.

10.186.3.2 void gazebo::rendering::Visual::AttachLineVertex (DynamicLines * *_line*, unsigned int *_index*)

Attach a vertex of a line to the position of the visual.

Parameters

in	*_line*	Line to attach to this visual.
in	*_index*	Index of the line vertex to attach.

10.186.3.3 Ogre::MovableObject* gazebo::rendering::Visual::AttachMesh (const std::string & *_meshName*, const std::string & *_subMesh* = " ", bool *_centerSubmesh* = false, const std::string & *_objName* = " ")

Attach a mesh to this visual by name.

Parameters

in	*_meshName*	Name of the mesh.
in	*_subMesh*	Name of the submesh. Empty string to use all submeshes.
in	*_centerSubmesh*	True to center a submesh.
in	*_objName*	Name of the attached Object to put the mesh onto.

10.186.3.4 void gazebo::rendering::Visual::AttachObject (Ogre::MovableObject * *_obj*)

Attach a renerable object to the visual.

Parameters

in	*_obj*	A movable object to attach to the visual.

10.186.3.5 void gazebo::rendering::Visual::AttachVisual (VisualPtr *_vis*)

Attach a visual to this visual.

Parameters

in	*_vis*	**Visual** (p. 920) to attach.

10.186.3.6 void gazebo::rendering::Visual::ClearParent ( )

Clear parents.

10.186.3.7 VisualPtr gazebo::rendering::Visual::Clone (const std::string & *_name*, VisualPtr *_newParent*)

Clone the visual with a new name.

Parameters

in	*_name*	Name of the cloned **Visual** (p. 920).
in	*_newParent*	Parent of the cloned **Visual** (p. 920).

Returns

The visual.

10.186.3.8 DynamicLines* gazebo::rendering::Visual::CreateDynamicLine (RenderOpType *_type* = RENDERING_LINE_STRIP)

Add a line to the visual.

Parameters

in	*_type*	The type of line to make.

Returns

A pointer to the new dynamic line.

10.186.3.9 void gazebo::rendering::Visual::DeleteDynamicLine (DynamicLines * *_line*)

Delete a dynamic line.

Parameters

in	*_line*	Pointer to the line to delete.

10.186.3.10 void gazebo::rendering::Visual::DetachObjects ( )

Detach all objects.

10.186.3.11 void gazebo::rendering::Visual::DetachVisual (VisualPtr *_vis*)

Detach a visual.

Parameters

in	*_vis*	**Visual** (p. 920) to detach.

10.186.3.12 void gazebo::rendering::Visual::DetachVisual (const std::string & *_name*)

Detach a visual.

Parameters

in	*_name*	Name of the visual to detach.

10.186.3.13 void gazebo::rendering::Visual::DisableTrackVisual ( )

Disable tracking of a visual.

10.186.3.14 void gazebo::rendering::Visual::EnableTrackVisual (VisualPtr *_vis*)

Set one visual to track/follow another.

Parameters

in	*_vis*	**Visual** (p. 920) to track.

10.186.3.15 void gazebo::rendering::Visual::Fini ( )

Helper for the destructor.

10.186.3.16 unsigned int gazebo::rendering::Visual::GetAttachedObjectCount ( ) const

Return the number of attached movable objects.

Returns

The number of attached movable objects.

10.186.3.17 math::Box gazebo::rendering::Visual::GetBoundingBox ( ) const

Get the bounding box for the visual.

Returns

The bounding box in world coordinates.

10.186.3.18 VisualPtr gazebo::rendering::Visual::GetChild (unsigned int *_index*)

Get an attached visual based on an index.

Index should be between 0 and **Visual::GetChildCount** (p. 928).

Parameters

in	*_index*	Index of the child to retreive.

Returns

Pointer to the child visual, NULL if index is invalid.

10.186.3.19 unsigned int gazebo::rendering::Visual::GetChildCount ( )

Get the number of attached visuals.

Returns

The number of children.

10.186.3.20 std::string gazebo::rendering::Visual::GetMaterialName ( ) const

Get the name of the material.

Returns

The name of the visual applied to this visual.

10.186.3.21 std::string gazebo::rendering::Visual::GetMeshName ( ) const

The name of the mesh set in the visual's SDF.

Returns

Name of the mesh.

10.186.3.22 std::string gazebo::rendering::Visual::GetName ( ) const

Get the name of the visual.

Returns

The name of the visual.

10.186.3.23 std::string gazebo::rendering::Visual::GetNormalMap ( ) const

Get the normal map.

Returns

The name of the normal map material.

10.186.3.24 VisualPtr gazebo::rendering::Visual::GetParent ( ) const

Get the parent visual, if one exists.

Returns

Pointer to the parent visual, NULL if no parent.

10.186.3.25 math::Pose gazebo::rendering::Visual::GetPose ( ) const

Get the pose of the visual.

Returns

The **Visual** (p. 920)'s pose.

10.186.3.26 math::Vector3 gazebo::rendering::Visual::GetPosition ( ) const

Get the position of the visual.

Returns

The visual's position.

10.186.3.27 VisualPtr gazebo::rendering::Visual::GetRootVisual ( )

Get the root visual.

Returns

The root visual, which is one level below the world visual.

10.186.3.28 math::Quaternion gazebo::rendering::Visual::GetRotation ( ) const

Get the rotation of the visual.

Returns

The visual's rotation.

10.186.3.29 math::Vector3 gazebo::rendering::Visual::GetScale ( )

Get the scale.

Returns

The scaling factor.

10.186.3.30 ScenePtr gazebo::rendering::Visual::GetScene ( ) const

Get current.

Returns

Pointer to the scene.

10.186.3.31 Ogre::SceneNode* gazebo::rendering::Visual::GetSceneNode ( ) const

Return the scene Node of this visual entity.

Returns

The **Ogre** (p. 110) scene node.

10.186.3.32 std::string gazebo::rendering::Visual::GetShaderType ( ) const

Get the shader type.

Returns

String of the shader type: "vertex", "pixel", "normal_map_object_space", "normal_map_tangent_space".

10.186.3.33 std::string gazebo::rendering::Visual::GetSubMeshName ( ) const

Get the name of the sub mesh set in the visual's SDF.

Returns

Name of the submesh. Empty string if no submesh is specified.

10.186.3.34 float gazebo::rendering::Visual::GetTransparency ( )

Get the transparency.

Returns

The transparency.

10.186.3.35 uint32_t gazebo::rendering::Visual::GetVisibilityFlags ( )

Get visibility flags for this visual and all children.

Returns

The visiblity flags.

See Also

GZ_VISIBILITY_ALL (p. 1112)
GZ_VISIBILITY_GUI (p. 1112)
GZ_VISIBILITY_NOT_SELECTABLE (p. 1112)

10.186.3.36 bool gazebo::rendering::Visual::GetVisible ( ) const

Get whether the visual is visible.

Returns

True if the visual is visible.

10.186.3.37 math::Pose gazebo::rendering::Visual::GetWorldPose ( ) const

Get the global pose of the node.

Returns

The pose in the world coordinate frame.

10.186.3.38 bool gazebo::rendering::Visual::HasAttachedObject (const std::string & *_name*)

Returns true if an object with _name is attached.

Parameters

in	*_name*	Name of an object to find.

10.186.3.39 void gazebo::rendering::Visual::Init ( )

Helper for the contructor.

10.186.3.40 void gazebo::rendering::Visual::InsertMesh (const std::string & *_meshName*, const std::string & *_subMesh* = " ", bool *_centerSubmesh* = false)

Insert a mesh into **Ogre** (p. 110).

Parameters

in	*_meshName*	Name of the mesh to insert.
in	*_subMesh*	Name of the mesh within _meshName to insert.
in	*_centerSubmesh*	True to center the submesh.

10.186.3.41 static void gazebo::rendering::Visual::InsertMesh (const **common::Mesh** * *_mesh*, const std::string & *_subMesh* = " ", bool *_centerSubmesh* = false) [static]

Insert a mesh into **Ogre** (p. 110).

Parameters

in	*_mesh*	Pointer to the mesh to insert.
in	*_subMesh*	Name of the mesh within _meshName to insert.
in	*_centerSubmesh*	True to center the submesh.

10.186.3.42 bool gazebo::rendering::Visual::IsPlane ( ) const

Return true if the visual is a plane.

Returns

True if a plane.

10.186.3.43 bool gazebo::rendering::Visual::IsStatic ( ) const

Return true if the visual is a static geometry.

Returns

True if the visual is static.

10.186.3.44 void gazebo::rendering::Visual::Load (**sdf::ElementPtr** *_sdf*)

Load the visual with a set of parameters.

Parameters

in	*_sdf*	Load from an SDF element.

10.186.3.45 virtual void gazebo::rendering::Visual::Load ( ) [virtual]

Load the visual with default parameters.

Reimplemented in **gazebo::rendering::ArrowVisual** (p. 135), **gazebo::rendering::SonarVisual** (p. 790), and **gazebo::rendering::AxisVisual** (p. 138).

10.186.3.46 void gazebo::rendering::Visual::LoadFromMsg (ConstVisualPtr & *_msg*)

Load from a message.

Parameters

in	*_msg*	A visual message.

10.186.3.47 void gazebo::rendering::Visual::LoadPlugin (const std::string & *_filename*, const std::string & *_name*, sdf::ElementPtr *_sdf*)

Load a plugin.

Parameters

_filename	The filename of the plugin
_name	A unique name for the plugin
_sdf	The SDF to pass into the plugin.

10.186.3.48 void gazebo::rendering::Visual::MakeStatic ( )

Make the visual objects static renderables.

10.186.3.49 void gazebo::rendering::Visual::MoveToPosition (const math::Pose & *_pose*, double *_time*)

Move to a pose and over a given time.

Parameters

in	*_pose*	Pose the visual will end at.
in	*_time*	Time it takes the visual to move to the pose.

10.186.3.50 void gazebo::rendering::Visual::MoveToPositions (const std::vector< math::Pose > & *_pts*, double *_time*, boost::function< void() > *_onComplete* = NULL)

Move to a series of pose and over a given time.

Parameters

in	*_poses*	Series of poses the visual will move to.
in	*_time*	Time it takes the visual to move to the pose.
in	*_onComplete*	Callback used when the move is complete.

10.186.3.51 void gazebo::rendering::Visual::RemovePlugin (const std::string & *_name*)

Remove a running plugin.

Parameters

_name	The unique name of the plugin to remove

10.186.3.52 void gazebo::rendering::Visual::SetAmbient (const **common::Color** & *_color*)

Set the ambient color of the visual.

Parameters

in	*_color*	The ambient color.

10.186.3.53 void gazebo::rendering::Visual::SetCastShadows (bool *_shadows*)

Set whether the visual should cast shadows.

Parameters

in	*_shadows*	True to enable shadows.

10.186.3.54 void gazebo::rendering::Visual::SetDiffuse (const **common::Color** & *_color*)

Set the diffuse color of the visual.

Parameters

in	*_color*	Set the diffuse color.

10.186.3.55 virtual void gazebo::rendering::Visual::SetEmissive (const **common::Color** & *_color*) [virtual]

Set the emissive value.

Parameters

in	*_color*	The emissive color.

Reimplemented in **gazebo::rendering::LaserVisual** (p. 433).

10.186.3.56 void gazebo::rendering::Visual::SetHighlighted (bool *_highlighted*)

Set the visual to be visually highlighted.

This is most often used when an object is selected by a user via the GUI.

Parameters

in	*_highlighted*	True to enable the highlighting.

10.186.3.57 void gazebo::rendering::Visual::SetMaterial (const std::string & *_materialName,* bool *_unique* = `true`)

Set the material.

Parameters

in	*_materialName*	The name of the material.
in	*_unique*	True to make the material unique, which allows the material to change without changing materials that originally had the same name.

10.186.3.58 void gazebo::rendering::Visual::SetName (const std::string & *_name*)

Set the name of the visual.

Parameters

in	*_name*	Name of the visual

10.186.3.59 void gazebo::rendering::Visual::SetNormalMap (const std::string & *_nmap*)

Set the normal map.

Parameters

in	*_nmap*	Name of the normal map material.

10.186.3.60 void gazebo::rendering::Visual::SetPose (const math::Pose & *_pose*)

Set the pose of the visual.

Parameters

in	*_pose*	The new pose of the visual.

10.186.3.61 void gazebo::rendering::Visual::SetPosition (const math::Vector3 & *_pos*)

Set the position of the visual.

Parameters

in	*_pos*	The position to set the visual to.

10.186.3.62 void gazebo::rendering::Visual::SetRibbonTrail (bool *_value,* const common::Color & *_initialColor,* const common::Color & *_changeColor*)

True on or off a ribbon trail.

Parameters

in	*_value*	True to enable ribbon trail.
in	*_initialColor*	The initial color of the ribbon trail.
in	*_changeColor*	Color to change too as the trail grows.

10.186.3.63 void gazebo::rendering::Visual::SetRotation (const math::Quaternion & *_rot*)

Set the rotation of the visual.

Parameters

in	*_rot*	The rotation of the visual.

10.186.3.64 void gazebo::rendering::Visual::SetScale (const math::Vector3 & *_scale*)

Set the scale.

Parameters

in	*_scale*	The scaling factor for the visual.

10.186.3.65 void gazebo::rendering::Visual::SetScene (ScenePtr *_scene*)

Set current scene.

Parameters

in	*_scene*	Pointer to the scene.

10.186.3.66 void gazebo::rendering::Visual::SetShaderType (const std::string & *_type*)

Set the shader type for the visual's material.

Parameters

in	*_type*	Shader type string: "vertex", "pixel", "normal_map_object_space", "normal_map_tangent_space".

10.186.3.67 void gazebo::rendering::Visual::SetSkeletonPose (const msgs::PoseAnimation & *_pose*)

Set animation skeleton pose.

Parameters

in	*_pose*	Skelton message

10.186.3.68 void gazebo::rendering::Visual::SetSpecular (const common::Color & *_color*)

Set the specular color of the visual.

Parameters

in	*_color*	Specular color.

10.186.3.69 void gazebo::rendering::Visual::SetTransparency (float *_trans*)

Set the transparency.

Parameters

in	*_trans*	The transparency, between 0 and 1 where 0 is no transparency.

10.186.3.70 void gazebo::rendering::Visual::SetVisibilityFlags (uint32_t *_flags*)

Set visibility flags for this visual and all children.

Parameters

in	*_flags*	The visiblity flags.

See Also

GZ_VISIBILITY_ALL (p. 1112)
GZ_VISIBILITY_GUI (p. 1112)
GZ_VISIBILITY_NOT_SELECTABLE (p. 1112)

10.186.3.71 void gazebo::rendering::Visual::SetVisible (bool *_visible*, bool *_cascade* = `true`)

Set whether the visual is visible.

Parameters

in	*_visible*	set this node visible.
in	*_cascade*	setting this parameter in children too.

10.186.3.72 void gazebo::rendering::Visual::SetWireframe (bool _show)

Enable or disable wireframe for this visual.

Parameters

in	*_show*	True to enable wireframe for this visual.

10.186.3.73 void gazebo::rendering::Visual::SetWorldPose (const math::Pose _pose)

Set the world pose of the visual.

Parameters

in	*_pose*	Pose of the visual in the world coordinate frame.

10.186.3.74 void gazebo::rendering::Visual::SetWorldPosition (const math::Vector3 & _pos)

Set the world linear position of the visual.

Parameters

in	*_pose*	Position in the world coordinate frame.

10.186.3.75 void gazebo::rendering::Visual::SetWorldRotation (const math::Quaternion & _rot)

Set the world orientation of the visual.

Parameters

in	*_rot*	Rotation in the world coordinate frame.

10.186.3.76 void gazebo::rendering::Visual::ShowBoundingBox ( )

Display the bounding box visual.

10.186.3.77 void gazebo::rendering::Visual::ShowCollision (bool _show)

Display the collision visuals.

Parameters

in	*_show*	True to show visuals labeled as collision objects.

10.186.3.78 void gazebo::rendering::Visual::ShowCOM (bool _show)

Display Center of Mass visuals.

Parameters

in	*_show*	True to show center of mass visualizations.

10.186.3.79 void gazebo::rendering::Visual::ShowJoints (bool *_show*)

Display joint visuals.

Parameters

in	*_show*	True to show joint visualizations.

10.186.3.80 void gazebo::rendering::Visual::ShowSkeleton (bool *_show*)

Display the skeleton visuals.

Parameters

in	*_show*	True to show skeleton visuals.

10.186.3.81 void gazebo::rendering::Visual::ToggleVisible ( )

Toggle whether this visual is visible.

10.186.3.82 void gazebo::rendering::Visual::Update ( )

Update the visual.

10.186.3.83 void gazebo::rendering::Visual::UpdateFromMsg (ConstVisualPtr & *_msg*)

Update a visual based on a message.

Parameters

in	*_msg*	The visual message.

10.186.4 Member Data Documentation

10.186.4.1 VisualPtr gazebo::rendering::Visual::parent [protected]

Parent visual.

10.186.4.2 ScenePtr gazebo::rendering::Visual::scene [protected]

Pointer to the visual's scene.

10.186.4.3 Ogre::SceneNode* gazebo::rendering::Visual::sceneNode [protected]

Pointer to the visual's scene node in **Ogre** (p. 110).

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

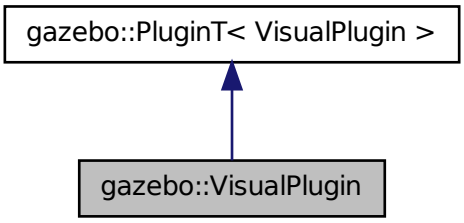
- **Visual.hh**

10.187 gazebo::VisualPlugin Class Reference

A plugin loaded within the gzserver on startup.

`#include <Plugin.hh>`

Inheritance diagram for gazebo::VisualPlugin:



Public Member Functions

- **VisualPlugin** ()
- virtual void **Init** ()

  Initialize the plugin.
- virtual void **Load** (**rendering::VisualPtr** _visual, **sdf::ElementPtr** _sdf)=0

  Load function.
- virtual void **Reset** ()

  Override this method for custom plugin reset behavior.

Additional Inherited Members

10.187.1 Detailed Description

A plugin loaded within the gzserver on startup.

See `reference`.

10.187.2 Constructor & Destructor Documentation

10.187.2.1 gazebo::VisualPlugin::VisualPlugin ( ) [inline]

References gazebo::PluginT< VisualPlugin >::type, and gazebo::VISUAL_PLUGIN.

10.187.3 Member Function Documentation

10.187.3.1 virtual void gazebo::VisualPlugin::Init ( ) [inline], [virtual]

Initialize the plugin.

Called after Gazebo has been loaded. Must not block.

10.187.3.2 virtual void gazebo::VisualPlugin::Load (rendering::VisualPtr *_visual,* sdf::ElementPtr *_sdf*) [pure virtual]

Load function.

Called when a Plugin is first created, and after the World has been loaded. This function should not be blocking.

Parameters

in	*_visual*	Pointer the Visual Object.
in	*_sdf*	Pointer the the SDF element of the plugin.

10.187.3.3 virtual void gazebo::VisualPlugin::Reset ( ) [inline], [virtual]

Override this method for custom plugin reset behavior.

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

- **common/Plugin.hh**

10.188 gazebo::rendering::WindowManager Class Reference

Class to mangage render windows.

`#include <rendering/rendering.hh>`

Public Member Functions

- **WindowManager** ()

  Constructor.
- virtual ~**WindowManager** ()

  Destructor.
- int **CreateWindow** (const std::string &_ogreHandle, uint32_t _width, uint32_t _height)

  Create a window.
- void **Fini** ()

Shutdown all the windows.

- float **GetAvgFPS** (uint32_t _id)

  Get the average FPS.
- uint32_t **GetTriangleCount** (uint32_t _id)

  Get the triangle count.
- Ogre::RenderWindow * **GetWindow** (uint32_t _id)

  Get the render window associated with the given id.
- void **Moved** (uint32_t _id)

  *Tells **Ogre** (p. 110) the window has moved, and needs updating.*
- void **Resize** (uint32_t _id, int _width, int _height)

  Resize a window.
- void **SetCamera** (int _windowId, **CameraPtr** _camera)

  Attach a camera to a window.

10.188.1 Detailed Description

Class to mangage render windows.

10.188.2 Constructor & Destructor Documentation

10.188.2.1 gazebo::rendering::WindowManager::WindowManager ( )

Constructor.

10.188.2.2 virtual gazebo::rendering::WindowManager::~WindowManager ( ) [virtual]

Destructor.

10.188.3 Member Function Documentation

10.188.3.1 int gazebo::rendering::WindowManager::CreateWindow (const std::string & *_ogreHandle*, uint32_t *_width*, uint32_t *_height*)

Create a window.

Parameters

in	*_ogreHandle*	String representing the ogre window handle.
in	*_width*	With of the window in pixels.
in	*_height*	Height of the window in pixels.

10.188.3.2 void gazebo::rendering::WindowManager::Fini ( )

Shutdown all the windows.

10.188.3.3 float gazebo::rendering::WindowManager::GetAvgFPS (uint32_t _id)

Get the average FPS.

Parameters

in	*_id*	ID of the window.

Returns

The frames per second.

10.188.3.4 uint32_t gazebo::rendering::WindowManager::GetTriangleCount (uint32_t _id)

Get the triangle count.

Parameters

in	*_id*	ID of the window.

Returns

The triangle count.

10.188.3.5 Ogre::RenderWindow* gazebo::rendering::WindowManager::GetWindow (uint32_t _id)

Get the render window associated with the given id.

Parameters

in	*_id*	ID of the window.

Returns

Pointer to the render window, NULL if the id is invalid.

10.188.3.6 void gazebo::rendering::WindowManager::Moved (uint32_t _id)

Tells **Ogre** (p. 110) the window has moved, and needs updating.

Parameters

in	*_id*	ID of the window.

10.188.3.7 void gazebo::rendering::WindowManager::Resize (uint32_t _id, int _width, int _height)

Resize a window.

Parameters

in	*_id*	Id of the window to resize.
in	*_width*	New width of the window.
in	*_height*	New height of the window.

10.188.3.8 void gazebo::rendering::WindowManager::SetCamera (int *_windowId,* CameraPtr *_camera*)

Attach a camera to a window.

Parameters

in	*_windowId*	Id of the window to add the camera to.
in	*_camera*	Pointer to the camera to attach.

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

- **WindowManager.hh**

10.189 gazebo::rendering::WireBox Class Reference

Draws a wireframe box.

`#include <rendering/rendering.hh>`

Public Member Functions

- **WireBox** (**VisualPtr** _parent, const **math::Box** &_box)
  Constructor.
- ~**WireBox** ()
  Destructor.
- void **Init** (const **math::Box** &_box)
  Builds the wireframe line list.
- void **SetVisible** (bool _visible)
  Set the visibility of the box.

10.189.1 Detailed Description

Draws a wireframe box.

10.189.2 Constructor & Destructor Documentation

10.189.2.1 gazebo::rendering::WireBox::WireBox (VisualPtr *_parent,* const math::Box & *_box*) [explicit]

Constructor.

Parameters

in	*_box*	Dimenision of the box to draw.

10.189.2.2 gazebo::rendering::WireBox::~WireBox ( )

Destructor.

10.189.3 Member Function Documentation

10.189.3.1 void gazebo::rendering::WireBox::Init (const math::Box & *_box*)

Builds the wireframe line list.

Parameters

in	*_box*	Box to build a wireframe from.

10.189.3.2 void gazebo::rendering::WireBox::SetVisible (bool *_visible*)

Set the visibility of the box.

Parameters

in	*_visible*	True to make the box visible, False to hide.

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

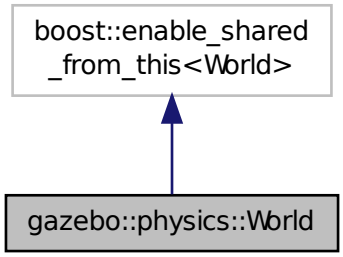
- **WireBox.hh**

10.190 gazebo::physics::World Class Reference

The world provides access to all other object within a simulated environment.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::World:



Public Member Functions

- **World** (const std::string &_name="")

  Constructor.
- ∼**World** ()

  Destructor.
- void **Clear** ()

  Remove all entities from the world.
- void **DisableAllModels** ()

  Disable all links in all the models.
- void **EnableAllModels** ()

  Enable all links in all the models.
- void **EnablePhysicsEngine** (bool _enable)

  enable/disable physics engine during World::Update.
- void **Fini** ()

  Finalize the world.
- **BasePtr GetByName** (const std::string &_name)

  Get an element by name.
- bool **GetEnablePhysicsEngine** ()

  check if physics engine is enabled/disabled.
- **EntityPtr GetEntity** (const std::string &_name)

  *Get a pointer to an **Entity** (p. 288) based on a name.*
- **EntityPtr GetEntityBelowPoint** (const **math::Vector3** &_pt)

  Get the nearest entity below a point.
- **ModelPtr GetModel** (unsigned int _index) const

  Get a model based on an index.
- **ModelPtr GetModel** (const std::string &_name)

  Get a model by name.
- **ModelPtr GetModelBelowPoint** (const **math::Vector3** &_pt)

  Get the nearest model below a point.
- unsigned int **GetModelCount** () const

  Get the number of models.
- **Model_V GetModels** () const

  Get a list of all the models.
- std::string **GetName** () const

  Get the name of the world.
- **common::Time GetPauseTime** () const

  Get the amount of time simulation has been paused.
- **PhysicsEnginePtr GetPhysicsEngine** () const

  Return the physics engine.
- **common::Time GetRealTime** () const

  Get the real time (elapsed time).
- bool **GetRunning** () const

  Return the running state of the world.
- **EntityPtr GetSelectedEntity** () const

  *Get the selected **Entity** (p. 288).*
- boost::mutex ∗ **GetSetWorldPoseMutex** () const

Get the set world pose mutex.

- **common::Time GetSimTime** () const

  *Get the world simulation time, note if you want the PC wall clock call **common::Time::GetWallTime** (p. 836).*

- **common::Time GetStartTime** () const

  Get the wall time simulation was started.

- void **Init** ()

  Initialize the world.

- void **InsertModelFile** (const std::string &_sdfFilename)

  Insert a model from an SDF file.

- void **InsertModelSDF** (const **sdf::SDF** &_sdf)

  Insert a model using SDF.

- void **InsertModelString** (const std::string &_sdfString)

  Insert a model from an SDF string.

- bool **IsLoaded** () const

  Return true if the world has been loaded.

- bool **IsPaused** () const

  Returns the state of the simulation true if paused.

- void **Load** (**sdf::ElementPtr** _sdf)

  Load the world using SDF parameters.

- void **LoadPlugin** (const std::string &_filename, const std::string &_name, **sdf::ElementPtr** _sdf)

  Load a plugin.

- void **PrintEntityTree** ()

  *Print **Entity** (p. 288) tree.*

- void **PublishModelPose** (**physics::ModelPtr** _model)

  Publish pose updates for a model.

- void **RemovePlugin** (const std::string &_name)

  Remove a running plugin.

- void **Reset** ()

  Reset time and model poses, configurations in simulation.

- void **ResetEntities** (**Base::EntityType** _type=**Base::BASE**)

  Reset with options.

- void **ResetTime** ()

  Reset simulation time back to zero.

- void **Run** (unsigned int _iterations=0)

  Run the world in a thread.

- void **Save** (const std::string &_filename)

  Save a world to a file.

- void **SetPaused** (bool _p)

  Set whether the simulation is paused.

- void **SetSimTime** (const **common::Time** &_t)

  Set the sim time.

- void **SetState** (const **WorldState** &_state)

  Set the current world state.

- void **StepWorld** (int _steps)

  Step callback.

- void **Stop** ()

  Stop the world.

- std::string **StripWorldName** (const std::string &_name) const

  Return a version of the name with "<world_name>::" removed.
- void **UpdateStateSDF** ()

  Update the state SDF value from the current state.

Public Attributes

- std::list< **Entity** * > **dirtyPoses**

  *when physics engine makes an update and changes a link pose, this flag is set to trigger **Entity::SetWorldPose** (p. 297) on the **physics::Link** (p. 439) in World::Update.*

10.190.1 Detailed Description

The world provides access to all other object within a simulated environment.

The **World** (p. 945) is the container for all models and their components (links, joints, sensors, plugins, etc), and **WorldPlugin** (p. 957) instances. Many core function are also handled in the **World** (p. 945), including physics update, model updates, and message processing.

10.190.2 Constructor & Destructor Documentation

10.190.2.1 gazebo::physics::World::World (const std::string & *_name* = " ") [explicit]

Constructor.

Constructor for the **World** (p. 945). Must specify a unique name.

Parameters

in	*_name*	Name of the world.

10.190.2.2 gazebo::physics::World::~World ( )

Destructor.

10.190.3 Member Function Documentation

10.190.3.1 void gazebo::physics::World::Clear ( )

Remove all entities from the world.

10.190.3.2 void gazebo::physics::World::DisableAllModels ( )

Disable all links in all the models.

Disable is a physics concept. Disabling means that the physics engine should not update an entity.

10.190.3.3 void gazebo::physics::World::EnableAllModels ( )

Enable all links in all the models.

Enable is a physics concept. Enabling means that the physics engine should update an entity.

10.190.3.4 void gazebo::physics::World::EnablePhysicsEngine (bool *_enable*) [inline]

enable/disable physics engine during World::Update.

Parameters

in	*_enable*	True to enable the physics engine.

10.190.3.5 void gazebo::physics::World::Fini ( )

Finalize the world.

Call this function to tear-down the world.

10.190.3.6 BasePtr gazebo::physics::World::GetByName (const std::string & *_name*)

Get an element by name.

Searches the list of entities, and return a pointer to the model with a matching _name.

Parameters

in	*_name*	The name of the **Model** (p. 516) to find.

Returns

A pointer to the entity, or NULL if no entity was found.

10.190.3.7 bool gazebo::physics::World::GetEnablePhysicsEngine ( ) [inline]

check if physics engine is enabled/disabled.

Parameters

True	if the physics engine is enabled.

10.190.3.8 EntityPtr gazebo::physics::World::GetEntity (const std::string & *_name*)

Get a pointer to an **Entity** (p. 288) based on a name.

This function is the same as GetByName, but limits the search to only Entities.

Parameters

in	*_name*	The name of the **Entity** (p. 288) to find.

Returns

A pointer to the **Entity** (p. 288), or NULL if no **Entity** (p. 288) was found.

10.190.3.9 EntityPtr gazebo::physics::World::GetEntityBelowPoint (const math::Vector3 & _pt)

Get the nearest entity below a point.

Projects a Ray down (-Z axis) starting at the given point. The first entity hit by the Ray is returned.

Parameters

in	*_pt*	The 3D point to search below

Returns

A pointer to nearest **Entity** (p. 288), NULL if none is found.

10.190.3.10 ModelPtr gazebo::physics::World::GetModel (unsigned int _index) const

Get a model based on an index.

Get a **Model** (p. 516) using an index, where index must be greater than zero and less than **World::GetModelCount()** (p. 951)

Parameters

in	*_index*	The index of the model [0..GetModelCount)

Returns

A pointer to the **Model** (p. 516). NULL if _index is invalid.

10.190.3.11 ModelPtr gazebo::physics::World::GetModel (const std::string & _name)

Get a model by name.

This function is the same as GetByName, but limits the search to only models.

Parameters

in	*_name*	The name of the **Model** (p. 516) to find.

Returns

A pointer to the **Model** (p. 516), or NULL if no model was found.

10.190.3.12 ModelPtr gazebo::physics::World::GetModelBelowPoint (const math::Vector3 & _pt)

Get the nearest model below a point.

This function makes use of **World::GetEntityBelowPoint** (p. 950).

Parameters

in	*_pt*	The 3D point to search below.

Returns

A pointer to nearest **Model** (p. 516), NULL if none is found.

10.190.3.13 unsigned int gazebo::physics::World::GetModelCount ( ) const

Get the number of models.

Returns

The number of models in the **World** (p. 945).

10.190.3.14 Model_V gazebo::physics::World::GetModels ( ) const

Get a list of all the models.

Returns

A list of all the Models in the world.

10.190.3.15 std::string gazebo::physics::World::GetName ( ) const

Get the name of the world.

Returns

The name of the world.

10.190.3.16 common::Time gazebo::physics::World::GetPauseTime ( ) const

Get the amount of time simulation has been paused.

Returns

The pause time.

10.190.3.17 PhysicsEnginePtr gazebo::physics::World::GetPhysicsEngine ( ) const

Return the physics engine.

Get a pointer to the physics engine used by the world.

Returns

Pointer to the physics engine.

10.190.3.18 common::Time gazebo::physics::World::GetRealTime ( ) const

Get the real time (elapsed time).

Returns

The real time.

10.190.3.19 bool gazebo::physics::World::GetRunning ( ) const

Return the running state of the world.

Returns

True if the world is running.

10.190.3.20 EntityPtr gazebo::physics::World::GetSelectedEntity ( ) const

Get the selected **Entity** (p. 288).

The selected entity is set via the GUI.

Returns

A point to the **Entity** (p. 288), NULL if nothing is selected.

10.190.3.21 boost::mutex* gazebo::physics::World::GetSetWorldPoseMutex ( ) const [inline]

Get the set world pose mutex.

Returns

Pointer to the mutex.

10.190.3.22 common::Time gazebo::physics::World::GetSimTime ( ) const

Get the world simulation time, note if you want the PC wall clock call **common::Time::GetWallTime** (p. 836).

Returns

The current simulation time

10.190.3.23 common::Time gazebo::physics::World::GetStartTime ( ) const

Get the wall time simulation was started.

Returns

The start time.

10.190.3.24 void gazebo::physics::World::Init ( )

Initialize the world.

This is called after Load.

10.190.3.25 void gazebo::physics::World::InsertModelFile (const std::string & *_sdfFilename*)

Insert a model from an SDF file.

Spawns a model into the world base on and SDF file.

Parameters

in	*_sdfFilename*	The name of the SDF file (including path).

10.190.3.26 void gazebo::physics::World::InsertModelSDF (const sdf::SDF & *_sdf*)

Insert a model using SDF.

Spawns a model into the world base on and SDF object.

Parameters

in	*_sdf*	A reference to an SDF object.

10.190.3.27 void gazebo::physics::World::InsertModelString (const std::string & *_sdfString*)

Insert a model from an SDF string.

Spawns a model into the world base on and SDF string.

Parameters

in	*_sdfString*	A string containing valid SDF markup.

10.190.3.28 bool gazebo::physics::World::IsLoaded ( ) const

Return true if the world has been loaded.

Returns

True if **World::Load** (p. 954) has completed.

10.190.3.29 bool gazebo::physics::World::IsPaused ( ) const

Returns the state of the simulation true if paused.

Returns

True if paused.

10.190.3.30 void gazebo::physics::World::Load (sdf::ElementPtr _sdf)

Load the world using SDF parameters.

Load a world from and SDF pointer.

Parameters

in	*_sdf*	SDF parameters.

10.190.3.31 void gazebo::physics::World::LoadPlugin (const std::string & _filename, const std::string & _name, sdf::ElementPtr _sdf)

Load a plugin.

Parameters

in	*_filename*	The filename of the plugin.
in	*_name*	A unique name for the plugin.
in	*_sdf*	The SDF to pass into the plugin.

10.190.3.32 void gazebo::physics::World::PrintEntityTree ( )

Print **Entity** (p. 288) tree.

Prints alls the entities to stdout.

10.190.3.33 void gazebo::physics::World::PublishModelPose (physics::ModelPtr _model)

Publish pose updates for a model.

This list of models to publish is processed and cleared once every iteration.

Parameters

in	*_model*	Pointer to the model to publish.

10.190.3.34 void gazebo::physics::World::RemovePlugin (const std::string & _name)

Remove a running plugin.

Parameters

in	*_name*	The unique name of the plugin to remove.

10.190.3.35 void gazebo::physics::World::Reset ( )

Reset time and model poses, configurations in simulation.

10.190.3.36 void gazebo::physics::World::ResetEntities (Base::EntityType _type = Base::BASE)

Reset with options.

The _type parameter specifies which type of eneities to reset. See **Base::EntityType** (p. 144).

Parameters

in	*_type*	The type of reset.

10.190.3.37 void gazebo::physics::World::ResetTime ( )

Reset simulation time back to zero.

10.190.3.38 void gazebo::physics::World::Run (unsigned int _iterations = 0)

Run the world in a thread.

Run the update loop.

Parameters

in	*_iterations*	Run for this many iterations, then stop. A value of zero disables run stop.

10.190.3.39 void gazebo::physics::World::Save (const std::string & _filename)

Save a world to a file.

Save the current world and its state to a file.

Parameters

in	*_filename*	Name of the file to save into.

10.190.3.40 void gazebo::physics::World::SetPaused (bool _p)

Set whether the simulation is paused.

Parameters

in	*_p*	True pauses the simulation. False runs the simulation.

10.190.3.41 void gazebo::physics::World::SetSimTime (const common::Time & _t)

Set the sim time.

Parameters

in	*_t*	The new simulation time

10.190.3.42 void gazebo::physics::World::SetState (const WorldState & *_state*)

Set the current world state.

Parameters

_state	The state to set the **World** (p. 945) to.

10.190.3.43 void gazebo::physics::World::StepWorld (int *_steps*)

Step callback.

Parameters

in	*_steps*	The number of steps the **World** (p. 945) should take.

10.190.3.44 void gazebo::physics::World::Stop ( )

Stop the world.

Stop the update loop.

10.190.3.45 std::string gazebo::physics::World::StripWorldName (const std::string & *_name*) const

Return a version of the name with "<world_name>::" removed.

Parameters

in	*_name*	Usually the name of an entity.

Returns

The stripped world name.

10.190.3.46 void gazebo::physics::World::UpdateStateSDF ( )

Update the state SDF value from the current state.

10.190.4 Member Data Documentation

10.190.4.1 std::list<Entity*> gazebo::physics::World::dirtyPoses

when physics engine makes an update and changes a link pose, this flag is set to trigger **Entity::SetWorldPose** (p. 297) on the **physics::Link** (p. 439) in World::Update.

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

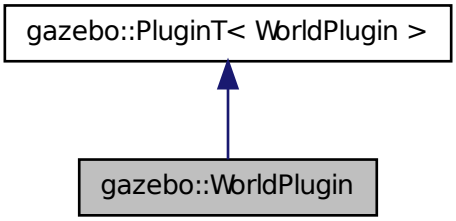
- **World.hh**

10.191 gazebo::WorldPlugin Class Reference

A plugin with access to **physics::World** (p. 945).

`#include <common/common.hh>`

Inheritance diagram for gazebo::WorldPlugin:



Public Member Functions

- **WorldPlugin** ()

  Constructor.
- virtual ~**WorldPlugin** ()

  Destructor.
- virtual void **Init** ()
- virtual void **Load** (**physics::WorldPtr** _world, **sdf::ElementPtr** _sdf)=0

  Load function.
- virtual void **Reset** ()

Additional Inherited Members

10.191.1 Detailed Description

A plugin with access to **physics::World** (p. 945).

See `reference`.

10.191.2 Constructor & Destructor Documentation

10.191.2.1 gazebo::WorldPlugin::WorldPlugin ( ) [inline]

Constructor.

References gazebo::PluginT< WorldPlugin >::type, and gazebo::WORLD_PLUGIN.

10.191.2.2 virtual gazebo::WorldPlugin::~WorldPlugin ( ) [inline], [virtual]

Destructor.

10.191.3 Member Function Documentation

10.191.3.1 virtual void gazebo::WorldPlugin::Init ( ) [inline], [virtual]

10.191.3.2 virtual void gazebo::WorldPlugin::Load (physics::WorldPtr *_world,* sdf::ElementPtr *_sdf*) [pure virtual]

Load function.

Called when a Plugin is first created, and after the World has been loaded. This function should not be blocking.

Parameters

in	*_world*	Pointer the World
in	*_sdf*	Pointer the the SDF element of the plugin.

10.191.3.3 virtual void gazebo::WorldPlugin::Reset ( ) [inline], [virtual]

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

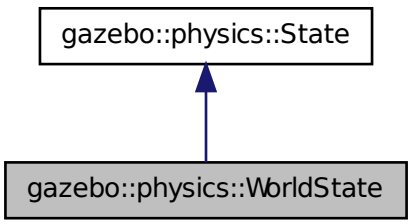
- **common/Plugin.hh**

10.192 gazebo::physics::WorldState Class Reference

Store state information of a **physics::World** (p. 945) object.

`#include <physics/physics.hh>`

Inheritance diagram for gazebo::physics::WorldState:



Public Member Functions

- **WorldState** ()

  Default constructor.
- **WorldState** (const **WorldPtr** _world)

  Constructor.
- **WorldState** (const **sdf::ElementPtr** _sdf)

  Constructor.
- virtual ~**WorldState** ()

  Destructor.
- void **FillSDF** (**sdf::ElementPtr** _sdf)

  Populate a state SDF element with data from the object.
- **ModelState GetModelState** (unsigned int _index) const **GAZEBO_DEPRECATED**(1.7)

  Deprecated.
- **ModelState GetModelState** (const std::string &_modelName) const

  Get a model state by model name.
- unsigned int **GetModelStateCount** () const

  Get the number of model states.
- **ModelState_M GetModelStates** (const boost::regex &_regex) const

  Get model states based on a regular expression.
- const **ModelState_M** & **GetModelStates** () const

  Get the model states.
- bool **HasModelState** (const std::string &_modelName) const

  *Return true if **WorldState** (p. 958) has a **ModelState** (p. 533) with the given name.*
- bool **IsZero** () const

  Return true if the values in the state are zero.
- void **Load** (const **WorldPtr** _world)

  *Load from a **World** (p. 945) pointer.*
- virtual void **Load** (const **sdf::ElementPtr** _elem)

  Load state from SDF element.
- **WorldState operator+** (const **WorldState** &_state) const

  Addition operator.
- **WorldState operator-** (const **WorldState** &_state) const

  Subtraction operator.
- **WorldState** & **operator=** (const **WorldState** &_state)

  Assignment operator.
- virtual void **SetRealTime** (const **common::Time** &_time)

  Set the real time when this state was generated.
- virtual void **SetSimTime** (const **common::Time** &_time)

  Set the sim time when this state was generated.
- virtual void **SetWallTime** (const **common::Time** &_time)

  Set the wall time when this state was generated.
- void **SetWorld** (const **WorldPtr** _world)

  Set the world.

Friends

- std::ostream & **operator**<< (std::ostream &_out, const **gazebo::physics::WorldState** &_state)

  Stream insertion operator.

Additional Inherited Members

10.192.1 Detailed Description

Store state information of a **physics::World** (p. 945) object.

Instances of this class contain the state of a **World** (p. 945) at a specific time. **World** (p. 945) state includes the state of all models, and their children.

10.192.2 Constructor & Destructor Documentation

10.192.2.1 gazebo::physics::WorldState::WorldState ( )

Default constructor.

10.192.2.2 gazebo::physics::WorldState::WorldState (const **WorldPtr** *_world*) [explicit]

Constructor.

Generate a **WorldState** (p. 958) from an instance of a **World** (p. 945).

Parameters

in	*_world*	Pointer to a world

10.192.2.3 gazebo::physics::WorldState::WorldState (const **sdf::ElementPtr** *_sdf*) [explicit]

Constructor.

Build a **WorldState** (p. 958) from SDF data

Parameters

in	*_sdf*	SDF data to load a world state from.

10.192.2.4 virtual gazebo::physics::WorldState::~WorldState ( ) [virtual]

Destructor.

10.192.3 Member Function Documentation

10.192.3.1 void gazebo::physics::WorldState::FillSDF (**sdf::ElementPtr** *_sdf*)

Populate a state SDF element with data from the object.

Parameters

out	*_sdf*	SDF element to populate.

10.192.3.2 ModelState gazebo::physics::WorldState::GetModelState (unsigned int *_index*) const

Deprecated.

10.192.3.3 ModelState gazebo::physics::WorldState::GetModelState (const std::string & *_modelName*) const

Get a model state by model name.

Parameters

in	*_modelName*	Name of the model state to get.

Returns

The model state.

Exceptions

common::Exception *(p. 325)*	When the _modelName doesn't exist.

10.192.3.4 unsigned int gazebo::physics::WorldState::GetModelStateCount ( ) const

Get the number of model states.

Returns the number of models in this instance.

Returns

Number of models.

10.192.3.5 ModelState_M gazebo::physics::WorldState::GetModelStates (const boost::regex & *_regex*) const

Get model states based on a regular expression.

Parameters

in	*_regex*	The regular expression.

Returns

List of model states whose names match the regular expression.

10.192.3.6 const ModelState_M& gazebo::physics::WorldState::GetModelStates ( ) const

Get the model states.

Returns

A vector of model states.

10.192.3.7 bool gazebo::physics::WorldState::HasModelState (const std::string & *_modelName*) const

Return true if **WorldState** (p. 958) has a **ModelState** (p. 533) with the given name.

Parameters

in	*_modelName*	Name of the model to search for.

Returns

True if the **ModelState** (p. 533) exists.

10.192.3.8 bool gazebo::physics::WorldState::IsZero ( ) const

Return true if the values in the state are zero.

This will check to see if the all model states are zero.

Returns

True if the values in the state are zero.

10.192.3.9 void gazebo::physics::WorldState::Load (const WorldPtr *_world*)

Load from a **World** (p. 945) pointer.

Generate a **WorldState** (p. 958) from an instance of a **World** (p. 945).

Parameters

in	*_world*	Pointer to a world

10.192.3.10 virtual void gazebo::physics::WorldState::Load (const sdf::ElementPtr *_elem*) [virtual]

Load state from SDF element.

Set a **WorldState** (p. 958) from an SDF element containing **WorldState** (p. 958) info.

Parameters

in	*_elem*	Pointer to the **WorldState** (p. 958) SDF element.

Reimplemented from **gazebo::physics::State** (p. 800).

10.192.3.11 WorldState gazebo::physics::WorldState::operator+ (const WorldState & *_state*) const

Addition operator.

Parameters

in	*_pt*	A state to add.

Returns

The resulting state.

