Torque 3D Documentation

Release 3.5.1

GarageGames, LLC

February 04, 2017

Contents

1 Introduction 3

2 World Editor 5

3 GUI Editor 263

4 Artists Guide 275

5 Scripting 285

6 Engine 901

7 License 903

Torque 3D is a large piece of software. Chances are that most of the applications you have worked on up to this point have only been a fraction of the size of this SDK. This reference manual exists for the sole purpose of giving you, the user, a strong foundation to rely on while learning the engine.

The documentation is divided into multiple sections, each of which contains information related to specific subject. This means of organization allows you to jump to different chapters containing information that is pertinent to what you wish to work on.

CHAPTER 1

Introduction

1.1 What is Torque 3D?

Torque 3D was created by GarageGames to make the development of games easier, faster, and more affordable. It is a professional Software Development Kit ("SDK") that will save you the effort required to build a rendering system, high speed multiplayer networking, real time editors, a scripting system, and much more.

As part of Torque 3D, you receive full access to 100% of our engine source code. This means that you can add to, alter, or optimize any component of the engine down to the lowest level C++ rendering calls. That being said, you don't need to be an experienced C++ programmer to use Torque 3D. In fact, you do not need to know C++ at all. Using TorqueScript and the collection of tools that are included with Torque 3D, you can build complete games (of many different genres) without ever touching a single line of C++ code.

To understand the basics of how the engine is setup and the tools available, a short reference is included below. Further sections in the documentation explain these tools in more depth.

1.1.1 The Engine

The engine handles all of the elements of a game that run in real time on your computer. The Torque 3D engine is written entirely in C++ and is fully accessible to you as a developer. This means you have access to the inner workings of the code to customize it for your needs. The end result is that Torque 3D allows developers to add functionality, increase optimization, and learn how everything works. Alternatively, you can build a game from scratch to release without delving into the source code. The choice of how to develop your game is up to you.

For example, if you wish to add MYSQL database functionality or integrate the Havok SDK to enhance your game, those paths are open to you. Another benefit of source code access is the ability to read through the comments and data structures to gain a better understanding of how the entire system is set up.

Do not be intimidated. This documentation will show you how to create games without touching the source code at all. There is no need to start working with the engine's C++ code until you feel comfortable. In the meantime, you can get going with Torque 3D right away!

1.1.2 TorqueScript

Much of your game play logic, camera controls, and user interface will be written in TorqueScript. It is a powerful and flexible scripting language with syntax similar to C++. The key benefit of TorqueScript is that you do not need to be a code guru or know the nitty-gritty specifics of a particular language like C++. If you are already familiar with basic programming concepts, you will have a head start on building your own game.

Another benefit of using TorqueScript, as opposed to editing the engine's underlying C++ source code, is that you do not have to recompile your executable to see changes in your game. You simply create or modify a script, save, and then run the game from the Toolbox.

There are several TorqueScript articles for new developers that will help you learn the syntax, functionality, and how to use the language with the engine and editors.

1.1.3 Editors

Learning to work with Torque 3D editors is a large part of your initial experience. The key is to remember that the editors work in real-time and are WYSIWYG (What You See Is What You Get). When you use the editors to modify your level, you will see the changes immediately in the game.

World Editor - The World Editor is a tool that will help you assemble your game levels. With this tool, you will add and position terrain, game objects, models, environmental effects, lighting, and more.

GUI Editor - GUI stands for Graphical User Interface. Some examples of GUIs include: splash screens, your main menu, options dialogs and in game Heads Up Displays ("HUDs"). With the GUI Editor, you can design and create your menus, player inventory system, health bars, loading screens, and so on.

1.1.4 The Asset Pipeline

You would not have much of a game without models, textures, and other art assets. For Torque 3D, the preferred file format for 3D art assets is COLLADA.

From the COLLADA website: "COLLADA is a COLLAborative Design Activity for establishing an open standard digital asset schema for interactive 3D applications." In-other-words, it is a 3D model file format supported by most major art applications used to make content for games. You can create a model in 3D Studio Max, Maya, Blender, or any other 3D editor that supports the COLLADA format.

For those of you familiar with previous Torque engines, you can still import DTS (static models) and DSQ (animation data) files for your 3D objects. This includes static shapes, players, buildings, and props. If you already have a library of DTS and DSQs, feel free to use them in Torque 3D. From this point on, we recommend you transition to the COLLADA open standard for new art assets.

1.2 Seting up the SDK

TBD

1.3 The Project Manager

TBD

1.4 A Tour of the SDK

TBD

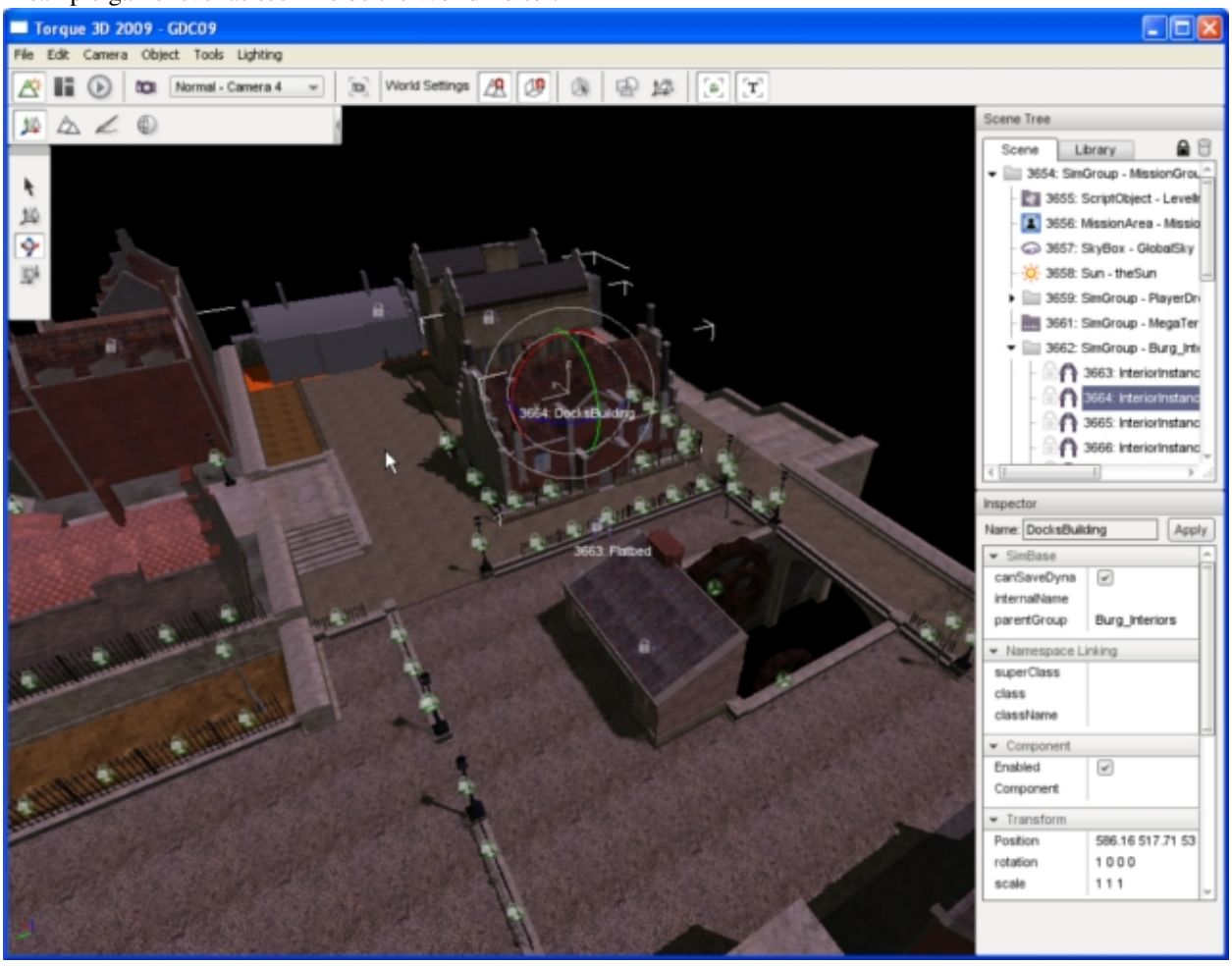
CHAPTER 2

World Editor

2.1 Overview of World Editor

The World Editor is used to build and edit game levels. This includes adding and modifying terrains, buildings, foliage, cloud layers, vehicles, environmental effects, lighting effects, and much more. Aside from the Toolbox, the World Editor is the first (and most important) tool a new user should learn.

A sample game level as seen inside the World Editor:



The World Editor is not a tool for creating game objects. Objects must be created using applications appropriate for

the object type (i.e., 3DS Max to create a 3D model). However, once an object is loaded it can be modified by the World Editor in a variety of ways. The simplest modification would be a change in scale (size), but more complex modifications are also possible. For example, the Torque Material Editor can be used to alter (or completely replace) textures on a 3D object or add shader effects.

A typical World Editor workflow might go as follows (in very simplified terms):

1. Create a 3D model in an application like 3DS Max, Maya, or Blender.
2. Save that model to a sub folder inside your game/art directory.
3. Launch World Editor (which will automatically find that model if step 2 was done correctly).
4. Add the model to your level; position, scale, rotate, and adjust its materials as desired.
5. Test your changes in-game with the push of a single button.
6. Return to the World Editor and continue to tweak your level.

Of course, there is a lot more to the World Editor than positioning 3D models. You will also be working with 2D assets like grey scale height-maps to create terrains, as well as specialized tools for creating rivers, forests, and roads.

Finally, it is worth noting that Torque 3D includes numerous art assets for you to play with... so you can skip steps 1 and 2 above and start building game levels right away!

2.1.1 Using the World Editor Documentation

The World Editor documentation follows a logical progression. Those who wish may work through it in a methodical way. Others may choose to skip difficult sections and jump directly to the tutorials at the end or to focus on only the features of interest.

Everyone learns differently, but we've found that a good way for new users to get started quickly is to follow these three steps:

1. Continue reading this document ("Overview") in its entirety. It covers: how to launch the World Editor, how to look and move around in a game level, and it offers a few important tips for new (and experienced) users.
2. With the World Editor open, quickly skim the next document, "Interface". You should only spend five-to-ten minutes getting an initial feel for the basic layout of the interface. Do not try to learn any features in detail.
3. Try to add moving clouds to your level by following the Basic Cloud Layer instructions. Whether you are successful or not, spend no more than five minutes on this task.

Do not be concerned if you have trouble completing Step 3. Its purpose is to give you a specific task that requires direct interaction with the interface. That small exposure to the interface will go a long way towards making the remainder of the documentation more meaningful and easy to follow.

Once you've completed the three steps above, how you proceed is up to you. For those who prefer to jump around, we recommend you start by carefully reviewing the Interface document.

2.1.2 How to Launch the World Editor

While your game is running, you can open or close the World Editor at any time using hot key combinations:

- **On Windows and Linux,** to open or close the World Editor, press the `F11` key.
- **On Mac OS X,** to open or close the World Editor, press `CMD+FN+F11`.