10.192.3.12 WorldState gazebo::physics::WorldState::operator- (const WorldState & *_state*) const

Subtraction operator.

Parameters

in	*_pt*	A state to substract.

Returns

The resulting state.

10.192.3.13 WorldState& gazebo::physics::WorldState::operator= (const WorldState & *_state*)

Assignment operator.

Parameters

in	*_state*	**State** (p. 797) value

Returns

Reference to this

10.192.3.14 virtual void gazebo::physics::WorldState::SetRealTime (const common::Time & *_time*) [virtual]

Set the real time when this state was generated.

Parameters

in	*_time*	Clock time since simulation was stated.

Reimplemented from **gazebo::physics::State** (p. 801).

10.192.3.15 virtual void gazebo::physics::WorldState::SetSimTime (const common::Time & *_time*) [virtual]

Set the sim time when this state was generated.

Parameters

in	*_time*	Simulation time when the data was recorded.

Reimplemented from **gazebo::physics::State** (p. 801).

10.192.3.16 virtual void gazebo::physics::WorldState::SetWallTime (const common::Time & _time) [virtual]

Set the wall time when this state was generated.

Parameters

in	*_time*	The absolute clock time when the **State** (p. 797) data was recorded.

Reimplemented from **gazebo::physics::State** (p. 801).

10.192.3.17 void gazebo::physics::WorldState::SetWorld (const WorldPtr _world)

Set the world.

Parameters

in	*_world*	Pointer to the world.

10.192.4 Friends And Related Function Documentation

10.192.4.1 std::ostream& operator<< (std::ostream & _out, const gazebo::physics::WorldState & _state) [friend]

Stream insertion operator.

Parameters

in	*_out*	output stream
in	*_state*	**World** (p. 945) state to output

Returns

the stream

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

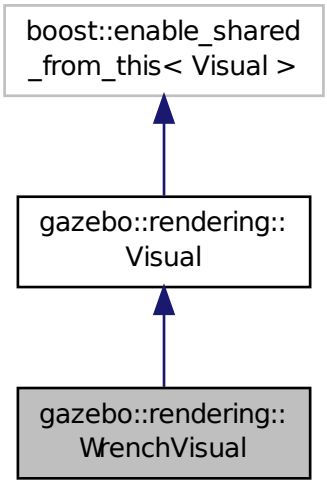
- **WorldState.hh**

10.193 gazebo::rendering::WrenchVisual Class Reference

Visualization for sonar data.

`#include <rendering/rendering.hh>`

Inheritance diagram for gazebo::rendering::WrenchVisual:



Public Member Functions

- **WrenchVisual** (const std::string &_name, **VisualPtr** _vis, const std::string &_topicName)

  Constructor.
- virtual ~**WrenchVisual** ()

  Destructor.
- void **Load** (ConstJointPtr &_msg)

  Load the visual based on a message.
- void **SetEnabled** (bool _enabled)

  Set to true to enable wrench visualization.

Additional Inherited Members

10.193.1 Detailed Description

Visualization for sonar data.

10.193.2 Constructor & Destructor Documentation

10.193.2.1 gazebo::rendering::WrenchVisual::WrenchVisual (const std::string & *_name*, VisualPtr *_vis*, const std::string & *_topicName*)

Constructor.

Parameters

in	*_name*	Name of the visual.
in	*_vis*	Pointer to the parent **Visual** (p. 920).
in	*_topicName*	Name of the topic that has sonar data.

10.193.2.2 virtual gazebo::rendering::WrenchVisual::~WrenchVisual ( ) [virtual]

Destructor.

10.193.3 Member Function Documentation

10.193.3.1 void gazebo::rendering::WrenchVisual::Load (ConstJointPtr & *_msg*)

Load the visual based on a message.

Parameters

in	*_msg*	Joint message

10.193.3.2 void gazebo::rendering::WrenchVisual::SetEnabled (bool *_enabled*)

Set to true to enable wrench visualization.

Parameters

in	*_enabled*	True to show wrenches, false to hide.

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

- **WrenchVisual.hh**

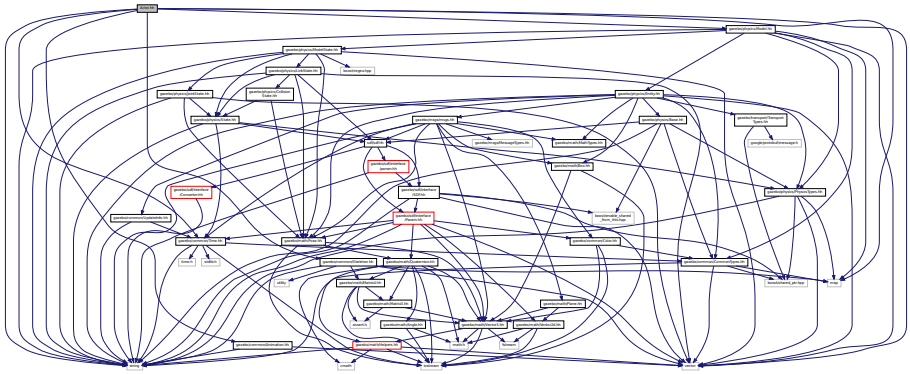
Chapter 11

File Documentation

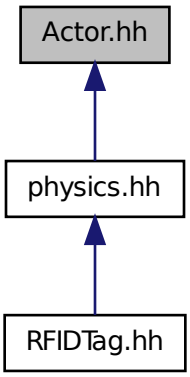
11.1 Actor.hh File Reference

```
#include <string>
#include <map>
#include <vector>
#include "gazebo/physics/Model.hh"
#include "gazebo/common/Time.hh"
#include "gazebo/common/Skeleton.hh"
#include "gazebo/common/Animation.hh"
```

Include dependency graph for Actor.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::Actor**

  Actor *(*p. *115) class enables GPU based mesh model / skeleton scriptable animation.*

- struct **gazebo::physics::TrajectoryInfo**

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::common**

  Common namespace.

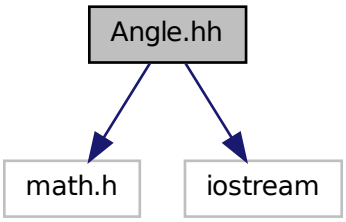
- namespace **gazebo::physics**

  namespace for physics

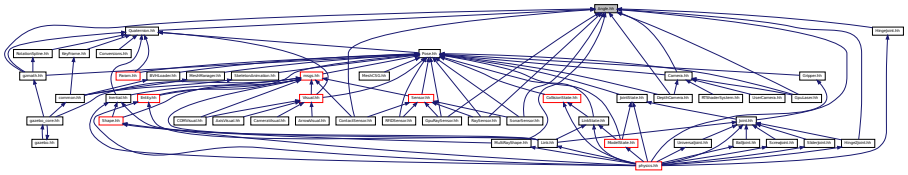
11.2 Angle.hh File Reference

```
#include <math.h>
#include <iostream>
```

Include dependency graph for Angle.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::math::Angle**

  An angle and related functions.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::math**

  Math namespace.

Macros

- #define **GZ_DTOR**(d) ((d) ∗ M_PI / 180)

  Converts degrees to radians.
- #define **GZ_NORMALIZE**(a) (atan2(sin(a), cos(a)))

  Macro tha normalizes an angle in the range -Pi to Pi.
- #define **GZ_RTOD**(r) ((r) ∗ 180 / M_PI)

  Macro that converts radians to degrees.

11.2.1 Macro Definition Documentation

11.2.1.1 #define GZ_DTOR(*d*) ((d) * M_PI / 180)

Converts degrees to radians.

Parameters

in	*degrees*	

Returns

radians

11.2.1.2 #define GZ_NORMALIZE(*a*) (atan2(sin(a), cos(a)))

Macro tha normalizes an angle in the range -Pi to Pi.

Parameters

in	*angle*	

Returns

the angle, in range

11.2.1.3 #define GZ_RTOD(*r*) ((r) * 180 / M_PI)

Macro that converts radians to degrees.

Parameters

in	*radians*	

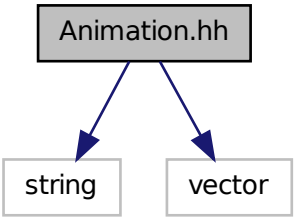
Returns

degrees

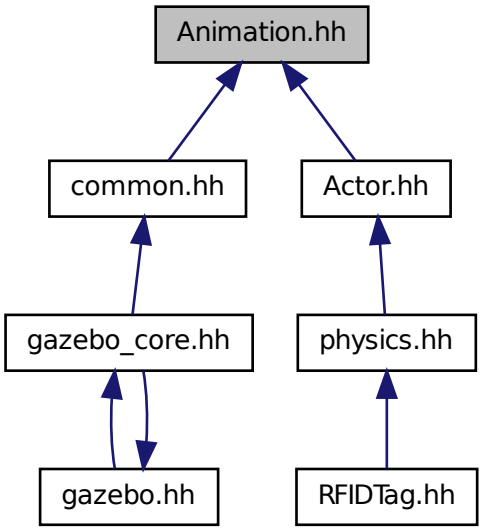
11.3 Animation.hh File Reference

```
#include <string>
#include <vector>
```

Include dependency graph for Animation.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::Animation**

  Manages an animation, which is a collection of keyframes and the ability to interpolate between the keyframes.
- class **gazebo::common::NumericAnimation**

  A numeric animation.
- class **gazebo::common::PoseAnimation**

  A pose animation.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

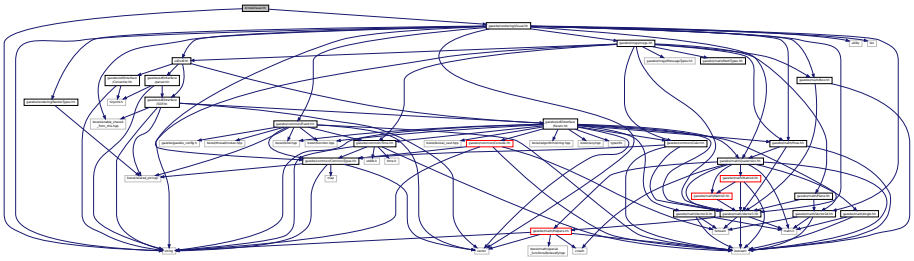
  Common namespace.
- namespace **gazebo::math**

  Math namespace.

11.4 ArrowVisual.hh File Reference

```
#include <string>
#include "gazebo/rendering/Visual.hh"
```

Include dependency graph for ArrowVisual.hh:



Classes

- class **gazebo::rendering::ArrowVisual**

  Basic arrow visualization.

Namespaces

- namespace **gazebo**

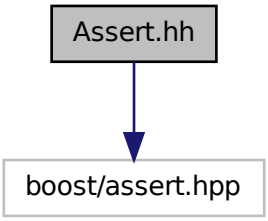
  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.
- namespace **ogre**

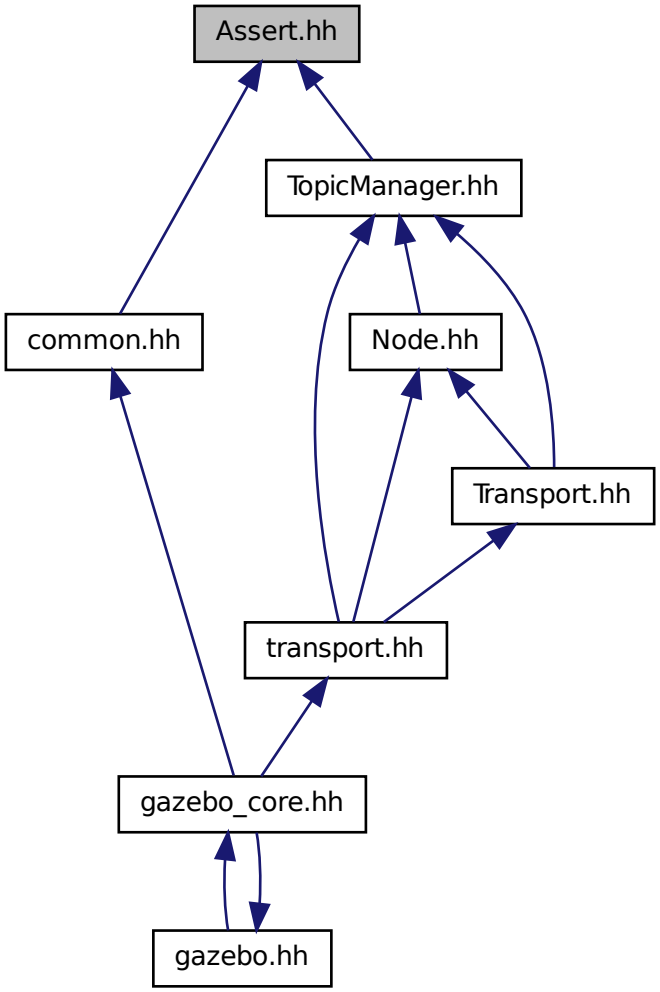
11.5 Assert.hh File Reference

```
#include <boost/assert.hpp>
```

Include dependency graph for Assert.hh:



This graph shows which files directly or indirectly include this file:



Macros

- #define **GZ_ASSERT**(_expr, _msg) BOOST_ASSERT_MSG(_expr, _msg)

  This macro define the standard way of launching an exception inside gazebo.

11.5.1 Macro Definition Documentation

11.5.1.1 #define GZ_ASSERT(*_expr, _msg*) BOOST_ASSERT_MSG(_expr, _msg)

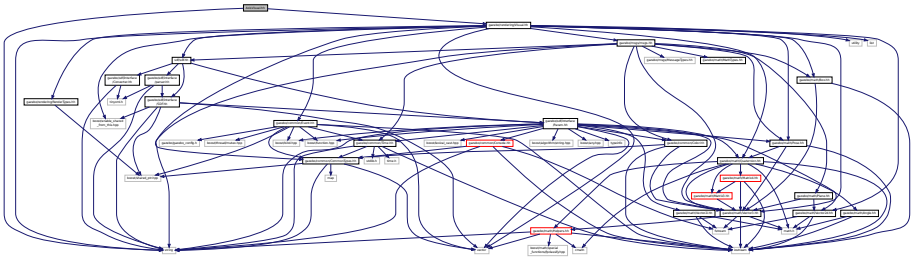
This macro define the standard way of launching an exception inside gazebo.

Referenced by gazebo::transport::TopicManager::Advertise().

11.6 AxisVisual.hh File Reference

```
#include <string>
#include "gazebo/rendering/Visual.hh"
```

Include dependency graph for AxisVisual.hh:



Classes

- class **gazebo::rendering::AxisVisual**

  Basic axis visualization.

Namespaces

- namespace **gazebo**

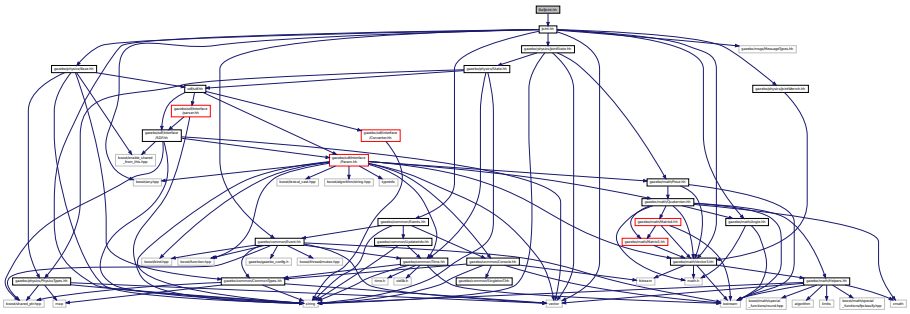
  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.

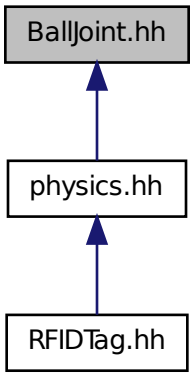
11.7 BallJoint.hh File Reference

```
#include "Joint.hh"
```

Include dependency graph for BallJoint.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::BallJoint**< **T** >

  Base *(*p. *141) class for a ball joint.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

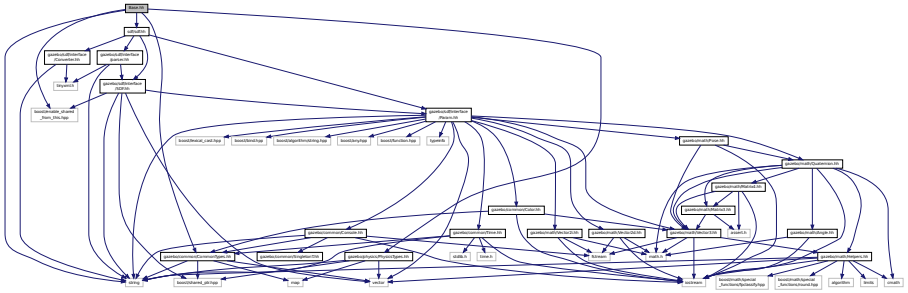
- namespace **gazebo::physics**

  namespace for physics

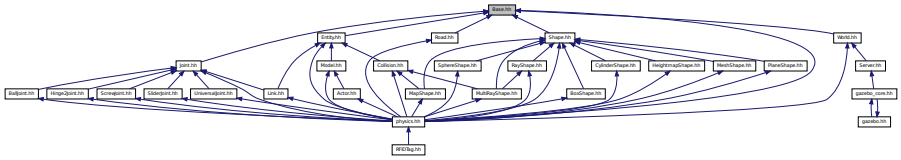
11.8 Base.hh File Reference

```
#include <boost/enable_shared_from_this.hpp>
#include <string>
#include <sdf/sdf.hh>
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/physics/PhysicsTypes.hh"
```

Include dependency graph for Base.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::Base**

  Base *(*p. *141) class for most physics classes.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::physics**

  namespace for physics

Variables

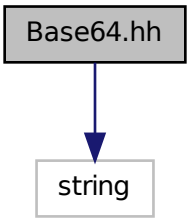
- static std::string **gazebo::physics::EntityTypename** [ ]

  String names for the different entity types.

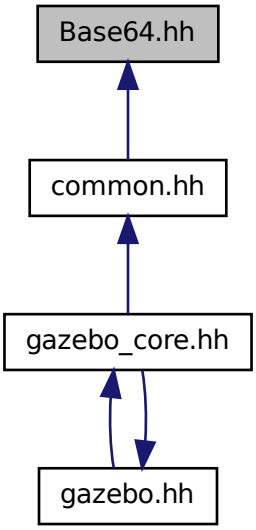
11.9 Base64.hh File Reference

```
#include <string>
```

Include dependency graph for Base64.hh:



This graph shows which files directly or indirectly include this file:



Functions

- std::string **Base64Decode** (const std::string &_encodedString)

  Decode a base64 string.

- void **Base64Encode** (const char *_bytesToEncode, unsigned int _len, std::string &_result)

  Encode a binary string into base 64.

11.9.1 Function Documentation

11.9.1.1 std::string Base64Decode (const std::string & *_encodedString*)

Decode a base64 string.

Parameters

in	*_encodedString*	A base 64 encoded string.

Returns

The decoded string.

11.9.1.2 void Base64Encode (const char * *_bytesToEncode*, unsigned int *_len*, std::string & *_result*)

Encode a binary string into base 64.

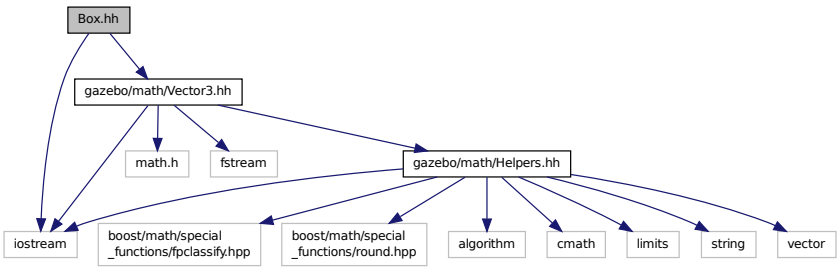
Parameters

in	*_bytesToEncode*	String of bytes to encode.
in	*_len*	Length of _bytesToEncode.
out	*_result*	Based64 string is appended to this string.

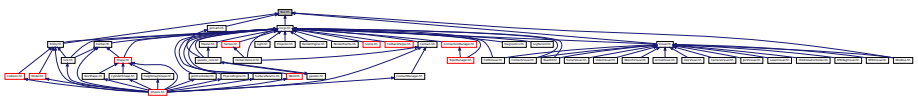
11.10 Box.hh File Reference

```
#include <iostream>
#include "gazebo/math/Vector3.hh"
```

Include dependency graph for Box.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::math::Box**

  Mathematical representation of a box and related functions.

Namespaces

- namespace **gazebo**

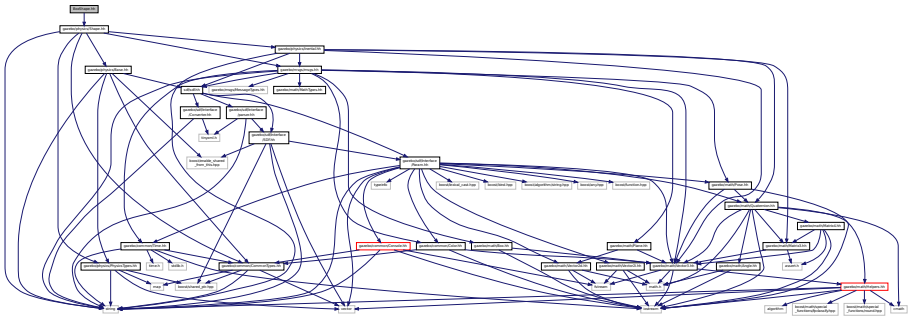
  Forward declarations for the common classes.
- namespace **gazebo::math**

  Math namespace.

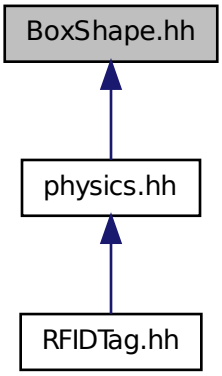
11.11 BoxShape.hh File Reference

```
#include "gazebo/physics/Shape.hh"
```

Include dependency graph for BoxShape.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::BoxShape**

  Box geometry primitive.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

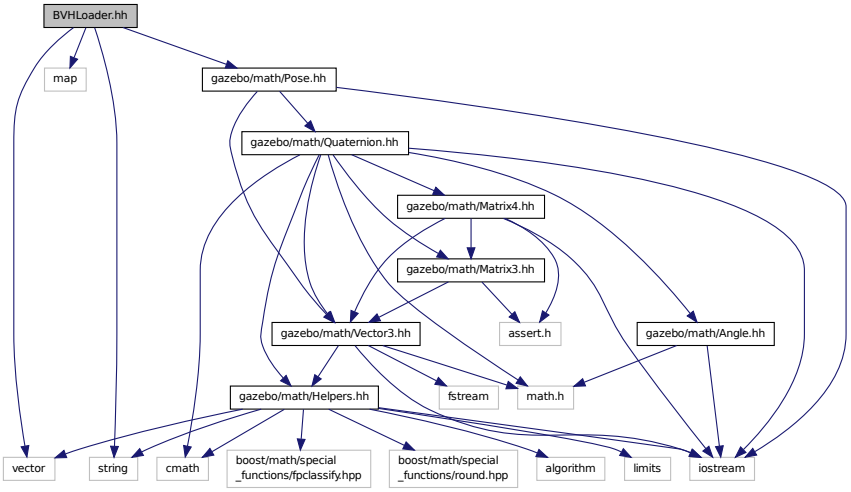
- namespace **gazebo::physics**

  namespace for physics

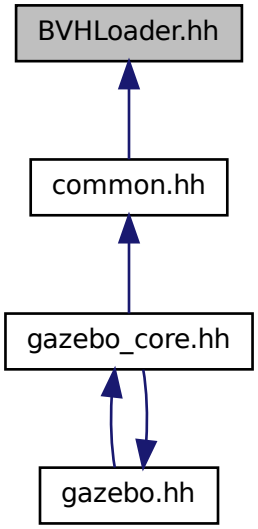
11.12 BVHLoader.hh File Reference

```
#include <vector>
#include <map>
#include <string>
#include "gazebo/math/Pose.hh"
```

Include dependency graph for BVHLoader.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::BVHLoader**

  Handles loading BVH animation files.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

  Common namespace.

Macros

- #define **X_POSITION** 0
- #define **X_ROTATION** 3
- #define **Y_POSITION** 1
- #define **Y_ROTATION** 4
- #define **Z_POSITION** 2
- #define **Z_ROTATION** 5

11.12.1 Macro Definition Documentation

11.12.1.1 #define X_POSITION 0

11.12.1.2 #define X_ROTATION 3

11.12.1.3 #define Y_POSITION 1

11.12.1.4 #define Y_ROTATION 4

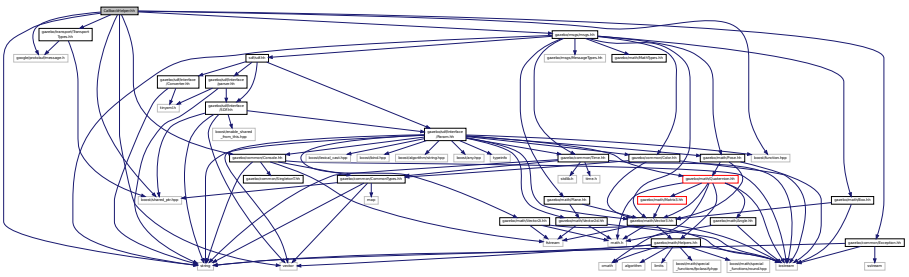
11.12.1.5 #define Z_POSITION 2

11.12.1.6 #define Z_ROTATION 5

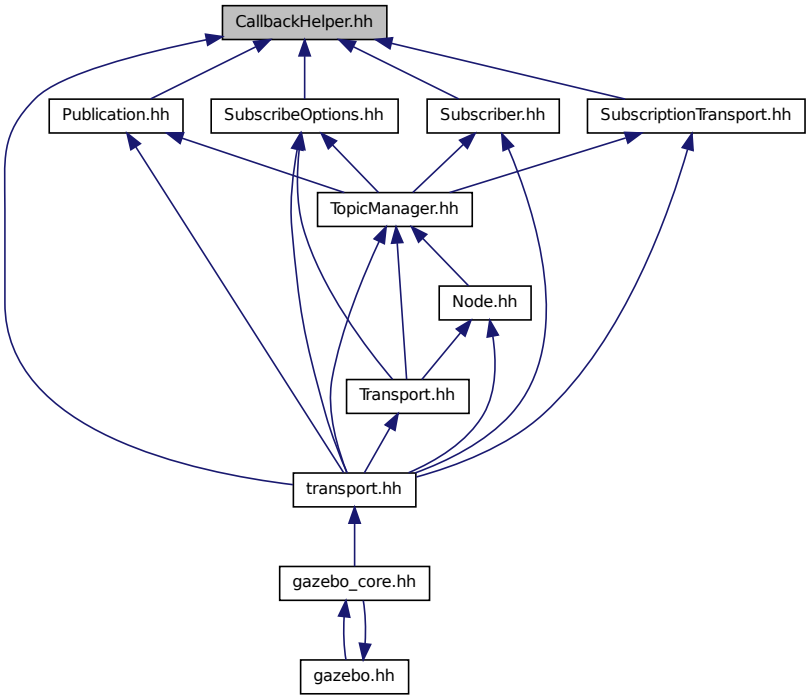
11.13 CallbackHelper.hh File Reference

```
#include <google/protobuf/message.h>
#include <boost/function.hpp>
#include <boost/shared_ptr.hpp>
#include <vector>
#include <string>
#include "gazebo/common/Console.hh"
#include "gazebo/msgs/msgs.hh"
#include "gazebo/common/Exception.hh"
#include "gazebo/transport/TransportTypes.hh"
```

Include dependency graph for CallbackHelper.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::transport::CallbackHelper**

  A helper class to handle callbacks when messages arrive.
- class **gazebo::transport::CallbackHelperT< M >**

  Callback helper Template.
- class **gazebo::transport::RawCallbackHelper**

  Used to connect publishers to subscribers, where the subscriber wants the raw data from the publisher.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::transport**

Typedefs

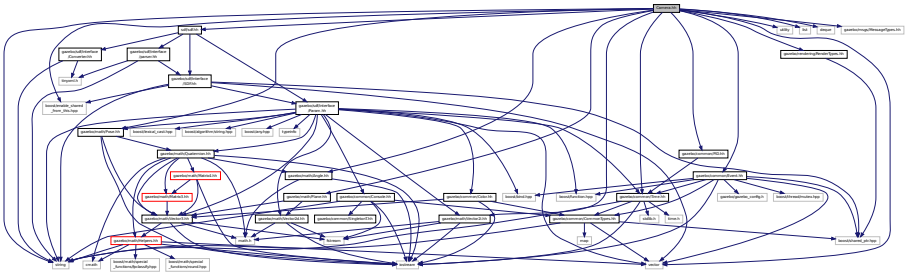
- typedef boost::shared_ptr
  < CallbackHelper > **gazebo::transport::CallbackHelperPtr**

  boost shared pointer to ***transport::CallbackHelper*** *(*p. 161*)*

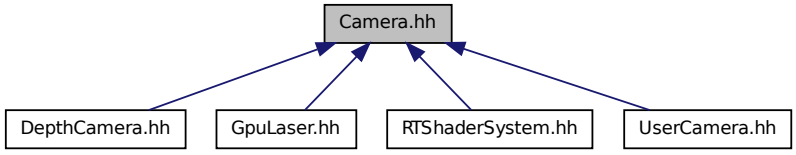
11.14 Camera.hh File Reference

```
#include <boost/enable_shared_from_this.hpp>
#include <string>
#include <utility>
#include <list>
#include <vector>
#include <deque>
#include <sdf/sdf.hh>
#include "gazebo/common/Event.hh"
#include "gazebo/common/PID.hh"
#include "gazebo/common/Time.hh"
#include "gazebo/math/Angle.hh"
#include "gazebo/math/Pose.hh"
#include "gazebo/math/Plane.hh"
#include "gazebo/math/Vector2i.hh"
#include "gazebo/msgs/MessageTypes.hh"
#include "gazebo/rendering/RenderTypes.hh"
```

Include dependency graph for Camera.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::rendering::Camera**
  Basic camera sensor.

Namespaces

- namespace **gazebo**

Forward declarations for the common classes.

- namespace **gazebo::rendering**

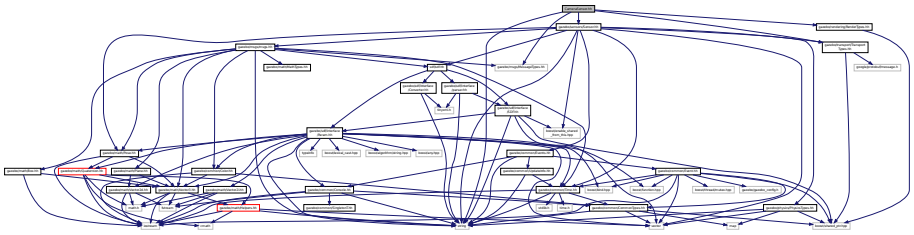
  Rendering namespace.

- namespace **Ogre**

11.15 CameraSensor.hh File Reference

```
#include <string>
#include "gazebo/sensors/Sensor.hh"
#include "gazebo/msgs/MessageTypes.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/rendering/RenderTypes.hh"
```

Include dependency graph for CameraSensor.hh:



Classes

- class **gazebo::sensors::CameraSensor**

  Basic camera sensor.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

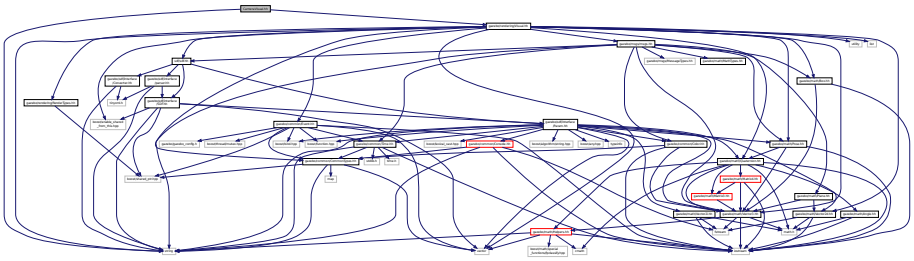
- namespace **gazebo::sensors**

  Sensors namespace.

11.16 CameraVisual.hh File Reference

```
#include <string>
#include "gazebo/rendering/Visual.hh"
```

Include dependency graph for CameraVisual.hh:



Classes

- class **gazebo::rendering::CameraVisual**

  Basic camera visualization.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

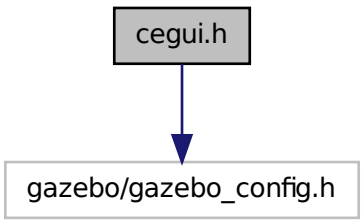
- namespace **gazebo::rendering**

  Rendering namespace.

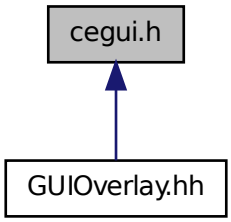
11.17 cegui.h File Reference

```
#include "gazebo/gazebo_config.h"
```

Include dependency graph for cegui.h:



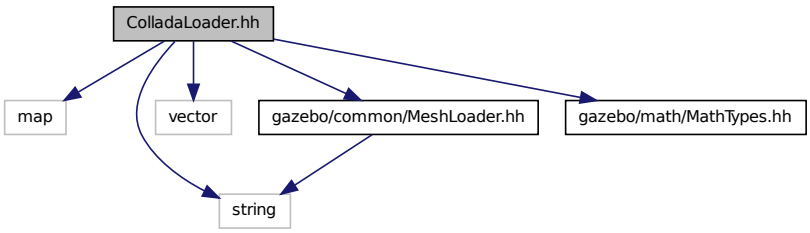
This graph shows which files directly or indirectly include this file:



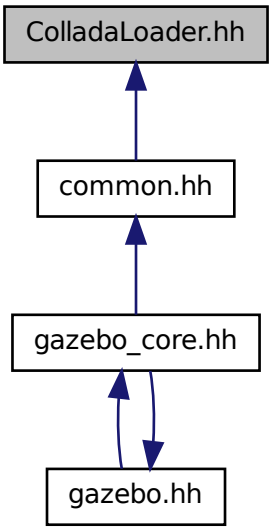
11.18 ColladaLoader.hh File Reference

```
#include <map>
#include <string>
#include <vector>
#include "gazebo/common/MeshLoader.hh"
#include "gazebo/math/MathTypes.hh"
```

Include dependency graph for ColladaLoader.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::ColladaLoader**

  Class used to load Collada mesh files.

Namespaces

- namespace **gazebo**

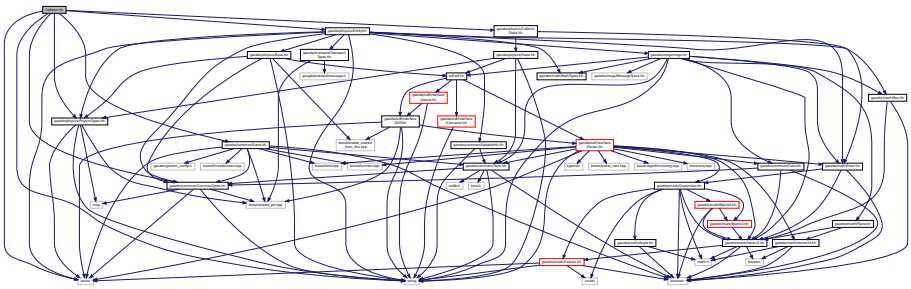
  Forward declarations for the common classes.
- namespace **gazebo::common**

  Common namespace.

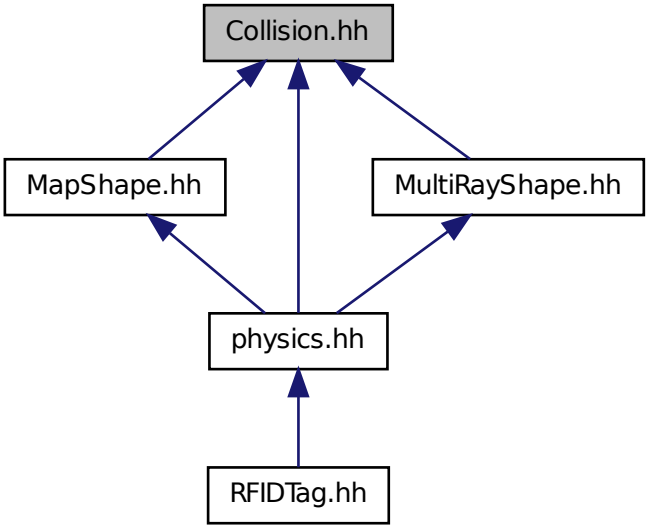
11.19 Collision.hh File Reference

```
#include <string>
#include <vector>
#include "gazebo/common/Event.hh"
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/physics/CollisionState.hh"
#include "gazebo/physics/Entity.hh"
```

Include dependency graph for Collision.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::Collision**

  Base *(p. 141) class for all collision entities.*

Namespaces

- namespace **gazebo**

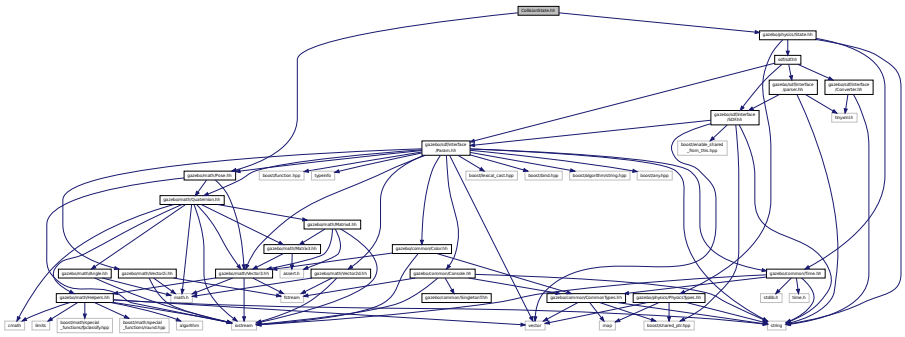
  Forward declarations for the common classes.
- namespace **gazebo::physics**

  namespace for physics

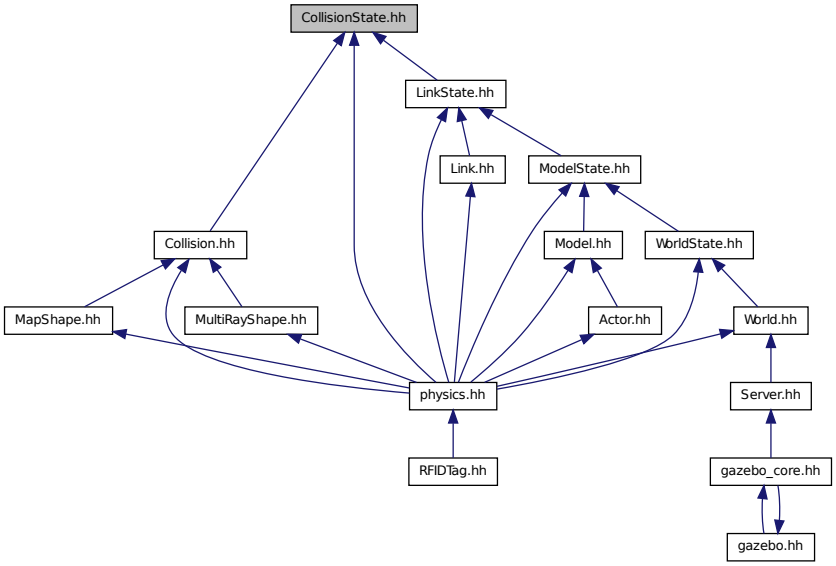
11.20 CollisionState.hh File Reference

```
#include "gazebo/physics/State.hh"
#include "gazebo/math/Pose.hh"
```

Include dependency graph for CollisionState.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::CollisionState**

  *Store state information of a **physics::Collision*** (p. *199*) *object.*

Namespaces

- namespace **gazebo**

Forward declarations for the common classes.

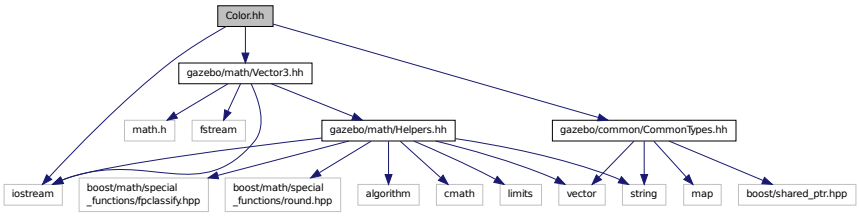
- namespace **gazebo::physics**

  namespace for physics

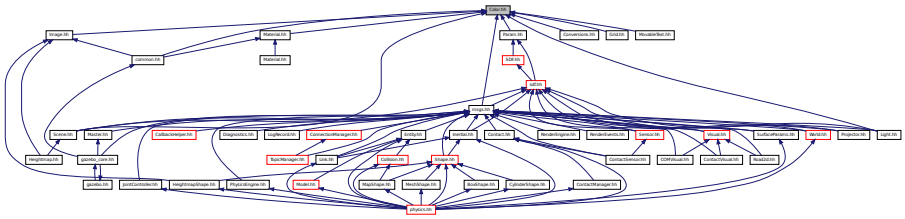
11.21 Color.hh File Reference

```
#include <iostream>
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/math/Vector3.hh"
```

Include dependency graph for Color.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::Color**

  Defines a color.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

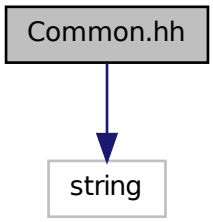
- namespace **gazebo::common**

  Common namespace.

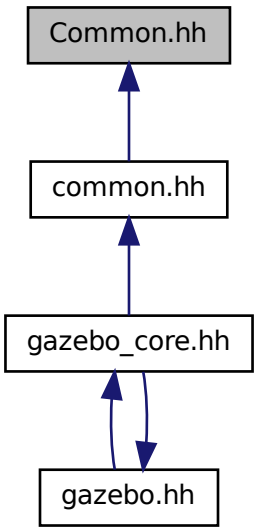
11.22 Common.hh File Reference

```
#include <string>
```

Include dependency graph for Common.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

Common namespace.

Functions

- void **gazebo::common::add_search_path_suffix** (const std::string &_suffix)

  *add path prefix to **common::SystemPaths*** (p. *824*)
- std::string **gazebo::common::find_file** (const std::string &_file)

  *search for file in **common::SystemPaths*** (p. *824*)
- std::string **gazebo::common::find_file** (const std::string &_file, bool _searchLocalPath)

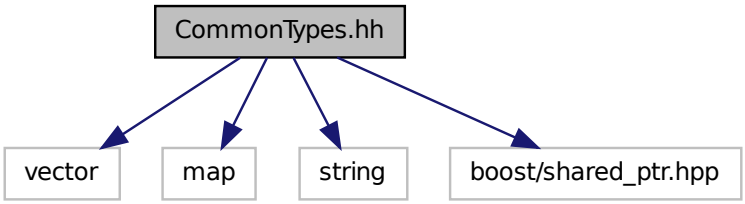
  *search for file in **common::SystemPaths*** (p. *824*)
- std::string **gazebo::common::find_file_path** (const std::string &_file)

  *search for a file in **common::SystemPaths*** (p. *824*)

11.23 CommonTypes.hh File Reference

```
#include <vector>
#include <map>
#include <string>
#include <boost/shared_ptr.hpp>
```

Include dependency graph for CommonTypes.hh:



This graph shows which files directly or indirectly include this file:

Classes

- class **ParamT**< **T** >

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

  Common namespace.
- namespace **gazebo::event**

  Event *(p. 299) namespace.*

Macros

- #define **GAZEBO_DEPRECATED**(version) ()
- #define **GAZEBO_FORCEINLINE**
- #define **NULL** 0

Typedefs

- typedef boost::shared_ptr < Animation > **gazebo::common::AnimationPtr**
- typedef std::vector < ConnectionPtr > **gazebo::event::Connection_V**
- typedef boost::shared_ptr < Connection > **gazebo::event::ConnectionPtr**
- typedef boost::shared_ptr < DiagnosticTimer > **gazebo::common::DiagnosticTimerPtr**
- typedef boost::shared_ptr < GUIPlugin > **gazebo::GUIPluginPtr**
- typedef boost::shared_ptr < ModelPlugin > **gazebo::ModelPluginPtr**
- typedef boost::shared_ptr < NumericAnimation > **gazebo::common::NumericAnimationPtr**
- typedef std::vector < common::Param * > **gazebo::common::Param_V**
- typedef boost::shared_ptr < PoseAnimation > **gazebo::common::PoseAnimationPtr**
- typedef boost::shared_ptr < SensorPlugin > **gazebo::SensorPluginPtr**
- typedef std::map< std::string, std::string > **gazebo::common::StrStr_M**
- typedef boost::shared_ptr < SystemPlugin > **gazebo::SystemPluginPtr**
- typedef boost::shared_ptr < VisualPlugin > **gazebo::VisualPluginPtr**
- typedef boost::shared_ptr < WorldPlugin > **gazebo::WorldPluginPtr**

11.23.1 Macro Definition Documentation

11.23.1.1 #define GAZEBO_DEPRECATED(*version*) ()

11.23.1.2 #define GAZEBO_FORCEINLINE

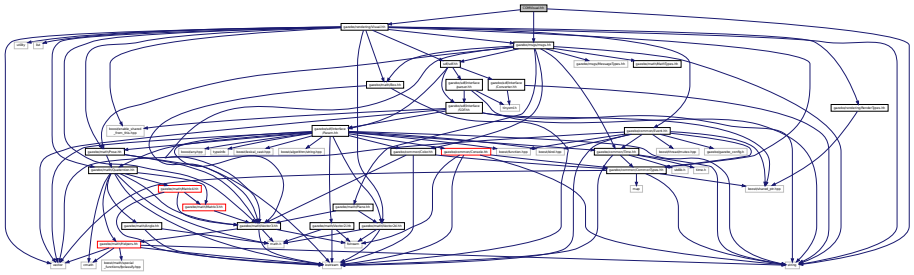
11.23.1.3 #define NULL 0

Referenced by gazebo::transport::TopicManager::Advertise(), gazebo::PluginT< ModelPlugin >::Create(), gazebo::event::EventT< T >::Disconnect(), gazebo::transport::PublishTask::execute(), gazebo::transport::ConnectionReadTask::execute(), gazebo::transport::CallbackHelperT< M >::GetMsgType(), and gazebo::transport::SubscribeOptions::Init().

11.24 COMVisual.hh File Reference

```
#include <string>
#include "gazebo/rendering/Visual.hh"
#include "gazebo/msgs/msgs.hh"
```

Include dependency graph for COMVisual.hh:



Classes

- class **gazebo::rendering::COMVisual**

  Basic Center of Mass visualization.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::rendering**

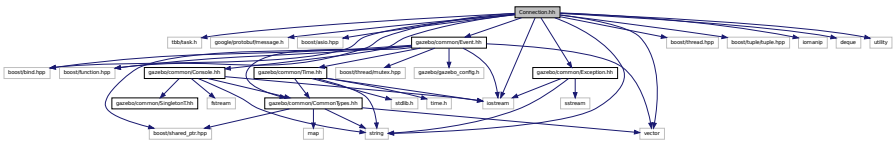
  Rendering namespace.

- namespace **ogre**

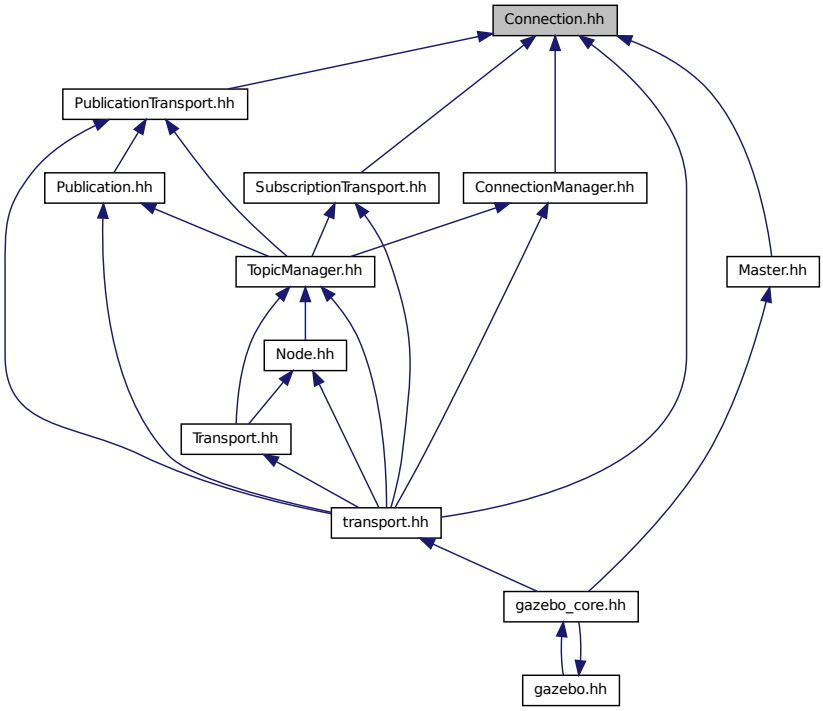
11.25 Connection.hh File Reference

```
#include <tbb/task.h>
#include <google/protobuf/message.h>
#include <boost/asio.hpp>
#include <boost/bind.hpp>
#include <boost/function.hpp>
#include <boost/thread.hpp>
#include <boost/tuple/tuple.hpp>
#include <string>
#include <vector>
#include <iostream>
#include <iomanip>
#include <deque>
#include <utility>
#include "gazebo/common/Event.hh"
#include "gazebo/common/Console.hh"
#include "gazebo/common/Exception.hh"
```

Include dependency graph for Connection.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::transport::Connection**

  Single TCP/IP connection manager.
- class **gazebo::transport::ConnectionReadTask**

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::transport**

Macros

- #define **HEADER_LENGTH** 8

Typedefs

- typedef boost::shared_ptr
  < Connection > **gazebo::transport::ConnectionPtr**

Functions

- bool **gazebo::transport::is_stopped** ()

  Is the transport system stopped?

11.25.1 Macro Definition Documentation

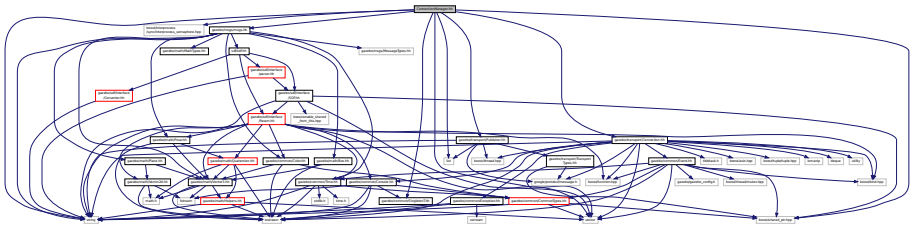
11.25.1.1 #define HEADER_LENGTH 8

Referenced by gazebo::transport::Connection::AsyncRead().

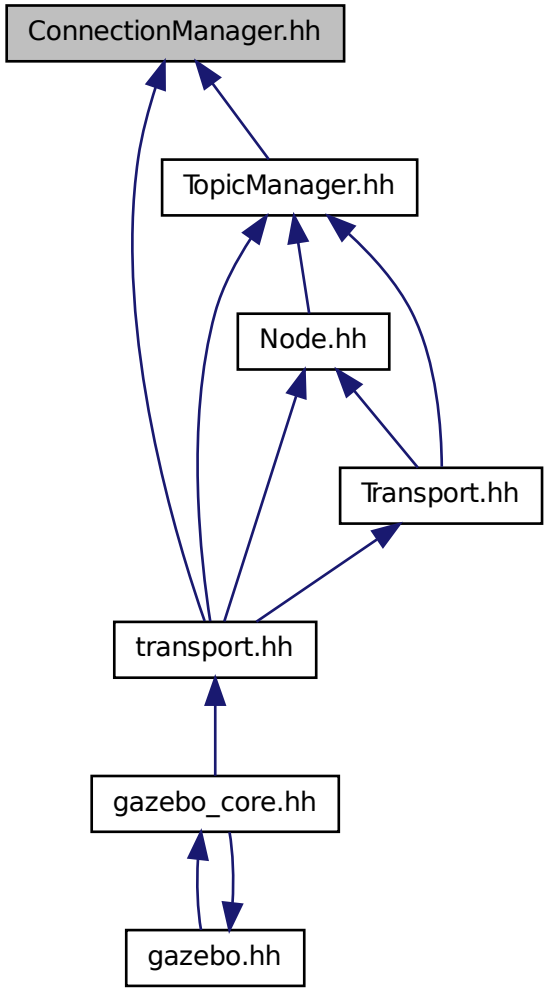
11.26 ConnectionManager.hh File Reference

```
#include <boost/shared_ptr.hpp>
#include <boost/interprocess/sync/interprocess_semaphore.hpp>
#include <string>
#include <list>
#include <vector>
#include "gazebo/msgs/msgs.hh"
#include "gazebo/common/SingletonT.hh"
#include "gazebo/transport/Publisher.hh"
#include "gazebo/transport/Connection.hh"
```

Include dependency graph for ConnectionManager.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::transport::ConnectionManager**

  Manager of connections.

Namespaces

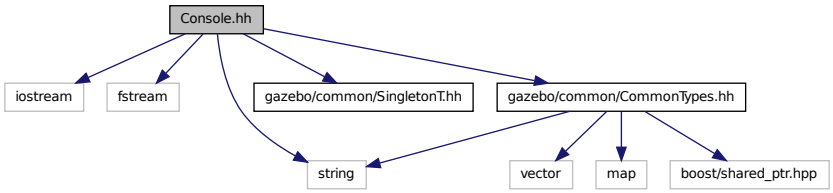
- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::transport**

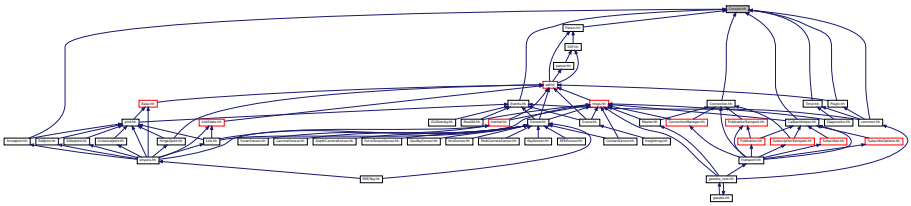
11.27 Console.hh File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include "gazebo/common/SingletonT.hh"
#include "gazebo/common/CommonTypes.hh"
```

Include dependency graph for Console.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::Console**

  Message, error, warning functionality.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

  Common namespace.

Macros

- #define **gzclr_end** "\033[0m"

  End marker.
- #define **gzclr_start**(clr) "\033[1;33m"

  Start marker.

- #define **gzdbg** (**gazebo::common::Console::Instance**()->ColorMsg("Dbg", 36))

  Output a debug message.

- #define **gzerr**

  Output an error message.

- #define **gzlog** (**gazebo::common::Console::Instance**()->Log())

  Output a message to a log file.

- #define **gzmsg** (**gazebo::common::Console::Instance**()->ColorMsg("Msg", 32))

  Output a message.

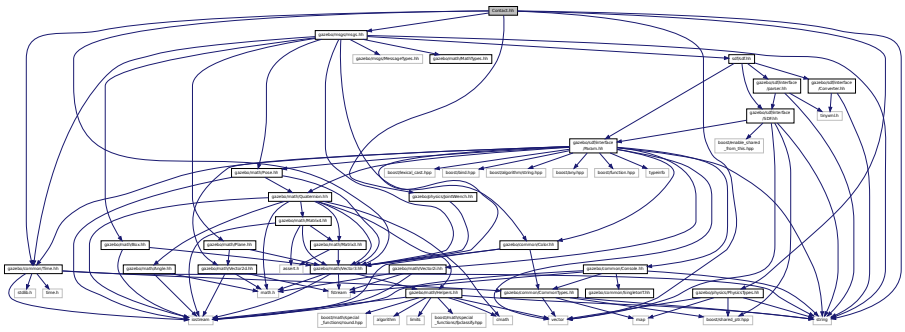
- #define **gzwarn**

  Output a warning message.

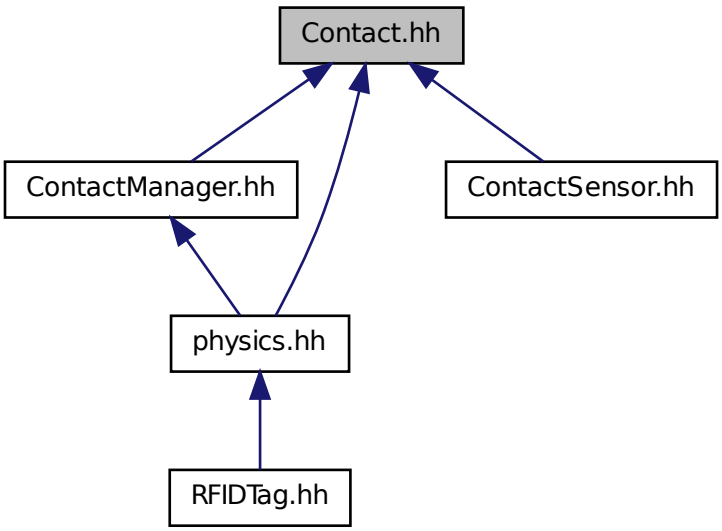
11.28 Contact.hh File Reference

```
#include <vector>
#include <string>
#include "gazebo/msgs/msgs.hh"
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/common/Time.hh"
#include "gazebo/math/Vector3.hh"
#include "gazebo/physics/JointWrench.hh"
```

Include dependency graph for Contact.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::Contact**

  A contact between two collisions.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::physics**

  namespace for physics

Macros

- #define **MAX_COLLIDE_RETURNS** 250
- #define **MAX_CONTACT_JOINTS** 32

11.28.1 Macro Definition Documentation

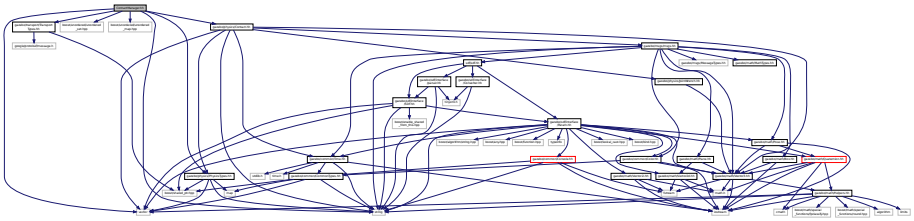
11.28.1.1 #define MAX_COLLIDE_RETURNS 250

11.28.1.2 #define MAX_CONTACT_JOINTS 32

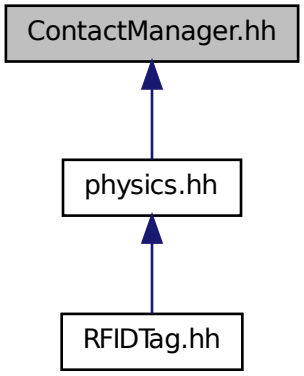
11.29 ContactManager.hh File Reference

```
#include <vector>
#include <string>
#include <boost/unordered/unordered_set.hpp>
#include <boost/unordered/unordered_map.hpp>
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/physics/Contact.hh"
```

Include dependency graph for ContactManager.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::ContactManager**

  Aggregates all the contact information generated by the collision detection engine.

- class **gazebo::physics::ContactPublisher**

  *A custom contact publisher created for each contact filter in the **Contact** (p. 241) Manager.*

Namespaces

- namespace **gazebo**

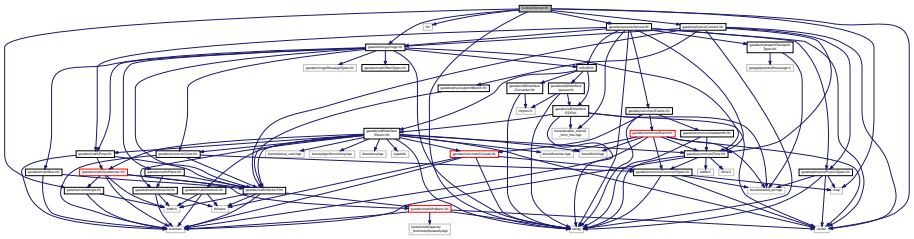
  Forward declarations for the common classes.
- namespace **gazebo::physics**

  namespace for physics

11.30 ContactSensor.hh File Reference

```
#include <vector>
#include <map>
#include <list>
#include <string>
#include "gazebo/msgs/msgs.hh"
#include "gazebo/math/Angle.hh"
#include "gazebo/sensors/Sensor.hh"
#include "gazebo/physics/Contact.hh"
```

Include dependency graph for ContactSensor.hh:



Classes

- class **gazebo::sensors::ContactSensor**

  Contact sensor.

Namespaces

- namespace **gazebo**

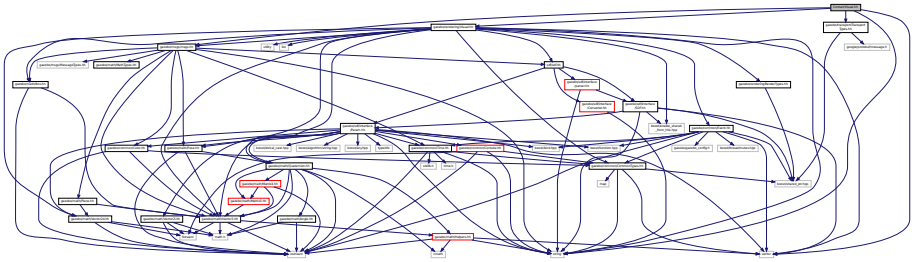
  Forward declarations for the common classes.
- namespace **gazebo::sensors**

  Sensors namespace.

11.31 ContactVisual.hh File Reference

```
#include <string>
#include <vector>
#include "gazebo/rendering/Visual.hh"
#include "gazebo/msgs/msgs.hh"
#include "gazebo/transport/TransportTypes.hh"
```

Include dependency graph for ContactVisual.hh:



Classes

- class **gazebo::rendering::ContactVisual**

  Contact visualization.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.
- namespace **Ogre**

11.32 Conversions.hh File Reference

```
#include "gazebo/rendering/ogre_gazebo.h"
#include "gazebo/common/Color.hh"
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Quaternion.hh"
```

Include dependency graph for Conversions.hh:

Classes

- class **gazebo::rendering::Conversions**

  Conversions (p. *254*) ***Conversions.hh*** (p. *1006*) ***rendering/Conversions.hh*** (p. *1006*).

Namespaces

- namespace **gazebo**

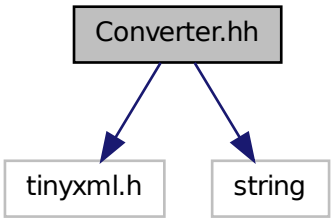
  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.

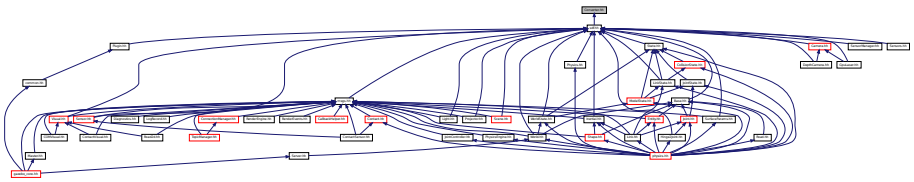
11.33 Converter.hh File Reference

```
#include <tinyxml.h>
#include <string>
```

Include dependency graph for Converter.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **sdf::Converter**

  *Convert from one version of **SDF** (p. 728) to another.*

Namespaces

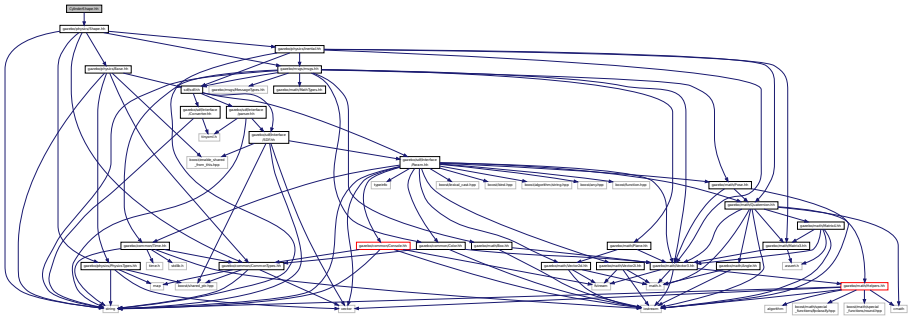
- namespace **sdf**

  namespace for Simulation Description Format parser

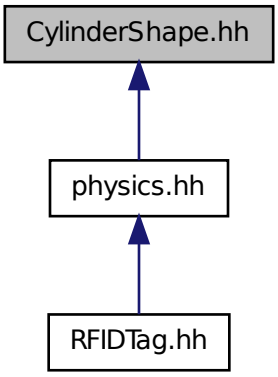
11.34 CylinderShape.hh File Reference

```
#include "gazebo/physics/Shape.hh"
```

Include dependency graph for CylinderShape.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::CylinderShape**

  Cylinder collision.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

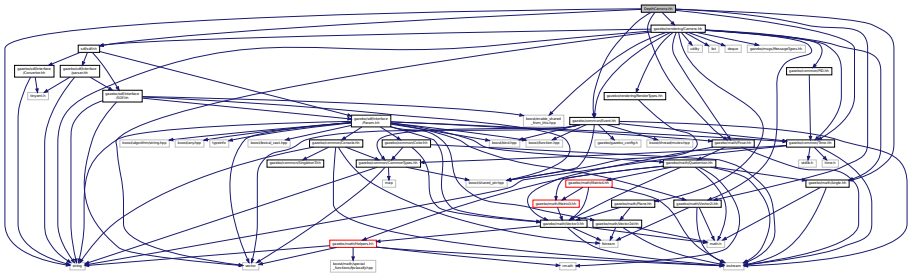
- namespace **gazebo::physics**

  namespace for physics

11.35 DepthCamera.hh File Reference

```
#include <string>
#include <sdf/sdf.hh>
#include "gazebo/common/Event.hh"
#include "gazebo/common/Time.hh"
#include "gazebo/math/Angle.hh"
#include "gazebo/math/Pose.hh"
#include "gazebo/math/Vector2i.hh"
#include "gazebo/rendering/Camera.hh"
```

Include dependency graph for DepthCamera.hh:



Classes

- class **gazebo::rendering::DepthCamera**

  Depth camera used to render depth data into an image buffer.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::rendering**

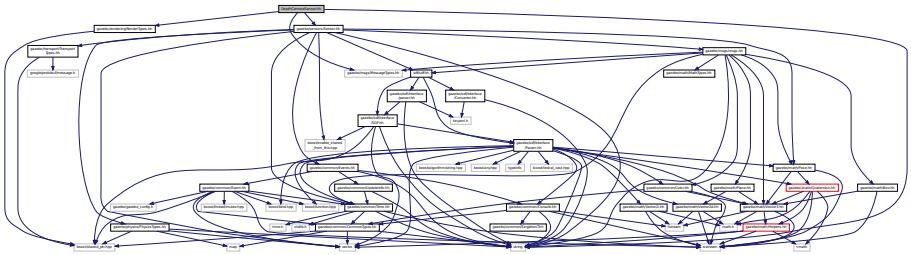
  Rendering namespace.

- namespace **Ogre**

11.36 DepthCameraSensor.hh File Reference

```
#include <string>
#include "gazebo/sensors/Sensor.hh"
#include "gazebo/msgs/MessageTypes.hh"
#include "gazebo/rendering/RenderTypes.hh"
```

Include dependency graph for DepthCameraSensor.hh:



Classes

- class **gazebo::sensors::DepthCameraSensor**

Namespaces

- namespace **gazebo**

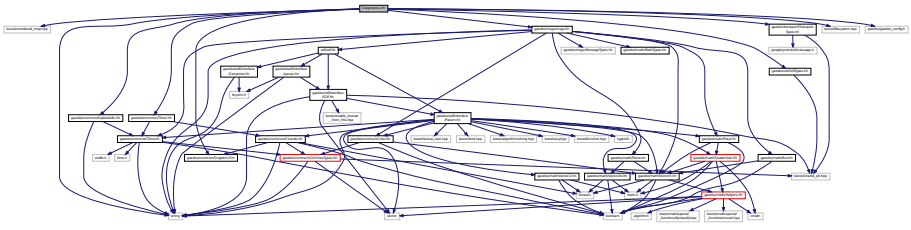
  Forward declarations for the common classes.
- namespace **gazebo::sensors**

  Sensors namespace.

11.37 Diagnostics.hh File Reference

```
#include <boost/unordered_map.hpp>
#include <string>
#include <boost/filesystem.hpp>
#include "gazebo/gazebo_config.h"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/msgs/msgs.hh"
#include "gazebo/common/UpdateInfo.hh"
#include "gazebo/common/SingletonT.hh"
#include "gazebo/common/Timer.hh"
#include "gazebo/util/UtilTypes.hh"
```

Include dependency graph for Diagnostics.hh:



Classes

- class **gazebo::util::DiagnosticManager**

A diagnostic manager class.

- class **gazebo::util::DiagnosticTimer**

  A timer designed for diagnostics.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::util**

Macros

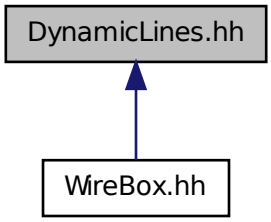
- #define **DIAG_TIMER_LAP**(_name, _prefix) ((void)0)
- #define **DIAG_TIMER_START**(_name) ((void) 0)
- #define **DIAG_TIMER_STOP**(_name) ((void) 0)

11.38 DynamicLines.hh File Reference

```
#include <vector>
#include <string>
#include "gazebo/math/Vector3.hh"
#include "gazebo/rendering/DynamicRenderable.hh"
```

Include dependency graph for DynamicLines.hh:

This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::rendering::DynamicLines**

  Class for drawing lines that can change.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::rendering**

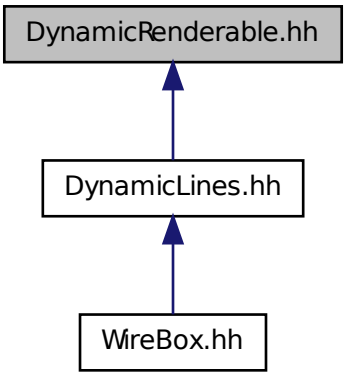
  Rendering namespace.

11.39 DynamicRenderable.hh File Reference

```
#include "gazebo/rendering/ogre_gazebo.h"
#include "gazebo/rendering/RenderTypes.hh"
```

Include dependency graph for DynamicRenderable.hh:

This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::rendering::DynamicRenderable**

  Abstract base class providing mechanisms for dynamically growing hardware buffers.

Namespaces

- namespace **gazebo**

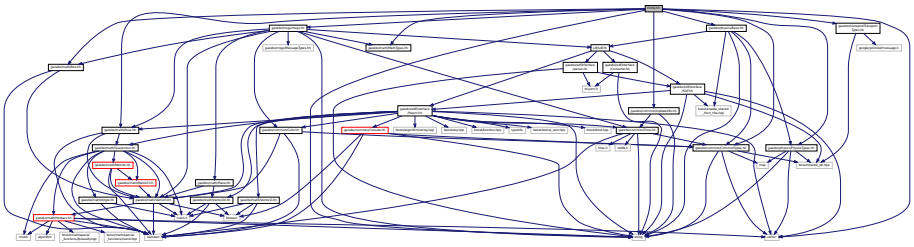
  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.

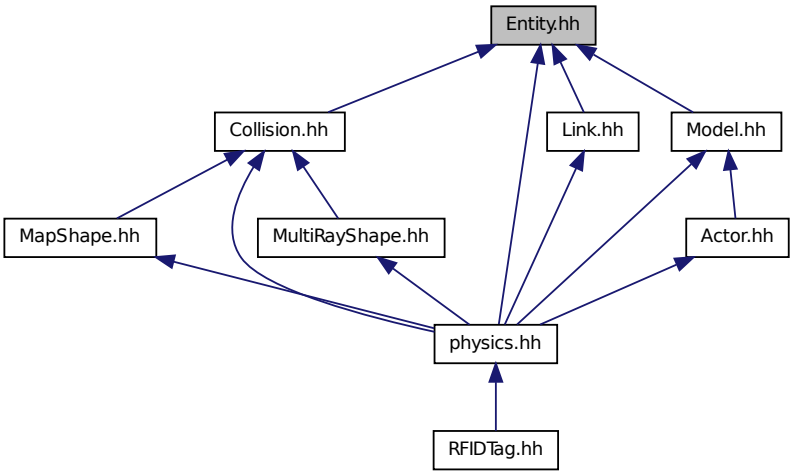
11.40 Entity.hh File Reference

```
#include <string>
#include <vector>
#include "gazebo/msgs/msgs.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/common/UpdateInfo.hh"
#include "gazebo/math/MathTypes.hh"
#include "gazebo/math/Box.hh"
#include "gazebo/math/Pose.hh"
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/physics/Base.hh"
```

Include dependency graph for Entity.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::Entity**

  Base *(p. 141) class for all physics objects in Gazebo.*

Namespaces

- namespace **boost**
- namespace **gazebo**

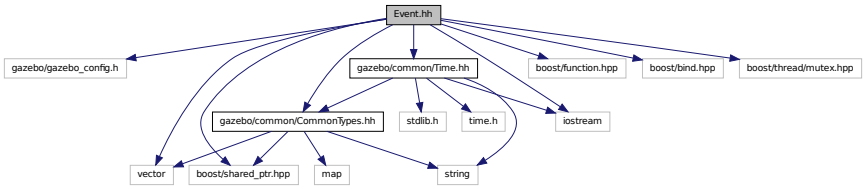
  Forward declarations for the common classes.
- namespace **gazebo::physics**

  namespace for physics

11.41 Event.hh File Reference

```
#include <gazebo/gazebo_config.h>
#include <gazebo/common/Time.hh>
#include <gazebo/common/CommonTypes.hh>
#include <boost/function.hpp>
#include <boost/bind.hpp>
#include <boost/shared_ptr.hpp>
#include <boost/thread/mutex.hpp>
#include <iostream>
#include <vector>
```

Include dependency graph for Event.hh:



This graph shows which files directly or indirectly include this file:

Classes

- class **gazebo::event::Connection**

  A class that encapsulates a connection.
- class **gazebo::event::Event**

  Base class for all events.
- class **gazebo::event::EventT< T >**

  A class for event processing.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

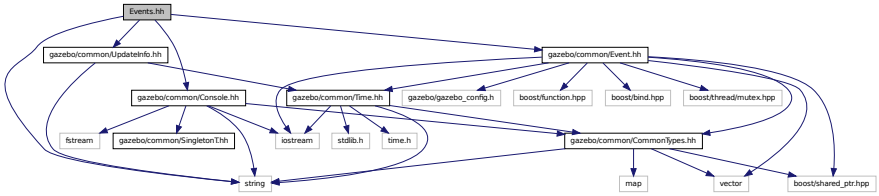
- namespace **gazebo::event**

  Event (p. *299*) *namespace.*

11.42 Events.hh File Reference

```
#include <string>
#include "gazebo/common/Console.hh"
#include "gazebo/common/UpdateInfo.hh"
#include "gazebo/common/Event.hh"
```

Include dependency graph for Events.hh:



This graph shows which files directly or indirectly include this file:

Classes

- class **gazebo::event::Events**

  An ***Event*** (p. *299*) *class to get notifications for simulator events.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

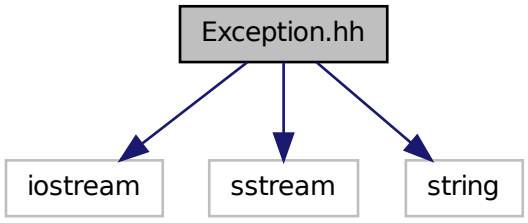
- namespace **gazebo::event**

  Event (p. *299*) *namespace.*

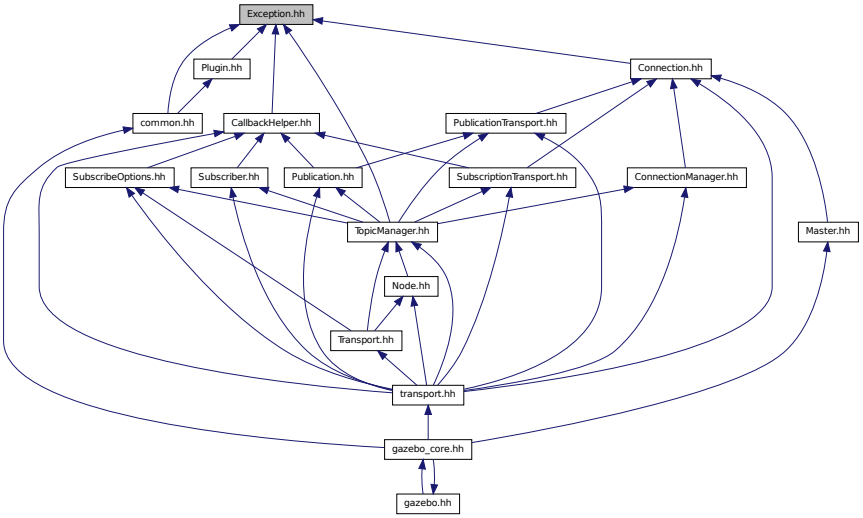
11.43 Exception.hh File Reference

```
#include <iostream>
#include <sstream>
#include <string>
```

Include dependency graph for Exception.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::AssertionInternalError**

  Class for generating Exceptions which come from gazebo assertions.
- class **gazebo::common::Exception**

  Class for generating exceptions.
- class **gazebo::common::InternalError**

  Class for generating Internal Gazebo Errors: those errors which should never happend and represent programming bugs.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

  Common namespace.

Macros

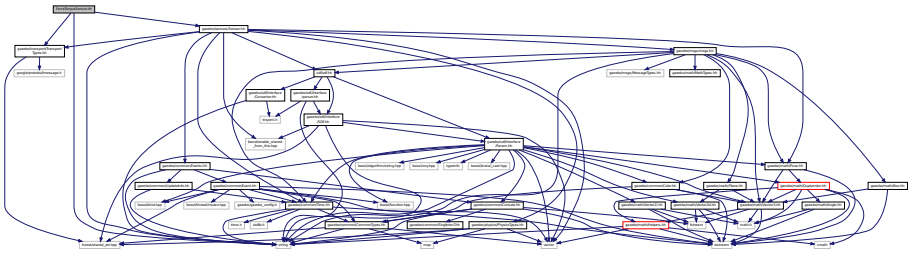
- #define **gzthrow**(msg)

  This macro logs an error to the throw stream and throws an exception that contains the file name and line number.

11.44 ForceTorqueSensor.hh File Reference

```
#include <string>
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/sensors/Sensor.hh"
```

Include dependency graph for ForceTorqueSensor.hh:



Classes

- class **gazebo::sensors::ForceTorqueSensor**

  Sensor *(p. 731) for measure force and torque on a joint.*

Namespaces

- namespace **gazebo**

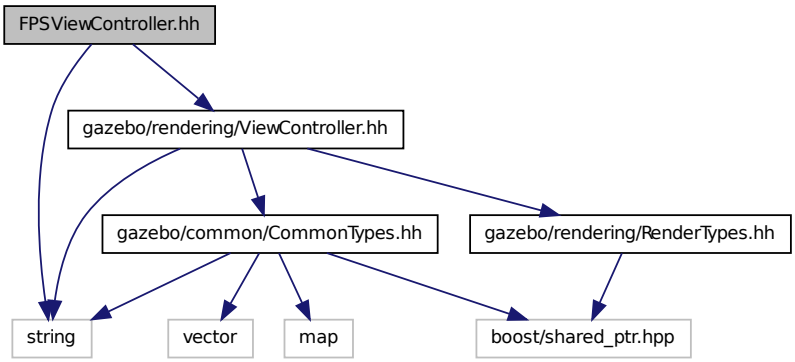
  Forward declarations for the common classes.
- namespace **gazebo::sensors**

  Sensors namespace.

11.45 FPSViewController.hh File Reference

```
#include <string>
#include "gazebo/rendering/ViewController.hh"
```

Include dependency graph for FPSViewController.hh:



Classes

- class **gazebo::rendering::FPSViewController**

  First Person Shooter style view controller.

Namespaces

- namespace **gazebo**

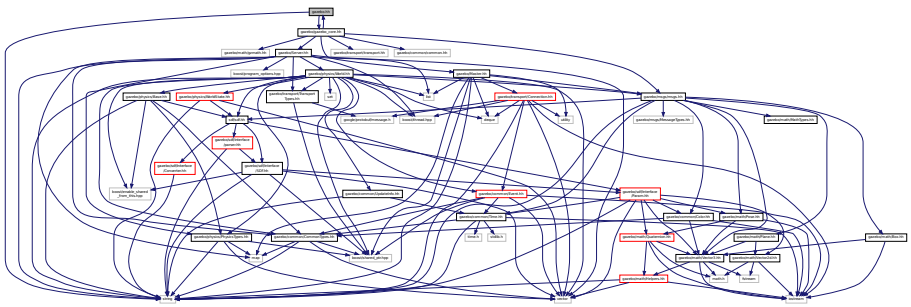
  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.

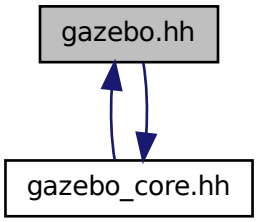
11.46 gazebo.hh File Reference

```
#include <gazebo/gazebo_core.hh>
#include <string>
```

Include dependency graph for gazebo.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

Functions

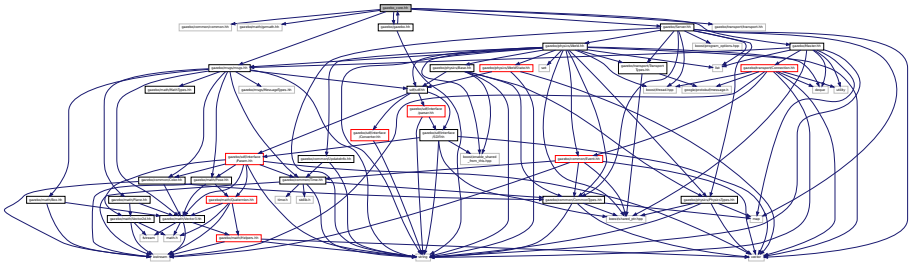
- void **gazebo::add_plugin** (const std::string &_filename)
- std::string **gazebo::find_file** (const std::string &_file)

  Find a file in the gazebo search paths.
- void **gazebo::fini** ()
- bool **gazebo::init** ()
- bool **gazebo::load** (int _argc=0, char **_argv=0)
- void **gazebo::print_version** ()
- void **gazebo::run** ()
- void **gazebo::stop** ()

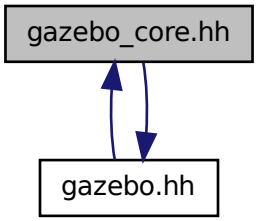
11.47 gazebo_core.hh File Reference

```
#include <gazebo/common/common.hh>
#include <gazebo/math/gzmath.hh>
#include <gazebo/msgs/msgs.hh>
#include <gazebo/transport/transport.hh>
#include <gazebo/Server.hh>
#include <gazebo/Master.hh>
#include <gazebo/gazebo.hh>
```

Include dependency graph for gazebo_core.hh:



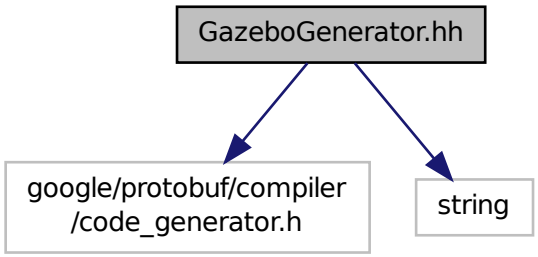
This graph shows which files directly or indirectly include this file:



11.48 GazeboGenerator.hh File Reference

```
#include <google/protobuf/compiler/code_generator.h>
#include <string>
```

Include dependency graph for GazeboGenerator.hh:



Classes

- class **google::protobuf::compiler::cpp::GazeboGenerator**

  *Google protobuf message generator for **gazebo::msgs*** (p. 91).

Namespaces

- namespace **google**
- namespace **google::protobuf**
- namespace **google::protobuf::compiler**
- namespace **google::protobuf::compiler::cpp**

11.49 GpuLaser.hh File Reference

```
#include <string>
#include <vector>
#include <sdf/sdf.hh>
#include "gazebo/rendering/ogre_gazebo.h"
#include "gazebo/rendering/Camera.hh"
#include "gazebo/rendering/RenderTypes.hh"
#include "gazebo/common/Event.hh"
#include "gazebo/common/Time.hh"
#include "gazebo/math/Angle.hh"
#include "gazebo/math/Pose.hh"
#include "gazebo/math/Vector2i.hh"
```

Include dependency graph for GpuLaser.hh:

Classes

- class **gazebo::rendering::GpuLaser**

  GPU based laser distance sensor.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

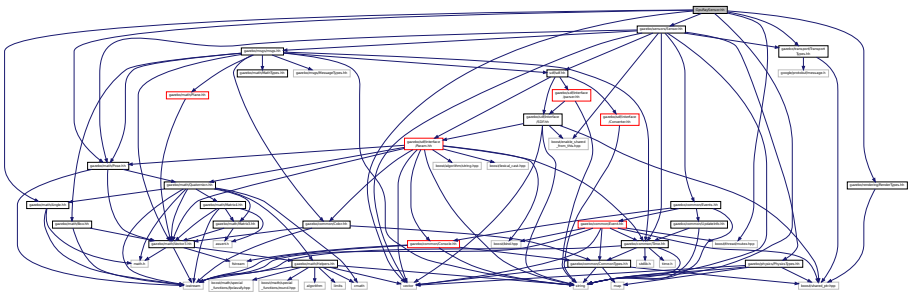
  Common namespace.
- namespace **gazebo::rendering**

  Rendering namespace.
- namespace **Ogre**

11.50 GpuRaySensor.hh File Reference

```
#include <vector>
#include <string>
#include <boost/thread/mutex.hpp>
#include "gazebo/math/Angle.hh"
#include "gazebo/math/Pose.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/sensors/Sensor.hh"
#include "gazebo/rendering/RenderTypes.hh"
```

Include dependency graph for GpuRaySensor.hh:



Classes

- class **gazebo::sensors::GpuRaySensor**

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::sensors**

  Sensors namespace.

11.51 Grid.hh File Reference

```
#include <stdint.h>
#include <vector>
#include <string>
#include "gazebo/rendering/ogre_gazebo.h"
#include "gazebo/common/Color.hh"
```

Include dependency graph for Grid.hh:

Classes

- class **gazebo::rendering::Grid**

  Displays a grid of cells, drawn with lines.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::rendering**

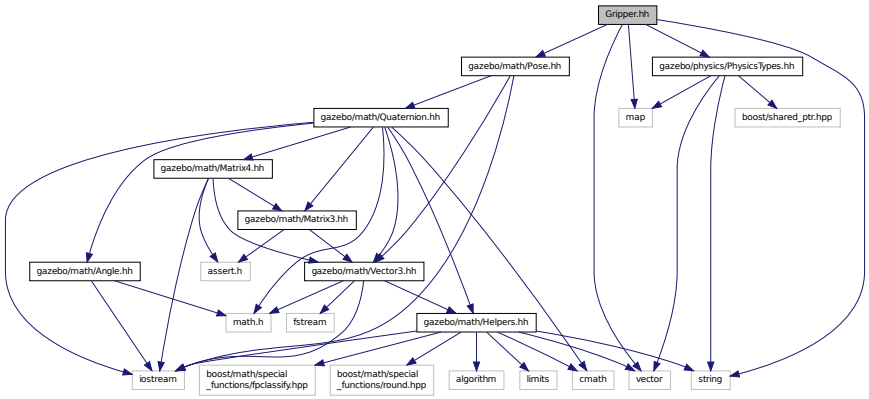
  Rendering namespace.

- namespace **Ogre**

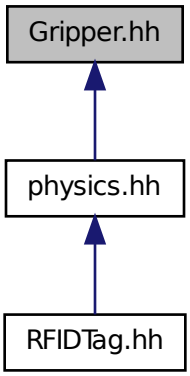
11.52 Gripper.hh File Reference

```
#include <map>
#include <vector>
#include <string>
#include "gazebo/math/Pose.hh"
#include "gazebo/physics/PhysicsTypes.hh"
```

Include dependency graph for Gripper.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::Gripper**

  A gripper abstraction.

Namespaces

- namespace **gazebo**

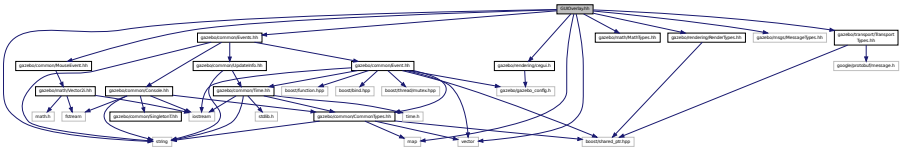
  Forward declarations for the common classes.
- namespace **gazebo::physics**

  namespace for physics

11.53 GUIOverlay.hh File Reference

```
#include <string>
#include <map>
#include <vector>
#include "gazebo/rendering/cegui.h"
#include "gazebo/common/MouseEvent.hh"
#include "gazebo/common/Events.hh"
#include "gazebo/math/MathTypes.hh"
#include "gazebo/rendering/RenderTypes.hh"
#include "gazebo/msgs/MessageTypes.hh"
#include "gazebo/transport/TransportTypes.hh"
```

Include dependency graph for GUIOverlay.hh:



Classes

- class **gazebo::rendering::GUIOverlay**

  A class that creates a CEGUI overlay on a render window.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.
- namespace **Ogre**

11.54 Heightmap.hh File Reference

```
#include <string>
#include <vector>
#include "gazebo/rendering/ogre_gazebo.h"
#include "gazebo/common/Image.hh"
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Vector2d.hh"
#include "gazebo/rendering/Scene.hh"
```

Include dependency graph for Heightmap.hh:

Classes

- class **gazebo::rendering::GzTerrainMatGen**
- class **gazebo::rendering::Heightmap**

  Rendering a terrain using heightmap information.
- class **gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperCg**

  Keeping the CG shader for reference.
- class **gazebo::rendering::GzTerrainMatGen::SM2Profile::ShaderHelperGLSL**

  Utility class to help with generating shaders for GLSL.
- class **gazebo::rendering::GzTerrainMatGen::SM2Profile**

  Shader model 2 profile target.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::rendering**

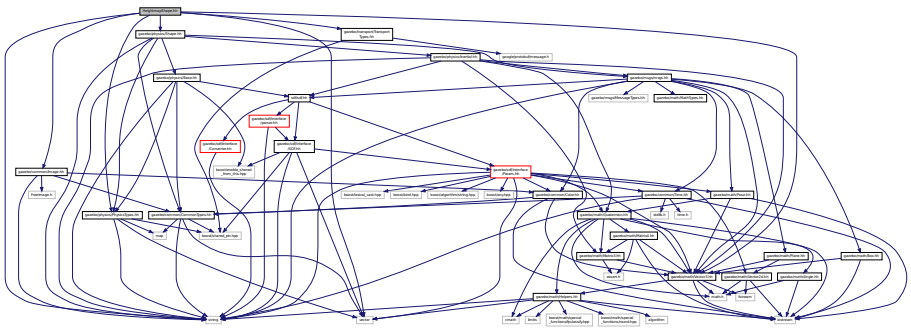
  Rendering namespace.

- namespace **Ogre**

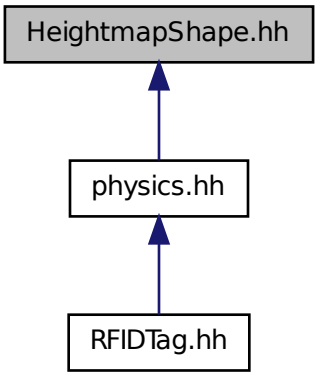
11.55 HeightmapShape.hh File Reference