Note: When you first launch the World Editor, it is likely you will do so from the Toolbox. However, after you have modified your level, if you decide to test it out by clicking the Play Game button (as described in the "Interface"

document), you will need to use the F11 hotkey to get back to the World Editor. Otherwise, you would be forced to quit your game and relaunch the World Editor from the Toolbox.

2.1.3 Looking and Moving Around

While working in the World Editor, you will need to move and look around to inspect your level.

- **Forward/Back/Left/Right** movement is controlled by the corresponding arrow keys on your keyboard (the WASD keys can also be used). If you have a mouse-wheel, it can be used to move forward or backward.
- **Look Left/Right/Up/Down** by holding the right mouse button down while moving the mouse.
- **Pan Left/Right/Up/Down** by holding down the middle mouse button (Mouse 3) while moving the mouse. On most mice with a scroll wheel, this is achieved by depressing (not scrolling) the mouse wheel.

Note: There are a number of Camera options, discussed further in the Interface document, which in some cases may alter the behavior of these controls in minor ways.

When play testing your game outside of the World Editor, default control is typical of most First Person Shooters and can be remapped by pressing Ctrl-O (Windows) to bring up an options dialog. A few important controls are listed below:

- Forward/Back/Left/Right movement is controlled by the corresponding arrow keys on your keyboard (the WASD key can also be used).
- Look around by moving the mouse.
- Fire/Alt Fire are triggered by the left and right mouse buttons.
- Jump is activated by the Space Bar.
- First/Third person view is toggled by pressing TAB.
- Change weapons by scrolling the mouse wheel (or press Q key).
- Exit vehicles by pressing Control-F.
- Return to World Editor by using the F11 hotkey (as discussed above).

2.1.4 Tips

The following is a general list of knowledge you should keep in mind while editing a level in your game:

- **Try to design your levels outside of the editor first.** Sometimes it is helpful to have a simple verbal or visual design ready before you actually start editing. Even if it is a simple blueprint on a napkin, a level editor/artist with a reference to work from will cover ground much more quickly.
- **Prioritize your object placement.** It makes sense to polish certain aspects of a level before others. For example, try to finish your Sky, Sun, and Terrain before you move on to adding rivers, foliage, and other objects. Performing major adjustments to a terrain with hundreds of objects already placed could be tedious and counterproductive.
- **Play your level regularly.** After you reach a major milestone, try actually doing the things in your level as a player would. There is a big difference between the experience of a player in a game and that of a designer with a free-floating camera in the World Editor.

- **Do not forget to optimize.** Some specific World Editor objects are more appropriate than others. Use Ground Cover instead of a 3D model with lots of grass or trees attached. As much as possible, use the Sun rather than numerous point lights to handle ambient lighting. There are other such optimizations which will become apparent towards the end of development.
- **SAVE AND SAVE OFTEN.** This cannot be stressed enough. Computers crash, power goes out, cats jump on keyboards, and in rare circumstances you may encounter a yet undiscovered issue which causes data corruption. Any number of accidents can result in hours of work being lost. We recommend you save as often as you can.

2.2 World Editor Interface

The default World Editor view consists of five main sections:

File Menu Found at the very top of the World Editor window, you will find menus that controls the global functionality of the editor, such as opening/saving levels, toggling camera modes, opening settings dialogs, and so on.

Tools Bar Located just below the File Menu, this bar contains shortcuts to all of the tools, their settings, and some options found in the File Menu.

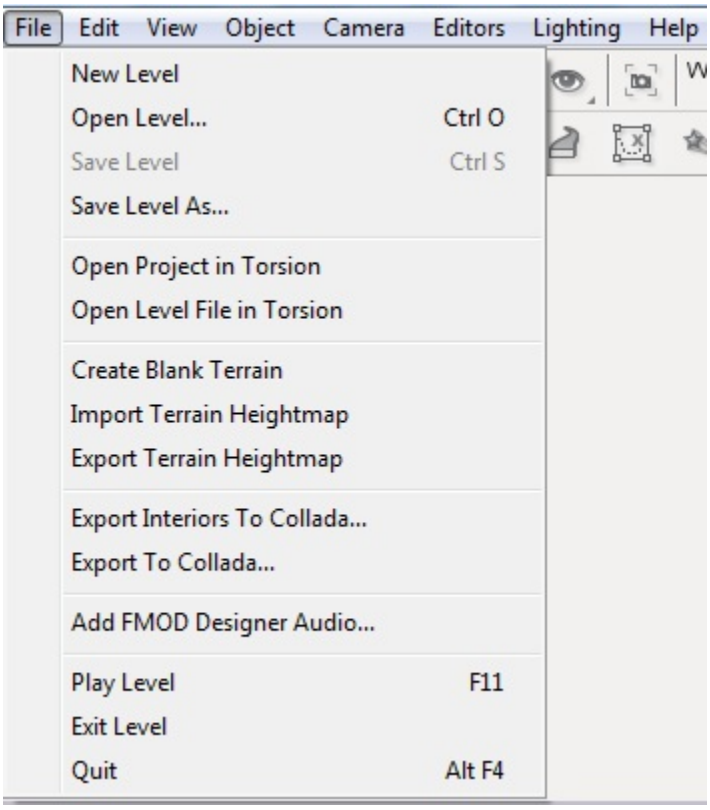
Tool Palette The Tool Palette changes based on what Tool you are currently using. For example, when using the Object Editor you will have icons for moving and rotating an object, wheras the Terrain will have icons for moving and rotating an object, whereas the Terrain Editor display icons for elevation tools.

Scene Tree Panel While using the Object Editor, one of the floating panels available to you is the Scene Tree. It is composed of two tabs: Scene and Library. The Scene tab contains a list of objects currently in your level. The Library tab is what you will use to add new objects to your level after which they will appear in the Scene tab.

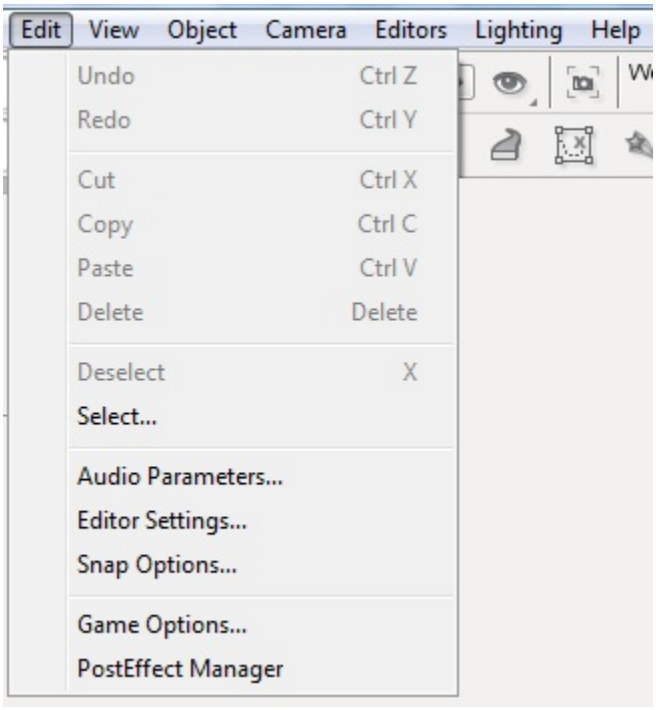
Inspector Panel While using the Object Editor, a selected object's properties will be shown in this panel. Most of your object editing will be performed here.

2.2.1 File Menu

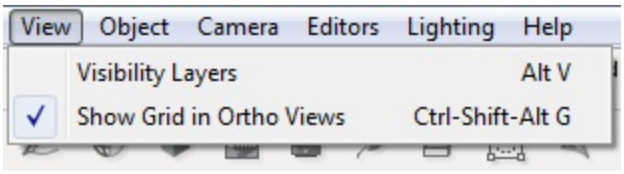
File Menu allows you to: Create, save, open, and close levels; Open, import, and export level data to/from other tools; Run your level to test it and exit the World Editor.



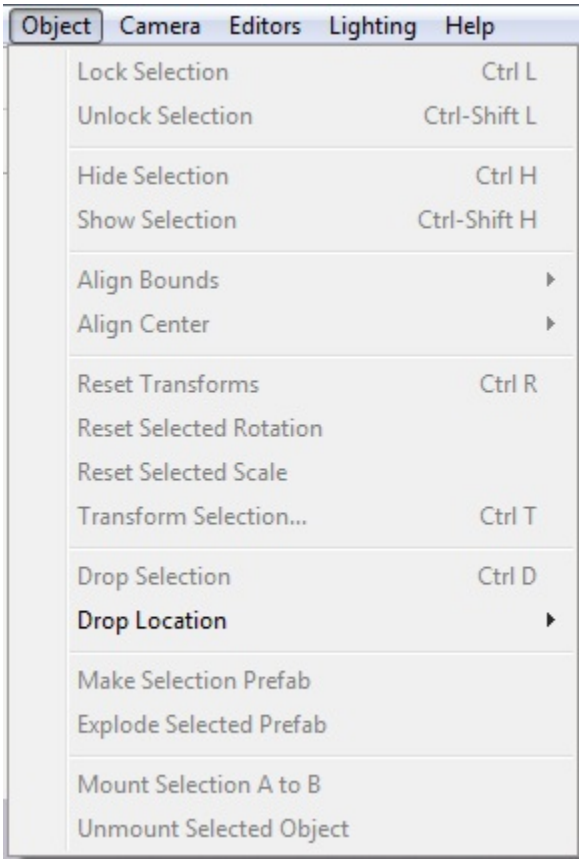
The Edit Menu allows you to: Control editor actions such as undo and redo; Cut, copy, paste, and delete objects you have selected; Select objects using a name pattern or by type filtering; Access dialogs to control various World Editor settings.



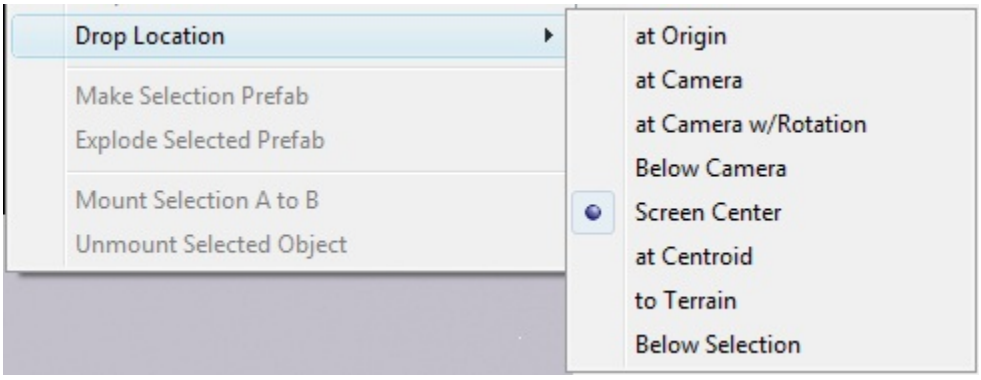
The View Menu: Opens the Visibility Layers dialog which toggles debug rendering modes; Toggle the visibility of other aspects of the editor.