```
#include <string>
#include <vector>
#include "gazebo/common/Image.hh"
#include "gazebo/math/Vector3.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/physics/Shape.hh"
```

Include dependency graph for HeightmapShape.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::HeightmapShape**

  HeightmapShape *(p. 371) collision shape builds a heightmap from an image.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

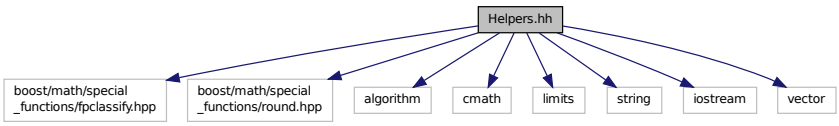
- namespace **gazebo::physics**

  namespace for physics

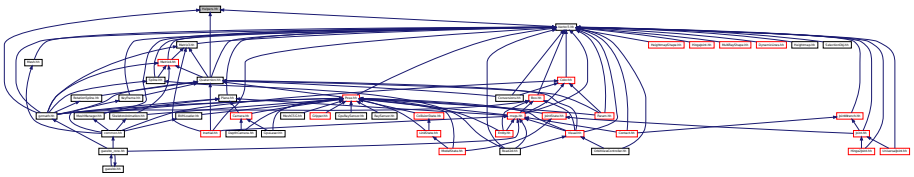
11.56 Helpers.hh File Reference

```
#include <boost/math/special_functions/fpclassify.hpp>
#include <boost/math/special_functions/round.hpp>
#include <algorithm>
#include <cmath>
#include <limits>
#include <string>
#include <iostream>
#include <vector>
```

Include dependency graph for Helpers.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::math**

  Math namespace.

Macros

- #define **GZ_DBL_MAX** std::numeric_limits<double>::max()
- #define **GZ_DBL_MIN** std::numeric_limits<double>::min()
- #define **GZ_FLT_MAX** std::numeric_limits<float>::max()
- #define **GZ_FLT_MIN** std::numeric_limits<float>::min()

Functions

- template<typename T >
  T **gazebo::math::clamp** (T _v, T _min, T _max)

  Simple clamping function.
- template<typename T >
  bool **gazebo::math::equal** (const T &_a, const T &_b, const T &_epsilon=1e-6)

  check if two values are equal, within a tolerance
- bool **gazebo::math::isnan** (float _v)

  check if a float is NaN
- bool **gazebo::math::isnan** (double _v)

  check if a double is NaN
- bool **gazebo::math::isPowerOfTwo** (unsigned int _x)

  is this a power of 2?

- template<typename T >
  T **gazebo::math::max** (const std::vector< T > &_values)

  get the maximum value of vector of values
- template<typename T >
  T **gazebo::math::mean** (const std::vector< T > &_values)

  get mean of vector of values
- template<typename T >
  T **gazebo::math::min** (const std::vector< T > &_values)

  get the minimum value of vector of values
- double **gazebo::math::parseFloat** (const std::string &_input)

  parse string into float
- int **gazebo::math::parseInt** (const std::string &_input)

  parse string into an integer
- template<typename T >
  T **gazebo::math::precision** (const T &_a, const unsigned int &_precision)

  get value at a specified precision
- template<typename T >
  T **gazebo::math::variance** (const std::vector< T > &_values)

  get variance of vector of values

Variables

- static const double **gazebo::math::NAN_D** = std::numeric_limits<double>::quiet_NaN()

  Returns the representation of a quiet not a number (NAN)
- static const int **gazebo::math::NAN_I** = std::numeric_limits<int>::quiet_NaN()

  Returns the representation of a quiet not a number (NAN)

11.56.1 Macro Definition Documentation

11.56.1.1 #define GZ_DBL_MAX std::numeric_limits<double>::max()

11.56.1.2 #define GZ_DBL_MIN std::numeric_limits<double>::min()

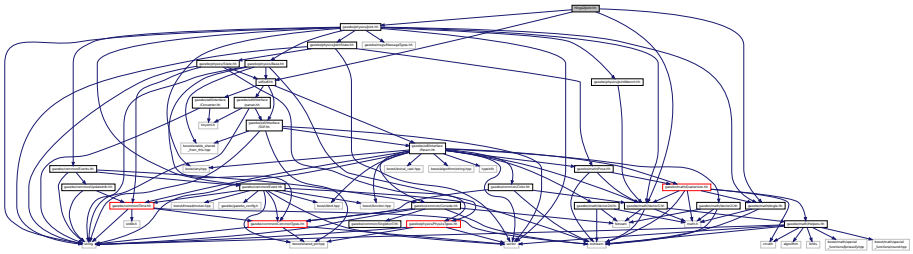
11.56.1.3 #define GZ_FLT_MAX std::numeric_limits<float>::max()

11.56.1.4 #define GZ_FLT_MIN std::numeric_limits<float>::min()

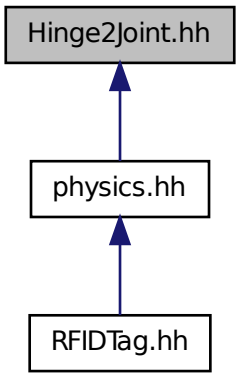
11.57 Hinge2Joint.hh File Reference

```
#include <sdf/sdf.hh>
#include "gazebo/math/Angle.hh"
#include "gazebo/math/Vector3.hh"
#include "gazebo/physics/Joint.hh"
```

Include dependency graph for Hinge2Joint.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::Hinge2Joint**< **T** >

  A two axis hinge joint.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::physics**

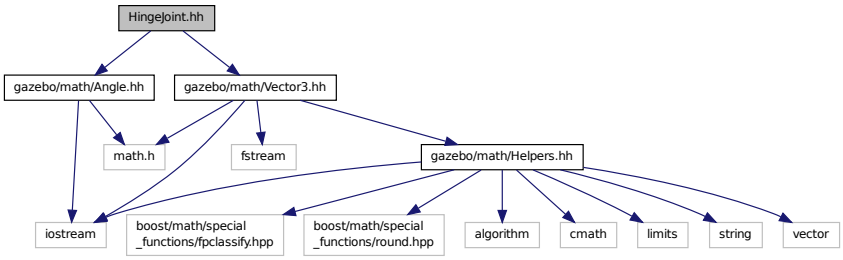
  namespace for physics

11.58 HingeJoint.hh File Reference

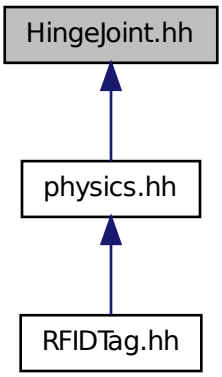
```
#include "gazebo/math/Angle.hh"
```

```
#include "gazebo/math/Vector3.hh"
```

Include dependency graph for HingeJoint.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::HingeJoint**< **T** >

  A single axis hinge joint.

Namespaces

- namespace **gazebo**

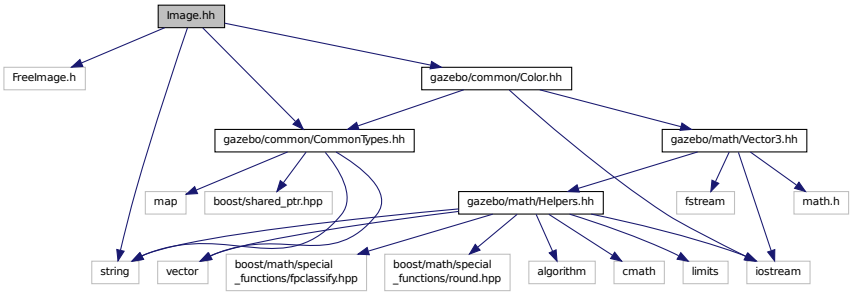
  Forward declarations for the common classes.
- namespace **gazebo::physics**

  namespace for physics

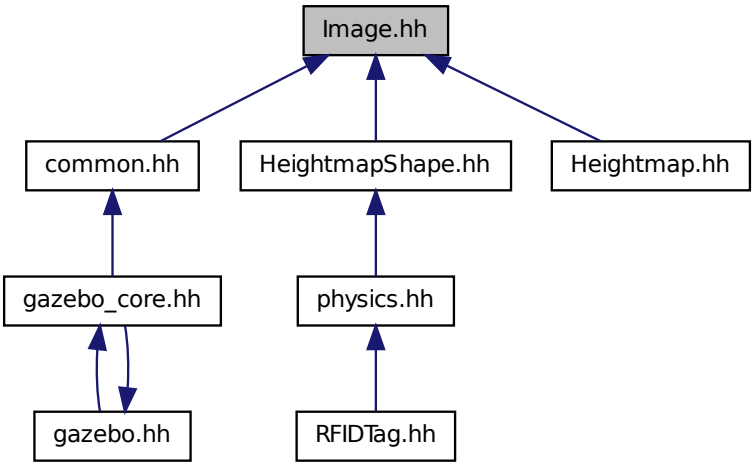
11.59 Image.hh File Reference

```
#include <FreeImage.h>
#include <string>
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/common/Color.hh"
```

Include dependency graph for Image.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::Image**

  Encapsulates an image.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::common**

  Common namespace.

Variables

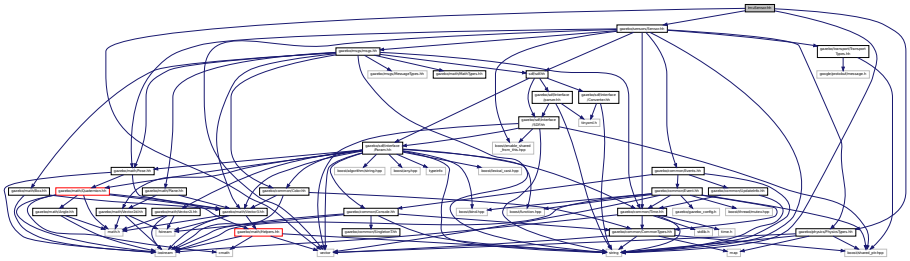
- static std::string **gazebo::common::PixelFormatNames** [ ]

  String names for the pixel formats.

11.60 ImuSensor.hh File Reference

```
#include <vector>
#include <string>
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/sensors/Sensor.hh"
```

Include dependency graph for ImuSensor.hh:



Classes

- class **gazebo::sensors::ImuSensor**

  An IMU sensor.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

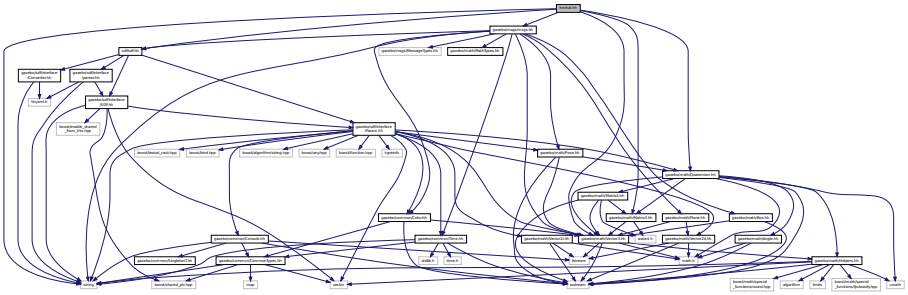
- namespace **gazebo::sensors**

  Sensors namespace.

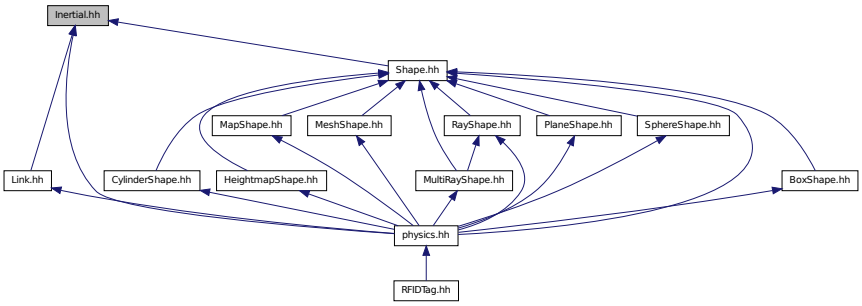
11.61 Inertial.hh File Reference

```
#include <string>
#include <sdf/sdf.hh>
#include "gazebo/msgs/msgs.hh"
#include "gazebo/math/Quaternion.hh"
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Matrix3.hh"
```

Include dependency graph for Inertial.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::Inertial**

  A class for inertial information about a link.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

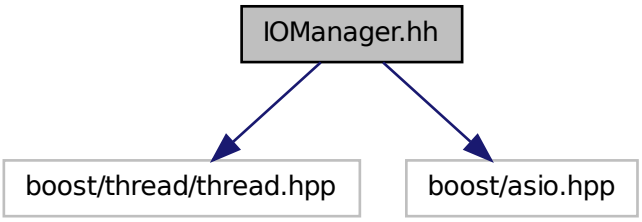
- namespace **gazebo::physics**

  namespace for physics

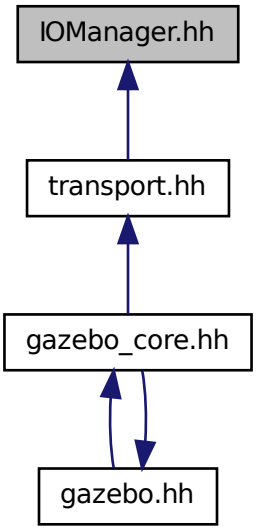
11.62 IOManager.hh File Reference

```
#include <boost/thread/thread.hpp>
#include <boost/asio.hpp>
```

Include dependency graph for IOManager.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::transport::IOManager**

  Manages boost::asio IO.

Namespaces

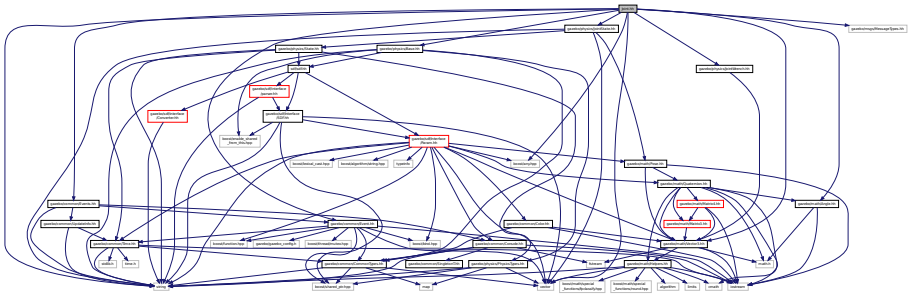
- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::transport**

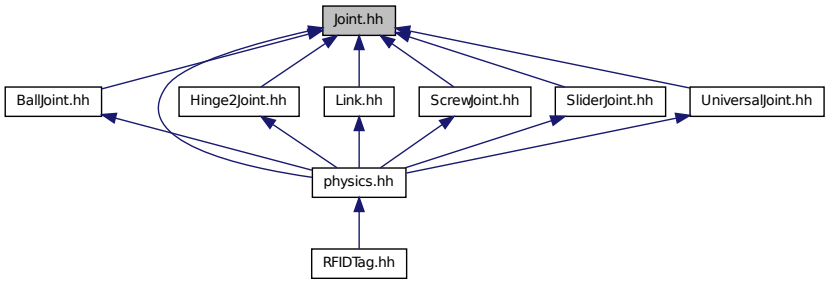
11.63 Joint.hh File Reference

```
#include <string>
#include <vector>
#include <boost/any.hpp>
#include "gazebo/common/Event.hh"
#include "gazebo/common/Events.hh"
#include "gazebo/math/Angle.hh"
#include "gazebo/math/Vector3.hh"
#include "gazebo/msgs/MessageTypes.hh"
#include "gazebo/physics/JointState.hh"
#include "gazebo/physics/Base.hh"
#include "gazebo/physics/JointWrench.hh"
```

Include dependency graph for Joint.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::Joint**

Base *(p. 141) class for all joints.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::physics**

  namespace for physics

Macros

- #define **MAX_JOINT_AXIS** 2

  maximum number of axis per joint anticipated.

11.63.1 Macro Definition Documentation

11.63.1.1 #define MAX_JOINT_AXIS 2

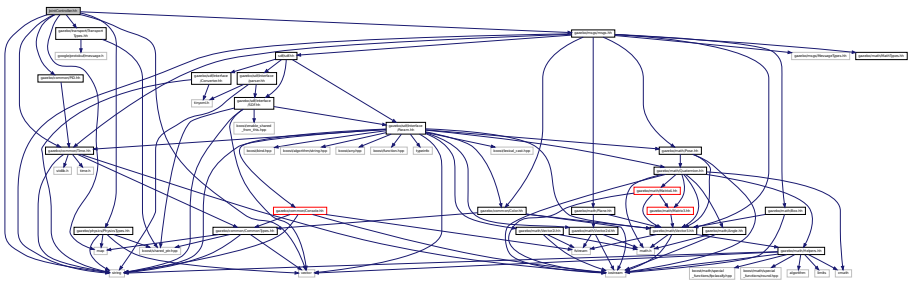
maximum number of axis per joint anticipated.

Currently, this is 2 as 3-axis joints (e.g. ball) actuation, control is not there yet.

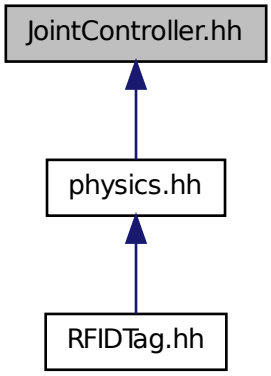
11.64 JointController.hh File Reference

```
#include <map>
#include <string>
#include <vector>
#include "gazebo/common/PID.hh"
#include "gazebo/common/Time.hh"
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/msgs/msgs.hh"
```

Include dependency graph for JointController.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::JointController**

  *A class for manipulating **physics::Joint*** (p. *401*).

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

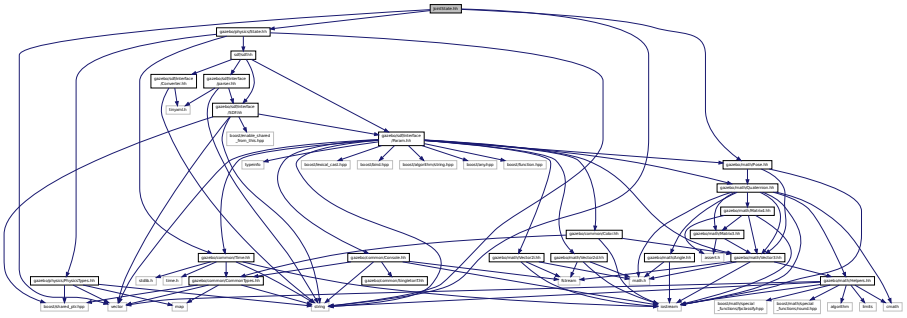
- namespace **gazebo::physics**

  namespace for physics

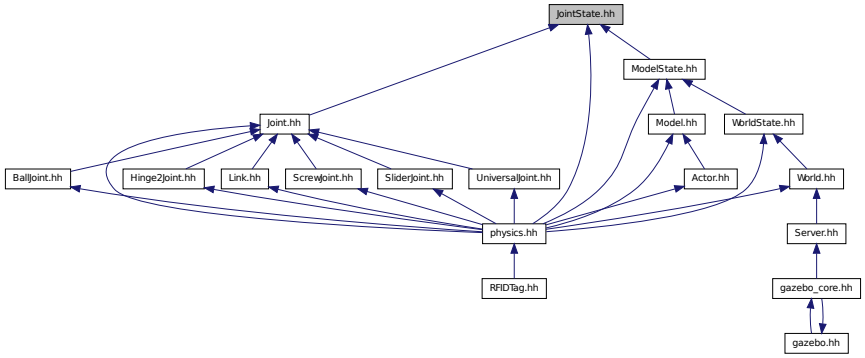
11.65 JointState.hh File Reference

```
#include <vector>
#include <string>
#include "gazebo/physics/State.hh"
#include "gazebo/math/Pose.hh"
```

Include dependency graph for JointState.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::JointState**

  *keeps track of state of a **physics::Joint*** (p. *401*)

Namespaces

- namespace **gazebo**

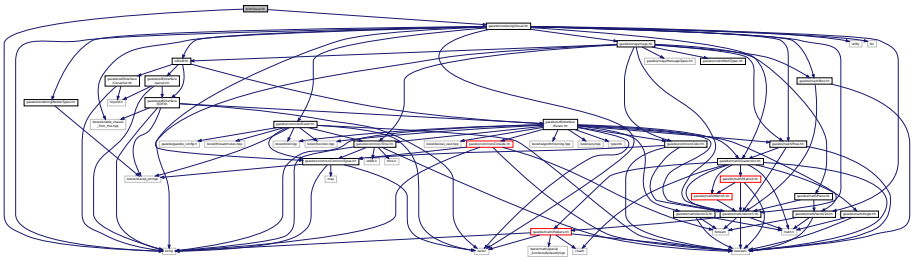
  Forward declarations for the common classes.
- namespace **gazebo::physics**

  namespace for physics

11.66 JointVisual.hh File Reference

```
#include <string>
#include "gazebo/rendering/Visual.hh"
```

Include dependency graph for JointVisual.hh:



Classes

- class **gazebo::rendering::JointVisual**

  Visualization for joints.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

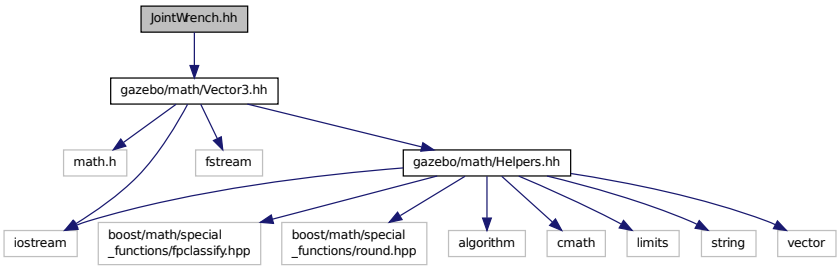
- namespace **gazebo::rendering**

  Rendering namespace.

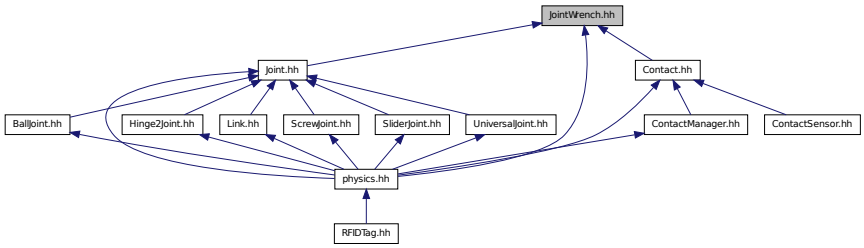
11.67 JointWrench.hh File Reference

```
#include "gazebo/math/Vector3.hh"
```

Include dependency graph for JointWrench.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::JointWrench**

  Wrench information from a joint.

Namespaces

- namespace **gazebo**

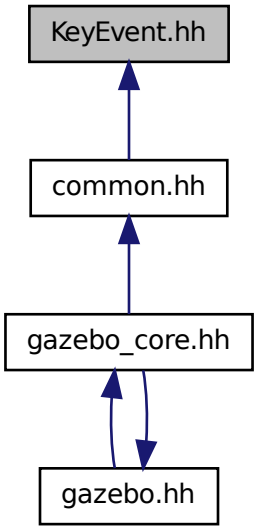
  Forward declarations for the common classes.

- namespace **gazebo::physics**

  namespace for physics

11.68 KeyEvent.hh File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::KeyEvent**

  Generic description of a keyboard event.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

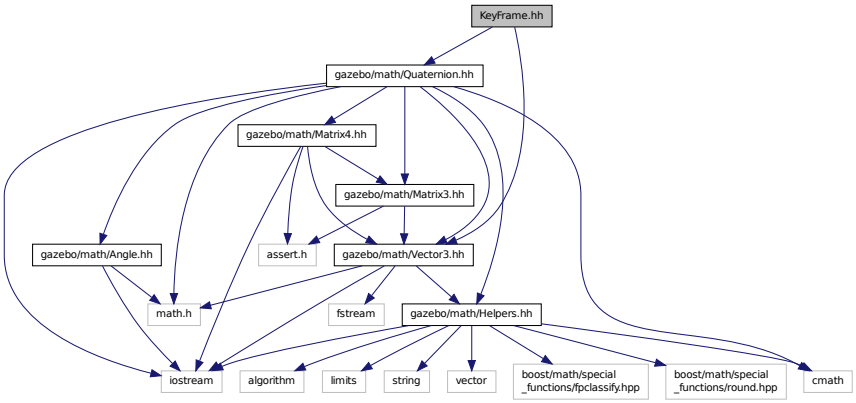
- namespace **gazebo::common**

  Common namespace.

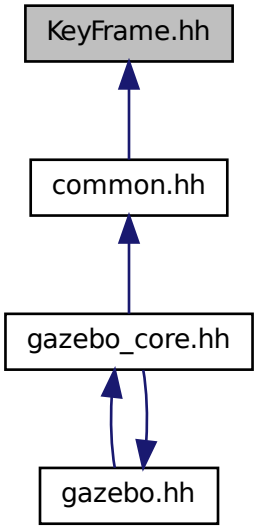
11.69 KeyFrame.hh File Reference

```
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Quaternion.hh"
```

Include dependency graph for KeyFrame.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::KeyFrame**

  A key frame in an animation.
- class **gazebo::common::NumericKeyFrame**

  *A keyframe for a **NumericAnimation** (p. 581).*

- class **gazebo::common::PoseKeyFrame**

  *A keyframe for a **PoseAnimation** (p. 635).*

Namespaces

- namespace **gazebo**

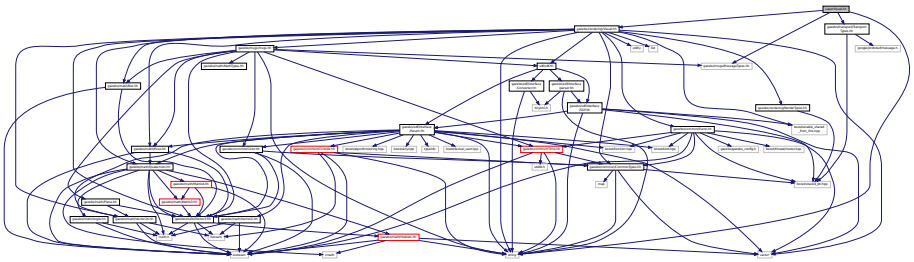
  Forward declarations for the common classes.
- namespace **gazebo::common**

  Common namespace.

11.70 LaserVisual.hh File Reference

```
#include <string>
#include "gazebo/rendering/Visual.hh"
#include "gazebo/msgs/MessageTypes.hh"
#include "gazebo/transport/TransportTypes.hh"
```

Include dependency graph for LaserVisual.hh:



Classes

- class **gazebo::rendering::LaserVisual**

  Visualization for laser data.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::rendering**

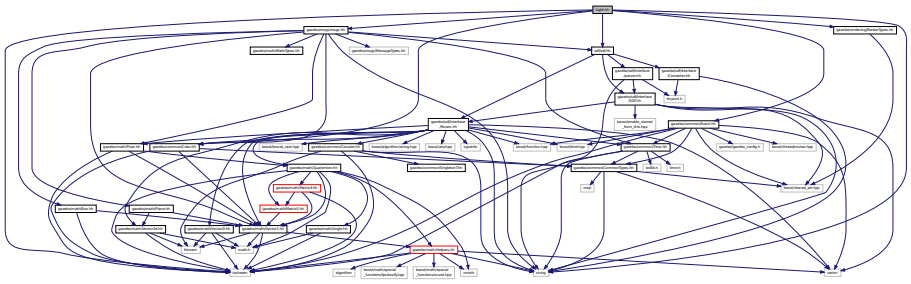
  Rendering namespace.

11.71 Light.hh File Reference

```
#include <string>
```

```
#include <iostream>
#include <sdf/sdf.hh>
#include "gazebo/msgs/msgs.hh"
#include "gazebo/rendering/RenderTypes.hh"
#include "gazebo/common/Event.hh"
#include "gazebo/common/Color.hh"
```

Include dependency graph for Light.hh:



Classes

- class **gazebo::rendering::Light**

  A light source.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::rendering**

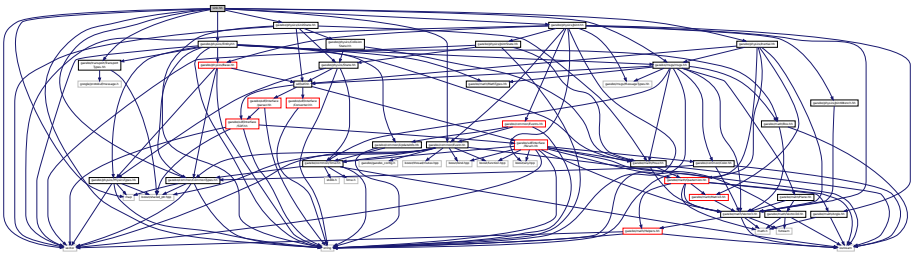
  Rendering namespace.

- namespace **Ogre**

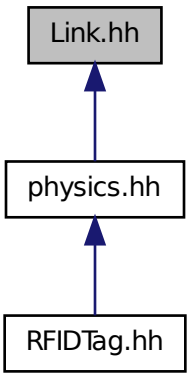
11.72 Link.hh File Reference

```
#include <map>
#include <vector>
#include <string>
#include "gazebo/msgs/msgs.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/common/Event.hh"
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/physics/LinkState.hh"
#include "gazebo/physics/Entity.hh"
#include "gazebo/physics/Inertial.hh"
#include "gazebo/physics/Joint.hh"
```

Include dependency graph for Link.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::Link**

  Link (p. *439*) *class defines a rigid body entity, containing information on inertia, visual and collision properties of a rigid body.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

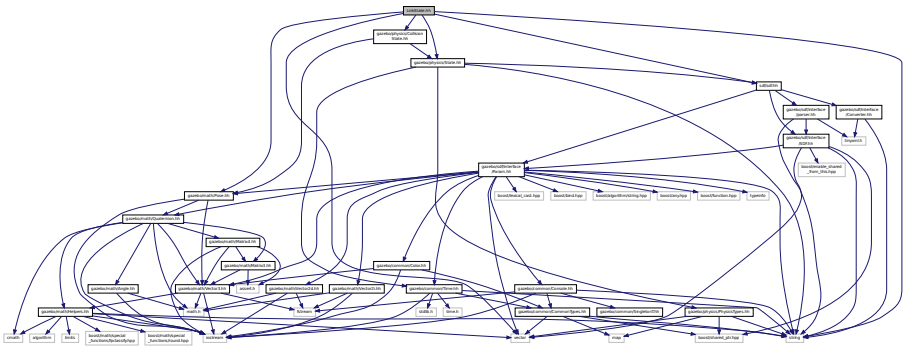
- namespace **gazebo::physics**

  namespace for physics

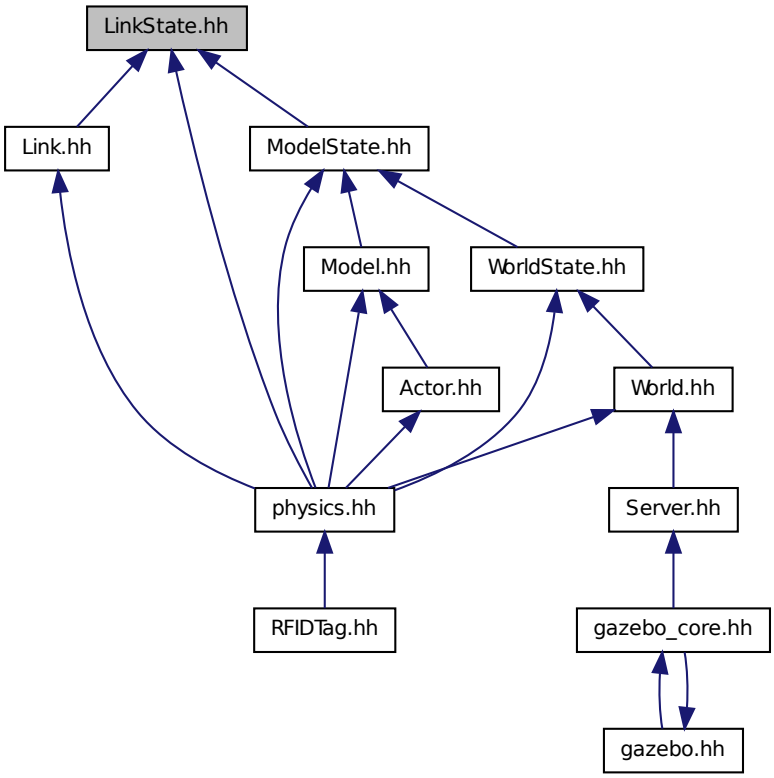
11.73 LinkState.hh File Reference

```
#include <vector>
#include <string>
#include <sdf/sdf.hh>
#include "gazebo/physics/State.hh"
#include "gazebo/physics/CollisionState.hh"
#include "gazebo/math/Pose.hh"
```

Include dependency graph for LinkState.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::LinkState**

  *Store state information of a **physics::Link** (p. 439) object.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::physics**

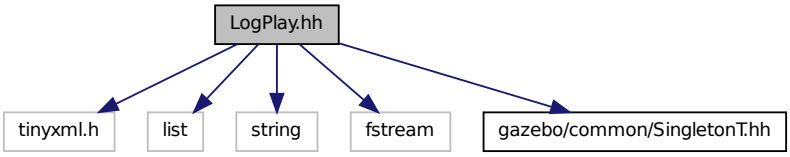
  namespace for physics

11.74 LogPlay.hh File Reference

```
#include <tinyxml.h>
```

```
#include <list>
#include <string>
#include <fstream>
#include "gazebo/common/SingletonT.hh"
```

Include dependency graph for LogPlay.hh:



Classes

- class **gazebo::util::LogPlay**

Namespaces

- namespace **gazebo**

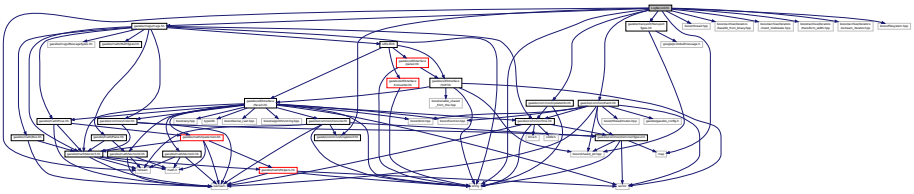
  Forward declarations for the common classes.

- namespace **gazebo::util**

11.75 LogRecord.hh File Reference

```
#include <fstream>
#include <string>
#include <map>
#include <boost/thread.hpp>
#include <boost/archive/iterators/base64_from_binary.hpp>
#include <boost/archive/iterators/insert_linebreaks.hpp>
#include <boost/archive/iterators/transform_width.hpp>
#include <boost/archive/iterators/ostream_iterator.hpp>
#include <boost/filesystem.hpp>
#include "gazebo/msgs/msgs.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/common/UpdateInfo.hh"
#include "gazebo/common/Event.hh"
#include "gazebo/common/SingletonT.hh"
```

Include dependency graph for LogRecord.hh:



Classes

- class **gazebo::util::LogRecord**

  addtogroup gazebo_util

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::util**

Macros

- #define **GZ_LOG_VERSION** "1.0"

11.75.1 Macro Definition Documentation

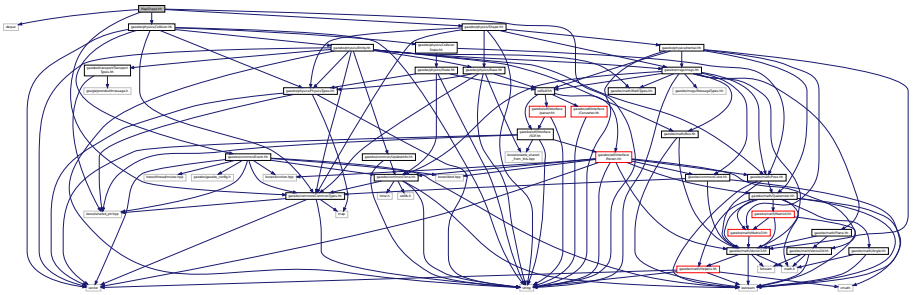
11.75.1.1 #define GZ_LOG_VERSION "1.0"

11.76 mainpage.html File Reference

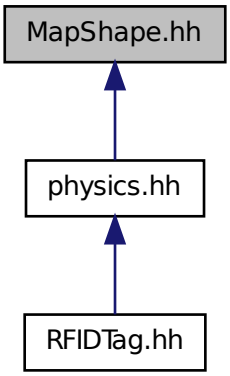
11.77 MapShape.hh File Reference

```
#include <deque>
#include <string>
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/physics/Collision.hh"
#include "gazebo/physics/Shape.hh"
```

Include dependency graph for MapShape.hh:



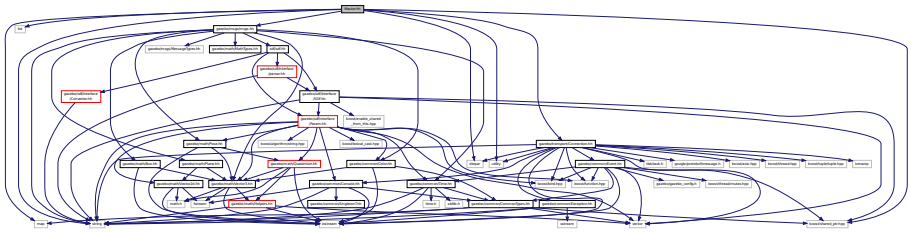
This graph shows which files directly or indirectly include this file:



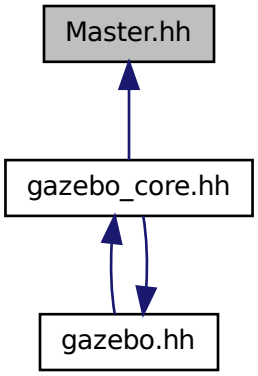
11.78 Master.hh File Reference

```
#include <string>
#include <list>
#include <deque>
#include <utility>
#include <map>
#include <boost/shared_ptr.hpp>
#include "gazebo/msgs/msgs.hh"
#include "gazebo/transport/Connection.hh"
```

Include dependency graph for Master.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::Master**

  A ROS Master-like manager that directs gztopic connections, enables each gazebo network client to locate one another for peer-to-peer communication.

Namespaces

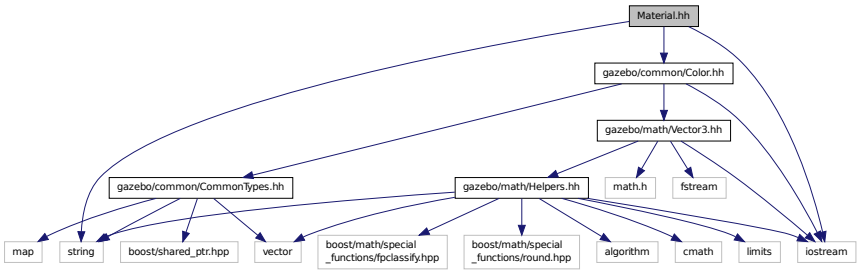
- namespace **gazebo**

  Forward declarations for the common classes.

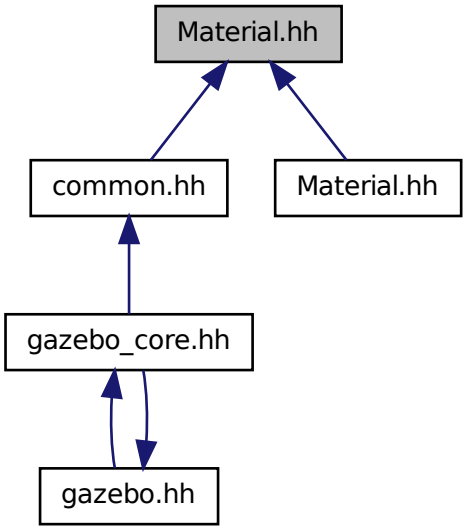
11.79 Material.hh File Reference

```
#include <string>
#include <iostream>
#include "gazebo/common/Color.hh"
```

Include dependency graph for common/Material.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::Material**

  Encapsulates description of a material.

Namespaces

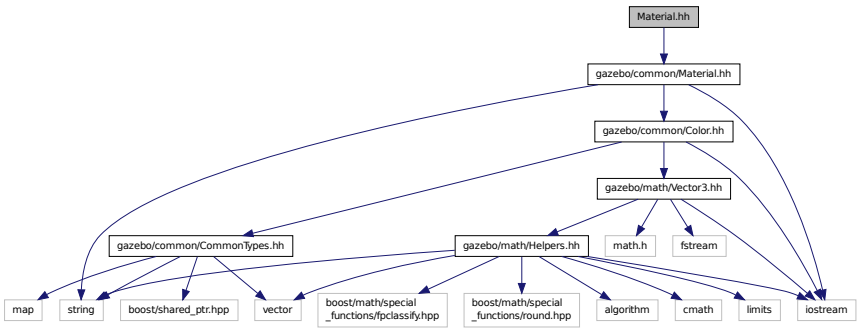
- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

Common namespace.

11.80 Material.hh File Reference

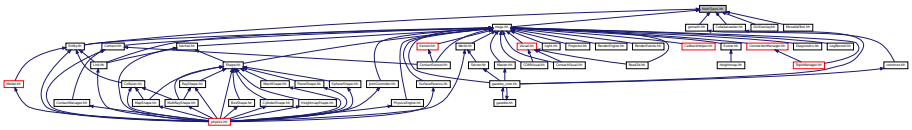
`#include "gazebo/common/Material.hh"`
Include dependency graph for rendering/Material.hh:



11.81 MathTypes.hh File Reference

Forward declarations for the math classes.

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::math**

  Math namespace.

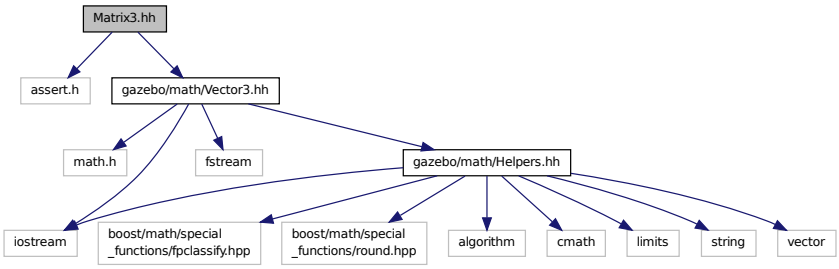
11.81.1 Detailed Description

Forward declarations for the math classes.

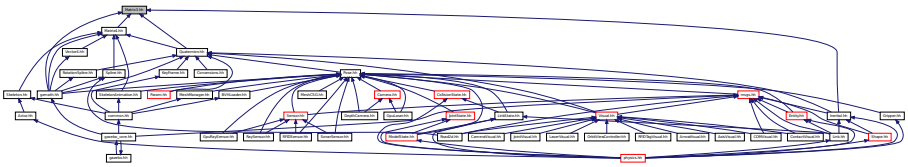
11.82 Matrix3.hh File Reference

```
#include <assert.h>
#include "gazebo/math/Vector3.hh"
```

Include dependency graph for Matrix3.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::math::Matrix3**

  A 3x3 matrix class.

Namespaces

- namespace **gazebo**

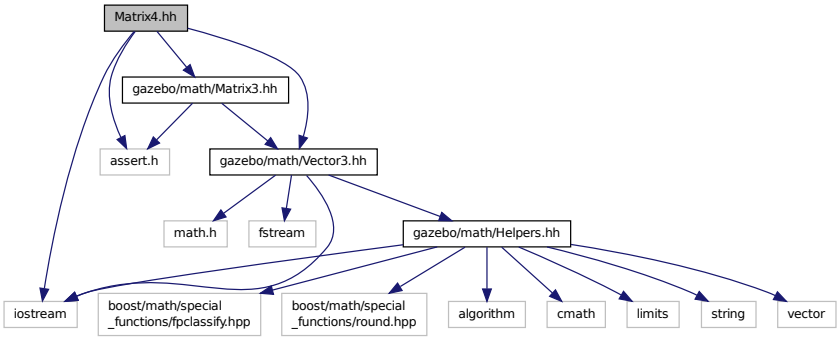
  Forward declarations for the common classes.
- namespace **gazebo::math**

  Math namespace.

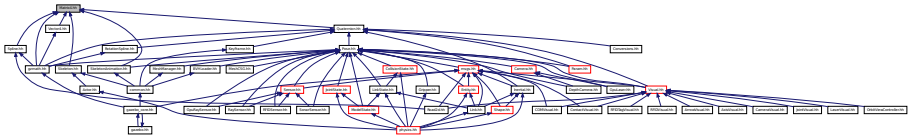
11.83 Matrix4.hh File Reference

```
#include <assert.h>
#include <iostream>
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Matrix3.hh"
```

Include dependency graph for Matrix4.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::math::Matrix4**

  A 3x3 matrix class.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

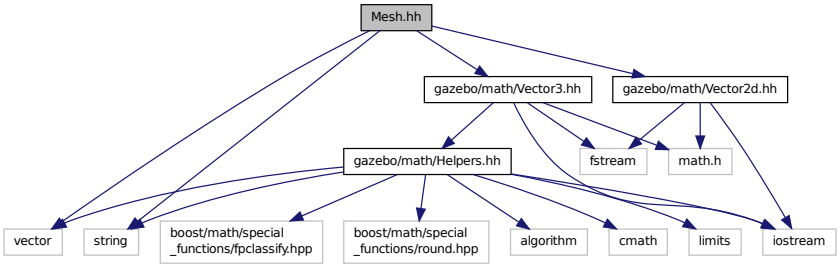
- namespace **gazebo::math**

  Math namespace.

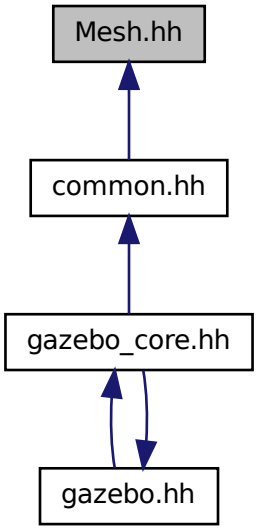
11.84 Mesh.hh File Reference