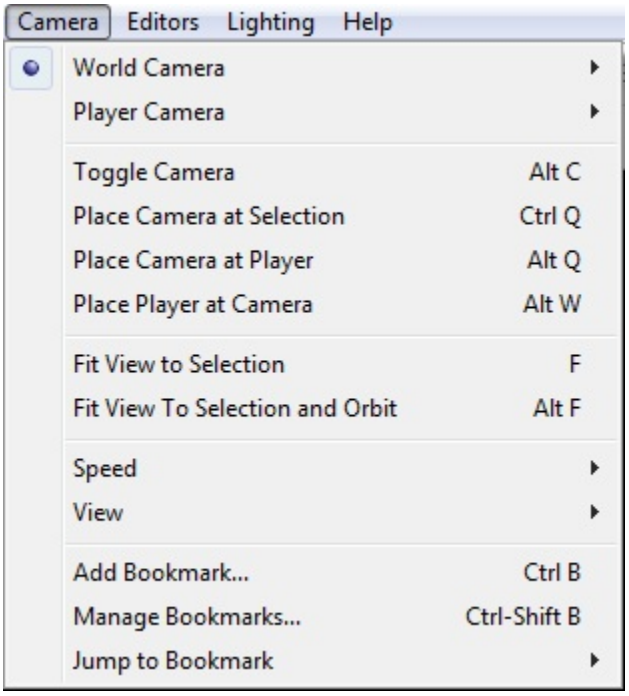
The Object Menu allows you to: Manipulate a selected object's settings by locking/unlocking it, hiding/showing the object, resetting its transforms, and so on.



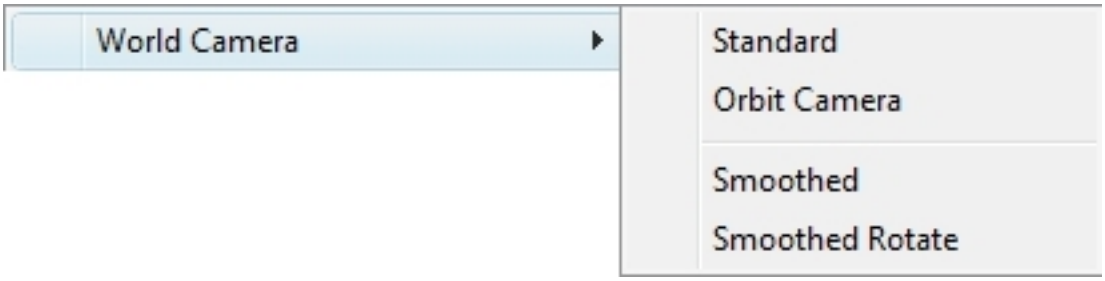
The Drop Location sub-menu selection informs the World Editor where it should place newly created objects.



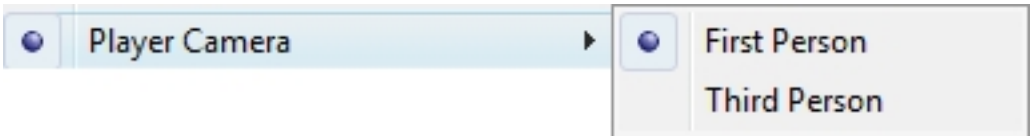
The Camera Menu allows you to choose your camera type, adjust its speed and motion, and drop it at certain locations.



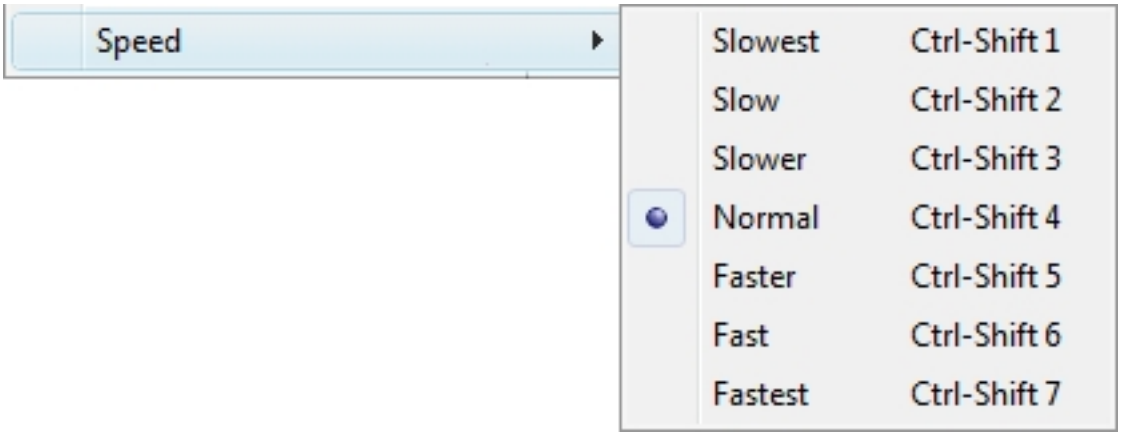
The World Camera sub-menu allows you to change the way the camera moves.



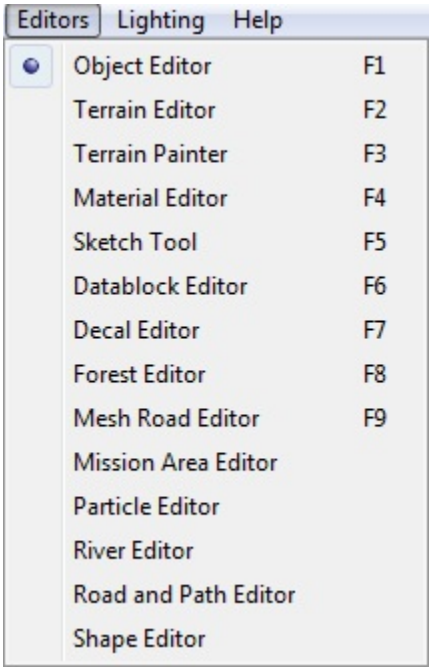
The Player Camera sub-menu allows you to switch between perspectives while moving around as a player.



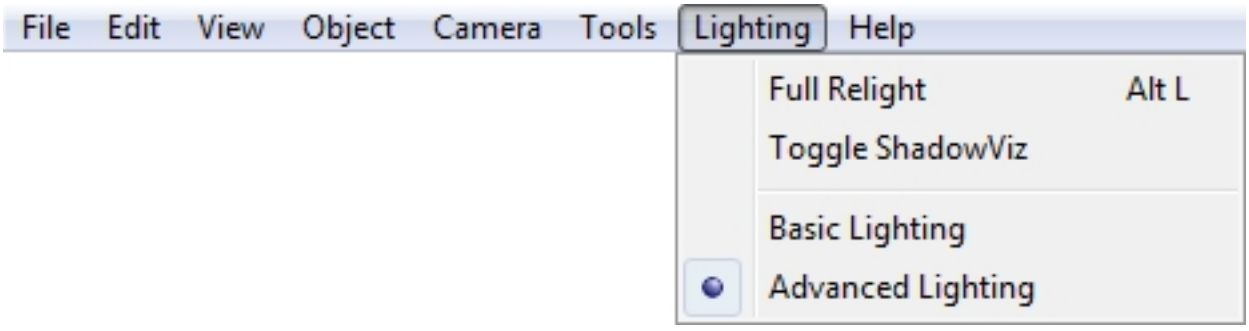
The Camera Speed sub-menu allows you to adjust how fast the camera moves.



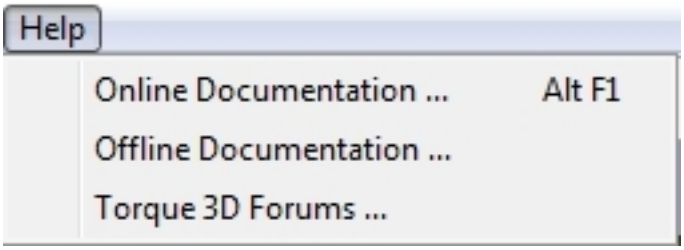
The Editors Menu allows you to select which set of editing tools is currently active in the World Editor.



The Lighting Menu allows you to switch between Advanced and Basic lighting modes, as well as perform level relights.

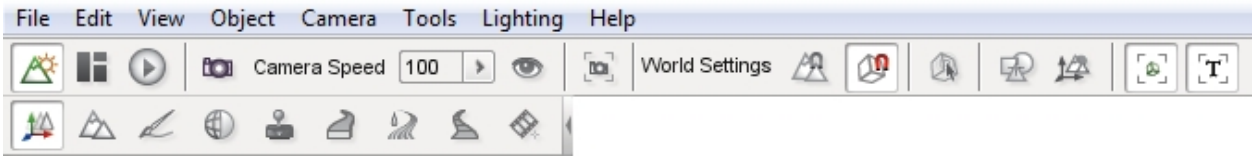


Contains shortcuts to documentation and forums for Torque 3D.



2.2.2 Tools Bar

The Tools Bar is the best way to switch between tools. It is made of two components: Tool Settings (top bar) and Tools Selector (bottom bar).



Tool Settings is made of up three sub-sections: the editor selector, camera settings, and Object Editor. The editor selector and camera setting will always be displayed. The Object Editor will display available settings for the currently selected tool. The Tools Selector will always display the same shortcuts for selecting tools.

This section focuses on the elements of Tool Settings.

The first three icons switch between the editor's operating modes. Each icon represents a different editing mode and only one mode can be active at any time. There are three modes: World Editor, GUI Editor, and Game Mode. The World Editor is represented by the mountain icon. The GUI Editor is represented by the boxes icon. The Game Mode is represented by the arrow icon.

World Editor mode provides tools for manipulating the "world" of your game including terrain, creatures, and so on.

GUI Editor mode provides tools for manipulating the Graphical User Interface (GUI) of your game such as health meters, cursors, and so on.

Play Game Mode runs your game and lets you play through it.

Note: When you use this icon to play your game the World Editor actually closes completely. To return to the World Editor you must press F11 or exit the game and relaunch the World Editor from the Toolbox.

Next to the editor selector, you will find the camera and visibility settings.

The camera icon will let you choose your camera type. The drop-down menu next to it will let you switch between camera speeds. The eye icon is the visualization settings which toggle debug rendering modes for various graphical modules, such as normal mapping, wireframe, specular shading, etc. The icon that looks like a camera in a box will move your camera to whatever object you have selected, filling up your view with its boundaries.

The World Settings make up the rest of this bar when using the tools. The first icon lets you determine your snapping options (snapping to terrain, a bounding box of an object, which axis, etc.). The next icon toggles snapping to a grid. The magnet icon determines soft snapping to other objects. The numeric indicator determines the distance of the snap option.

The box icon with an arrow is a selection tool that allows you to select an object according to its bounding box. This makes selecting small, detailed objects much easier. The next icon that looks like a bullseye will change the selection target from the object center to the bounding box center. The small icon with arrows and mountains will change the object transform and the world transform.

The next two icons show descriptors in your scene. The first icon that looks like a box in a square will display object icons for the various objects in your scene. The second icon will show text descriptors for the objects in your scene.

The last two icons in the bar are prefab icons. The first icon lets you group selected items into a "prefab" (or prefabricated collection) of objects. The second icon will ungroup your prefab items.

2.2.3 Tool Selector and Palette

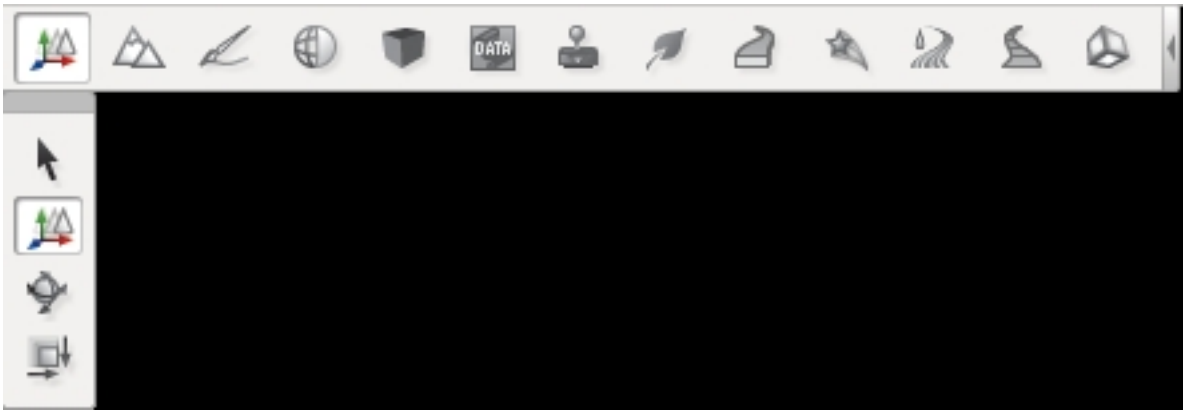


Fig. 2.1: Object Editor

2.2.4 Scene Tree

The Scene Tree panel is available while using the Object Editor tool. It is composed of two tabs: Scene and Library. The Scene tab contains a list of objects currently in your level. You can select specific objects to modify them.