```
#include <vector>
#include <string>
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Vector2d.hh"
```

Include dependency graph for Mesh.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::Mesh**

  A 3D mesh.

- struct **gazebo::common::NodeAssignment**

  Vertex to node weighted assignement for skeleton animation visualization.

- class **gazebo::common::SubMesh**

  A child mesh.

Namespaces

- namespace **gazebo**

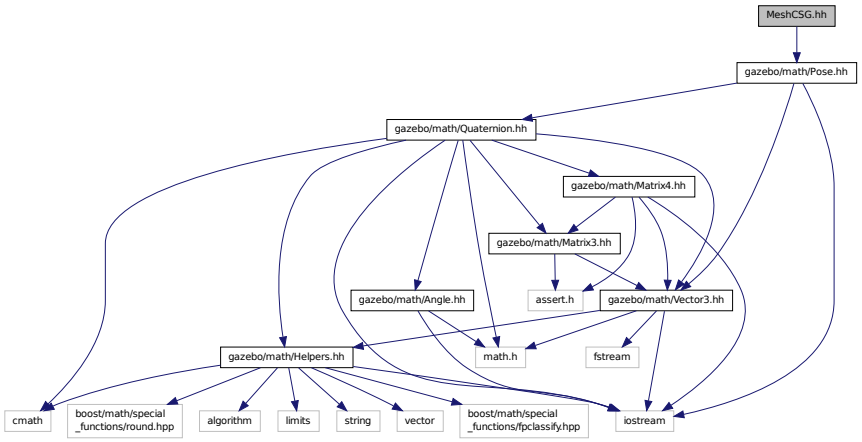
  Forward declarations for the common classes.
- namespace **gazebo::common**

  Common namespace.

11.85 MeshCSG.hh File Reference

```
#include "gazebo/math/Pose.hh"
```

Include dependency graph for MeshCSG.hh:



Classes

- class **gazebo::common::MeshCSG**

  Creates CSG meshes.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

  Common namespace.

Typedefs

- typedef _GPtrArray **GPtrArray**
- typedef _GtsSurface **GtsSurface**

11.85.1 Typedef Documentation

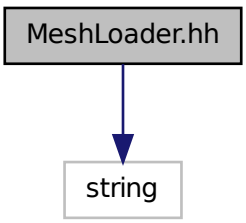
11.85.1.1 typedef _GPtrArray GPtrArray

11.85.1.2 typedef _GtsSurface GtsSurface

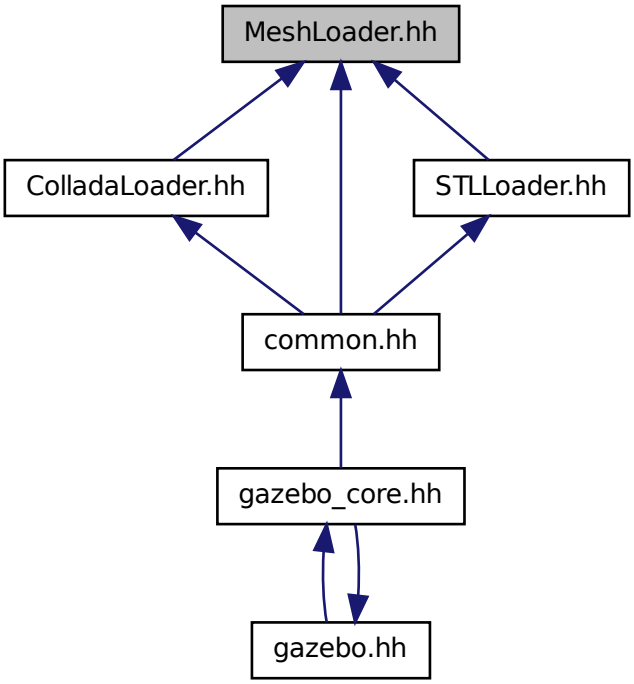
11.86 MeshLoader.hh File Reference

```
#include <string>
```

Include dependency graph for MeshLoader.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::MeshLoader**

  Base class for loading meshes.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

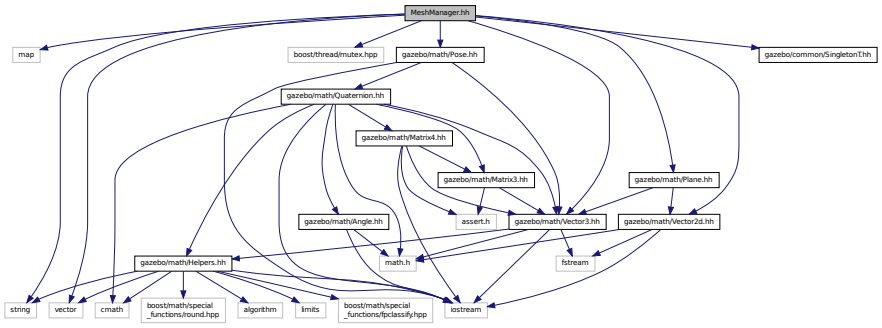
  Common namespace.

11.87 MeshManager.hh File Reference

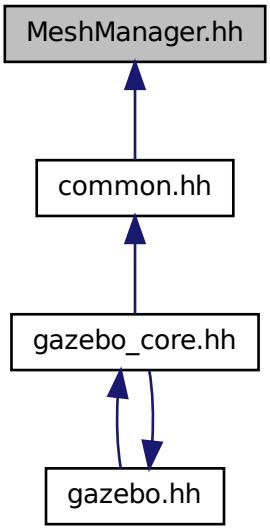
```
#include <map>
```

```
#include <string>
#include <vector>
#include <boost/thread/mutex.hpp>
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Vector2d.hh"
#include "gazebo/math/Pose.hh"
#include "gazebo/math/Plane.hh"
#include "gazebo/common/SingletonT.hh"
```

Include dependency graph for MeshManager.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::MeshManager**

  Maintains and manages all meshes.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

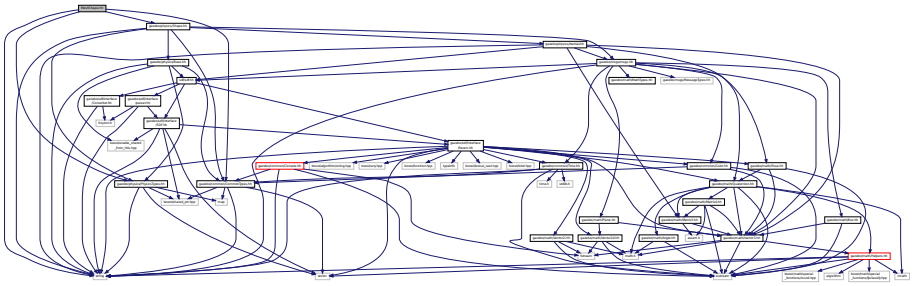
- namespace **gazebo::common**

  Common namespace.

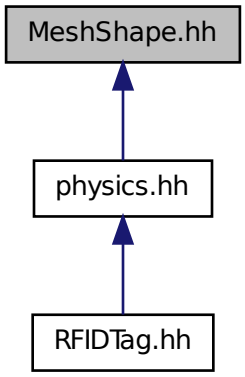
11.88 MeshShape.hh File Reference

```
#include <string>
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/physics/Shape.hh"
```

Include dependency graph for MeshShape.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::MeshShape**

  Triangle mesh collision shape.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

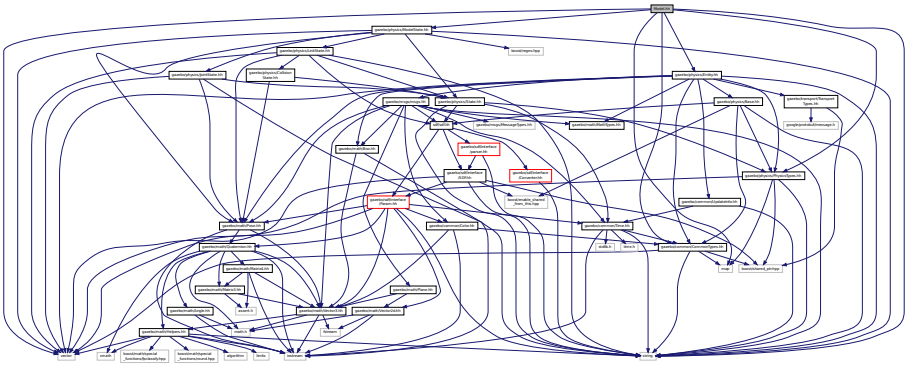
- namespace **gazebo::physics**

  namespace for physics

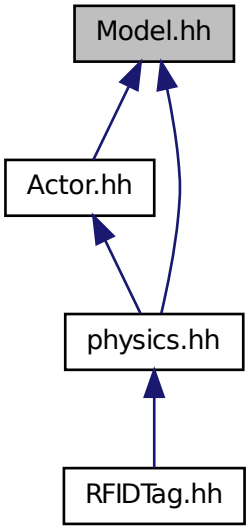
11.89 Model.hh File Reference

```
#include <string>
#include <map>
#include <vector>
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/physics/ModelState.hh"
#include "gazebo/physics/Entity.hh"
```

Include dependency graph for Model.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::Model**

  A model is a collection of links, joints, and plugins.

Namespaces

- namespace **boost**
- namespace **gazebo**

Forward declarations for the common classes.

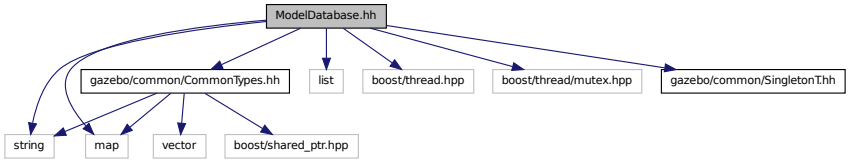
- namespace **gazebo::physics**

  namespace for physics

11.90 ModelDatabase.hh File Reference

```
#include <string>
#include <map>
#include <list>
#include <boost/thread.hpp>
#include <boost/thread/mutex.hpp>
#include "gazebo/common/SingletonT.hh"
#include "gazebo/common/CommonTypes.hh"
```

Include dependency graph for ModelDatabase.hh:



Classes

- class **gazebo::common::ModelDatabase**

  Connects to model database, and has utility functions to find models.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

  Common namespace.

Macros

- #define **GZ_MODEL_DB_MANIFEST_FILENAME** "database.config"

  The file name of model database XML configuration.
- #define **GZ_MODEL_MANIFEST_FILENAME** "model.config"

  The file name of model XML configuration.

11.90.1 Macro Definition Documentation

11.90.1.1 #define GZ_MODEL_DB_MANIFEST_FILENAME "database.config"

The file name of model database XML configuration.

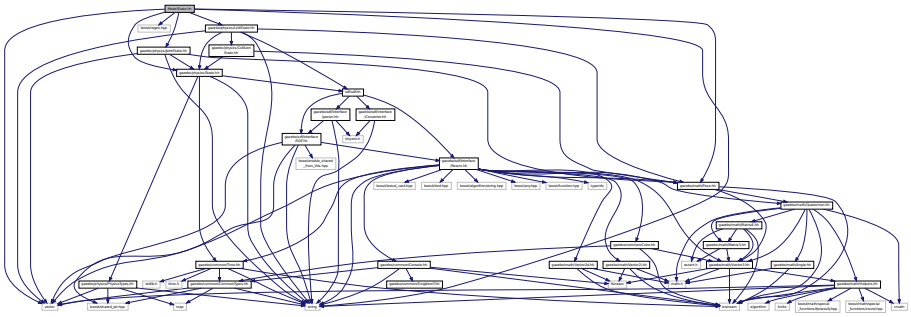
11.90.1.2 #define GZ_MODEL_MANIFEST_FILENAME "model.config"

The file name of model XML configuration.

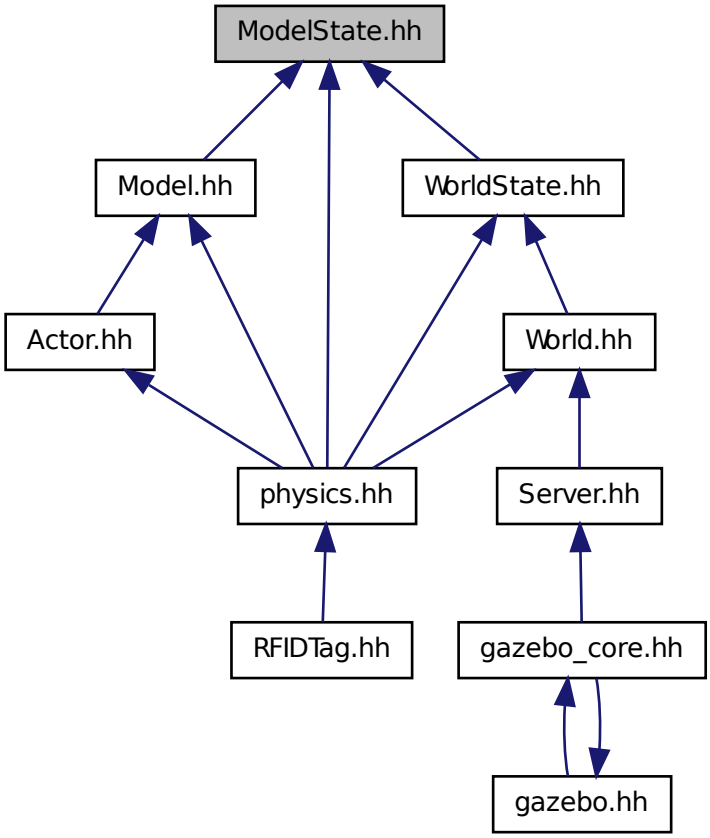
11.91 ModelState.hh File Reference

```
#include <vector>
#include <string>
#include <boost/regex.hpp>
#include "gazebo/math/Pose.hh"
#include "gazebo/physics/State.hh"
#include "gazebo/physics/LinkState.hh"
#include "gazebo/physics/JointState.hh"
```

Include dependency graph for ModelState.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::ModelState**

  *Store state information of a **physics::Model** (p. 516) object.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

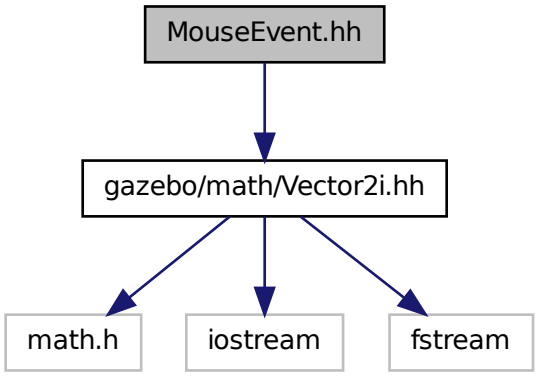
- namespace **gazebo::physics**

  namespace for physics

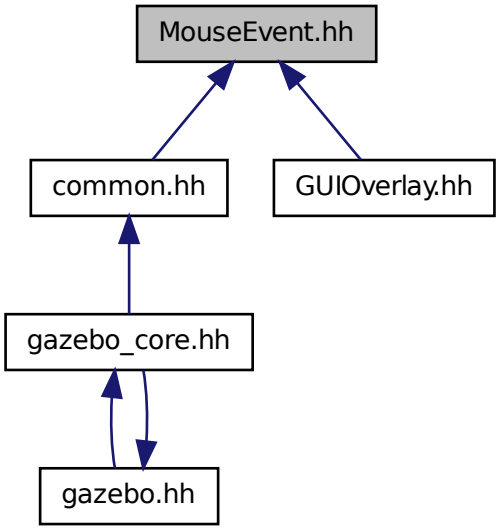
11.92 MouseEvent.hh File Reference

```
#include "gazebo/math/Vector2i.hh"
```

Include dependency graph for MouseEvent.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::MouseEvent**

  Generic description of a mouse event.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

  Common namespace.

11.93 MovableText.hh File Reference

```
#include <string>
#include "gazebo/rendering/ogre_gazebo.h"
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/common/Color.hh"
#include "gazebo/math/MathTypes.hh"
```

Include dependency graph for MovableText.hh:

Classes

- class **gazebo::rendering::MovableText**

  Movable text.

Namespaces

- namespace **boost**
- namespace **gazebo**

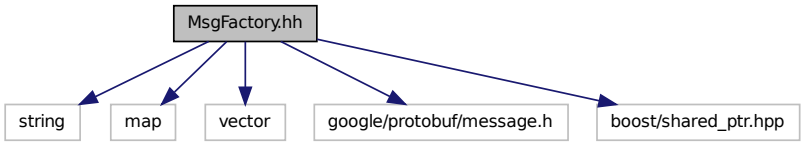
  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.

11.94 MsgFactory.hh File Reference

```
#include <string>
#include <map>
#include <vector>
#include <google/protobuf/message.h>
#include <boost/shared_ptr.hpp>
```

Include dependency graph for MsgFactory.hh:



Classes

- class **gazebo::msgs::MsgFactory**

  A factory that generates protobuf message based on a string type.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::msgs**

  Messages namespace.

Macros

- #define **GZ_REGISTER_STATIC_MSG**(_msgtype, _classname)

  Static message registration macro.

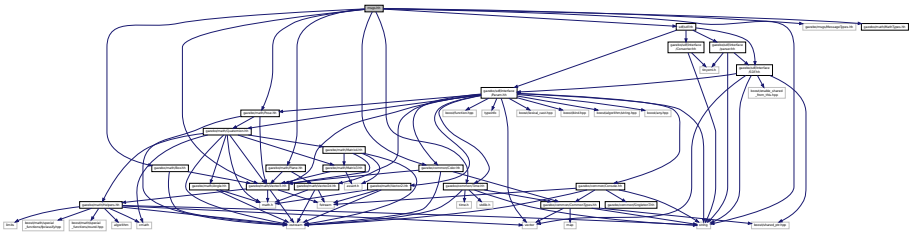
Typedefs

- typedef boost::shared_ptr
  < google::protobuf::Message >(* **gazebo::msgs::MsgFactoryFn**)()

11.95 msgs.hh File Reference

```
#include <string>
#include <sdf/sdf.hh>
#include "gazebo/msgs/MessageTypes.hh"
#include "gazebo/math/MathTypes.hh"
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Pose.hh"
#include "gazebo/math/Plane.hh"
#include "gazebo/math/Box.hh"
#include "gazebo/common/Color.hh"
#include "gazebo/common/Time.hh"
```

Include dependency graph for msgs.hh:



This graph shows which files directly or indirectly include this file:

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::msgs**

  Messages namespace.

Functions

- msgs::Vector3d **gazebo::msgs::Convert** (const math::Vector3 &_v)

  *Convert a **math::Vector3** (p. 890) to a msgs::Vector3d.*
- msgs::Quaternion **gazebo::msgs::Convert** (const math::Quaternion &_q)

  *Convert a **math::Quaternion** (p. 654) to a msgs::Quaternion.*
- msgs::Pose **gazebo::msgs::Convert** (const math::Pose &_p)

  *Convert a **math::Pose** (p. 626) to a msgs::Pose.*
- msgs::Color **gazebo::msgs::Convert** (const common::Color &_c)

  *Convert a **common::Color** (p. 213) to a msgs::Color.*
- msgs::Time **gazebo::msgs::Convert** (const common::Time &_t)

  *Convert a **common::Time** (p. 831) to a msgs::Time.*
- msgs::PlaneGeom **gazebo::msgs::Convert** (const math::Plane &_p)

  *Convert a **math::Plane** (p. 617) to a msgs::PlaneGeom.*
- math::Vector3 **gazebo::msgs::Convert** (const msgs::Vector3d &_v)

  Convert a msgs::Vector3d to a math::Vector.
- math::Quaternion **gazebo::msgs::Convert** (const msgs::Quaternion &_q)

  *Convert a msgs::Quaternion to a **math::Quaternion** (p. 654).*
- math::Pose **gazebo::msgs::Convert** (const msgs::Pose &_p)

  *Convert a msgs::Pose to a **math::Pose** (p. 626).*
- common::Color **gazebo::msgs::Convert** (const msgs::Color &_c)

*Convert a msgs::Color to a **common::Color** (p. 213).*

- common::Time **gazebo::msgs::Convert** (const msgs::Time &_t)

  *Convert a msgs::Time to a **common::Time** (p. 831).*
- math::Plane **gazebo::msgs::Convert** (const msgs::PlaneGeom &_p)

  Convert a msgs::PlaneGeom to a common::Plane.
- msgs::Request * **gazebo::msgs::CreateRequest** (const std::string &_request, const std::string &_data="")

  Create a request message.
- msgs::Fog **gazebo::msgs::FogFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::Fog from a fog SDF element.
- msgs::Geometry **gazebo::msgs::GeometryFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::Geometry from a geometry SDF element.
- msgs::Header * **gazebo::msgs::GetHeader** (google::protobuf::Message &_message)

  Get the header from a protobuf message.
- msgs::GUI **gazebo::msgs::GUIFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::GUI from a GUI SDF element.
- void **gazebo::msgs::Init** (google::protobuf::Message &_message, const std::string &_id="")

  Initialize a message.
- msgs::Light **gazebo::msgs::LightFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::Light from a light SDF element.
- msgs::MeshGeom **gazebo::msgs::MeshFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::MeshGeom from a mesh SDF element.
- msgs::Scene **gazebo::msgs::SceneFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::Scene from a scene SDF element.
- void **gazebo::msgs::Set** (common::Image &_img, const msgs::Image &_msg)

  *Convert a msgs::Image to a **common::Image** (p. 379).*
- void **gazebo::msgs::Set** (msgs::Image *_msg, const common::Image &_i)

  *Set a msgs::Image from a **common::Image** (p. 379).*
- void **gazebo::msgs::Set** (msgs::Vector3d *_pt, const math::Vector3 &_v)

  *Set a msgs::Vector3d from a **math::Vector3** (p. 890).*
- void **gazebo::msgs::Set** (msgs::Vector2d *_pt, const math::Vector2d &_v)

  *Set a msgs::Vector2d from a **math::Vector3** (p. 890).*
- void **gazebo::msgs::Set** (msgs::Quaternion *_q, const math::Quaternion &_v)

  *Set a msgs::Quaternion from a **math::Quaternion** (p. 654).*
- void **gazebo::msgs::Set** (msgs::Pose *_p, const math::Pose &_v)

  *Set a msgs::Pose from a **math::Pose** (p. 626).*
- void **gazebo::msgs::Set** (msgs::Color *_c, const common::Color &_v)

  *Set a msgs::Color from a **common::Color** (p. 213).*
- void **gazebo::msgs::Set** (msgs::Time *_t, const common::Time &_v)

  *Set a msgs::Time from a **common::Time** (p. 831).*
- void **gazebo::msgs::Set** (msgs::PlaneGeom *_p, const math::Plane &_v)

  *Set a msgs::Plane from a **math::Plane** (p. 617).*
- void **gazebo::msgs::Stamp** (msgs::Header *_header)

  Time stamp a header.
- void **gazebo::msgs::Stamp** (msgs::Time *_time)

  Set the time in a time message.
- msgs::TrackVisual **gazebo::msgs::TrackVisualFromSDF** (**sdf::ElementPtr** _sdf)

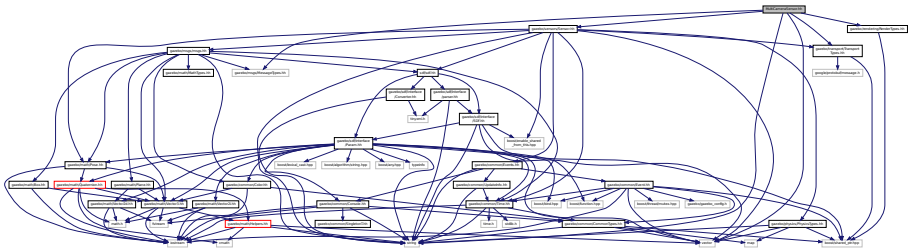
  Create a msgs::TrackVisual from a track visual SDF element.
- msgs::Visual **gazebo::msgs::VisualFromSDF** (**sdf::ElementPtr** _sdf)

  Create a msgs::Visual from a visual SDF element.

11.96 MultiCameraSensor.hh File Reference

```
#include <string>
#include <vector>
#include "gazebo/sensors/Sensor.hh"
#include "gazebo/msgs/MessageTypes.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/rendering/RenderTypes.hh"
```

Include dependency graph for MultiCameraSensor.hh:



Classes

- class **gazebo::sensors::MultiCameraSensor**

  Multiple camera sensor.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

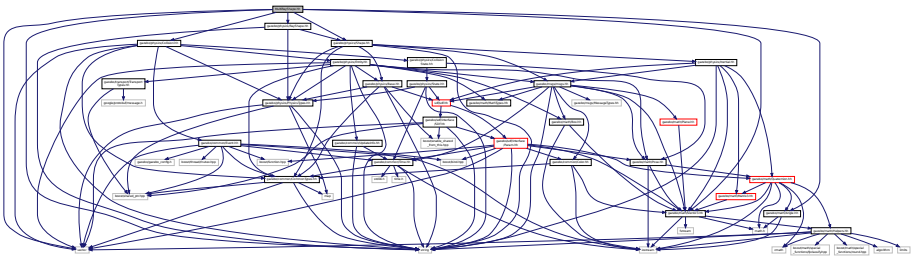
- namespace **gazebo::sensors**

  Sensors namespace.

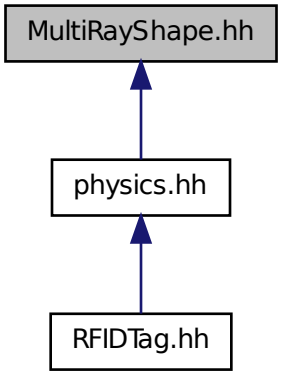
11.97 MultiRayShape.hh File Reference

```
#include <vector>
#include <string>
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Angle.hh"
#include "gazebo/physics/Collision.hh"
#include "gazebo/physics/Shape.hh"
#include "gazebo/physics/RayShape.hh"
```

Include dependency graph for MultiRayShape.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::MultiRayShape**

  Laser collision contains a set of ray-collisions, structured to simulate a laser range scanner.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::physics**

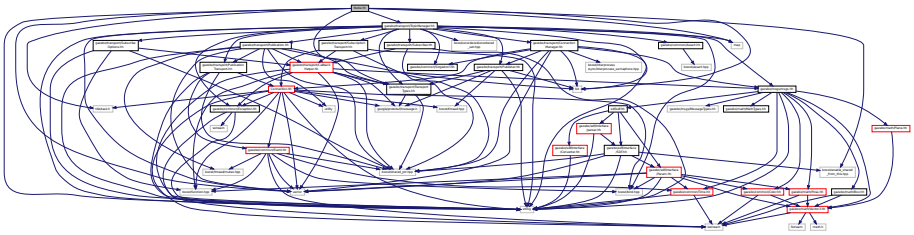
  namespace for physics

11.98 Node.hh File Reference

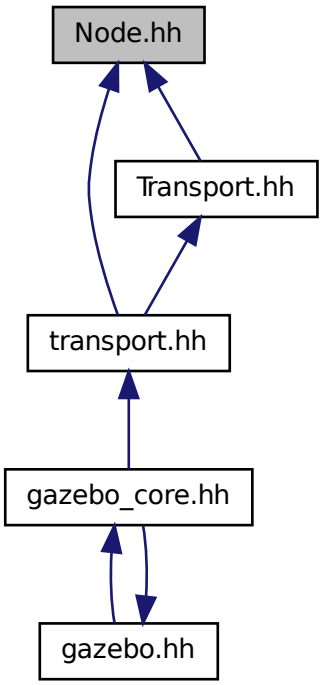
```
#include <tbb/task.h>
```

```
#include <boost/enable_shared_from_this.hpp>
#include <map>
#include <list>
#include <string>
#include <vector>
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/transport/TopicManager.hh"
```

Include dependency graph for Node.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::transport::Node**

A node can advertise and subscribe topics, publish on advertised topics and listen to subscribed topics.

- class **gazebo::transport::PublishTask**

Namespaces

- namespace **gazebo**

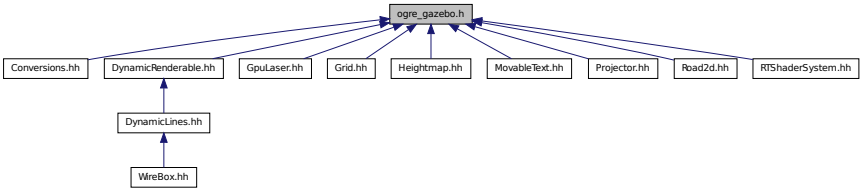
  Forward declarations for the common classes.
- namespace **gazebo::transport**

11.99 ogre_gazebo.h File Reference

```
#include <OGRE/Ogre.h>
#include <OGRE/OgreImageCodec.h>
#include <OGRE/OgreMovableObject.h>
#include <OGRE/OgreRenderable.h>
#include <OGRE/OgrePlugin.h>
#include <OGRE/OgreDataStream.h>
#include <OGRE/OgreLogManager.h>
#include <OGRE/OgreWindowEventUtilities.h>
#include <OGRE/OgreSceneQuery.h>
#include <OGRE/OgreRoot.h>
#include <OGRE/OgreSceneManager.h>
#include <OGRE/OgreSceneNode.h>
#include <OGRE/OgreVector3.h>
#include <OGRE/OgreManualObject.h>
#include <OGRE/OgreMaterialManager.h>
#include <OGRE/OgreColourValue.h>
#include <OGRE/OgreQuaternion.h>
#include <OGRE/OgreMesh.h>
#include <OGRE/OgreFontManager.h>
#include <OGRE/OgreHardwareBufferManager.h>
#include <OGRE/OgreCamera.h>
#include <OGRE/OgreNode.h>
#include <OGRE/OgreSimpleRenderable.h>
#include <OGRE/OgreFrameListener.h>
#include <OGRE/OgreTexture.h>
#include <OGRE/OgreRenderObjectListener.h>
#include <OGRE/Terrain/OgreTerrainMaterialGeneratorA.h>
#include <OGRE/Terrain/OgreTerrain.h>
#include <OGRE/Terrain/OgreTerrainGroup.h>
#include <OGRE/OgreTechnique.h>
#include <OGRE/OgrePass.h>
#include <OGRE/OgreTextureUnitState.h>
#include <OGRE/OgreGpuProgramManager.h>
#include <OGRE/OgreHighLevelGpuProgramManager.h>
#include <OGRE/OgreHardwarePixelBuffer.h>
#include <OGRE/OgreShadowCameraSetupPSSM.h>
```

Include dependency graph for ogre_gazebo.h:

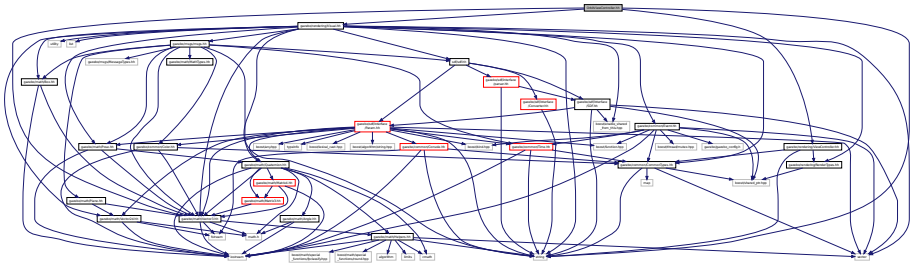
This graph shows which files directly or indirectly include this file:



11.100 OrbitViewController.hh File Reference

```
#include <string>
#include "gazebo/rendering/Visual.hh"
#include "gazebo/rendering/ViewController.hh"
#include "gazebo/math/Vector3.hh"
```

Include dependency graph for OrbitViewController.hh:



Classes

- class **gazebo::rendering::OrbitViewController**

  Orbit view controller.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::rendering**

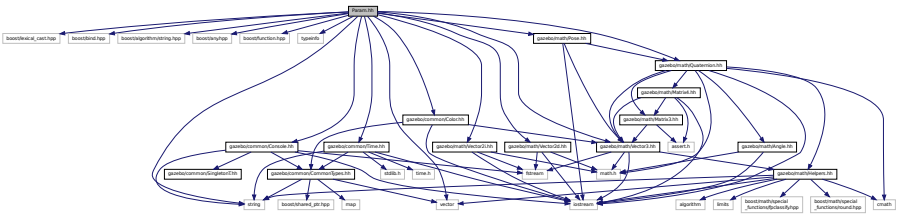
  Rendering namespace.

11.101 Param.hh File Reference

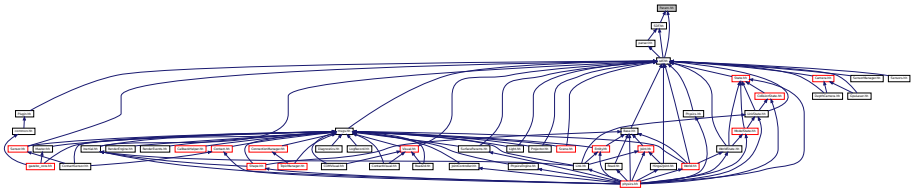
```
#include <boost/lexical_cast.hpp>
```

```
#include <boost/bind.hpp>
#include <boost/algorithm/string.hpp>
#include <boost/any.hpp>
#include <boost/function.hpp>
#include <typeinfo>
#include <string>
#include <vector>
#include "gazebo/common/Console.hh"
#include "gazebo/common/Color.hh"
#include "gazebo/common/Time.hh"
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Vector2i.hh"
#include "gazebo/math/Vector2d.hh"
#include "gazebo/math/Pose.hh"
#include "gazebo/math/Quaternion.hh"
```

Include dependency graph for Param.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **sdf::Param**

  A parameter class.

- class **sdf::ParamT< T >**

  Templatized parameter class.

Namespaces

- namespace **sdf**

  namespace for Simulation Description Format parser

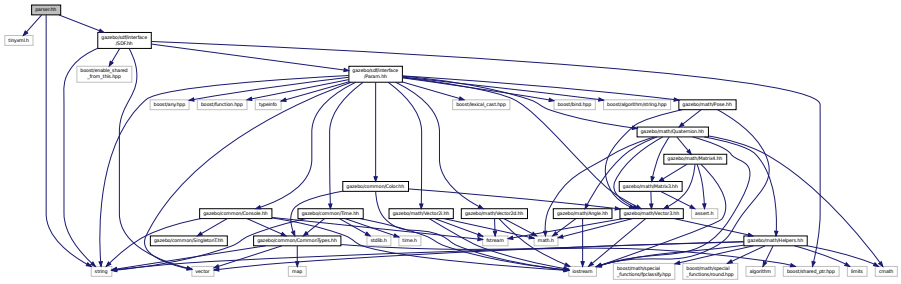
Typedefs

- typedef std::vector< ParamPtr > **sdf::Param_V**
- typedef boost::shared_ptr< Param > **sdf::ParamPtr**

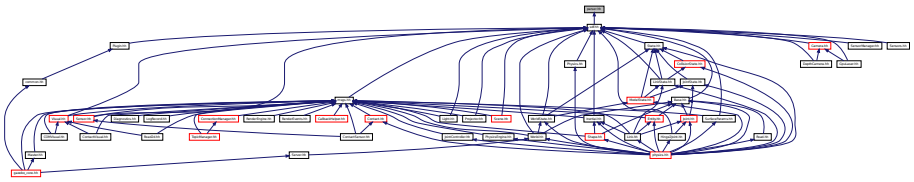
11.102 parser.hh File Reference

```
#include <tinyxml.h>
#include <string>
#include "gazebo/sdf/interface/SDF.hh"
```

Include dependency graph for parser.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **sdf**

  namespace for Simulation Description Format parser

Functions

- void **sdf::addNestedModel** (ElementPtr _sdf, ElementPtr _includeSDF)
- void **sdf::copyChildren** (ElementPtr _sdf, TiXmlElement *_xml)
- bool **sdf::init** (SDFPtr _sdf)

  Init based on the installed sdf_format.xml file.
- bool **sdf::initDoc** (TiXmlDocument *_xmlDoc, SDFPtr _sdf)
- bool **sdf::initDoc** (TiXmlDocument *_xmlDoc, ElementPtr _sdf)
- bool **sdf::initFile** (const std::string &_filename, SDFPtr _sdf)
- bool **sdf::initFile** (const std::string &_filename, ElementPtr _sdf)

- bool **sdf::initString** (const std::string &_xmlString, SDFPtr _sdf)
- bool **sdf::initXml** (TiXmlElement *_xml, ElementPtr _sdf)
- bool **sdf::readDoc** (TiXmlDocument *_xmlDoc, SDFPtr _sdf, const std::string &_source)

  *Populate the **SDF** (p. 728) values from a TinyXML document.*

- bool **sdf::readDoc** (TiXmlDocument *_xmlDoc, ElementPtr _sdf, const std::string &_source)
- bool **sdf::readFile** (const std::string &_filename, SDFPtr _sdf)

  *Populate the **SDF** (p. 728) values from a file.*

- bool **sdf::readString** (const std::string &_xmlString, SDFPtr _sdf)

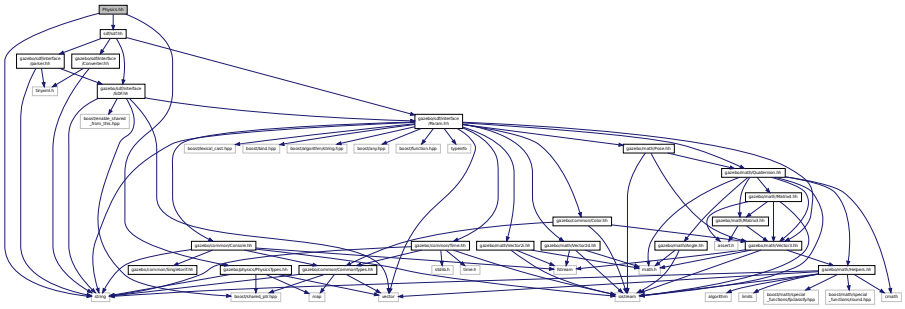
  *Populate the **SDF** (p. 728) values from a string.*

- bool **sdf::readString** (const std::string &_xmlString, ElementPtr _sdf)
- bool **sdf::readXml** (TiXmlElement *_xml, ElementPtr _sdf)

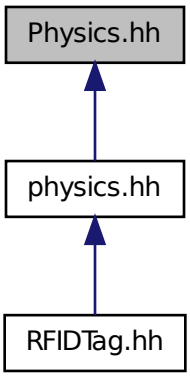
11.103 Physics.hh File Reference

```
#include <string>
#include <sdf/sdf.hh>
#include "gazebo/physics/PhysicsTypes.hh"
```

Include dependency graph for Physics.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::physics**

  namespace for physics

Functions

- WorldPtr **gazebo::physics::create_world** (const std::string &_name="")

  Create a world given a name.
- bool **gazebo::physics::fini** ()

  *Finalize transport by calling **gazebo::transport::fini** (p. 78).*
- WorldPtr **gazebo::physics::get_world** (const std::string &_name="")

  Returns a pointer to a world by name.
- void **gazebo::physics::init_world** (WorldPtr _world)

  Init world given a pointer to it.
- void **gazebo::physics::init_worlds** ()

  initialize multiple worlds stored in static variable gazebo::g_worlds
- bool **gazebo::physics::load** ()

  *Setup **gazebo::SystemPlugin** (p. 829)'s and call **gazebo::transport::init** (p. 79).*
- void **gazebo::physics::load_world** (WorldPtr _world, **sdf::ElementPtr** _sdf)

  *Load world from **sdf::Element** (p. 280) pointer.*
- void **gazebo::physics::load_worlds** (**sdf::ElementPtr** _sdf)

  *load multiple worlds from single **sdf::Element** (p. 280) pointer*
- void **gazebo::physics::pause_world** (WorldPtr _world, bool _pause)

  *Pause world by calling **World::SetPaused** (p. 955).*

- void **gazebo::physics::pause_worlds** (bool pause)

  pause multiple worlds stored in static variable gazebo::g_worlds

- void **gazebo::physics::remove_worlds** ()

  remove multiple worlds stored in static variable gazebo::g_worlds

- void **gazebo::physics::run_world** (WorldPtr _world, unsigned int _iterations=0)

  *Run world by calling **World::Run()** (p. 955) given a pointer to it.*

- void **gazebo::physics::run_worlds** (unsigned int _iterations=0)

  Run multiple worlds stored in static variable gazebo::g_worlds.

- void **gazebo::physics::stop_world** (WorldPtr _world)

  *Stop world by calling **World::Stop()** (p. 956) given a pointer to it.*

- void **gazebo::physics::stop_worlds** ()

  stop multiple worlds stored in static variable gazebo::g_worlds

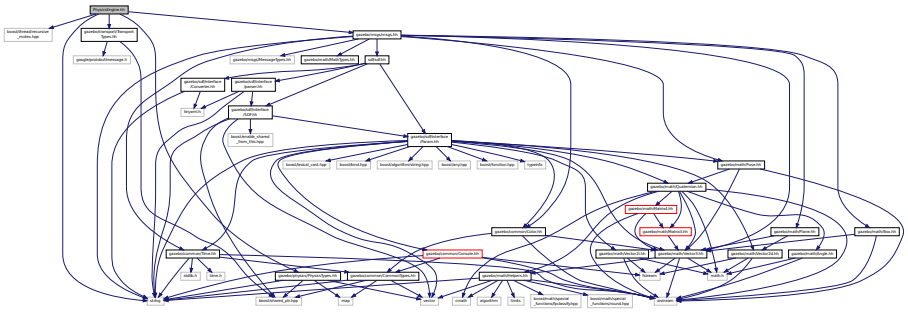
- bool **gazebo::physics::worlds_running** ()

  Return true if any world is running.

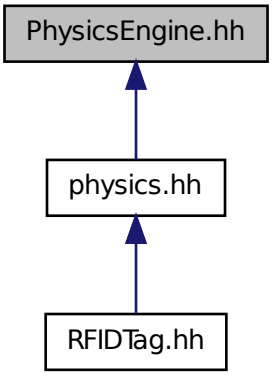
11.104 PhysicsEngine.hh File Reference

```
#include <boost/thread/recursive_mutex.hpp>
#include <string>
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/msgs/msgs.hh"
#include "gazebo/physics/PhysicsTypes.hh"
```

Include dependency graph for PhysicsEngine.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::PhysicsEngine**

  Base *(p. 141) class for a physics engine.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

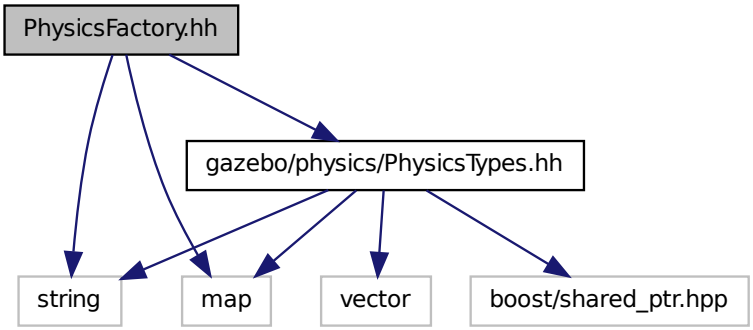
- namespace **gazebo::physics**

  namespace for physics

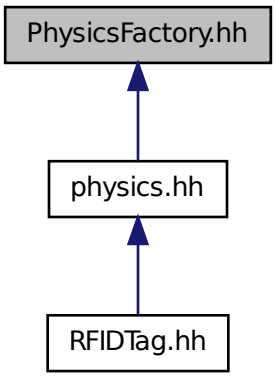
11.105 PhysicsFactory.hh File Reference

```
#include <string>
#include <map>
#include "gazebo/physics/PhysicsTypes.hh"
```

Include dependency graph for PhysicsFactory.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::PhysicsFactory**

  The physics factory instantiates different physics engines.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::physics**

  namespace for physics

Macros

- #define **GZ_REGISTER_PHYSICS_ENGINE**(name, classname)

  Static physics registration macro.

Typedefs

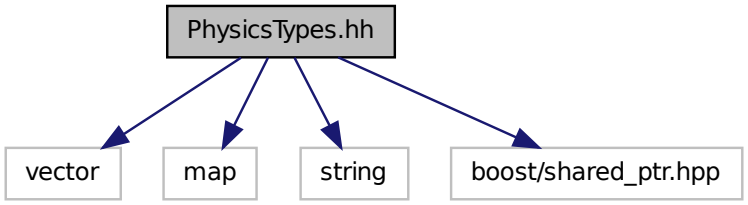
- typedef PhysicsEnginePtr(∗ **gazebo::physics::PhysicsFactoryFn**)(WorldPtr world)

11.106 PhysicsTypes.hh File Reference

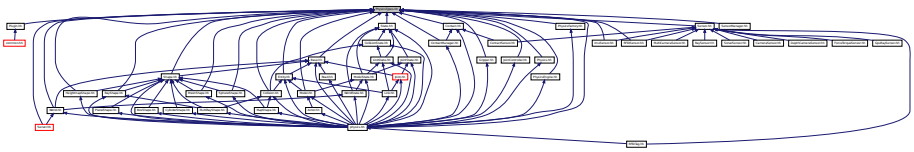
default namespace for gazebo