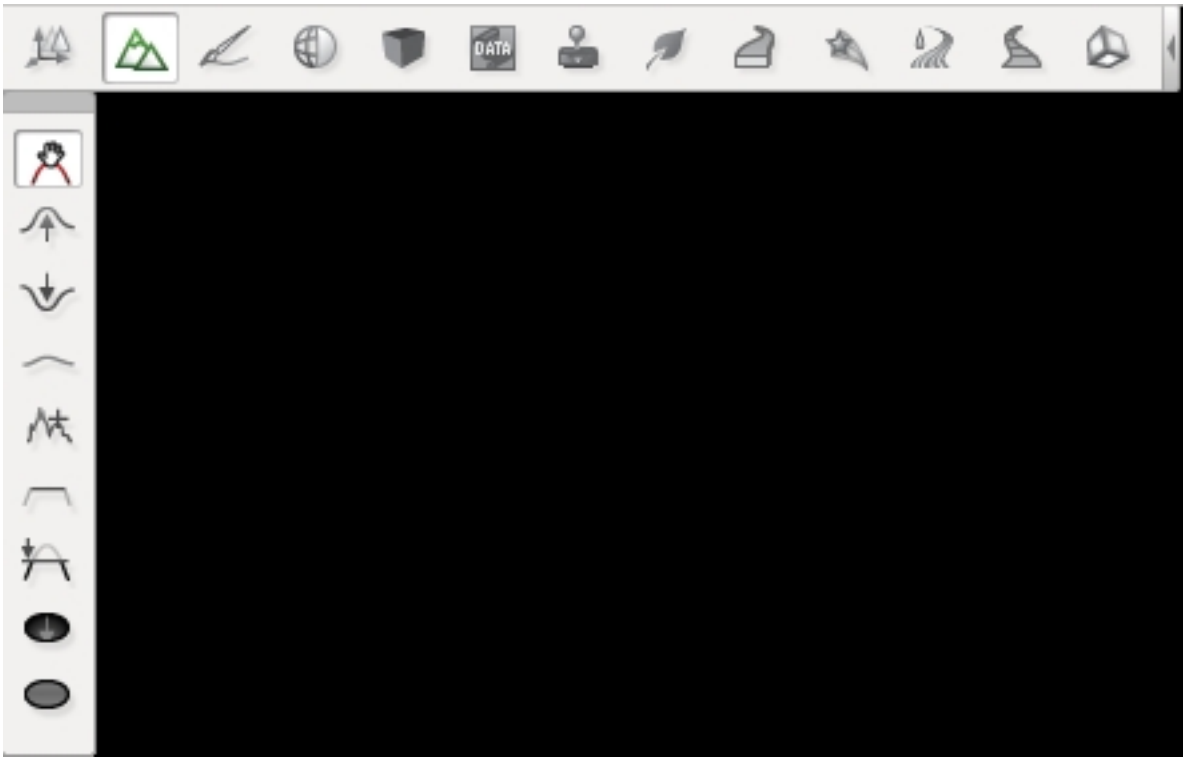


Fig. 2.2: Terrain Editor

Fig. 2.3: Terrain Painter

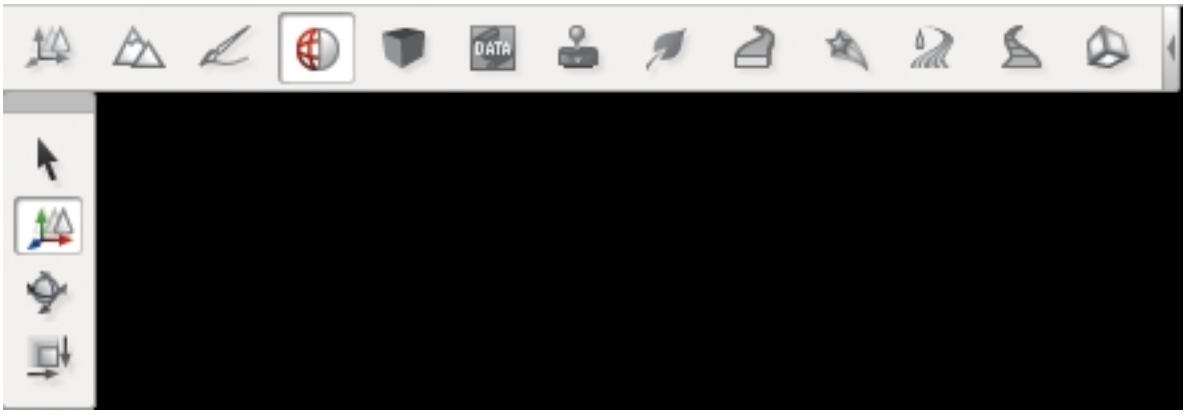


Fig. 2.4: Material Editor

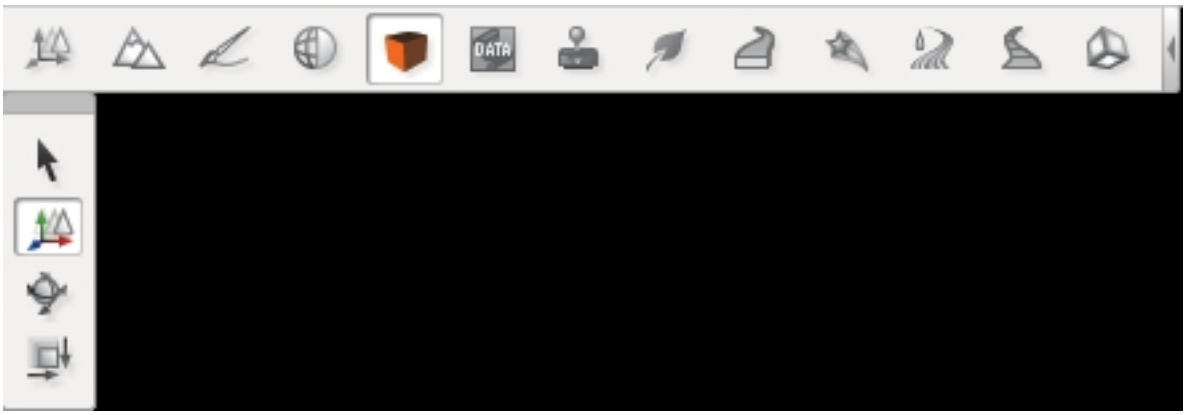


Fig. 2.5: Sketch Tool

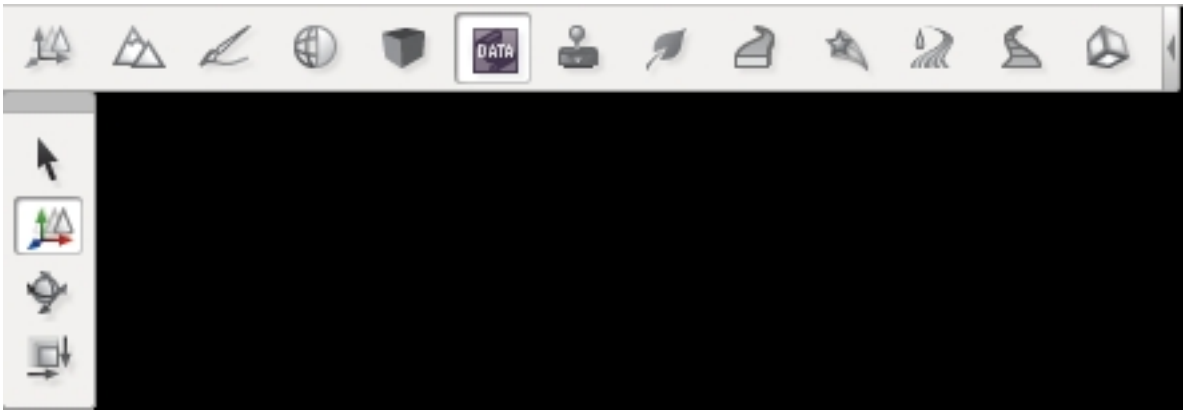


Fig. 2.6: Datablock Editor

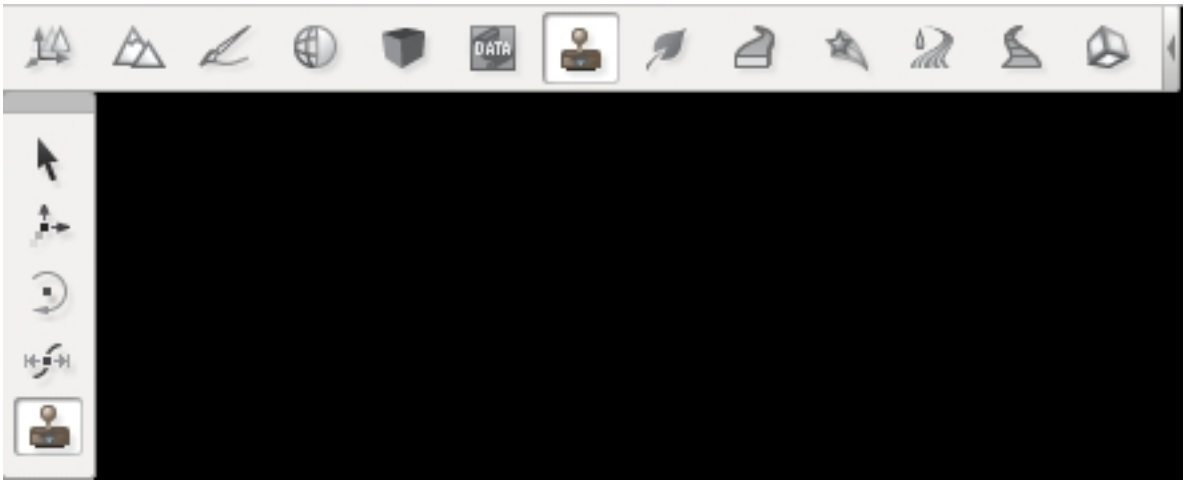


Fig. 2.7: Decal Editor

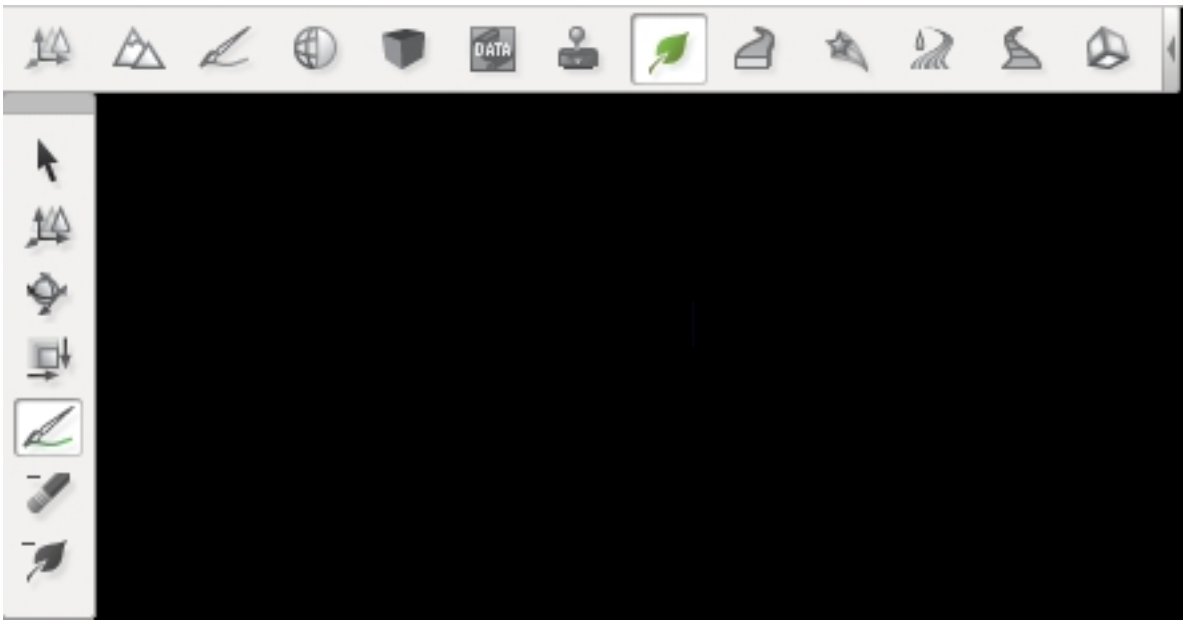


Fig. 2.8: Forest Editor

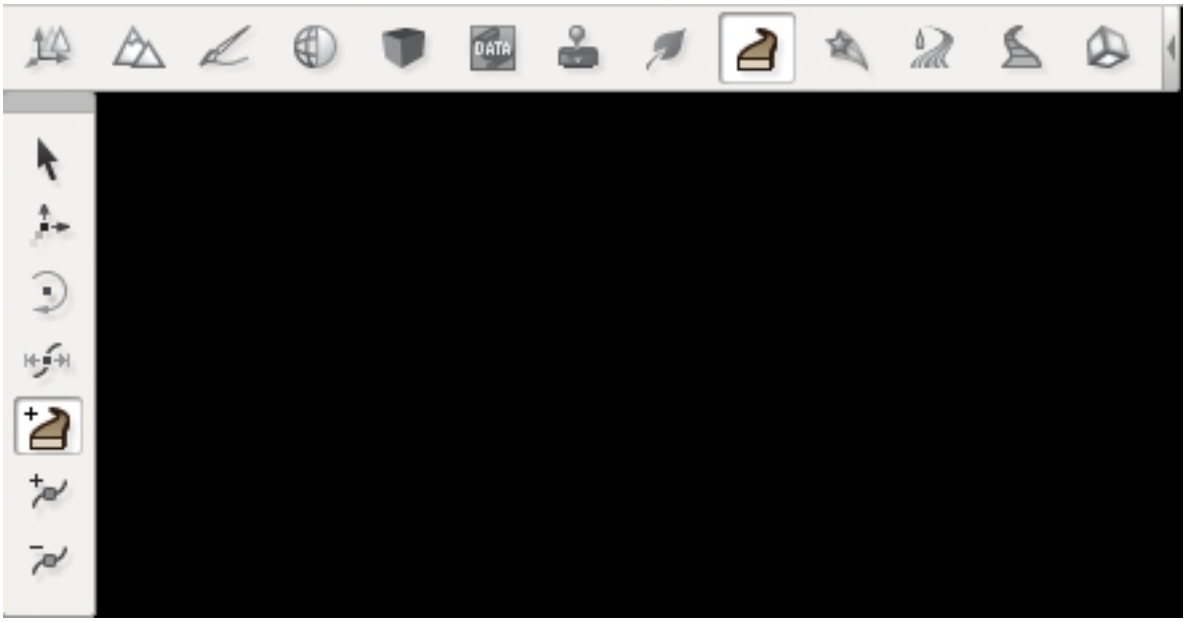


Fig. 2.9: Mesh Road Tool

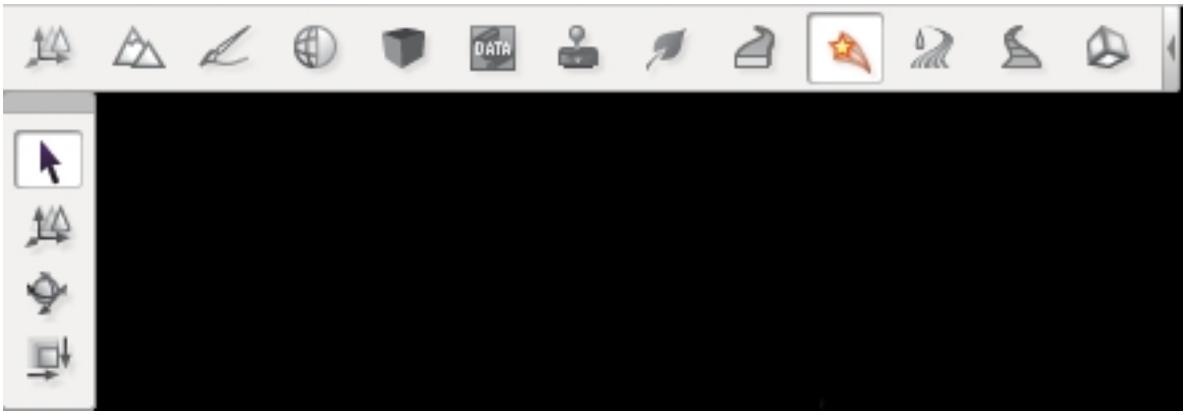


Fig. 2.10: Particle Editor

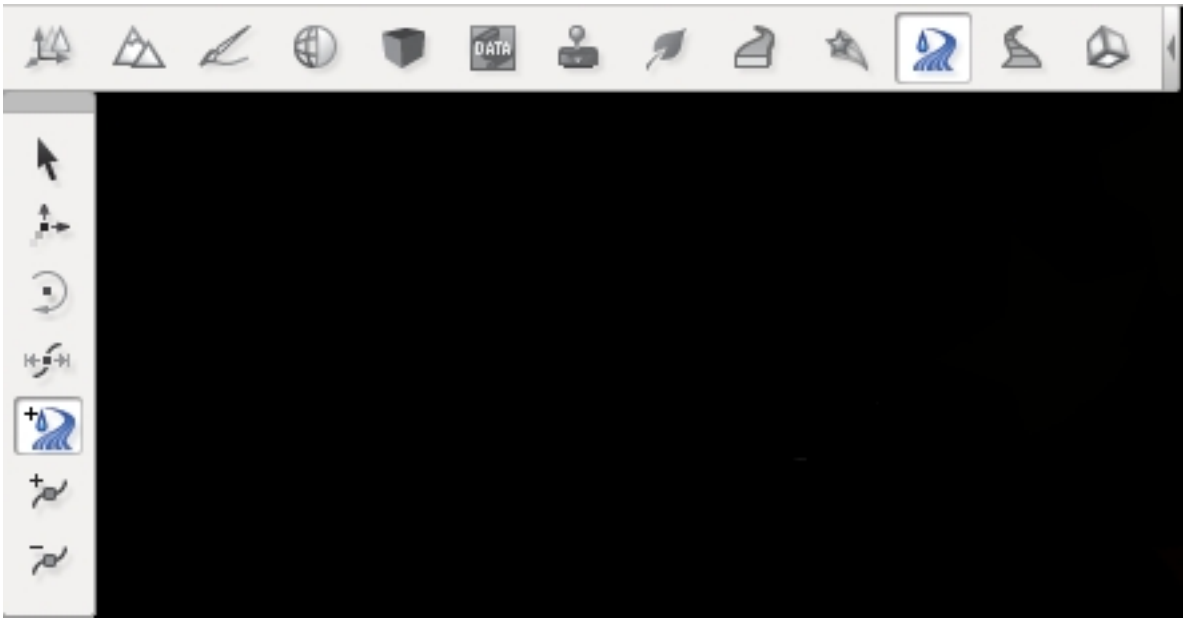


Fig. 2.11: River Tool

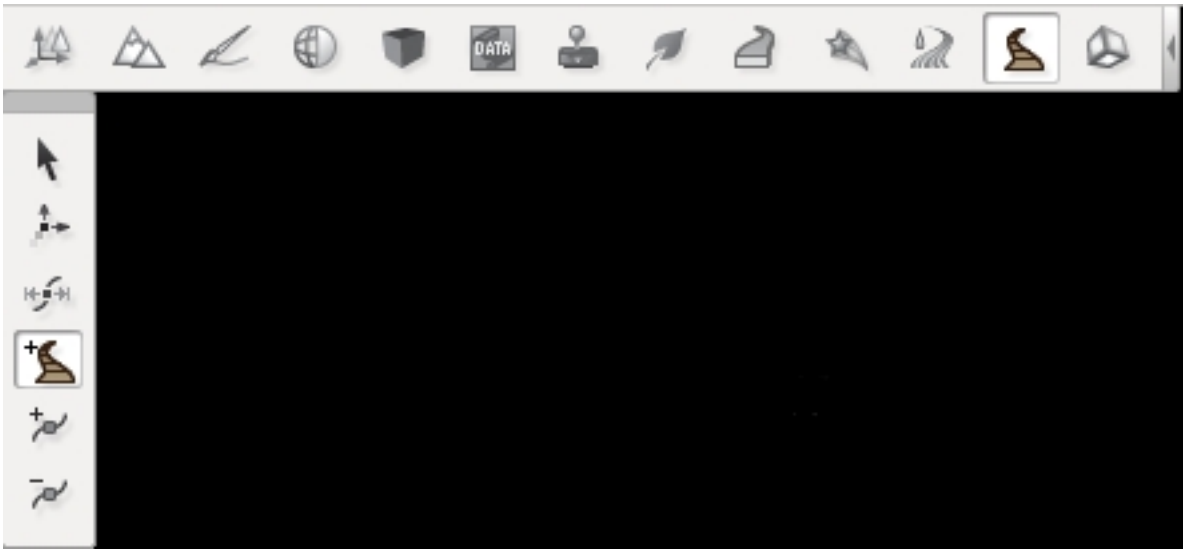


Fig. 2.12: Decal Road Tool

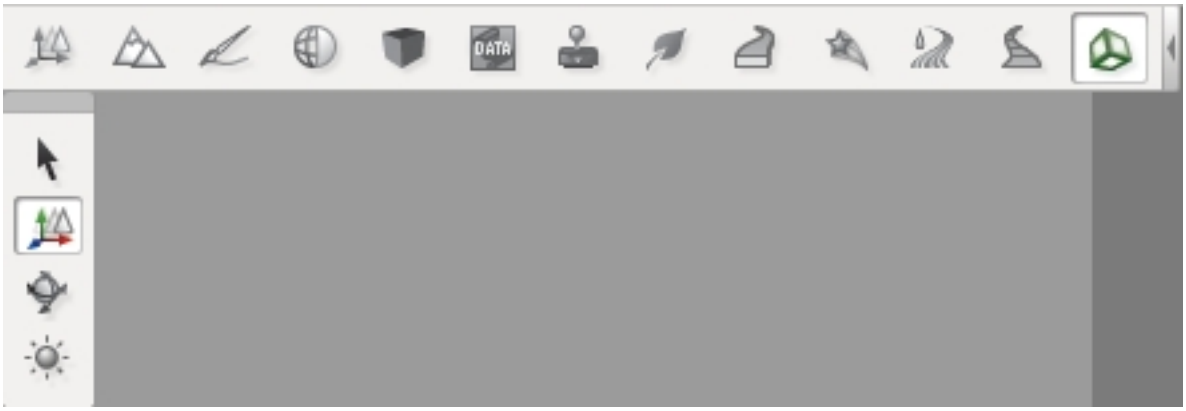
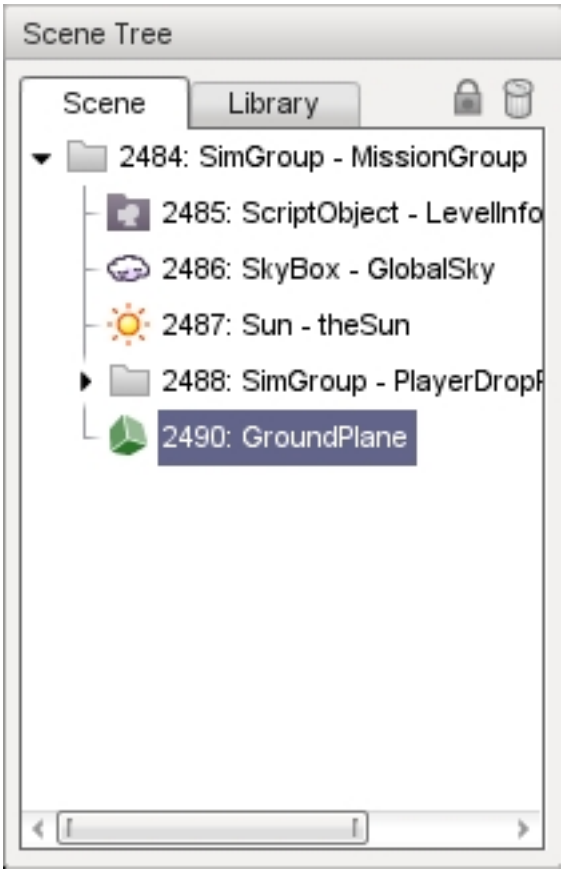


Fig. 2.13: Shape Editor

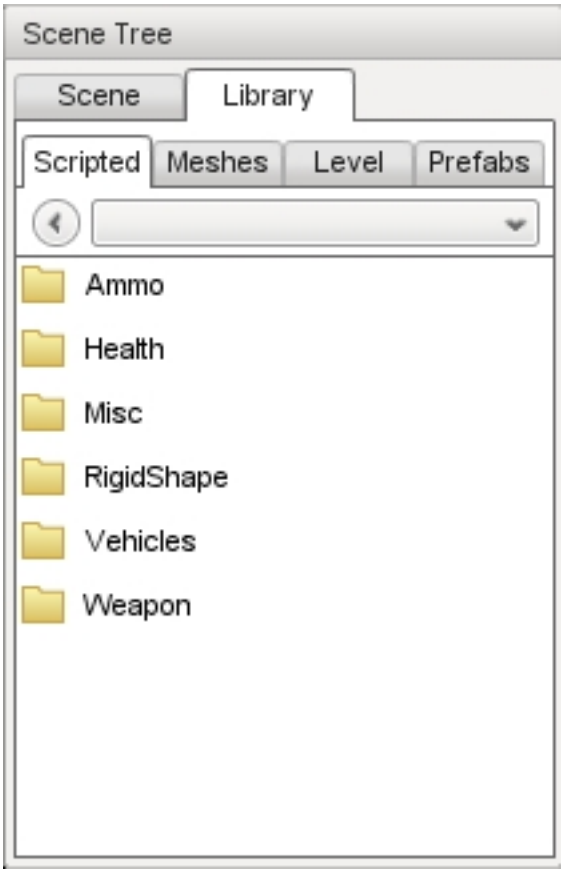


Each object in the tree has an icon, unique ID, an object type, and a name. Whenever you click on an object in the tree, it is selected in the level and vice versa. Most of your objects can stand alone in the tree, but you can also use a SimGroup object to organize related entries.

At first glance, a SimGroup looks like a folder and acts much like one to help organize your tree. It does not physically exist in your level, but you can reference it by name or ID from script or the engine. This is handy for grouping several game objects you might need to iterate through and invoke an action on. Even if you do not use that feature, it is still a good idea to group similar objects under a SimGroup to help organize and better navigate your trees as some levels can grow to a large number of objects.