```
#include <vector>
#include <map>
#include <string>
#include <boost/shared_ptr.hpp>
```

Include dependency graph for PhysicsTypes.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::physics**

  namespace for physics

Macros

- #define **GZ_ALL_COLLIDE** 0x0FFFFFFF

  Default collision bitmask.
- #define **GZ_FIXED_COLLIDE** 0x00000001

  Collision object will collide only with fixed objects.
- #define **GZ_GHOST_COLLIDE** 0x10000000

  Collides with everything else but other ghost.
- #define **GZ_NONE_COLLIDE** 0x00000000

  Collision object will collide with nothing.
- #define **GZ_SENSOR_COLLIDE** 0x00000002

  Collision object will collide only with sensors.

Typedefs

- typedef std::vector< ActorPtr > **gazebo::physics::Actor_V**
- typedef boost::shared_ptr< Actor > **gazebo::physics::ActorPtr**
- typedef std::vector< BasePtr > **gazebo::physics::Base_V**
- typedef boost::shared_ptr< Base > **gazebo::physics::BasePtr**
- typedef boost::shared_ptr
  < BoxShape > **gazebo::physics::BoxShapePtr**
- typedef std::vector< CollisionPtr > **gazebo::physics::Collision_V**
- typedef boost::shared_ptr
  < Collision > **gazebo::physics::CollisionPtr**
- typedef boost::shared_ptr
  < Contact > **gazebo::physics::ContactPtr**
- typedef boost::shared_ptr
  < CylinderShape > **gazebo::physics::CylinderShapePtr**
- typedef boost::shared_ptr< Entity > **gazebo::physics::EntityPtr**
- typedef boost::shared_ptr
  < HeightmapShape > **gazebo::physics::HeightmapShapePtr**
- typedef boost::shared_ptr
  < Inertial > **gazebo::physics::InertialPtr**
- typedef std::vector< JointPtr > **gazebo::physics::Joint_V**
- typedef std::vector
  < JointControllerPtr > **gazebo::physics::JointController_V**
- typedef boost::shared_ptr
  < JointController > **gazebo::physics::JointControllerPtr**
- typedef boost::shared_ptr< Joint > **gazebo::physics::JointPtr**
- typedef std::map< std::string,
  JointState > **gazebo::physics::JointState_M**
- typedef std::vector< LinkPtr > **gazebo::physics::Link_V**
- typedef boost::shared_ptr< Link > **gazebo::physics::LinkPtr**
- typedef std::map< std::string,
  LinkState > **gazebo::physics::LinkState_M**
- typedef boost::shared_ptr
  < MeshShape > **gazebo::physics::MeshShapePtr**

- typedef std::vector< ModelPtr > **gazebo::physics::Model_V**
- typedef boost::shared_ptr< Model > **gazebo::physics::ModelPtr**
- typedef std::map< std::string, ModelState > **gazebo::physics::ModelState_M**
- typedef boost::shared_ptr < MultiRayShape > **gazebo::physics::MultiRayShapePtr**
- typedef boost::shared_ptr < PhysicsEngine > **gazebo::physics::PhysicsEnginePtr**
- typedef boost::shared_ptr < RayShape > **gazebo::physics::RayShapePtr**
- typedef boost::shared_ptr< **Road** > **gazebo::physics::RoadPtr**
- typedef boost::shared_ptr< Shape > **gazebo::physics::ShapePtr**
- typedef boost::shared_ptr < SphereShape > **gazebo::physics::SphereShapePtr**
- typedef boost::shared_ptr < SurfaceParams > **gazebo::physics::SurfaceParamsPtr**
- typedef boost::shared_ptr< World > **gazebo::physics::WorldPtr**

11.106.1 Detailed Description

default namespace for gazebo

11.106.2 Macro Definition Documentation

11.106.2.1 #define GZ_ALL_COLLIDE 0x0FFFFFFF

Default collision bitmask.

Collision objects will collide with everything.

11.106.2.2 #define GZ_FIXED_COLLIDE 0x00000001

Collision object will collide only with fixed objects.

11.106.2.3 #define GZ_GHOST_COLLIDE 0x10000000

Collides with everything else but other ghost.

11.106.2.4 #define GZ_NONE_COLLIDE 0x00000000

Collision object will collide with nothing.

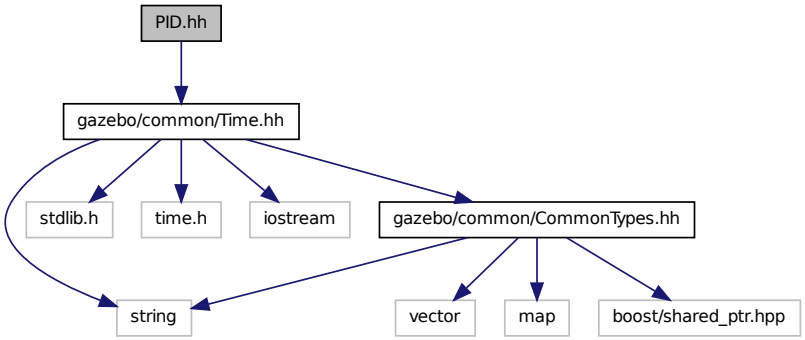
11.106.2.5 #define GZ_SENSOR_COLLIDE 0x00000002

Collision object will collide only with sensors.

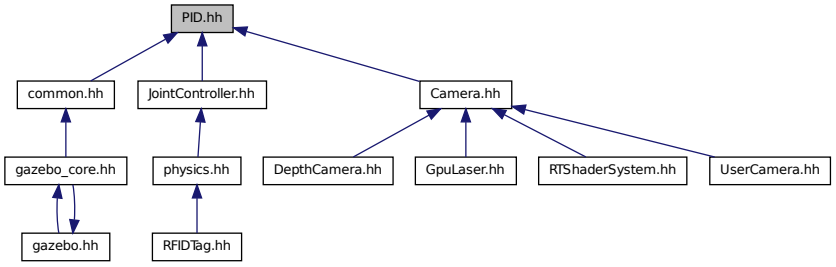
11.107 PID.hh File Reference

```
#include "gazebo/common/Time.hh"
```

Include dependency graph for PID.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::PID**

  *Generic **PID** (p. 613) controller class.*

Namespaces

- namespace **gazebo**

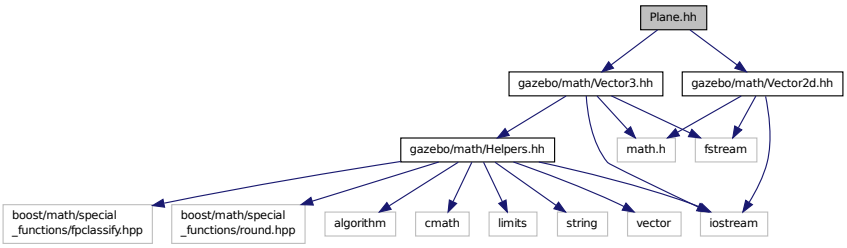
  Forward declarations for the common classes.
- namespace **gazebo::common**

  Common namespace.

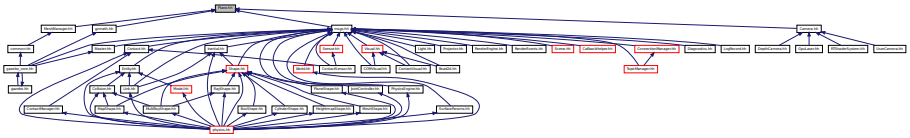
11.108 Plane.hh File Reference

```
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Vector2d.hh"
```

Include dependency graph for Plane.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::math::Plane**

  A plane and related functions.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

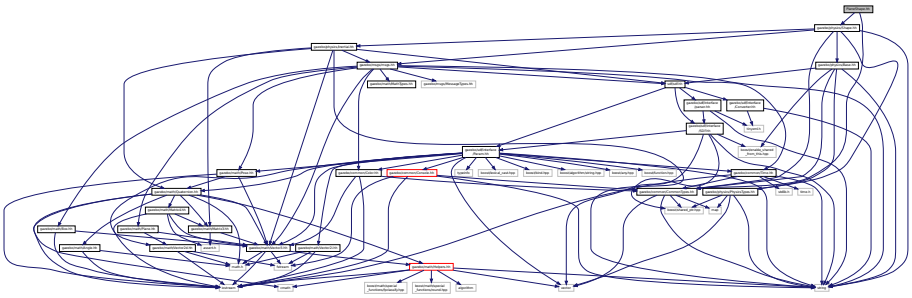
- namespace **gazebo::math**

  Math namespace.

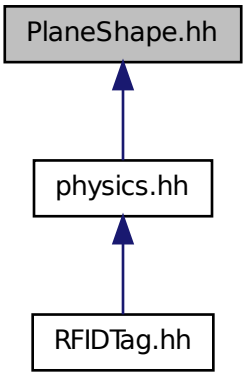
11.109 PlaneShape.hh File Reference

```
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/physics/Shape.hh"
```

Include dependency graph for PlaneShape.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::PlaneShape**

  Collision (p. *199*) *for an infinite plane.*

Namespaces

- namespace **gazebo**

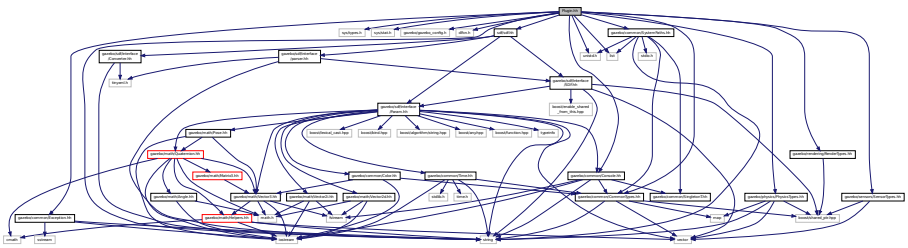
  Forward declarations for the common classes.
- namespace **gazebo::physics**

  namespace for physics

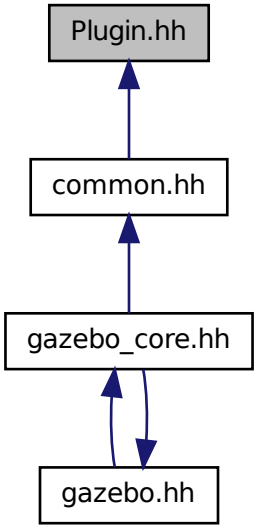
11.110 Plugin.hh File Reference

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <gazebo/gazebo_config.h>
#include <dlfcn.h>
#include <list>
#include <string>
#include <sdf/sdf.hh>
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/common/SystemPaths.hh"
#include "gazebo/common/Console.hh"
#include "gazebo/common/Exception.hh"
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/sensors/SensorTypes.hh"
#include "gazebo/rendering/RenderTypes.hh"
```

Include dependency graph for common/Plugin.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::ModelPlugin**

  *A plugin with access to **physics::Model*** (p. *516*).
- class **gazebo::PluginT< T >**

  A class which all plugins must inherit from.
- class **gazebo::SensorPlugin**

  A plugin with access to physics::Sensor.
- class **gazebo::SystemPlugin**

  A plugin loaded within the gzserver on startup.
- class **gazebo::VisualPlugin**

  A plugin loaded within the gzserver on startup.
- class **gazebo::WorldPlugin**

  *A plugin with access to **physics::World*** (p. *945*).

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

Macros

- #define **GZ_REGISTER_MODEL_PLUGIN**(classname)

  Plugin registration function for model plugin.
- #define **GZ_REGISTER_SENSOR_PLUGIN**(classname)

  Plugin registration function for sensors.
- #define **GZ_REGISTER_SYSTEM_PLUGIN**(classname)

  Plugin registration function for system plugin.
- #define **GZ_REGISTER_VISUAL_PLUGIN**(classname)

  Plugin registration function for visual plugin.
- #define **GZ_REGISTER_WORLD_PLUGIN**(classname)

  Plugin registration function for world plugin.

Enumerations

- enum **gazebo::PluginType** {
  gazebo::WORLD_PLUGIN, **gazebo::MODEL_PLUGIN**, **gazebo::SENSOR_PLUGIN**, **gazebo::SYSTEM_PLUGIN**,
  gazebo::VISUAL_PLUGIN }

  Used to specify the type of plugin.

11.110.1 Macro Definition Documentation

11.110.1.1 #define GZ_REGISTER_MODEL_PLUGIN(*classname*)

Value:

```
extern "C" gazebo::ModelPlugin *RegisterPlugin();    gazebo::ModelPlugin *RegisterPlugin() \
  {\
    return new classname();\
  }
```

Plugin registration function for model plugin.

Part of the shared object interface. This function is called when loading the shared library to add the plugin to the registered list.

Returns

the name of the registered plugin

11.110.1.2 #define GZ_REGISTER_SENSOR_PLUGIN(*classname*)

Value:

```
extern "C" gazebo::SensorPlugin *RegisterPlugin();    gazebo::SensorPlugin *RegisterPlugin() \
  {\
    return new classname();\
  }
```

Plugin registration function for sensors.

Part of the shared object interface. This function is called when loading the shared library to add the plugin to the registered list.

Returns

the name of the registered plugin

11.110.1.3 #define GZ_REGISTER_SYSTEM_PLUGIN(*classname*)

Value:

```
extern "C" gazebo::SystemPlugin *RegisterPlugin();    gazebo::SystemPlugin *RegisterPlugin() \
  {\
    return new classname();\
  }
```

Plugin registration function for system plugin.

Part of the shared object interface. This function is called when loading the shared library to add the plugin to the registered list.

Returns

the name of the registered plugin

11.110.1.4 #define GZ_REGISTER_VISUAL_PLUGIN(*classname*)

Value:

```
extern "C" gazebo::VisualPlugin *RegisterPlugin();    gazebo::VisualPlugin *RegisterPlugin() \
  {\
    return new classname();\
  }
```

Plugin registration function for visual plugin.

Part of the shared object interface. This function is called when loading the shared library to add the plugin to the registered list.

Returns

the name of the registered plugin

11.110.1.5 #define GZ_REGISTER_WORLD_PLUGIN(*classname*)

Value:

```
extern "C" gazebo::WorldPlugin *RegisterPlugin();    gazebo::WorldPlugin *RegisterPlugin() \
  {\
    return new classname();\
  }
```

Plugin registration function for world plugin.

Part of the shared object interface. This function is called when loading the shared library to add the plugin to the registered list.

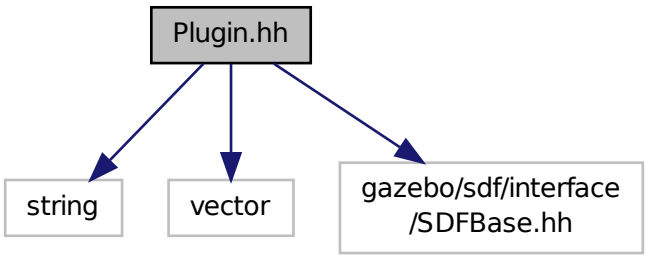
Returns

the name of the registered plugin

11.111 Plugin.hh File Reference

```
#include <string>
#include <vector>
#include "gazebo/sdf/interface/SDFBase.hh"
```

Include dependency graph for sdf/interface/Plugin.hh:



Classes

- class **sdf::Plugin**

Namespaces

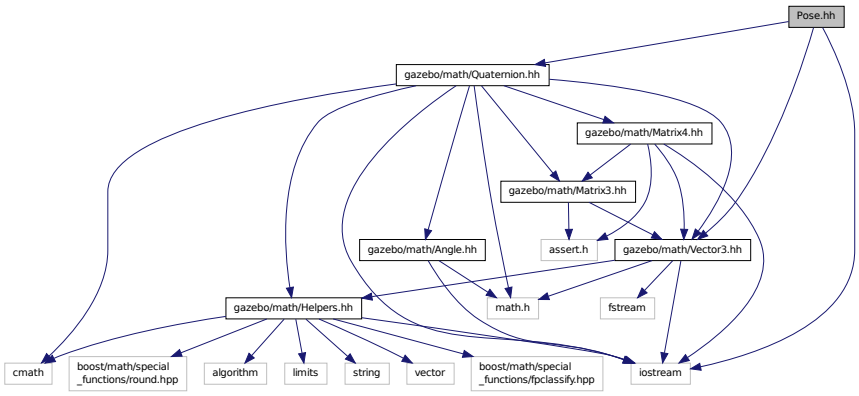
- namespace **sdf**

  namespace for Simulation Description Format parser

11.112 Pose.hh File Reference

```
#include <iostream>
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Quaternion.hh"
```

Include dependency graph for Pose.hh:



This graph shows which files directly or indirectly include this file:

Classes

- class **gazebo::math::Pose**

  Encapsulates a position and rotation in three space.

Namespaces

- namespace **gazebo**

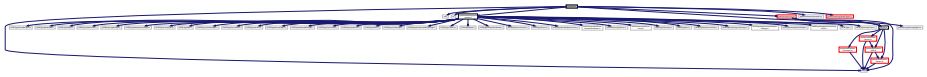
  Forward declarations for the common classes.
- namespace **gazebo::math**

  Math namespace.

11.113 Projector.hh File Reference

```
#include <string>
#include <map>
#include <list>
#include <sdf/sdf.hh>
#include "gazebo/rendering/ogre_gazebo.h"
#include "gazebo/msgs/msgs.hh"
#include "gazebo/transport/transport.hh"
#include "gazebo/rendering/RenderTypes.hh"
```

Include dependency graph for Projector.hh:



Classes

- class **gazebo::rendering::Projector**

  Projects a material onto surface, light a light projector.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

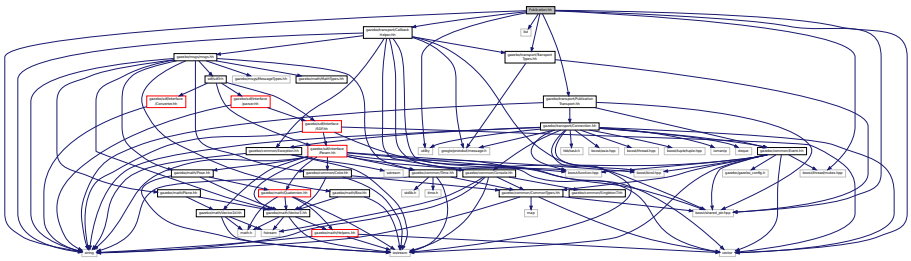
- namespace **gazebo::rendering**

  Rendering namespace.

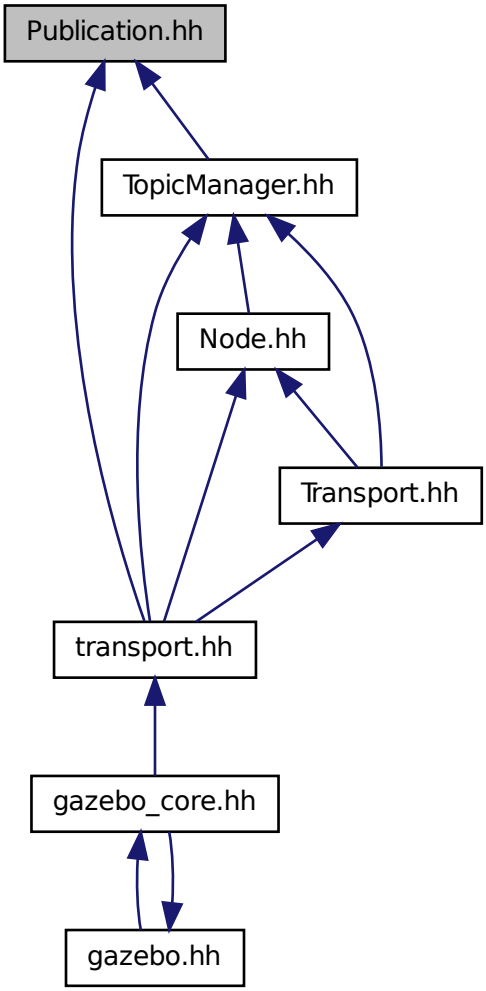
11.114 Publication.hh File Reference

```
#include <utility>
#include <boost/shared_ptr.hpp>
#include <boost/thread/mutex.hpp>
#include <list>
#include <string>
#include <vector>
#include "gazebo/transport/CallbackHelper.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/transport/PublicationTransport.hh"
```

Include dependency graph for Publication.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::transport::Publication**

  A publication for a topic.

Namespaces

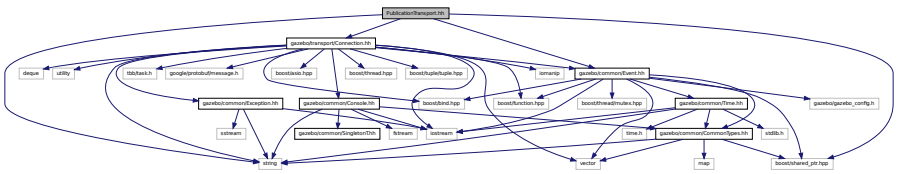
- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::transport**

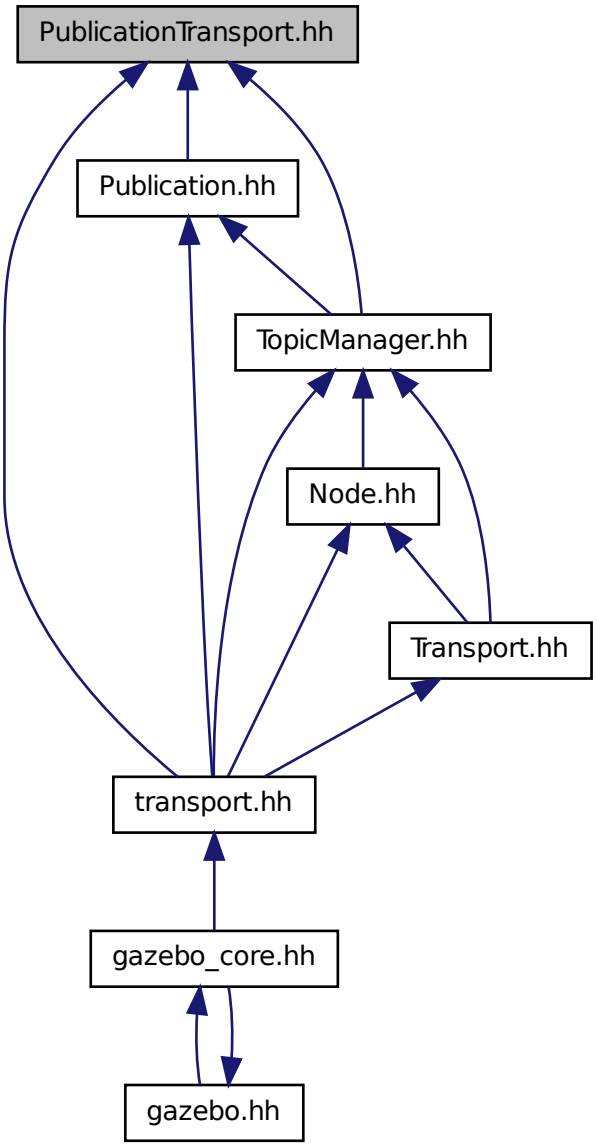
11.115 PublicationTransport.hh File Reference

```
#include <boost/shared_ptr.hpp>
#include <string>
#include "gazebo/transport/Connection.hh"
#include "gazebo/common/Event.hh"
```

Include dependency graph for PublicationTransport.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::transport::PublicationTransport**

  transport/transport.hh

Namespaces

- namespace **gazebo**

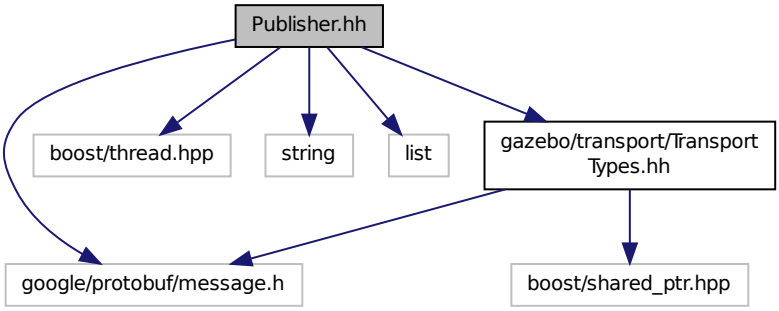
  Forward declarations for the common classes.

- namespace **gazebo::transport**

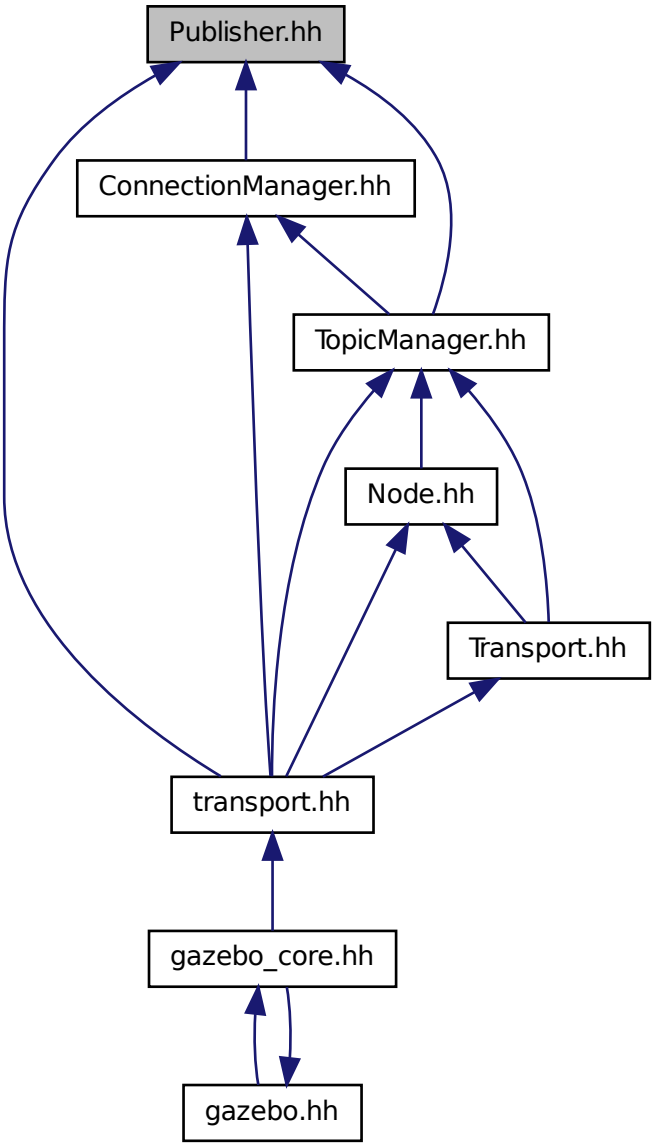
11.116 Publisher.hh File Reference

```
#include <google/protobuf/message.h>
#include <boost/thread.hpp>
#include <string>
#include <list>
#include "gazebo/transport/TransportTypes.hh"
```

Include dependency graph for Publisher.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::transport::Publisher**

  A publisher of messages on a topic.

Namespaces

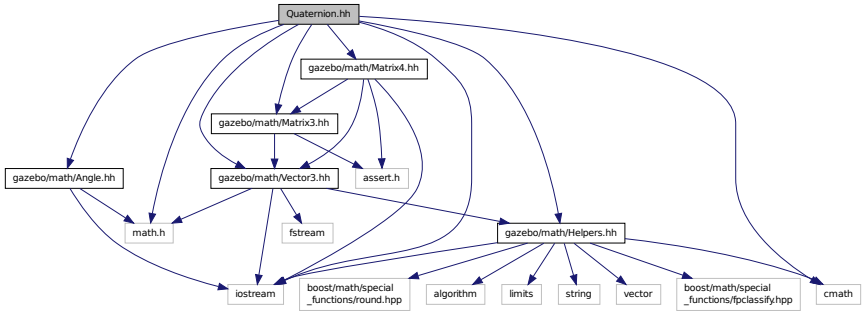
- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::transport**

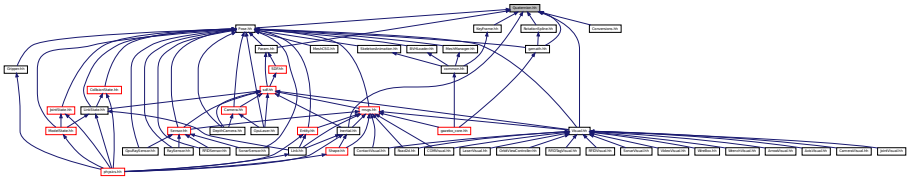
11.117 Quaternion.hh File Reference

```
#include <math.h>
#include <iostream>
#include <cmath>
#include "gazebo/math/Helpers.hh"
#include "gazebo/math/Angle.hh"
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Matrix3.hh"
#include "gazebo/math/Matrix4.hh"
```

Include dependency graph for Quaternion.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::math::Quaternion**

  A quaternion class.

Namespaces

- namespace **gazebo**

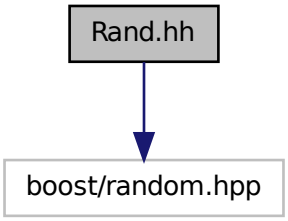
Forward declarations for the common classes.

- namespace **gazebo::math**

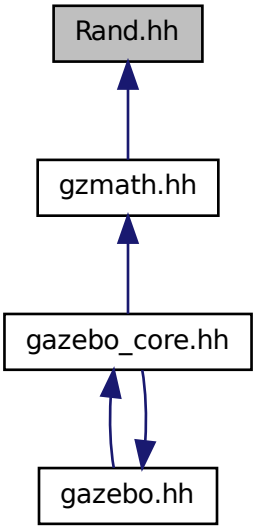
  Math namespace.

11.118 Rand.hh File Reference

`#include <boost/random.hpp>`
Include dependency graph for Rand.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::math::Rand**

  Random number generator class.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::math**

  Math namespace.

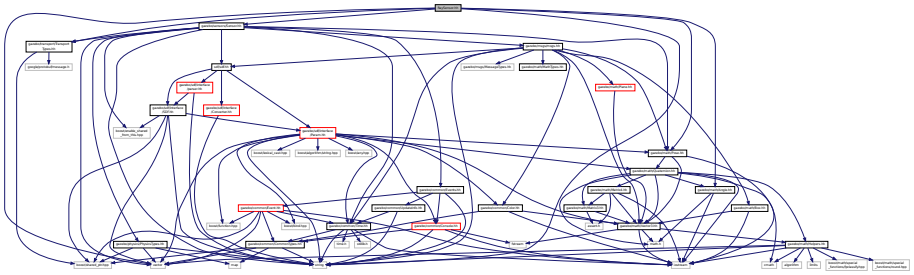
Typedefs

- typedef boost::mt19937 **gazebo::math::GeneratorType**
- typedef boost::normal_distribution < double > **gazebo::math::NormalRealDist**
- typedef boost::variate_generator < GeneratorType &, NormalRealDist > **gazebo::math::NRealGen**
- typedef boost::variate_generator < GeneratorType &, UniformIntDist > **gazebo::math::UIntGen**
- typedef boost::uniform_int< int > **gazebo::math::UniformIntDist**
- typedef boost::uniform_real < double > **gazebo::math::UniformRealDist**
- typedef boost::variate_generator < GeneratorType &, UniformRealDist > **gazebo::math::URealGen**

11.119 RaySensor.hh File Reference

```
#include <vector>
#include <string>
#include "gazebo/math/Angle.hh"
#include "gazebo/math/Pose.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/sensors/Sensor.hh"
```

Include dependency graph for RaySensor.hh:



Classes

- class **gazebo::sensors::RaySensor**

  Sensor (p. 731) *with one or more rays.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

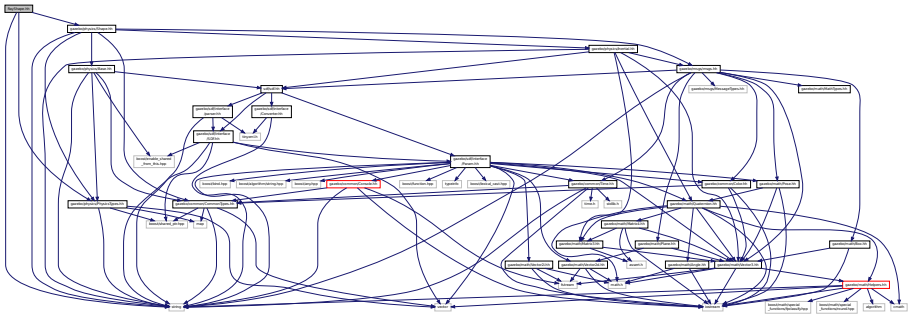
- namespace **gazebo::sensors**

  Sensors namespace.

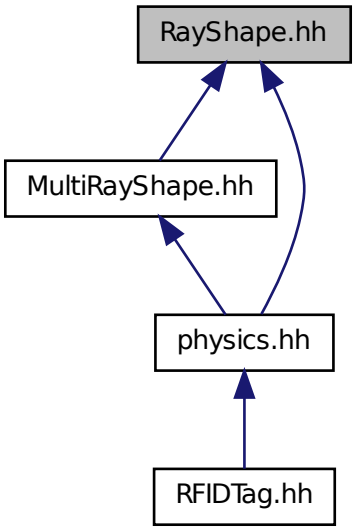
11.120 RayShape.hh File Reference

```
#include <string>
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/physics/Shape.hh"
```

Include dependency graph for RayShape.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::RayShape**

  Base (p. *141*) *class for Ray collision geometry.*

Namespaces

- namespace **gazebo**

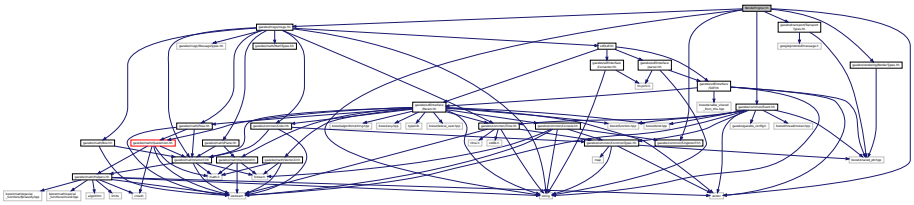
  Forward declarations for the common classes.
- namespace **gazebo::physics**

  namespace for physics

11.121 RenderEngine.hh File Reference

```
#include <vector>
#include <string>
#include "gazebo/msgs/msgs.hh"
#include "gazebo/common/SingletonT.hh"
#include "gazebo/common/Event.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/rendering/RenderTypes.hh"
```

Include dependency graph for RenderEngine.hh:



Classes

- class **gazebo::rendering::RenderEngine**

  Adaptor to Ogre3d.

Namespaces

- namespace **gazebo**

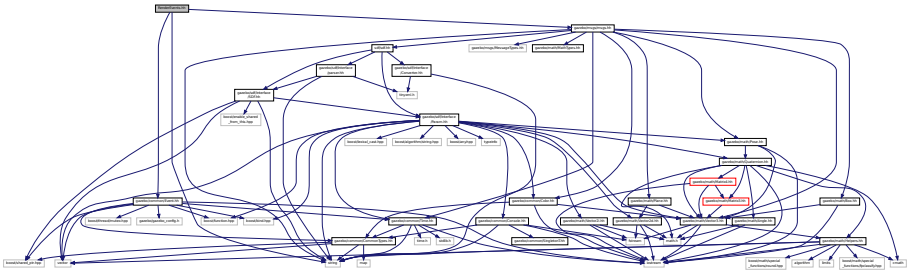
  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.
- namespace **Ogre**

11.122 RenderEvents.hh File Reference

```
#include <string>
#include "gazebo/common/Event.hh"
#include "gazebo/msgs/msgs.hh"
```

Include dependency graph for RenderEvents.hh:



Classes

- class **gazebo::rendering::Events**

  Base class for rendering events.

Namespaces

- namespace **gazebo**

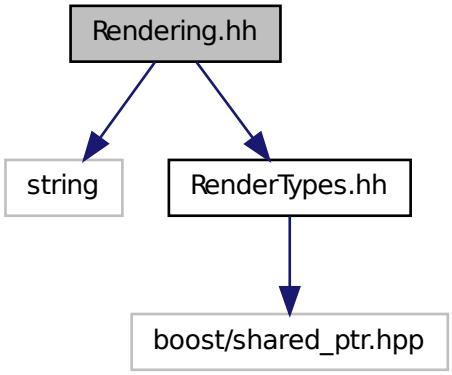
  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.

11.123 Rendering.hh File Reference

```
#include <string>
#include "RenderTypes.hh"
```

Include dependency graph for Rendering.hh:



Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.

Functions

- rendering::ScenePtr **gazebo::rendering::create_scene** (const std::string &_name, bool _enableVisualizations)

  *create **rendering::Scene*** (p. *707*) *by name.*
- bool **gazebo::rendering::fini** ()

  teardown rendering engine.
- rendering::ScenePtr **gazebo::rendering::get_scene** (const std::string &_name="")

  *get pointer to **rendering::Scene*** (p. *707*) *by name.*

- bool **gazebo::rendering::init** ()

  init rendering engine.
- bool **gazebo::rendering::load** ()

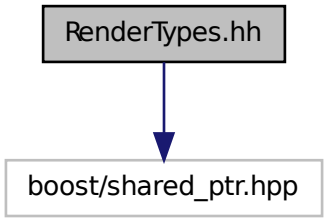
  load rendering engine.
- void **gazebo::rendering::remove_scene** (const std::string &_name)

  *remove a **rendering::Scene** (p. 707) by name*

11.124 RenderTypes.hh File Reference

```
#include <boost/shared_ptr.hpp>
```

Include dependency graph for RenderTypes.hh:



This graph shows which files directly or indirectly include this file:

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.

Macros

- #define **GZ_VISIBILITY_ALL** 0x0FFFFFFF

  Render everything visibility mask.
- #define **GZ_VISIBILITY_GUI** 0x00000001

  Render GUI visuals mask.

- #define **GZ_VISIBILITY_NOT_SELECTABLE** 0x00000002

  Render visuals that are not selectable mask.
- #define **GZ_VISIBILITY_SELECTION** 0x10000000

  Renders only objects that can be selected.

Typedefs

- typedef boost::shared_ptr < ArrowVisual > **gazebo::rendering::ArrowVisualPtr**
- typedef boost::shared_ptr < AxisVisual > **gazebo::rendering::AxisVisualPtr**
- typedef boost::shared_ptr< Camera > **gazebo::rendering::CameraPtr**
- typedef boost::shared_ptr < CameraVisual > **gazebo::rendering::CameraVisualPtr**
- typedef boost::shared_ptr < COMVisual > **gazebo::rendering::COMVisualPtr**
- typedef boost::shared_ptr < ContactVisual > **gazebo::rendering::ContactVisualPtr**
- typedef boost::shared_ptr < DepthCamera > **gazebo::rendering::DepthCameraPtr**
- typedef boost::shared_ptr < DynamicLines > **gazebo::rendering::DynamicLinesPtr**
- typedef boost::shared_ptr < GpuLaser > **gazebo::rendering::GpuLaserPtr**
- typedef boost::shared_ptr < JointVisual > **gazebo::rendering::JointVisualPtr**
- typedef boost::shared_ptr < LaserVisual > **gazebo::rendering::LaserVisualPtr**
- typedef boost::shared_ptr< Light > **gazebo::rendering::LightPtr**
- typedef boost::shared_ptr < RFIDTagVisual > **gazebo::rendering::RFIDTagVisualPtr**
- typedef boost::shared_ptr < RFIDVisual > **gazebo::rendering::RFIDVisualPtr**
- typedef boost::shared_ptr< Scene > **gazebo::rendering::ScenePtr**
- typedef boost::shared_ptr < SonarVisual > **gazebo::rendering::SonarVisualPtr**
- typedef boost::shared_ptr < UserCamera > **gazebo::rendering::UserCameraPtr**
- typedef boost::shared_ptr< Visual > **gazebo::rendering::VisualPtr**
- typedef boost::shared_ptr < WindowManager > **gazebo::rendering::WindowManagerPtr**
- typedef boost::shared_ptr < WrenchVisual > **gazebo::rendering::WrenchVisualPtr**

Enumerations

- enum **gazebo::rendering::RenderOpType** { **gazebo::rendering::RENDERING_POINT_LIST** = 0, **gazebo::rendering::RENDERING_LINE_LIST** = 1, **gazebo::rendering::RENDERING_LINE_STRIP** = 2, **gazebo::rendering::RENDERING_TRIANGLE_LIST** = 3, **gazebo::rendering::RENDERING_TRIANGLE_STRIP** = 4, **gazebo::rendering::RENDERING_TRIANGLE_FAN** = 5, **gazebo::rendering::RENDERING_MESH_RESOURCE** = 6 }

Type of render operation for a drawable.

11.124.1 Macro Definition Documentation

11.124.1.1 #define GZ_VISIBILITY_ALL 0x0FFFFFFF

Render everything visibility mask.

11.124.1.2 #define GZ_VISIBILITY_GUI 0x00000001

Render GUI visuals mask.

11.124.1.3 #define GZ_VISIBILITY_NOT_SELECTABLE 0x00000002

Render visuals that are not selectable mask.

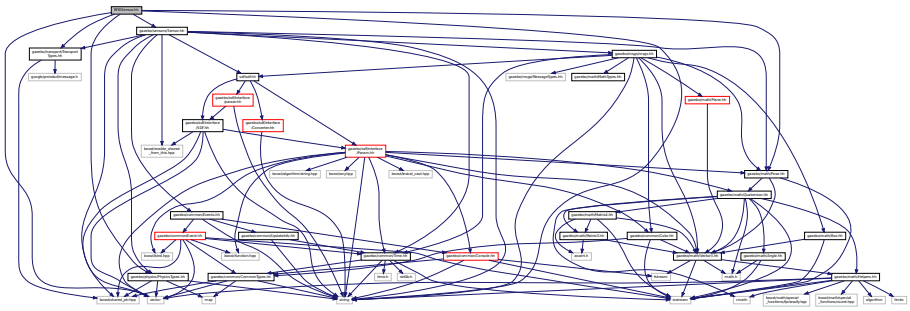
11.124.1.4 #define GZ_VISIBILITY_SELECTION 0x10000000

Renders only objects that can be selected.

11.125 RFIDSensor.hh File Reference

```
#include <vector>
#include <string>
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/math/Pose.hh"
#include "gazebo/sensors/Sensor.hh"
```

Include dependency graph for RFIDSensor.hh:



Classes

- class **gazebo::sensors::RFIDSensor**

  Sensor (p. *731*) *class for RFID type of sensor.*

Namespaces

- namespace **gazebo**

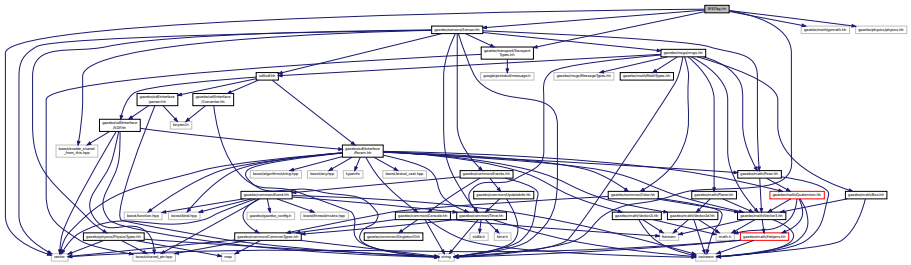
  Forward declarations for the common classes.
- namespace **gazebo::sensors**

  Sensors namespace.

11.126 RFIDTag.hh File Reference

```
#include <vector>
#include <string>
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/sensors/Sensor.hh"
#include "gazebo/math/gzmath.hh"
#include "gazebo/physics/physics.hh"
```

Include dependency graph for RFIDTag.hh:



Classes

- class **gazebo::sensors::RFIDTag**

  RFIDTag *(*p. 691*) to interact with RFIDTagSensors.*

Namespaces

- namespace **gazebo**

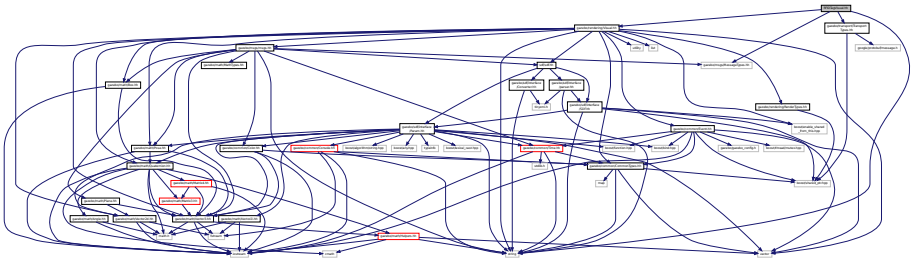
  Forward declarations for the common classes.
- namespace **gazebo::sensors**

  Sensors namespace.

11.127 RFIDTagVisual.hh File Reference

```
#include <string>
#include "gazebo/rendering/Visual.hh"
#include "gazebo/msgs/MessageTypes.hh"
#include "gazebo/transport/TransportTypes.hh"
```

Include dependency graph for RFIDTagVisual.hh:



Classes

- class **gazebo::rendering::RFIDTagVisual**

  Visualization for RFID tags sensor.

Namespaces

- namespace **gazebo**

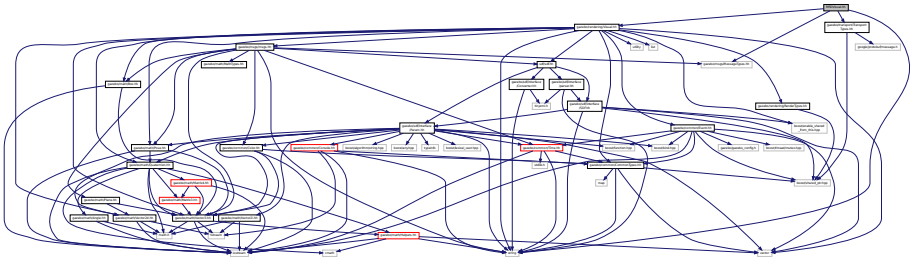
  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.

11.128 RFIDVisual.hh File Reference

```
#include <string>
#include "gazebo/rendering/Visual.hh"
#include "gazebo/msgs/MessageTypes.hh"
#include "gazebo/transport/TransportTypes.hh"
```

Include dependency graph for RFIDVisual.hh:



Classes

- class **gazebo::rendering::RFIDVisual**

  Visualization for RFID sensor.

Namespaces

- namespace **gazebo**

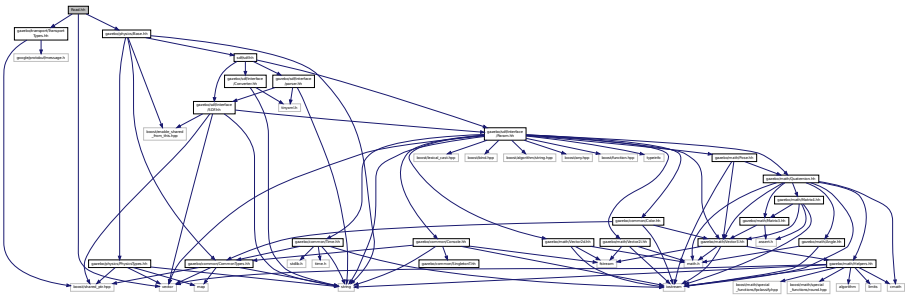
  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.