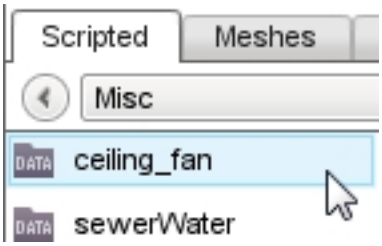
2.2.5 Library Tab

The Library tab is what you will use to add objects to your level. Once an object has been added to your level, it will appear in the Scene tab (described above). There are four sub-categories on the Library tab, which are separated as sub-tabs: Scripted, Meshes, Level, and Prefabs. Each category contains objects that serve very specific purposes.



Scripted Tab

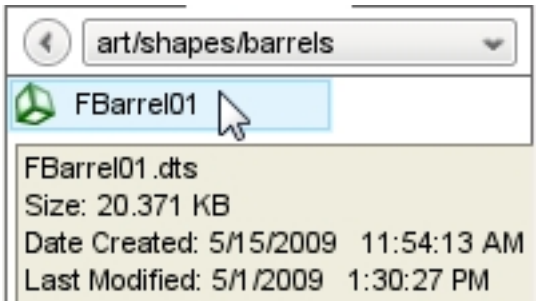
The first tab, Scripted, is automatically populated with game objects that have been created via script. For example, let's say you have a ceiling fan object with an associated script which controls how and when the fan blades rotate. It would appear in the Scripted tab as follows:



A discussion of scripting and how to associate scripts with an object is beyond the scope of this document. See the TorqueScript Tutorial for more information.

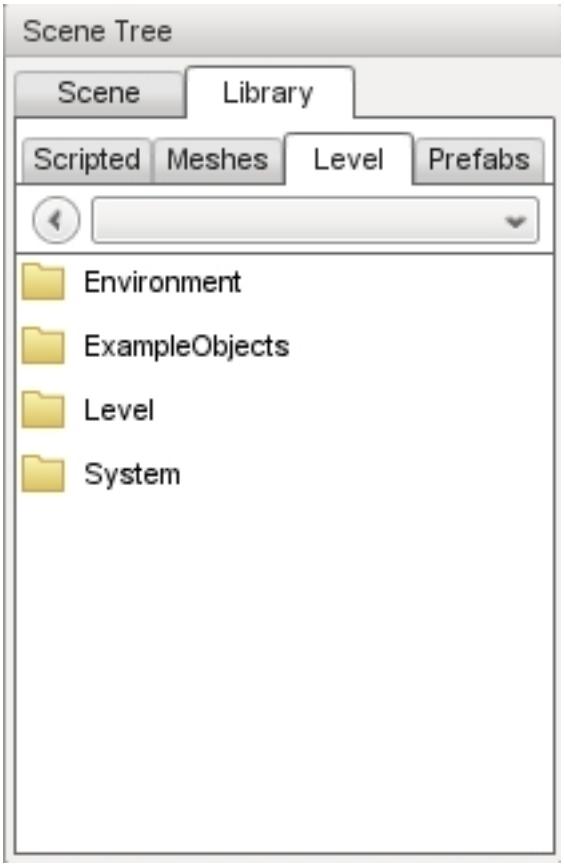
Meshes Tab

When you simply wish to add a 3D art asset, you will use the Meshes Tab. You can browse the various folders containing assets in your project's "art" directory. From here you can add DTS, COLLADA, and DIF files.

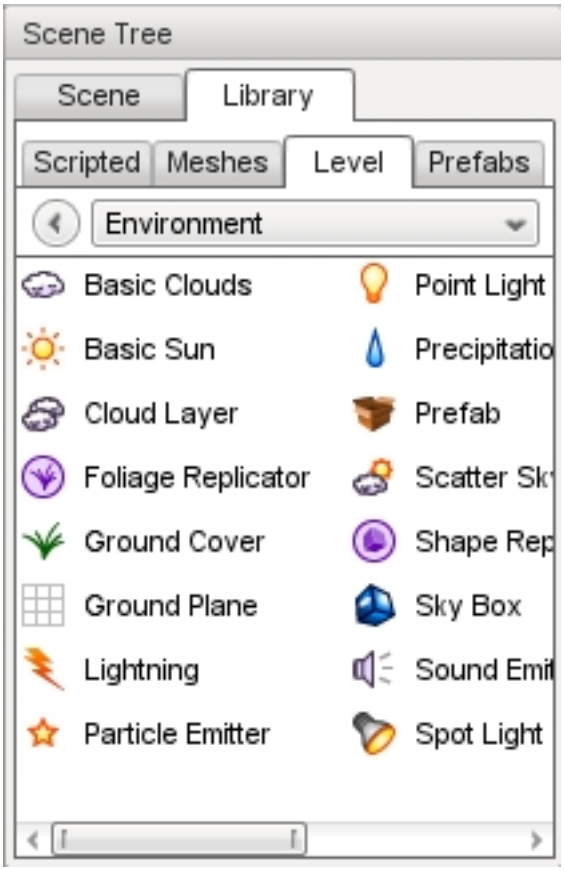


Level Tab

The Level Tab lists all the Torque 3D objects that can be used to populate your level. Objects are further divided into category folders. To view what is in a folder, double click it. To leave a folder and view the folder list, click the left pointing arrow icon. To move directly to another folder, select it from the drop down list.