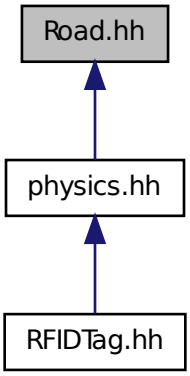
11.129 Road.hh File Reference

```
#include <vector>
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/physics/Base.hh"
```

Include dependency graph for Road.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::Road**

*for building a **Road** (p. 697) from SDF*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::physics**

  namespace for physics

11.130 Road2d.hh File Reference

```
#include <string>
#include <vector>
#include <list>
#include "gazebo/msgs/msgs.hh"
#include "gazebo/common/Events.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/rendering/ogre_gazebo.h"
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Spline.hh"
#include "gazebo/rendering/Visual.hh"
```

Include dependency graph for Road2d.hh:

Classes

- class **gazebo::rendering::Road2d**

Namespaces

- namespace **gazebo**

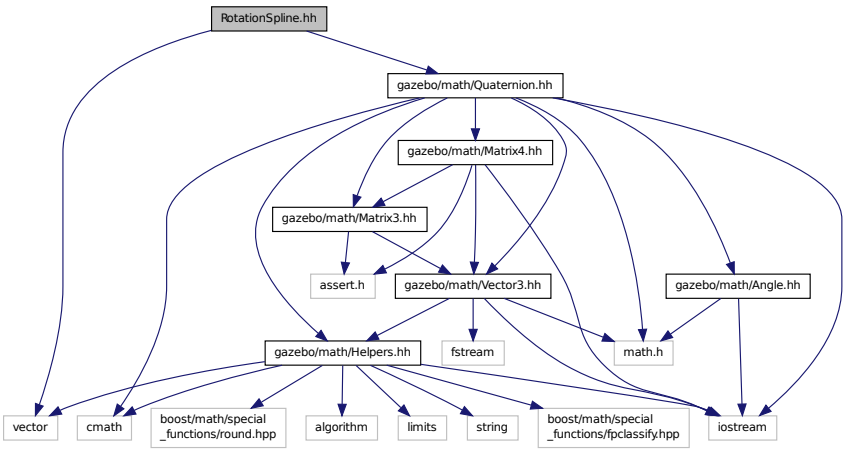
  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.

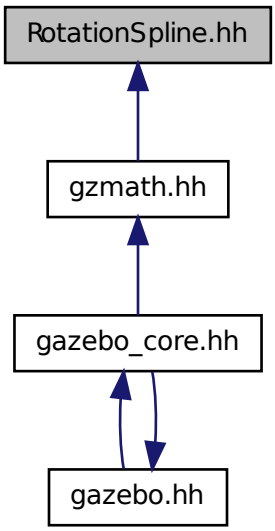
11.131 RotationSpline.hh File Reference

```
#include <vector>
#include "gazebo/math/Quaternion.hh"
```

Include dependency graph for RotationSpline.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::math::RotationSpline**

  Spline *(p. 793) for rotations.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::math**

  Math namespace.

11.132 RTShaderSystem.hh File Reference

```
#include <list>
#include <string>
#include <vector>
#include "gazebo/rendering/ogre_gazebo.h"
#include "gazebo/gazebo_config.h"
#include "gazebo/rendering/Camera.hh"
#include "gazebo/common/SingletonT.hh"
```

Include dependency graph for RTShaderSystem.hh:

Classes

- class **gazebo::rendering::RTShaderSystem**

  *Implements **Ogre** (p. 110)'s Run-Time Shader system.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::rendering**

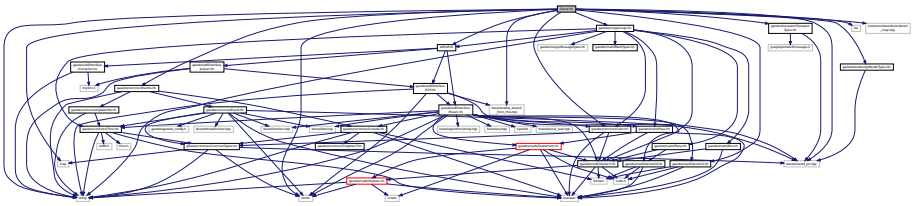
  Rendering namespace.

11.133 Scene.hh File Reference

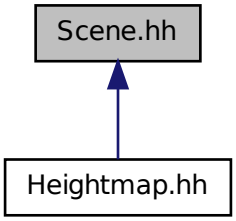
```
#include <vector>
```

```
#include <map>
#include <string>
#include <list>
#include <boost/enable_shared_from_this.hpp>
#include <boost/shared_ptr.hpp>
#include <boost/unordered/unordered_map.hpp>
#include <sdf/sdf.hh>
#include "gazebo/msgs/msgs.hh"
#include "gazebo/rendering/RenderTypes.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/common/Events.hh"
#include "gazebo/common/Color.hh"
#include "gazebo/math/Vector2i.hh"
```

Include dependency graph for Scene.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::rendering::Scene**

  Representation of an entire scene graph.

Namespaces

- namespace **boost**
- namespace **gazebo**

  Forward declarations for the common classes.

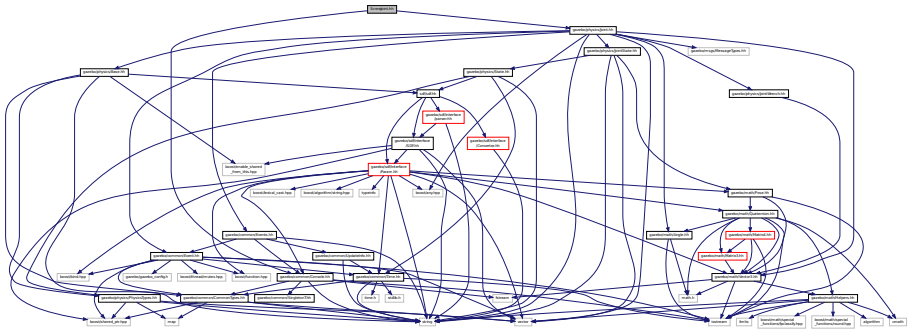
- namespace **gazebo::rendering**

  Rendering namespace.
- namespace **Ogre**
- namespace **SkyX**

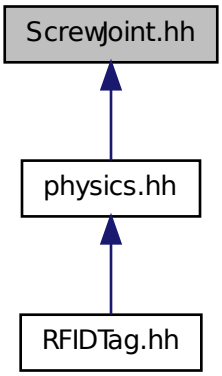
11.134 ScrewJoint.hh File Reference

```
#include "gazebo/physics/Joint.hh"
#include "gazebo/common/Console.hh"
```

Include dependency graph for ScrewJoint.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::ScrewJoint**< **T** >

  A screw joint, which has both prismatic and rotational DOFs.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

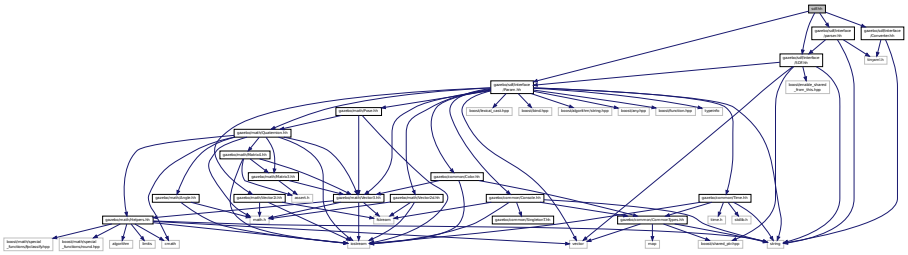
- namespace **gazebo::physics**

  namespace for physics

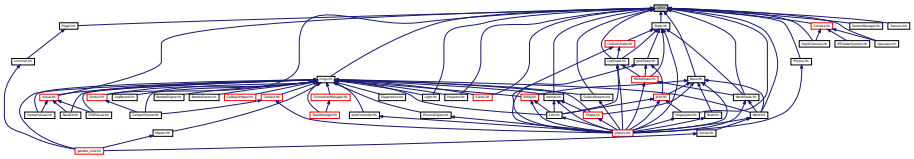
11.135 sdf.hh File Reference

```
#include "gazebo/sdf/interface/SDF.hh"
#include "gazebo/sdf/interface/Param.hh"
#include "gazebo/sdf/interface/parser.hh"
#include "gazebo/sdf/interface/Converter.hh"
```

Include dependency graph for sdf.hh:



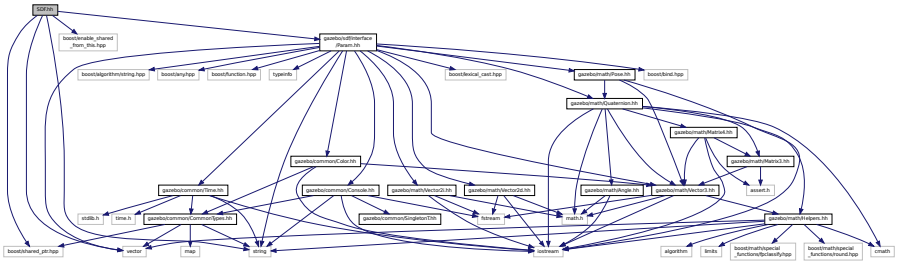
This graph shows which files directly or indirectly include this file:



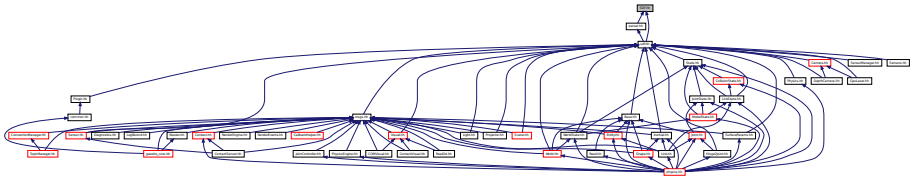
11.136 SDF.hh File Reference

```
#include <vector>
#include <string>
#include <boost/shared_ptr.hpp>
#include <boost/enable_shared_from_this.hpp>
#include "gazebo/sdf/interface/Param.hh"
```

Include dependency graph for SDF.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **sdf::Element**

  SDF *(*p. *728)* ***Element*** *(*p. *280) class.*

- class **sdf::SDF**

  Base ***SDF*** *(*p. *728) class.*

Namespaces

- namespace **sdf**

  namespace for Simulation Description Format parser

Macros

- #define **SDF_VERSION** "1.4"

Typedefs

- typedef boost::shared_ptr
  < Element > **sdf::ElementPtr**
- typedef std::vector< ElementPtr > **sdf::ElementPtr_V**
- typedef boost::shared_ptr< SDF > **sdf::SDFPtr**

Functions

- void **sdf::addURIPath** (const std::string &_uri, const std::string &_path)

  *A function that is used in the external **SDF** (p. 728) library.*
- void **sdf::setFindCallback** (boost::function< std::string(const std::string &)> _cb)

  *A function that is used in the external **SDF** (p. 728) library.*

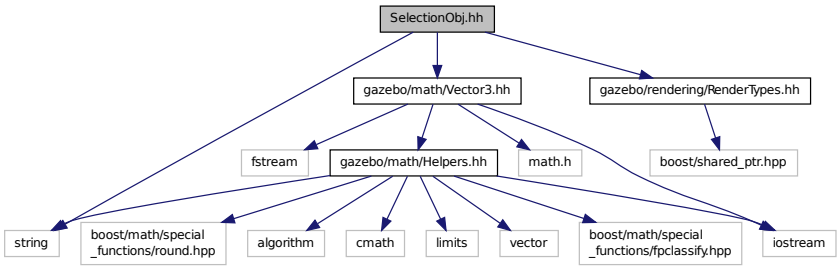
11.136.1 Macro Definition Documentation

11.136.1.1 #define SDF_VERSION "1.4"

11.137 SelectionObj.hh File Reference

```
#include <string>
#include "gazebo/math/Vector3.hh"
#include "gazebo/rendering/RenderTypes.hh"
```

Include dependency graph for SelectionObj.hh:



Classes

- class **gazebo::rendering::SelectionObj**

  A graphical selection object.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::rendering**

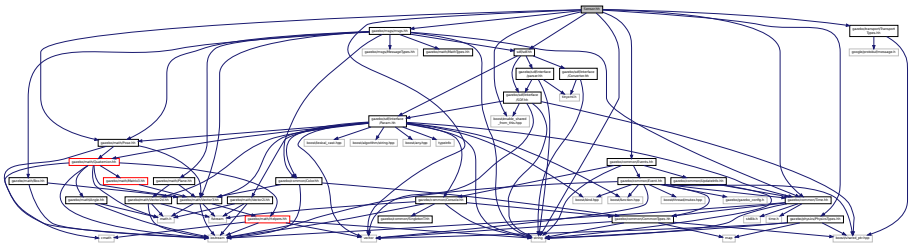
  Rendering namespace.

11.138 Sensor.hh File Reference

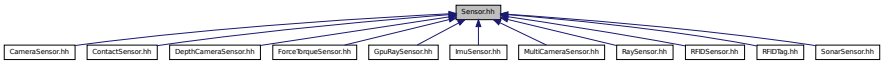
```
#include <boost/enable_shared_from_this.hpp>
```

```
#include <vector>
#include <string>
#include <sdf/sdf.hh>
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/msgs/msgs.hh"
#include "gazebo/common/Events.hh"
#include "gazebo/common/Time.hh"
#include "gazebo/math/Pose.hh"
#include "gazebo/transport/TransportTypes.hh"
```

Include dependency graph for Sensor.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::sensors::Sensor**

  Base class for sensors.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::sensors**

  Sensors namespace.

Enumerations

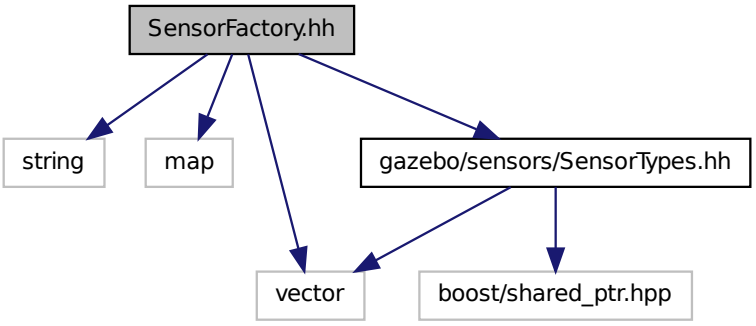
- enum **gazebo::sensors::SensorCategory** { **gazebo::sensors::IMAGE** = 0, **gazebo::sensors::RAY** = 1, **gazebo::sensors::OTHER** = 2, **gazebo::sensors::CATEGORY_COUNT** = 3 }

  SensorClass is used to categorize sensors.

11.139 SensorFactory.hh File Reference

```
#include <string>
#include <map>
#include <vector>
#include "gazebo/sensors/SensorTypes.hh"
```

Include dependency graph for SensorFactory.hh:



Classes

- class **gazebo::sensors::SensorFactory**

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::sensors**

  Sensors namespace.

Macros

- #define **GZ_REGISTER_STATIC_SENSOR**(name, classname)

  Static sensor registration macro.

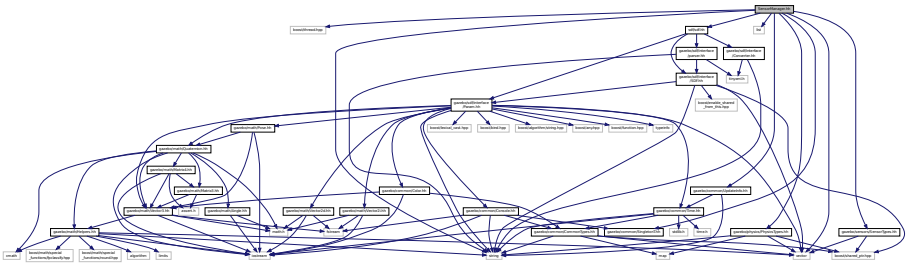
Typedefs

- typedef Sensor ∗(∗ **gazebo::sensors::SensorFactoryFn**)()

11.140 SensorManager.hh File Reference

```
#include <boost/thread.hpp>
#include <string>
#include <vector>
#include <list>
#include <sdf/sdf.hh>
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/common/SingletonT.hh"
#include "gazebo/common/UpdateInfo.hh"
#include "gazebo/sensors/SensorTypes.hh"
```

Include dependency graph for SensorManager.hh:



Classes

- class **gazebo::sensors::SensorManager**

  Class to manage and update all sensors.

- class **gazebo::sensors::SimTimeEvent**

- class **gazebo::sensors::SimTimeEventHandler**

  Monitors simulation time, and notifies conditions when a specified time has been reached.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

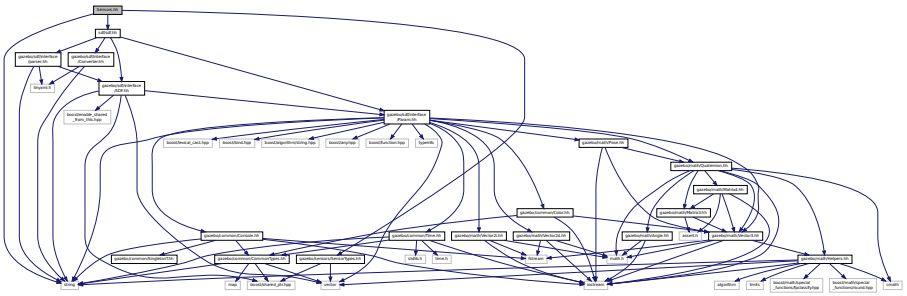
- namespace **gazebo::sensors**

  Sensors namespace.

11.141 Sensors.hh File Reference

```
#include <string>
#include <sdf/sdf.hh>
#include "gazebo/sensors/SensorTypes.hh"
```

Include dependency graph for Sensors.hh:



Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::sensors**

  Sensors namespace.

Functions

- std::string **gazebo::sensors::create_sensor** (**sdf::ElementPtr** _elem, const std::string &_worldName, const std::string &_parentName)

  Create a sensor using SDF.
- bool **gazebo::sensors::fini** ()

  shutdown the sensor generation loop.
- SensorPtr **gazebo::sensors::get_sensor** (const std::string &_name)

  Get a sensor using by name.
- bool **gazebo::sensors::init** ()

  initialize the sensor generation loop.
- bool **gazebo::sensors::load** ()

  Load the sensor library.
- void **gazebo::sensors::remove_sensor** (const std::string &_sensorName)

  Remove a sensor by name.
- bool **gazebo::sensors::remove_sensors** ()

  Remove all sensors.
- void **gazebo::sensors::run** () **GAZEBO_DEPRECATED**(1.5)

  Deprecated.
- void **gazebo::sensors::run_once** (bool _force=false)

  Run the sensor generation one step.
- void **gazebo::sensors::run_threads** ()

  Run sensors in a threads. This is a non-blocking call.
- void **gazebo::sensors::stop** ()

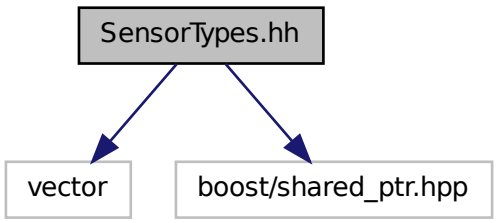
  Stop the sensor generation loop.

11.142 SensorTypes.hh File Reference

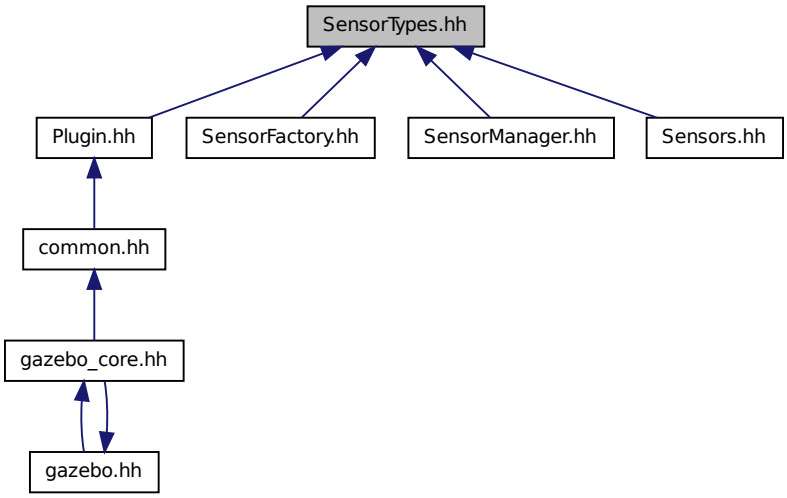
Forward declarations and typedefs for sensors.

```
#include <vector>
#include <boost/shared_ptr.hpp>
```

Include dependency graph for SensorTypes.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::sensors**

Sensors namespace.

Typedefs

- typedef std::vector < CameraSensorPtr > **gazebo::sensors::CameraSensor_V**
- typedef boost::shared_ptr < CameraSensor > **gazebo::sensors::CameraSensorPtr**
- typedef std::vector < ContactSensorPtr > **gazebo::sensors::ContactSensor_V**
- typedef boost::shared_ptr < ContactSensor > **gazebo::sensors::ContactSensorPtr**
- typedef std::vector < DepthCameraSensorPtr > **gazebo::sensors::DepthCameraSensor_V**
- typedef boost::shared_ptr < DepthCameraSensor > **gazebo::sensors::DepthCameraSensorPtr**
- typedef boost::shared_ptr < ForceTorqueSensor > **gazebo::sensors::ForceTorqueSensorPtr**
- typedef std::vector < GpuRaySensorPtr > **gazebo::sensors::GpuRaySensor_V**
- typedef boost::shared_ptr < GpuRaySensor > **gazebo::sensors::GpuRaySensorPtr**
- typedef std::vector< ImuSensorPtr > **gazebo::sensors::ImuSensor_V**
- typedef boost::shared_ptr < ImuSensor > **gazebo::sensors::ImuSensorPtr**
- typedef std::vector< RaySensorPtr > **gazebo::sensors::RaySensor_V**
- typedef boost::shared_ptr < RaySensor > **gazebo::sensors::RaySensorPtr**
- typedef std::vector< RFIDSensor > **gazebo::sensors::RFIDSensor_V**
- typedef boost::shared_ptr < RFIDSensor > **gazebo::sensors::RFIDSensorPtr**
- typedef std::vector< RFIDTag > **gazebo::sensors::RFIDTag_V**
- typedef boost::shared_ptr < RFIDTag > **gazebo::sensors::RFIDTagPtr**
- typedef std::vector< SensorPtr > **gazebo::sensors::Sensor_V**
- typedef boost::shared_ptr< Sensor > **gazebo::sensors::SensorPtr**
- typedef boost::shared_ptr < SonarSensor > **gazebo::sensors::SonarSensorPtr**

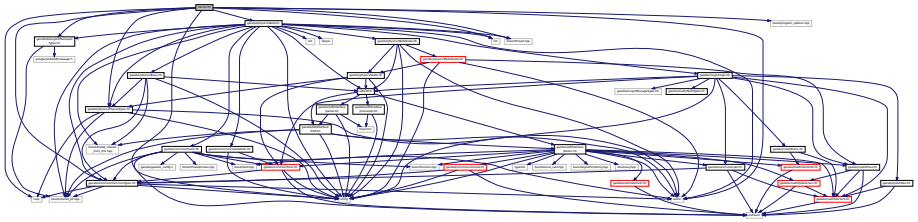
11.142.1 Detailed Description

Forward declarations and typedefs for sensors.

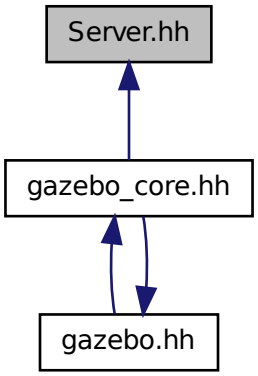
11.143 Server.hh File Reference

```
#include <string>
#include <vector>
#include <list>
#include <map>
#include <boost/program_options.hpp>
#include <boost/thread.hpp>
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/physics/World.hh"
```

Include dependency graph for Server.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::Server**

Namespaces

- namespace **boost**

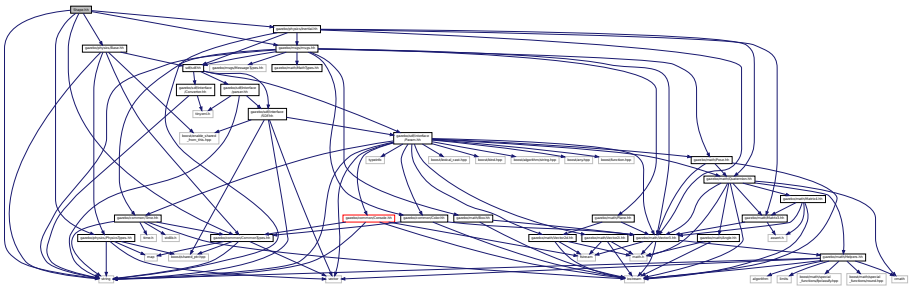
- namespace **gazebo**

  Forward declarations for the common classes.

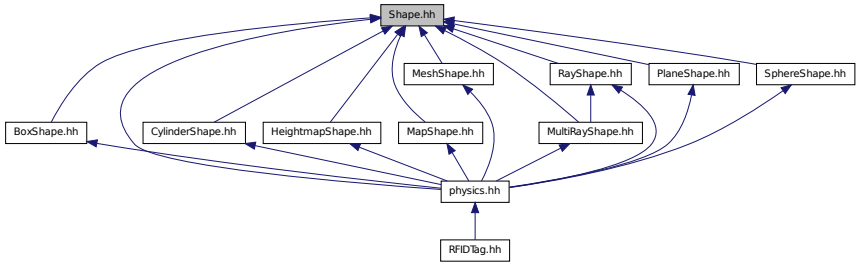
11.144 Shape.hh File Reference

```
#include <string>
#include "gazebo/msgs/msgs.hh"
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/physics/Inertial.hh"
#include "gazebo/physics/Base.hh"
```

Include dependency graph for Shape.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::Shape**

  Base *(p. 141) class for all shapes.*

Namespaces

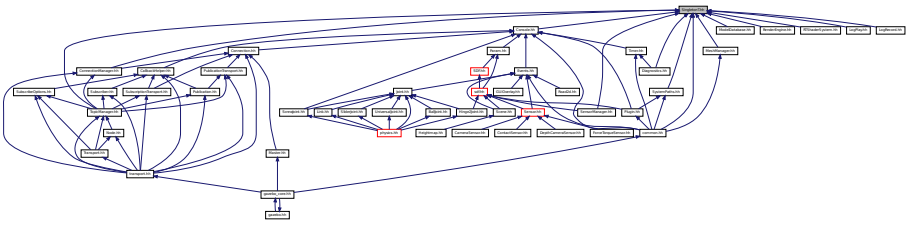
- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::physics**

  namespace for physics

11.145 SingletonT.hh File Reference

This graph shows which files directly or indirectly include this file:



Classes

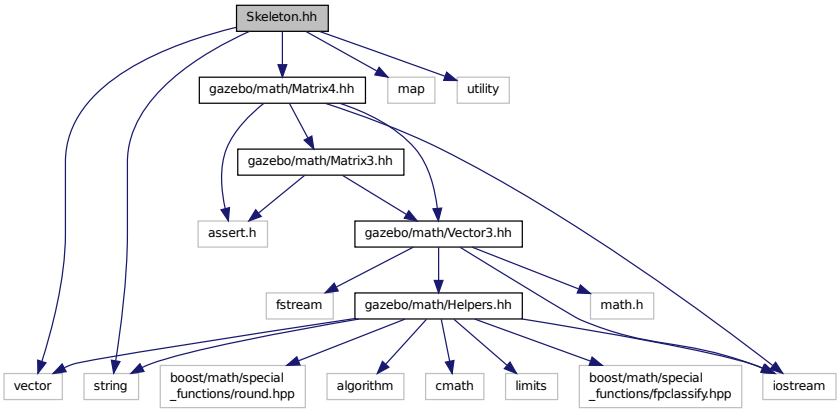
- class **SingletonT< T >**

  Singleton template class.

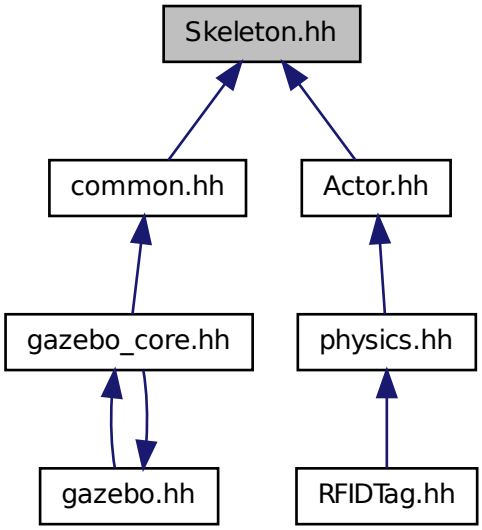
11.146 Skeleton.hh File Reference

```
#include <vector>
#include <string>
#include <map>
#include <utility>
#include "gazebo/math/Matrix4.hh"
```

Include dependency graph for Skeleton.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::NodeTransform**

  NodeTransform *(*p. *576)* ***Skeleton.hh*** *(*p. *1132) common/common.hh*
- class **gazebo::common::Skeleton**

  A skeleton.
- class **gazebo::common::SkeletonNode**

  A skeleton node.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

  Common namespace.

Typedefs

- typedef std::map< unsigned int, SkeletonNode * > **gazebo::common::NodeMap**
- typedef std::map< unsigned int, SkeletonNode * >::iterator **gazebo::common::NodeMapIter**

- typedef std::map< double,
  std::vector< NodeTransform > > **gazebo::common::RawNodeAnim**

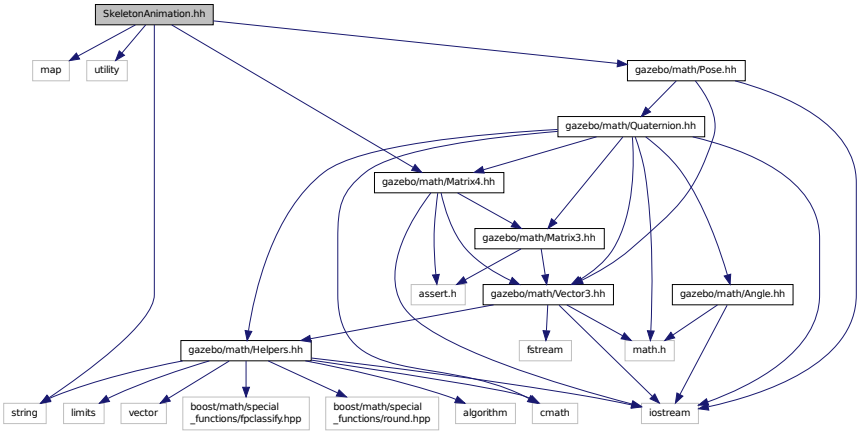
- typedef std::vector
  < std::vector< std::pair
  < std::string, double > > > **gazebo::common::RawNodeWeights**

- typedef std::map< std::string,
  RawNodeAnim > **gazebo::common::RawSkeletonAnim**

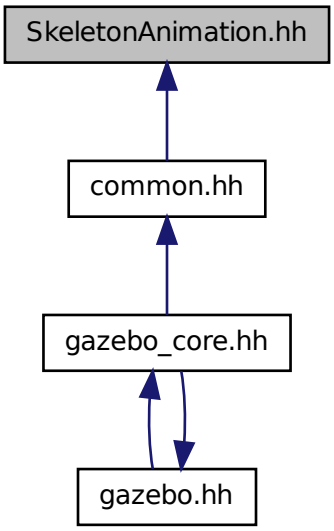
11.147 SkeletonAnimation.hh File Reference

```
#include <map>
#include <utility>
#include <string>
#include "gazebo/math/Matrix4.hh"
#include "gazebo/math/Pose.hh"
```

Include dependency graph for SkeletonAnimation.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::NodeAnimation**

  Node animation.

- class **gazebo::common::SkeletonAnimation**

  Skeleton *(p. 760) animation.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

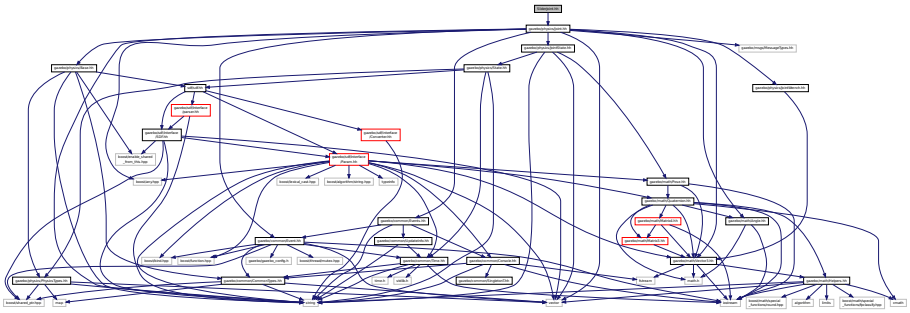
- namespace **gazebo::common**

  Common namespace.

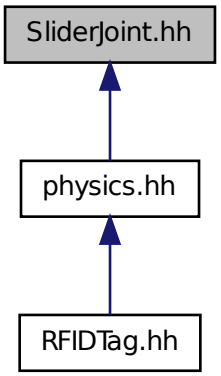
11.148 SliderJoint.hh File Reference

```
#include "gazebo/physics/Joint.hh"
```

Include dependency graph for SliderJoint.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::SliderJoint**< **T** >

  A slider joint.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

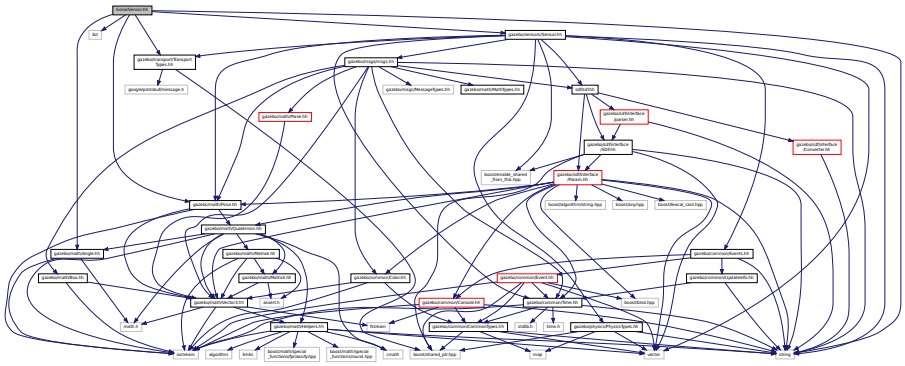
- namespace **gazebo::physics**

  namespace for physics

11.149 SonarSensor.hh File Reference

```
#include <string>
#include <list>
#include "gazebo/math/Angle.hh"
#include "gazebo/math/Pose.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/sensors/Sensor.hh"
```

Include dependency graph for SonarSensor.hh:



Classes

- class **gazebo::sensors::SonarSensor**

  Sensor (p. *731*) *with sonar cone.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

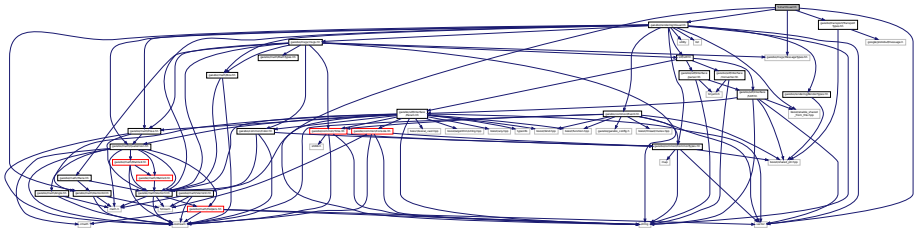
- namespace **gazebo::sensors**

  Sensors namespace.

11.150 SonarVisual.hh File Reference

```
#include <string>
#include <vector>
#include "gazebo/rendering/Visual.hh"
#include "gazebo/msgs/MessageTypes.hh"
#include "gazebo/transport/TransportTypes.hh"
```

Include dependency graph for SonarVisual.hh:



Classes

- class **gazebo::rendering::SonarVisual**

  Visualization for sonar data.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

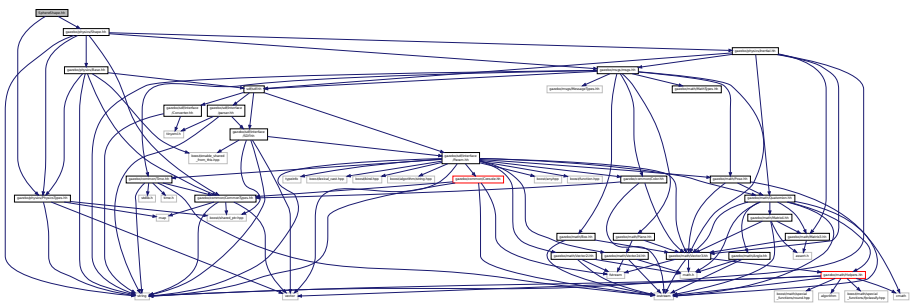
- namespace **gazebo::rendering**

  Rendering namespace.

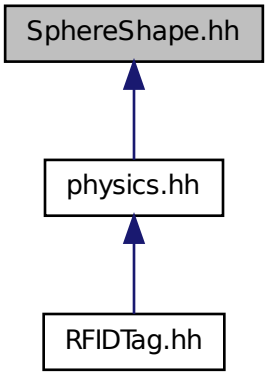
11.151 SphereShape.hh File Reference

```
#include "gazebo/physics/Shape.hh"
#include "gazebo/physics/PhysicsTypes.hh"
```

Include dependency graph for SphereShape.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::SphereShape**

  Sphere collision shape.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

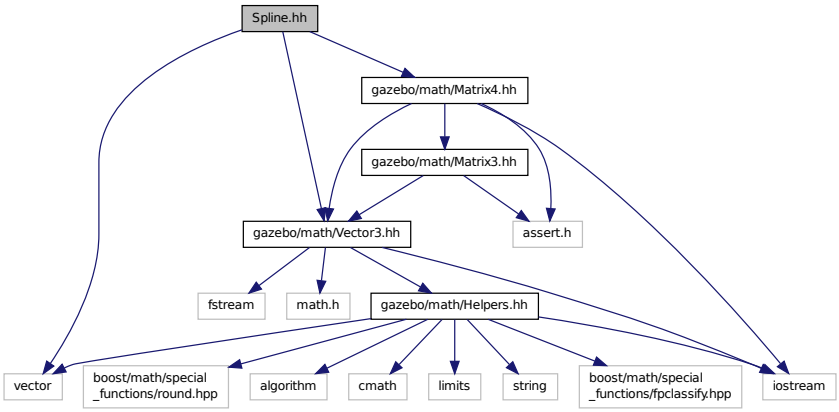
- namespace **gazebo::physics**

  namespace for physics

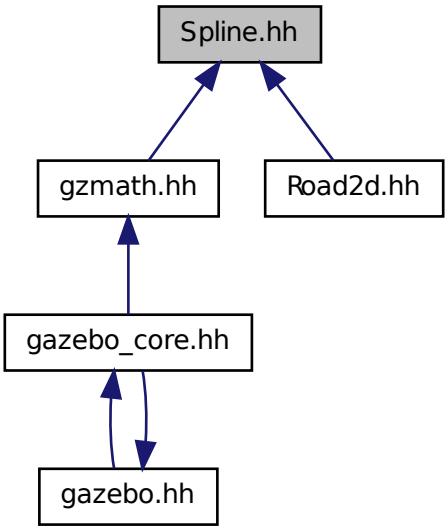
11.152 Spline.hh File Reference

```
#include <vector>
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Matrix4.hh"
```

Include dependency graph for Spline.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::math::Spline**

  Splines.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

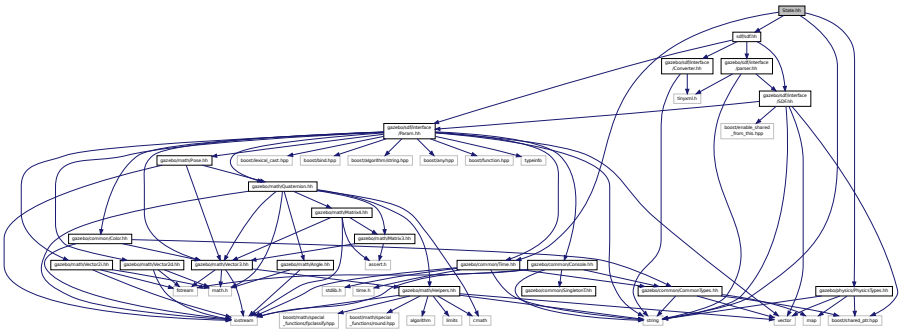
- namespace **gazebo::math**

  Math namespace.

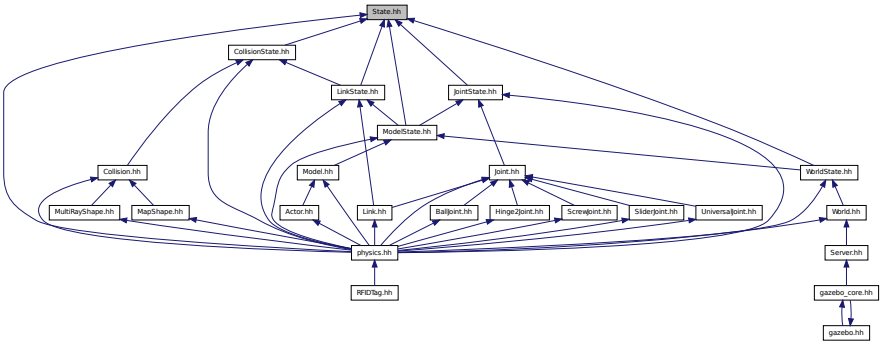
11.153 State.hh File Reference

```
#include <string>
#include <sdf/sdf.hh>
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/common/Time.hh"
```

Include dependency graph for State.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::State**

  State *(p. 797) of an entity.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

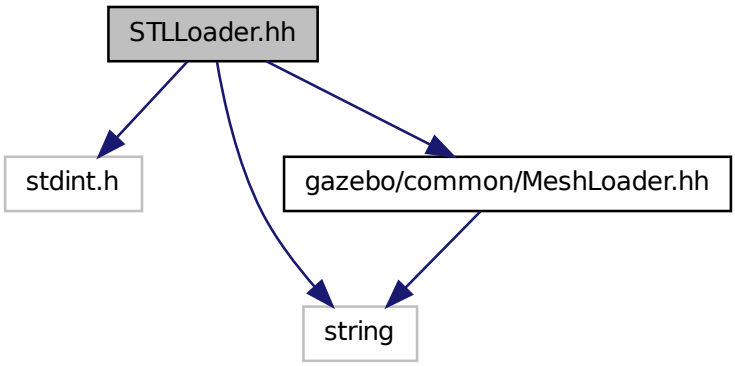
- namespace **gazebo::physics**

  namespace for physics

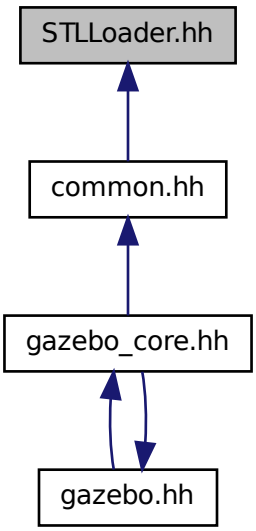
11.154 STLLoader.hh File Reference

```
#include <stdint.h>
#include <string>
#include "gazebo/common/MeshLoader.hh"
```

Include dependency graph for STLLoader.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::STLLoader**

  Class used to load STL mesh files.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

  Common namespace.

Macros

- #define **COR3_MAX** 200000
- #define **FACE_MAX** 200000
- #define **LINE_MAX_LEN** 256
- #define **ORDER_MAX** 10

11.154.1 Macro Definition Documentation

11.154.1.1 #define COR3_MAX 200000

11.154.1.2 #define FACE_MAX 200000

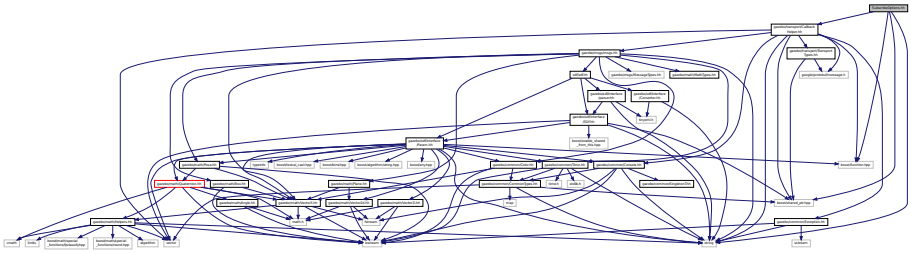
11.154.1.3 #define LINE_MAX_LEN 256

11.154.1.4 #define ORDER_MAX 10

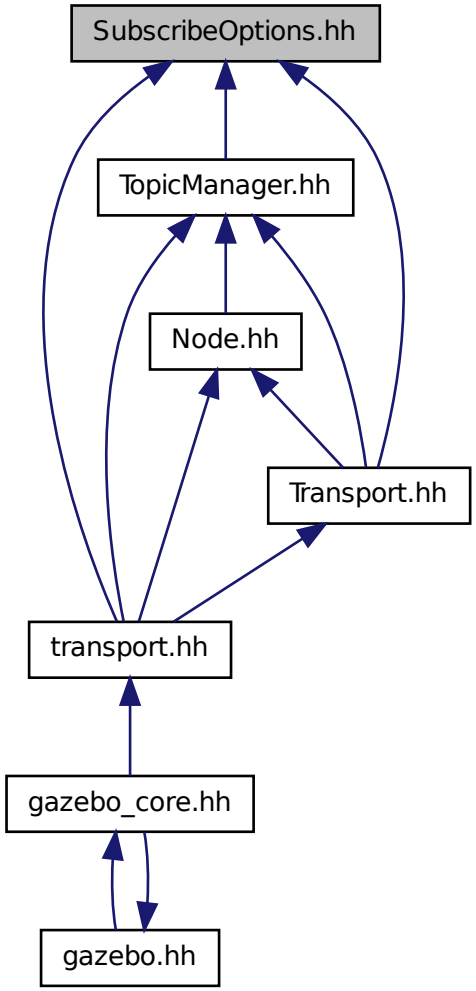
11.155 SubscribeOptions.hh File Reference

```
#include <boost/function.hpp>
#include <boost/shared_ptr.hpp>
#include <string>
#include "gazebo/transport/CallbackHelper.hh"
```

Include dependency graph for SubscribeOptions.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::transport::SubscribeOptions**

  Options for a subscription.