Each sub-category contains objects with similar themes:



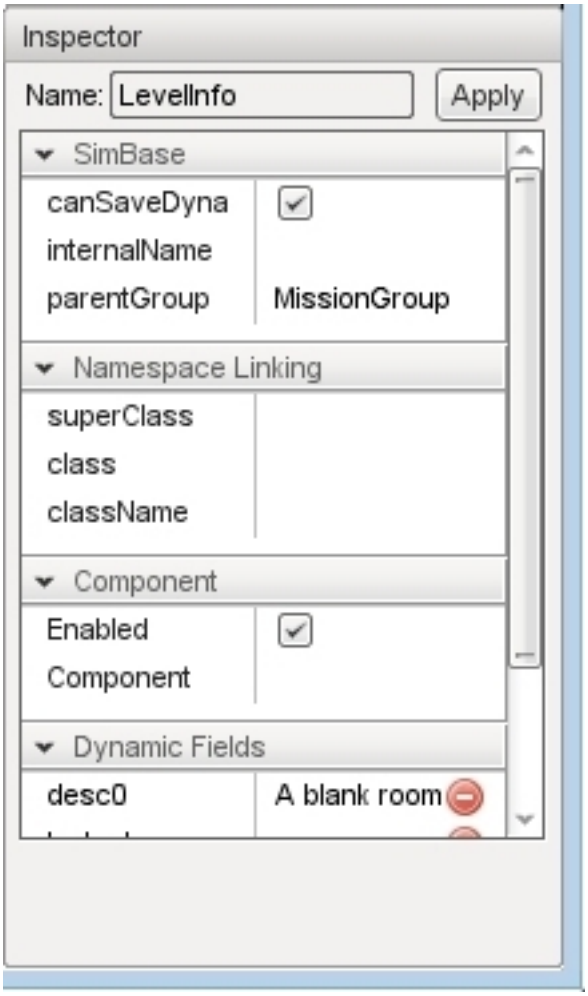
- The Environment sub-category contains most of the objects you will add to your level, such as Terrain, Sun, Clouds, Waterblocks, and similar objects.
- The ExampleObjects sub-category contains example rendering classes created in C++.
- The Level sub-category contains objects that manage Time of Day, level boundaries, and similar objects.
- The System sub-category contains engine-level objects such as SimGroups.

Prefabs Tab

The prefab system allows you to group multiple objects together and combine them into a single file. This new object can then be repeatedly placed into your level as a whole, making it easier for you to add complex groups of objects with only a few mouse clicks. When you create a prefab from multiple selections, you will save it to a *.prefab file using the group prefab icon. The World Editor will automatically load these files in the Prefabs tab.

2.2.6 Inspector

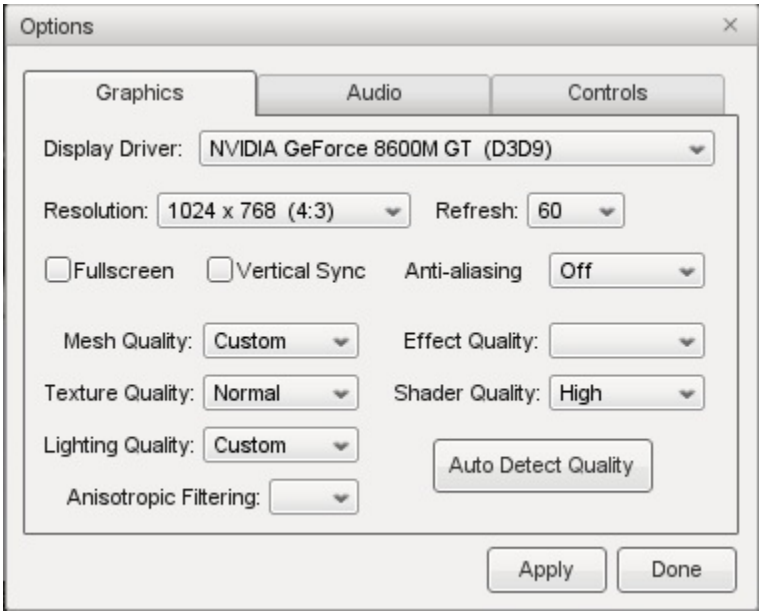
Whenever you add an object to a level, you will most likely start modifying them immediately. You can use the Inspector Panel to change the properties of an object



While there are a few shared property sections, most object types will have a unique set of properties. Editing is as simple as selecting an object in the level, locating a field that you want to change, such as "className" or "media", then either editing the existing value or entering a value if no default value is given. There are different types of values such as strings, numbers, check boxes, vectors, and even values that require the use of a file browser or color picker.

2.2.7 Options

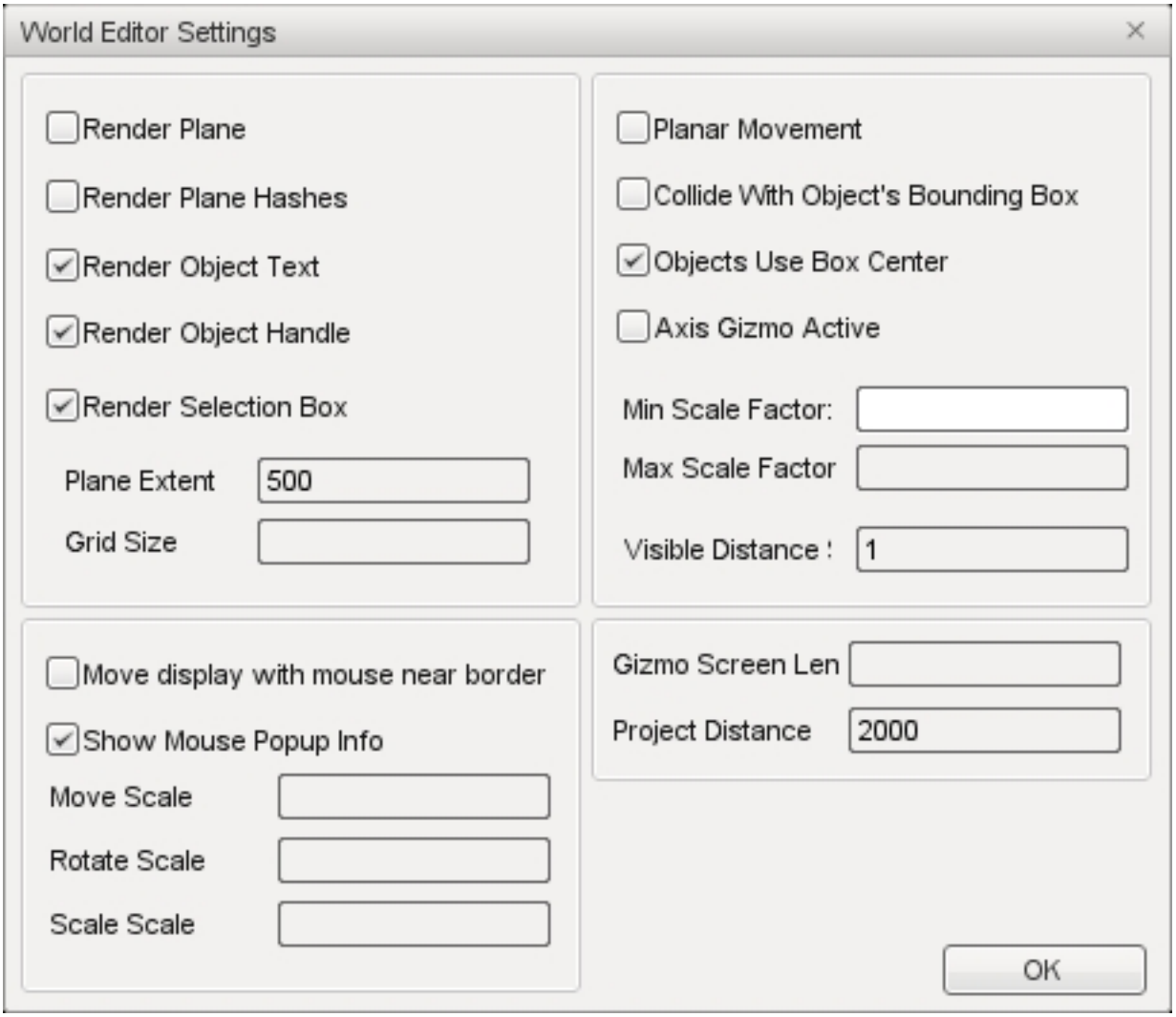
The Options dialog is used to change your current session's audio and video properties as well as mouse and keyboard control bindings. The Options dialog is accessed from the main menu by selecting Edit > Game Options...



You will use the Graphics tab to adjust your game resolution, screen mode, detail levels, and so on. The Audio tab allows you to adjust your current game's volume, both globally and channel specific.

2.2.8 World Editor Settings

The World Editor Setting dialog is important to editing.



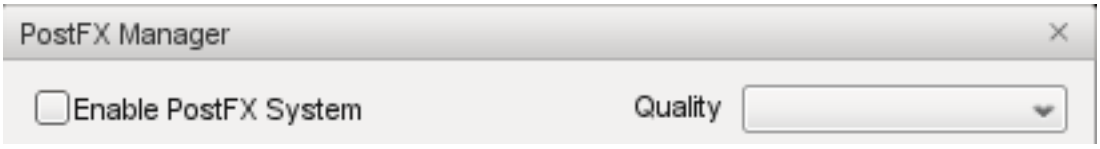
Through this dialog, you can change various aspects of how your tools render and function. The top left section will control what is rendered on your object, such as its text (name/ID), handle, and selection box. You can also adjust the rendering of the editing plane in relation to the object.

The bottom left section contains the control settings for your manipulators (Translate, Rotate, and Scale tools). You can tweak the sensitivity of the manipulators for more precise or dramatic modifications.

Both sections on the right have settings that adjust visibility and selection methods for your gizmos. The Visible Distance is also an important value, as that adjusts how far into the distance you can see while editing the level.

2.2.9 PostFX Manager

The PostFX Manager GUI allows level editors to control various post-processing effects. Select the *Enable PostFX* checkbox to toggle PostFX on and off.