Namespaces

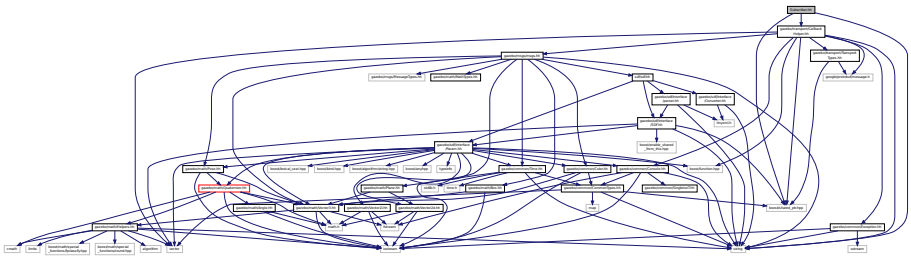
- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::transport**

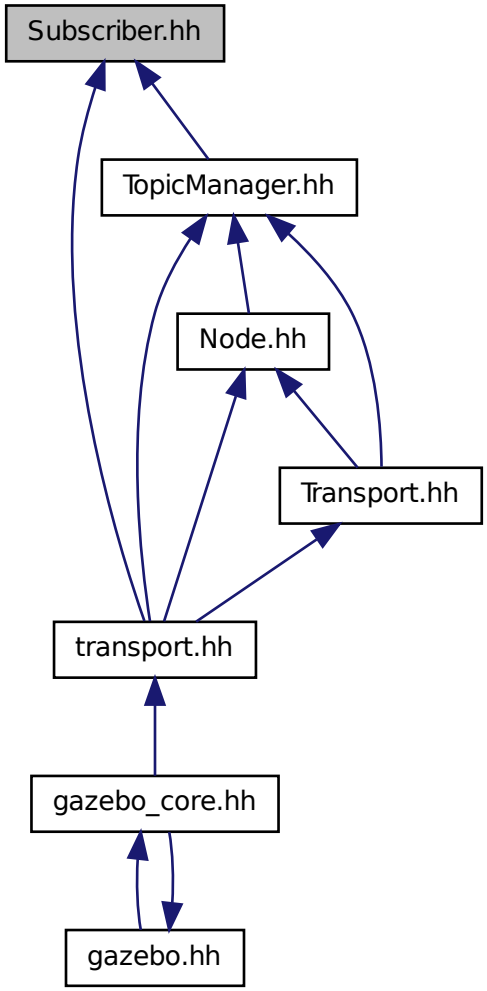
11.156 Subscriber.hh File Reference

```
#include <string>
#include <boost/shared_ptr.hpp>
#include "gazebo/transport/CallbackHelper.hh"
```

Include dependency graph for Subscriber.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::transport::Subscriber**

  A subscriber to a topic.

Namespaces

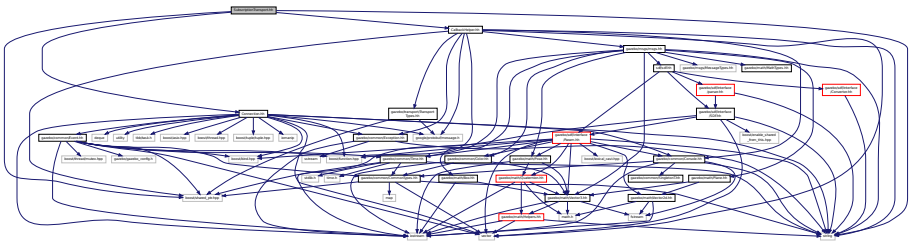
- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::transport**

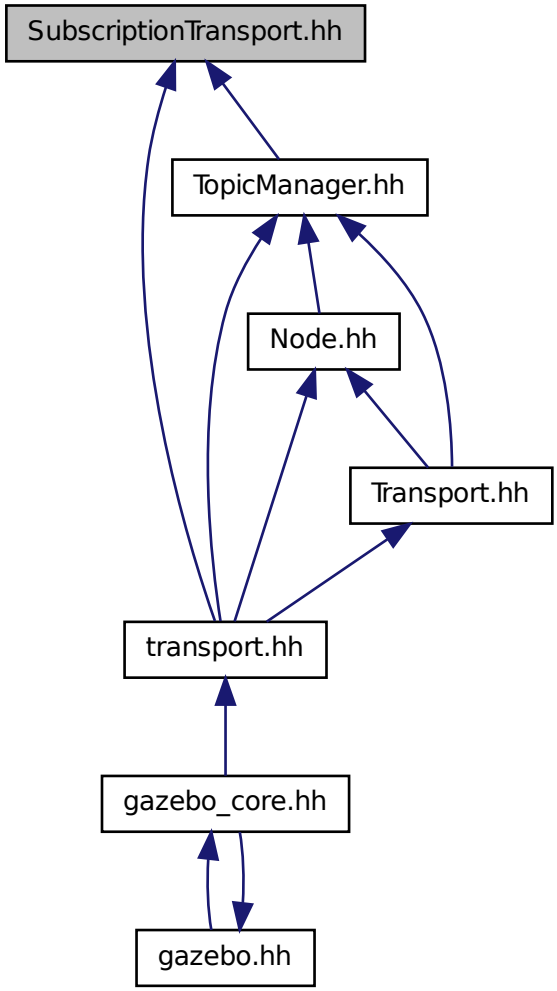
11.157 SubscriptionTransport.hh File Reference

```
#include <boost/shared_ptr.hpp>
#include <string>
#include "Connection.hh"
#include "CallbackHelper.hh"
```

Include dependency graph for SubscriptionTransport.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::transport::SubscriptionTransport**

  transport/transport.hh

Namespaces

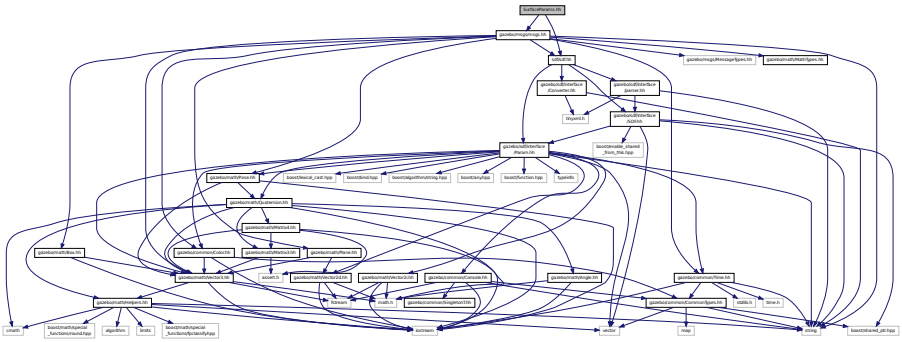
- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::transport**

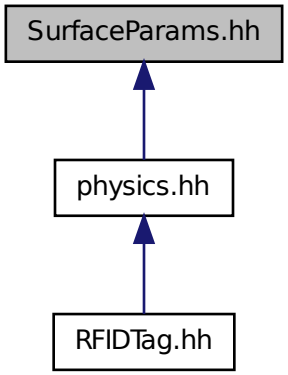
11.158 SurfaceParams.hh File Reference

```
#include <sdf/sdf.hh>
#include "gazebo/msgs/msgs.hh"
```

Include dependency graph for SurfaceParams.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::SurfaceParams**

  SurfaceParams (p. *820*) *defines various Surface contact parameters.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

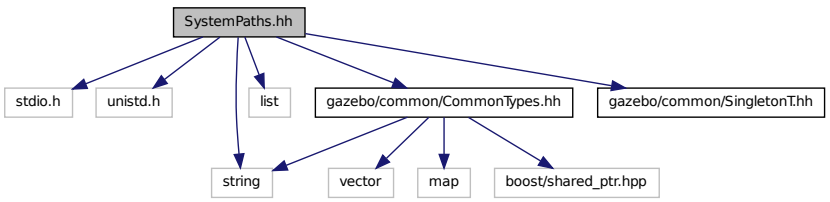
- namespace **gazebo::physics**

namespace for physics

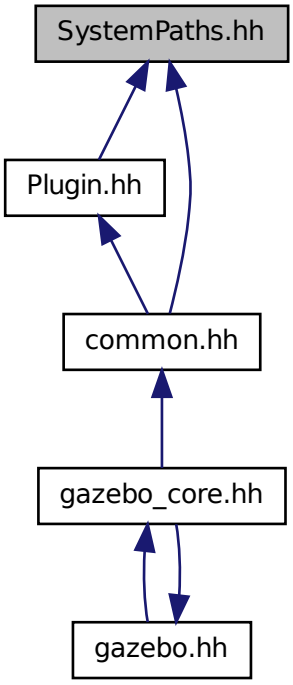
11.159 SystemPaths.hh File Reference

```
#include <stdio.h>
#include <unistd.h>
#include <string>
#include <list>
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/common/SingletonT.hh"
```

Include dependency graph for SystemPaths.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::SystemPaths**

  Functions to handle getting system paths, keeps track of:

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

  Common namespace.

Macros

- #define **GetCurrentDir** getcwd
- #define **LINUX**

11.159.1 Macro Definition Documentation

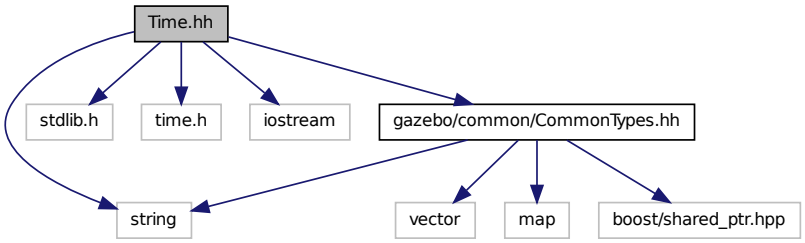
11.159.1.1 #define GetCurrentDir getcwd

11.159.1.2 #define LINUX

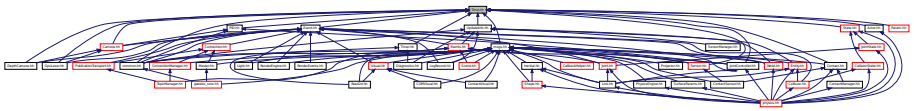
11.160 Time.hh File Reference

```
#include <string>
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include "gazebo/common/CommonTypes.hh"
```

Include dependency graph for Time.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::Time**

  *A **Time** (p. 831) class, can be used to hold wall- or sim-time.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

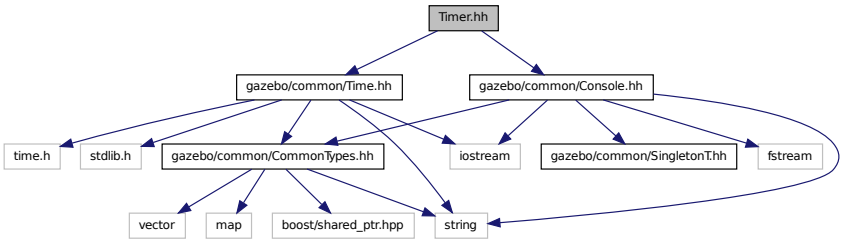
- namespace **gazebo::common**

  Common namespace.

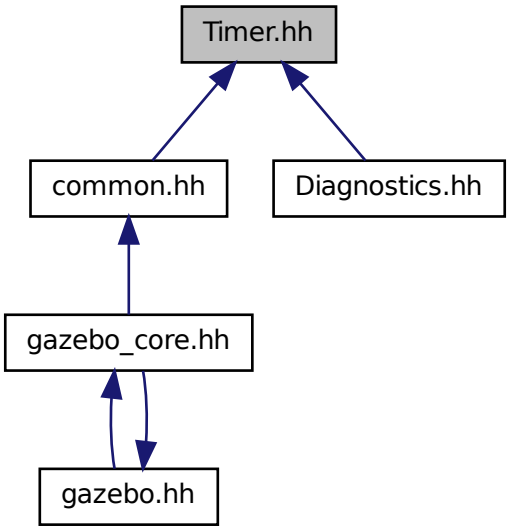
11.161 Timer.hh File Reference

```
#include "gazebo/common/Console.hh"
#include "gazebo/common/Time.hh"
```

Include dependency graph for Timer.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::Timer**

  A timer class, used to time things in real world walltime.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

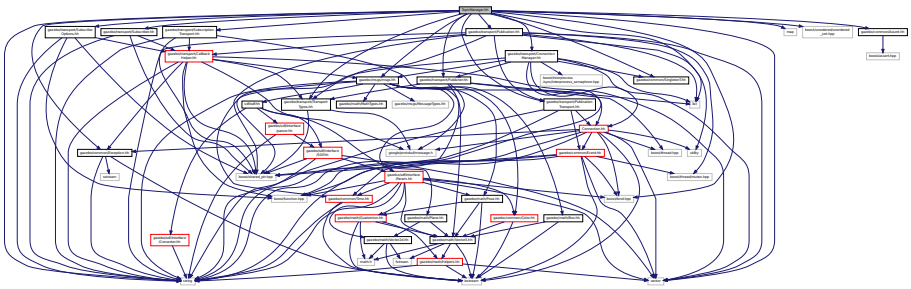
- namespace **gazebo::common**

  Common namespace.

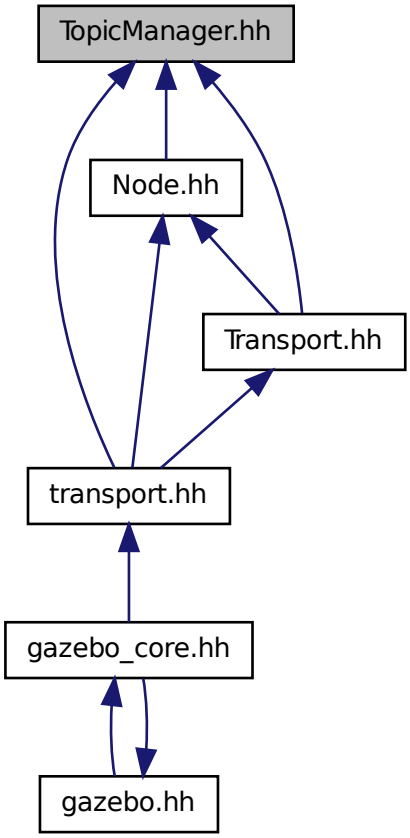
11.162 TopicManager.hh File Reference

```
#include <boost/bind.hpp>
#include <map>
#include <list>
#include <string>
#include <vector>
#include <boost/unordered/unordered_set.hpp>
#include "gazebo/common/Assert.hh"
#include "gazebo/common/Exception.hh"
#include "gazebo/msgs/msgs.hh"
#include "gazebo/common/SingletonT.hh"
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/transport/SubscribeOptions.hh"
#include "gazebo/transport/SubscriptionTransport.hh"
#include "gazebo/transport/PublicationTransport.hh"
#include "gazebo/transport/ConnectionManager.hh"
#include "gazebo/transport/Publisher.hh"
#include "gazebo/transport/Publication.hh"
#include "gazebo/transport/Subscriber.hh"
```

Include dependency graph for TopicManager.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::transport::TopicManager**

  Manages topics and their subscriptions.

Namespaces

- namespace **gazebo**

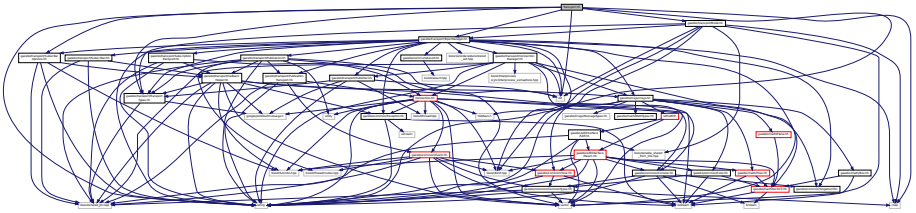
  Forward declarations for the common classes.
- namespace **gazebo::transport**

11.163 Transport.hh File Reference

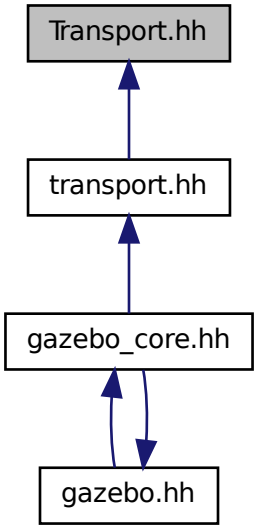
```
#include <boost/bind.hpp>
```

```
#include <string>
#include <list>
#include <map>
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/transport/SubscribeOptions.hh"
#include "gazebo/transport/Node.hh"
#include "gazebo/transport/TopicManager.hh"
```

Include dependency graph for Transport.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::transport**

Functions

- void **gazebo::transport::clear_buffers** ()

  Clear any remaining communication buffers.
- void **gazebo::transport::fini** ()

  Cleanup the transport component.
- bool **gazebo::transport::get_master_uri** (std::string &_master_host, unsigned int &_master_port)

  Get the hostname and port of the master from the GAZEBO_MASTER_URI environment variable.
- void **gazebo::transport::get_topic_namespaces** (std::list< std::string > &_namespaces)

  Return all the namespace (world names) on the master.
- std::map< std::string, std::list< std::string > > **gazebo::transport::getAdvertisedTopics** ()

  Get a list of all the topics and their message types.
- std::list< std::string > **gazebo::transport::getAdvertisedTopics** (const std::string &_msgType)

  Get a list of all the unique advertised topic names.
- bool **gazebo::transport::getMinimalComms** ()

  Get whether minimal comms has been enabled.
- std::string **gazebo::transport::getTopicMsgType** (const std::string &_topicName)

  Get the message typename that is published on the given topic.
- bool **gazebo::transport::init** (const std::string &_master_host="", unsigned int _master_port=0)

  Initialize the transport system.
- bool **gazebo::transport::is_stopped** ()

  Is the transport system stopped?
- void **gazebo::transport::pause_incoming** (bool _pause)

  Pause or unpause incoming messages.
- template<typename M > void **gazebo::transport::publish** (const std::string &_topic, const google::protobuf::Message &_message)

  A convenience function for a one-time publication of a message.
- boost::shared_ptr< msgs::Response > **gazebo::transport::request** (const std::string &_worldName, const std::string &_request, const std::string &_data="")

  Send a request and receive a response.
- void **gazebo::transport::requestNoReply** (const std::string &_worldName, const std::string &_request, const std::string &_data="")

  Send a request and don't wait for a response.
- void **gazebo::transport::requestNoReply** (NodePtr _node, const std::string &_request, const std::string &_data="")

  Send a request and don't wait for a response.
- void **gazebo::transport::run** ()

  Run the transport component.
- void **gazebo::transport::setMinimalComms** (bool _enabled)

  Set whether minimal comms should be used.
- void **gazebo::transport::stop** ()

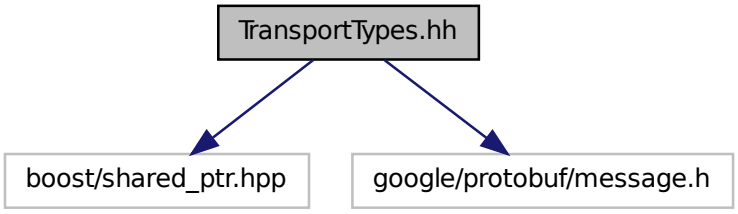
  Stop the transport component from running.

11.164 TransportTypes.hh File Reference

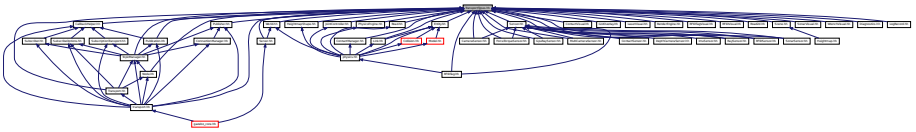
Forward declarations for transport.

```
#include <boost/shared_ptr.hpp>
#include <google/protobuf/message.h>
```

Include dependency graph for TransportTypes.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::transport**

Typedefs

- typedef boost::shared_ptr < google::protobuf::Message > **gazebo::transport::MessagePtr**
- typedef boost::shared_ptr< Node > **gazebo::transport::NodePtr**
- typedef boost::shared_ptr < Publication > **gazebo::transport::PublicationPtr**
- typedef boost::shared_ptr < PublicationTransport > **gazebo::transport::PublicationTransportPtr**
- typedef boost::shared_ptr < Publisher > **gazebo::transport::PublisherPtr**
- typedef boost::shared_ptr < Subscriber > **gazebo::transport::SubscriberPtr**
- typedef boost::shared_ptr < SubscriptionTransport > **gazebo::transport::SubscriptionTransportPtr**

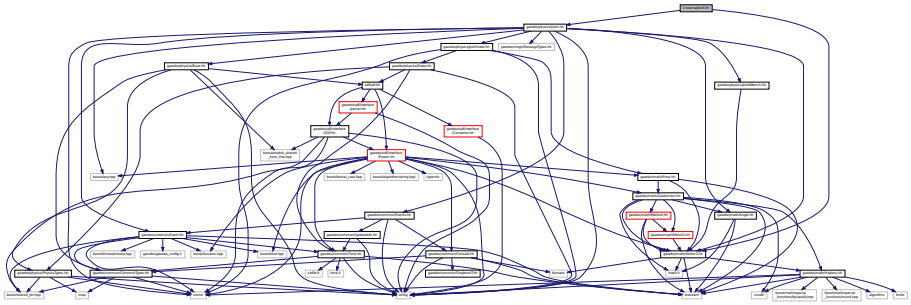
11.164.1 Detailed Description

Forward declarations for transport.

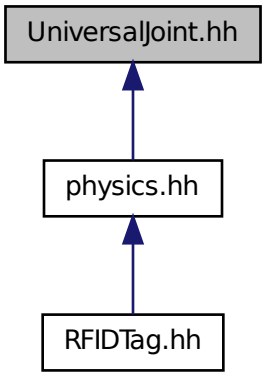
11.165 UniversalJoint.hh File Reference

```
#include "gazebo/math/Vector3.hh"
#include "gazebo/physics/Joint.hh"
```

Include dependency graph for UniversalJoint.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::UniversalJoint**< **T** >

  A universal joint.

Namespaces

- namespace **gazebo**

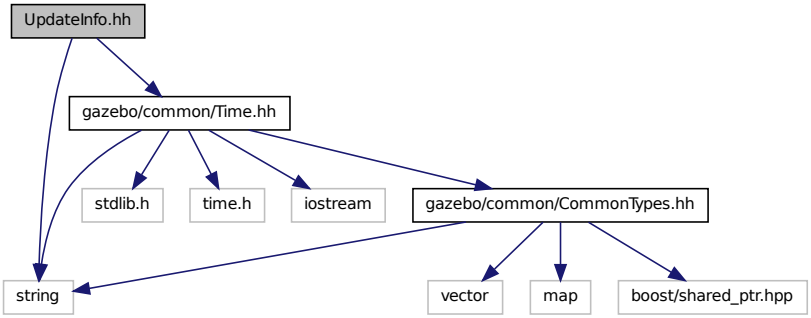
  Forward declarations for the common classes.
- namespace **gazebo::physics**

  namespace for physics

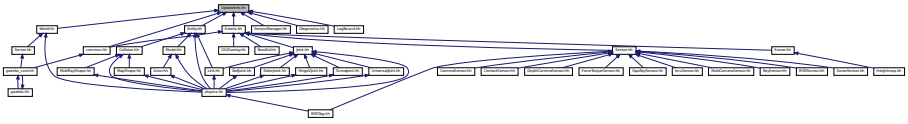
11.166 UpdateInfo.hh File Reference

```
#include <string>
#include "gazebo/common/Time.hh"
```

Include dependency graph for UpdateInfo.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::UpdateInfo**

  Information for use in an update event.

Namespaces

- namespace **gazebo**

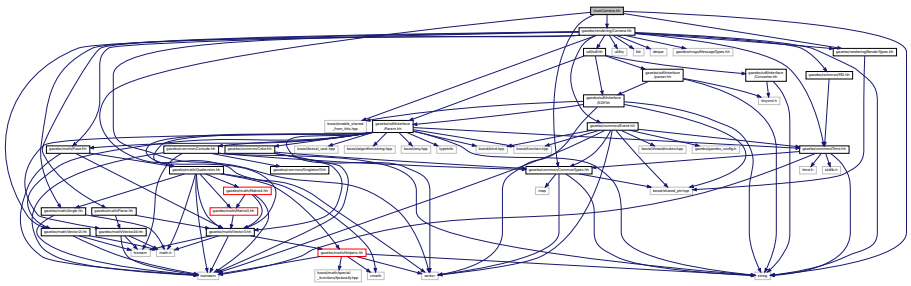
  Forward declarations for the common classes.
- namespace **gazebo::common**

  Common namespace.

11.167 UserCamera.hh File Reference

```
#include <string>
#include <vector>
#include "gazebo/rendering/Camera.hh"
#include "gazebo/rendering/RenderTypes.hh"
#include "gazebo/common/CommonTypes.hh"
```

Include dependency graph for UserCamera.hh:



Classes

- class **gazebo::rendering::UserCamera**

  A camera used for user visualization of a scene.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

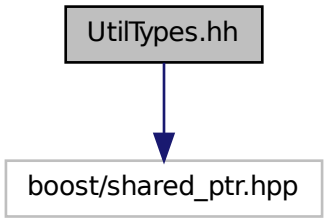
- namespace **gazebo::rendering**

  Rendering namespace.

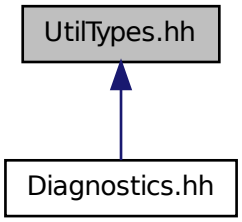
11.168 UtilTypes.hh File Reference

```
#include <boost/shared_ptr.hpp>
```

Include dependency graph for UtilTypes.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::util**

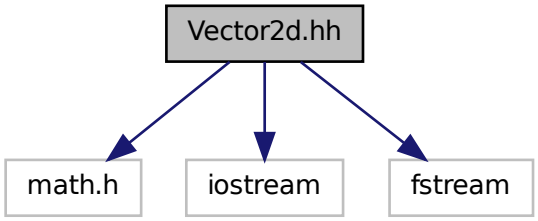
Typedefs

- typedef boost::shared_ptr
  < DiagnosticTimer > **gazebo::util::DiagnosticTimerPtr**

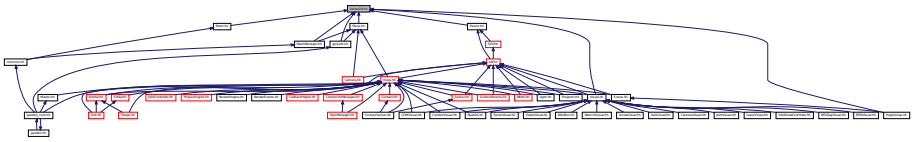
11.169 Vector2d.hh File Reference

```
#include <math.h>
#include <iostream>
#include <fstream>
```

Include dependency graph for Vector2d.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::math::Vector2d**

  Generic double x, y vector.

Namespaces

- namespace **gazebo**

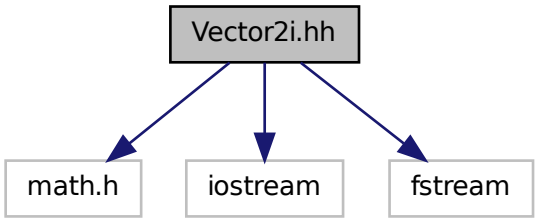
  Forward declarations for the common classes.
- namespace **gazebo::math**

  Math namespace.

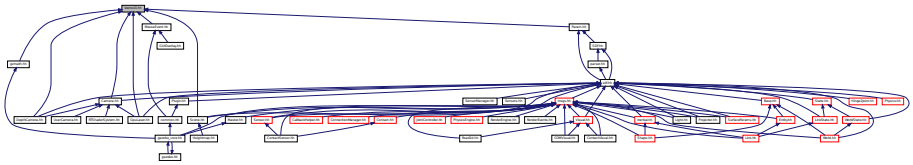
11.170 Vector2i.hh File Reference

```
#include <math.h>
#include <iostream>
#include <fstream>
```

Include dependency graph for Vector2i.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::math::Vector2i**

  Generic integer x, y vector.

Namespaces

- namespace **gazebo**

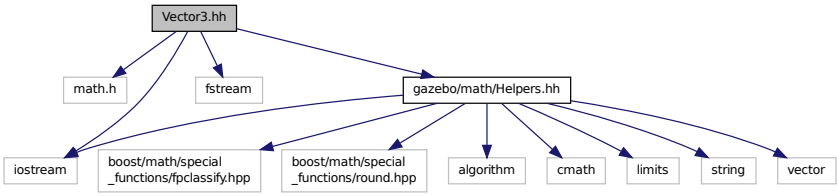
  Forward declarations for the common classes.
- namespace **gazebo::math**

  Math namespace.

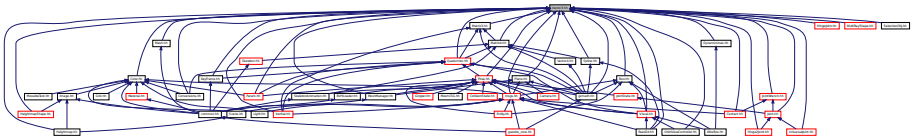
11.171 Vector3.hh File Reference

```
#include <math.h>
#include <iostream>
#include <fstream>
#include "gazebo/math/Helpers.hh"
```

Include dependency graph for Vector3.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::math::Vector3**

  *The **Vector3** (p. 890) class represents the generic vector containing 3 elements.*

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

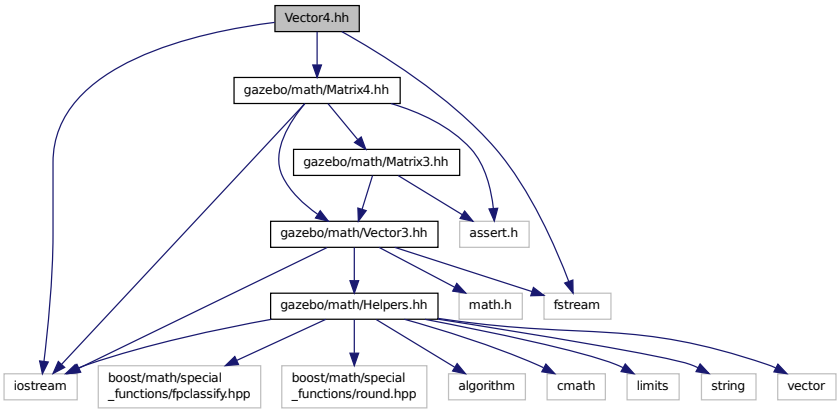
- namespace **gazebo::math**

  Math namespace.

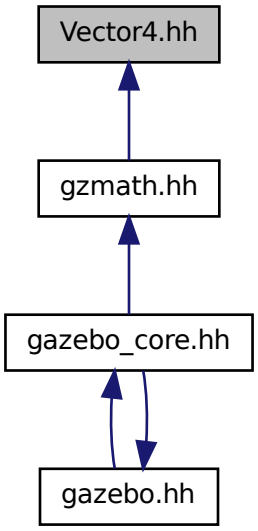
11.172 Vector4.hh File Reference

```
#include <iostream>
#include <fstream>
#include "gazebo/math/Matrix4.hh"
```

Include dependency graph for Vector4.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::math::Vector4**

  double Generic x, y, z, w vector

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

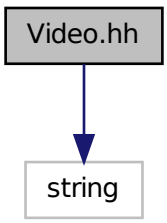
- namespace **gazebo::math**

  Math namespace.

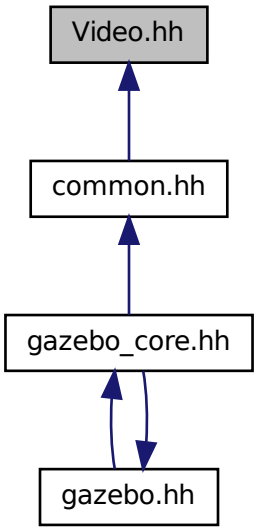
11.173 Video.hh File Reference

```
#include <string>
```

Include dependency graph for Video.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::common::Video**

  Handle video encoding and decoding using libavcodec.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

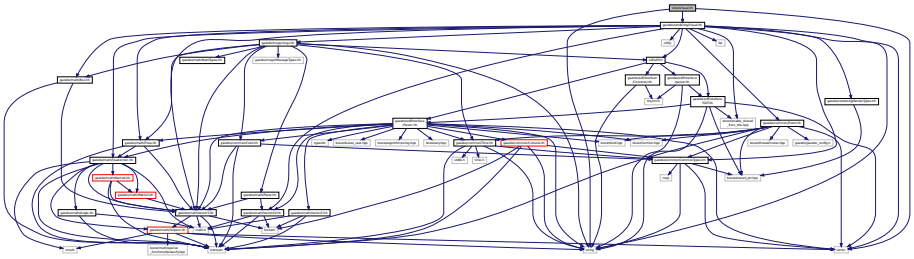
- namespace **gazebo::common**

  Common namespace.

11.174 VideoVisual.hh File Reference

```
#include <string>
#include <vector>
#include "gazebo/rendering/Visual.hh"
```

Include dependency graph for VideoVisual.hh:



Classes

- class **gazebo::rendering::VideoVisual**

  A visual element that displays a video as a texture.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::common**

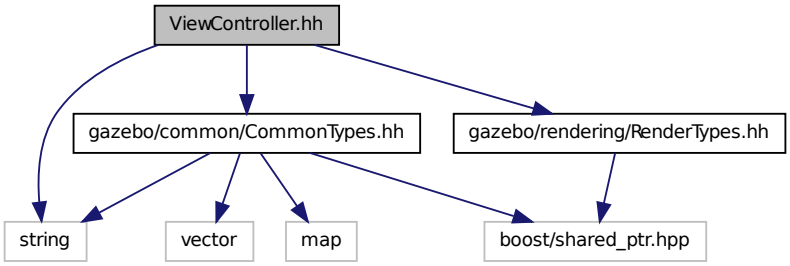
  Common namespace.
- namespace **gazebo::rendering**

  Rendering namespace.

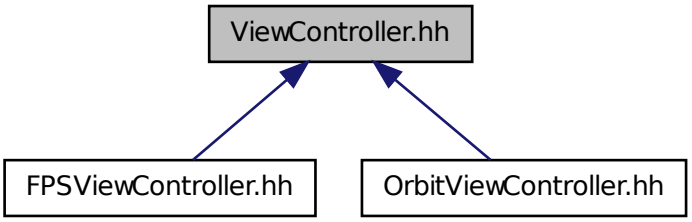
11.175 ViewController.hh File Reference

```
#include <string>
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/rendering/RenderTypes.hh"
```

Include dependency graph for ViewController.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::rendering::ViewController**

  Base class for view controllers.

Namespaces

- namespace **gazebo**

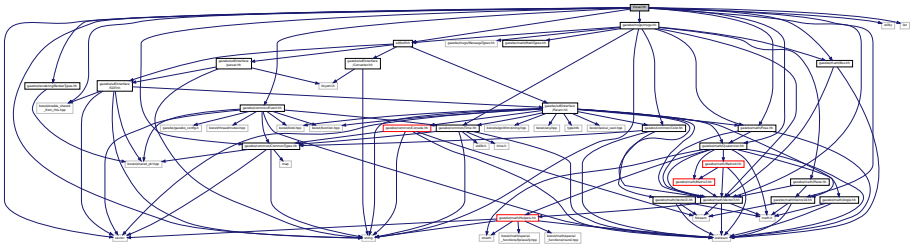
  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.

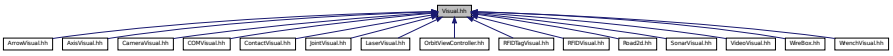
11.176 Visual.hh File Reference

```
#include <boost/enable_shared_from_this.hpp>
#include <string>
#include <utility>
#include <list>
#include <vector>
#include <sdf/sdf.hh>
#include "gazebo/msgs/msgs.hh"
#include "gazebo/common/Event.hh"
#include "gazebo/math/Box.hh"
#include "gazebo/math/Pose.hh"
#include "gazebo/math/Quaternion.hh"
#include "gazebo/math/Vector3.hh"
#include "gazebo/math/Vector2d.hh"
#include "gazebo/rendering/RenderTypes.hh"
#include "gazebo/common/CommonTypes.hh"
```

Include dependency graph for Visual.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::rendering::Visual**

  A renderable object.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::rendering**

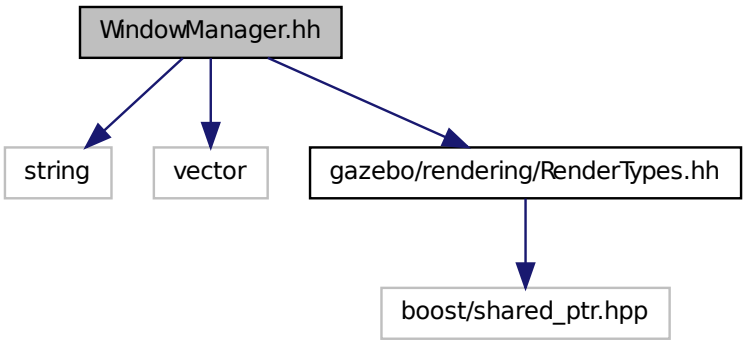
  Rendering namespace.

- namespace **Ogre**

11.177 WindowManager.hh File Reference

```
#include <string>
#include <vector>
#include "gazebo/rendering/RenderTypes.hh"
```

Include dependency graph for WindowManager.hh:



Classes

- class **gazebo::rendering::WindowManager**

  Class to mangage render windows.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

- namespace **gazebo::rendering**

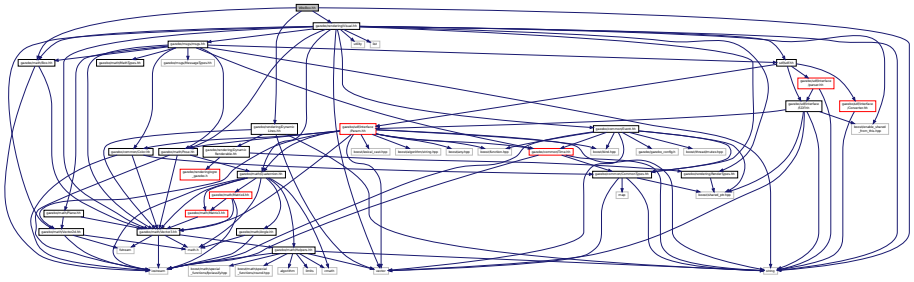
  Rendering namespace.

- namespace **Ogre**

11.178 WireBox.hh File Reference

```
#include <string>
#include "gazebo/math/Box.hh"
#include "gazebo/rendering/Visual.hh"
#include "gazebo/rendering/DynamicLines.hh"
```

Include dependency graph for WireBox.hh:



Classes

- class **gazebo::rendering::WireBox**

  Draws a wireframe box.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.

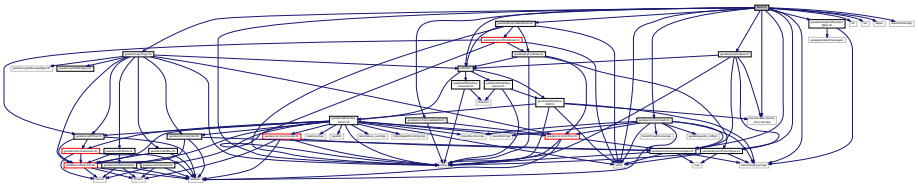
- namespace **gazebo::rendering**

  Rendering namespace.

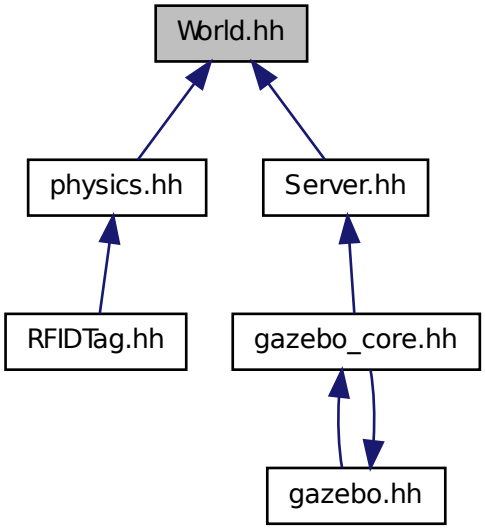
11.179 World.hh File Reference

```
#include <vector>
#include <list>
#include <set>
#include <deque>
#include <string>
#include <boost/thread.hpp>
#include <boost/enable_shared_from_this.hpp>
#include <boost/shared_ptr.hpp>
#include <sdf/sdf.hh>
#include "gazebo/transport/TransportTypes.hh"
#include "gazebo/msgs/msgs.hh"
#include "gazebo/common/CommonTypes.hh"
#include "gazebo/common/UpdateInfo.hh"
#include "gazebo/common/Event.hh"
#include "gazebo/physics/Base.hh"
#include "gazebo/physics/PhysicsTypes.hh"
#include "gazebo/physics/WorldState.hh"
```

Include dependency graph for World.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::World**

  The world provides access to all other object within a simulated environment.

Namespaces

- namespace **gazebo**

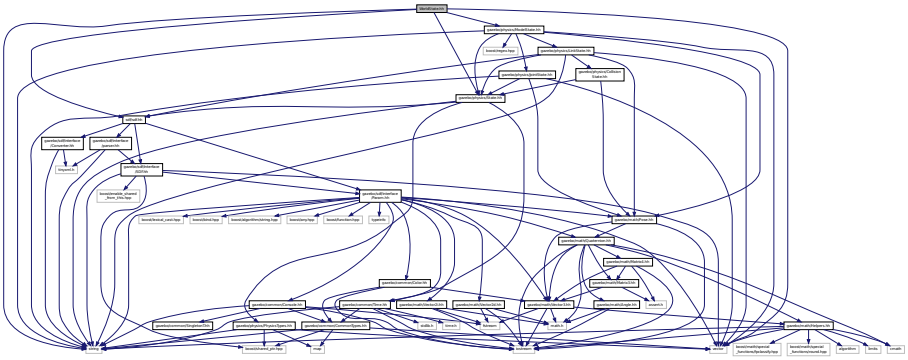
  Forward declarations for the common classes.
- namespace **gazebo::physics**

  namespace for physics

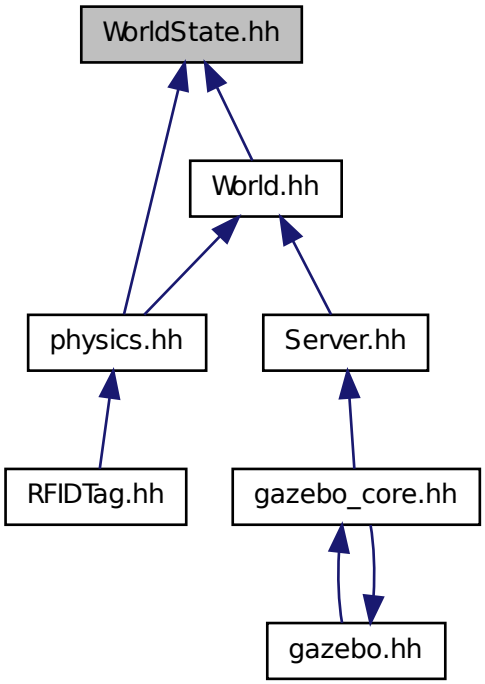
11.180 WorldState.hh File Reference

```
#include <string>
#include <vector>
#include <sdf/sdf.hh>
#include "gazebo/physics/State.hh"
#include "gazebo/physics/ModelState.hh"
```

Include dependency graph for WorldState.hh:



This graph shows which files directly or indirectly include this file:



Classes

- class **gazebo::physics::WorldState**

  *Store state information of a **physics::World** (p. 945) object.*

Namespaces

- namespace **gazebo**

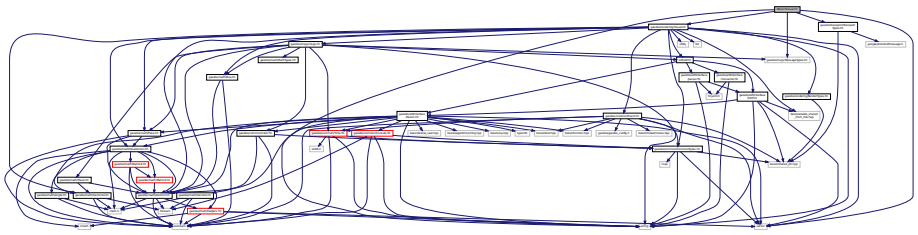
  Forward declarations for the common classes.
- namespace **gazebo::physics**

  namespace for physics

11.181 WrenchVisual.hh File Reference

```
#include <string>
#include <vector>
#include "gazebo/rendering/Visual.hh"
#include "gazebo/msgs/MessageTypes.hh"
#include "gazebo/transport/TransportTypes.hh"
```

Include dependency graph for WrenchVisual.hh:



Classes

- class **gazebo::rendering::WrenchVisual**

  Visualization for sonar data.

Namespaces

- namespace **gazebo**

  Forward declarations for the common classes.
- namespace **gazebo::rendering**

  Rendering namespace.

Index