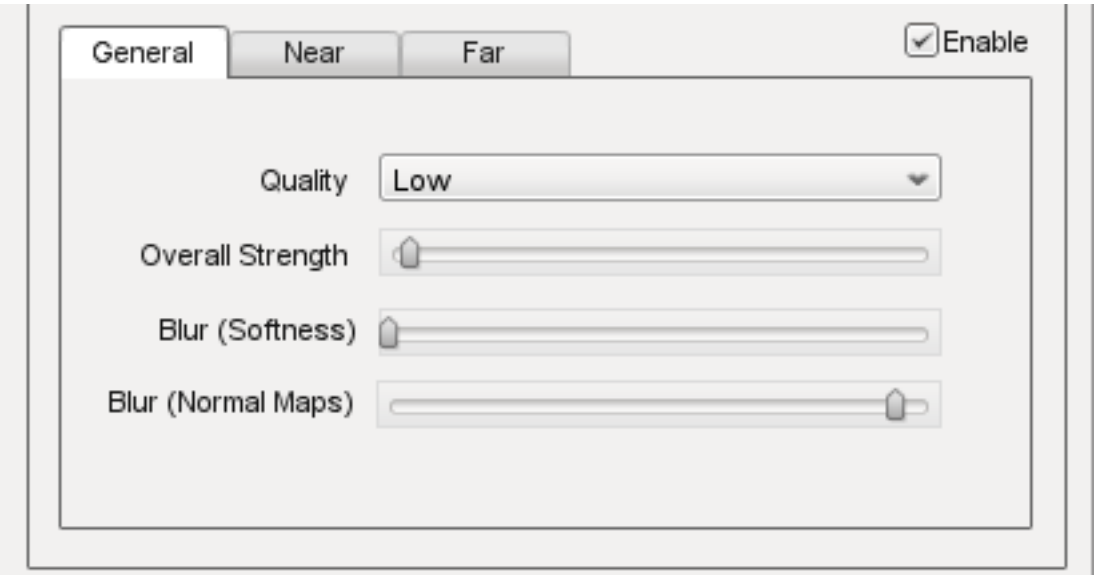


Use the effect tabs to access the effect settings.

PostFX settings can be saved to file and and loaded automatically with the level. To achieve this, simply save the settings with the same name as the level file. For example, for Burg.mis, save the PostFX settings in a file called Burg.postfxpreset.cs in the same folder as the level file.

SSAO

Screen space ambient occlusion (SSAO) is an approximation of true Ambient Occlusion. Enabling the effect will darken creases and surfaces that are close together. Outdoor areas with brighter ambient light will show the effect better.

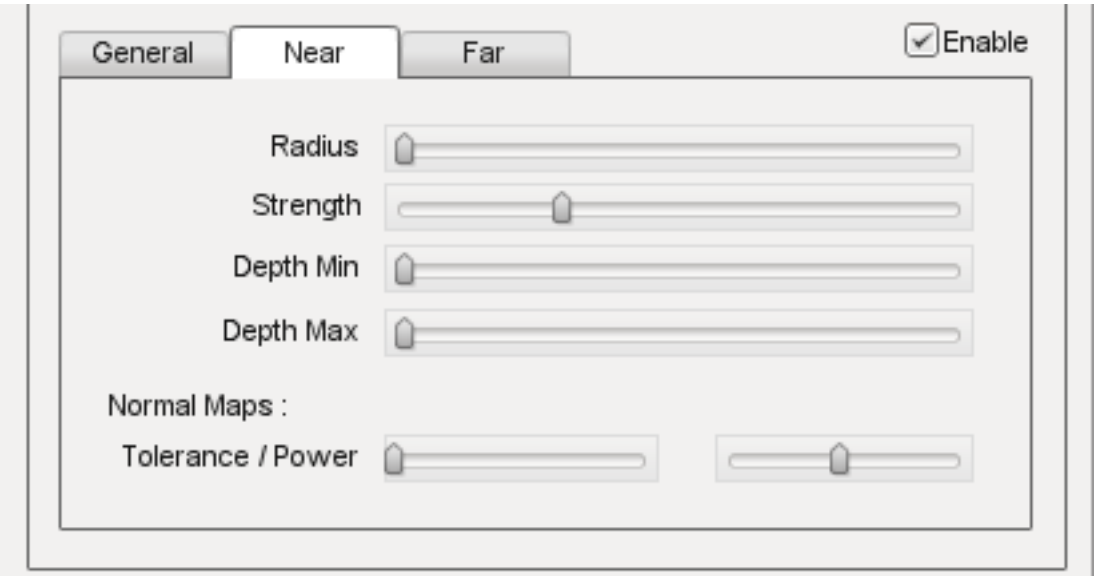


Quality Controls the number of ambient occlusion samples taken; higher quality is more expensive to compute.

Overall Strength Controls the overall intensity/darkness of the effect (applied on top of near/far strength).

Blur (Softness) Blur depth tolerance.

Blur (Normal Maps) Blur normal tolerance.



SSAO parameters for pixels near to the camera (small depth values).

Radius Occlusion radius.

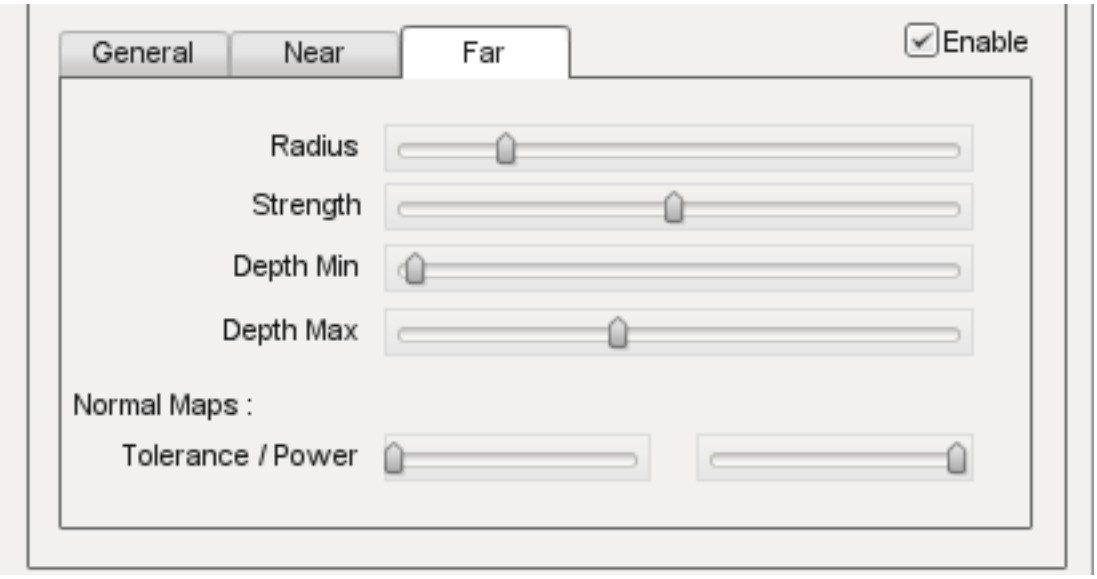
Strength Occlusion intensity/darkness.

Depth min Minimum screen depth at which to apply effect.

Depth max Maximum screen depth at which to apply effect.

Tolerance *Unused*

Power *Unused*



SSAO parameters for pixels far away from the camera (large depth values).

Radius Occlusion radius.

Strength Occlusion intensity/darkness.

Depth min Minimum screen depth at which to apply effect.

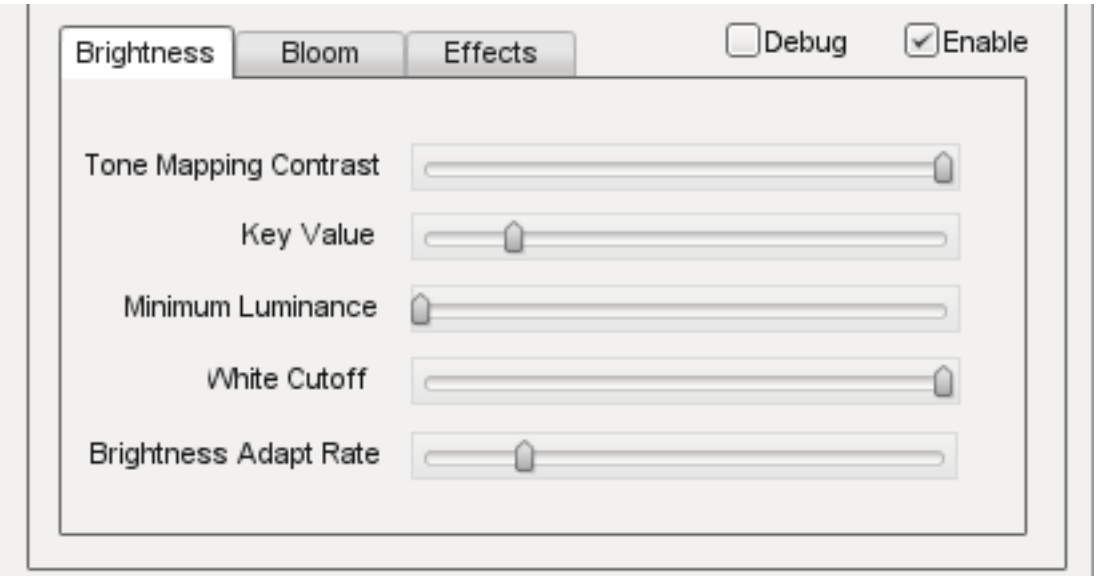
Depth max Maximum screen depth at which to apply effect.

Tolerance *Unused*

Power *Unused*

HDR

Control several High Dynamic Range (HDR) effects including Bloom and Tone mapping.



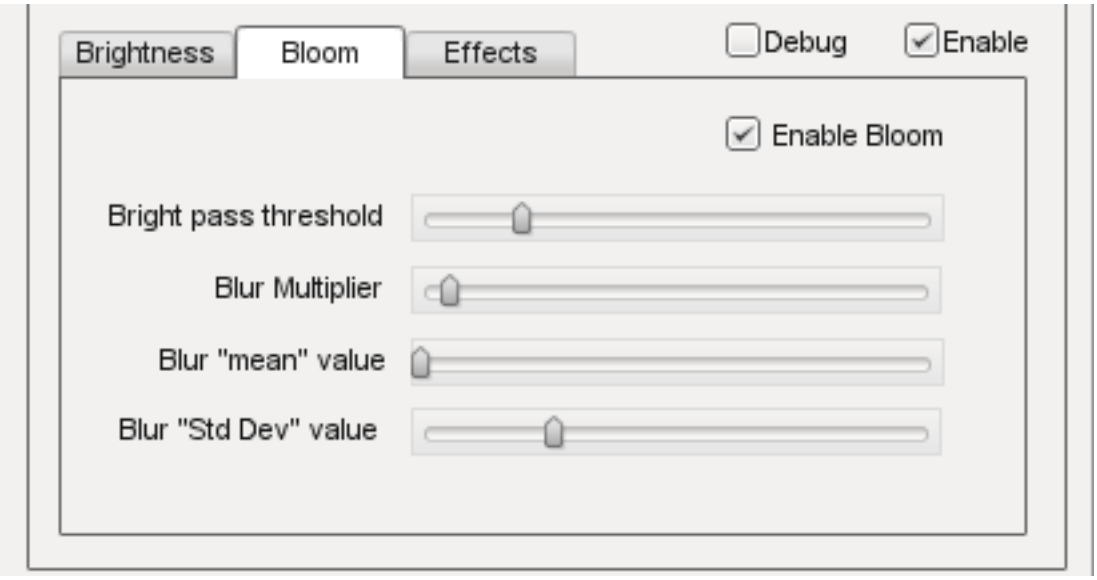
Tone Mapping Contrast Amount of interpolation between the scene and the tone mapped scene.

Key Value The tone mapping middle grey or exposure value used to adjust the overall "balance" of the image.

Minimum Luminence The minimum luninace value to allow when tone mapping the scene. Is particularly useful if your scene very dark or has a black ambient color in places.

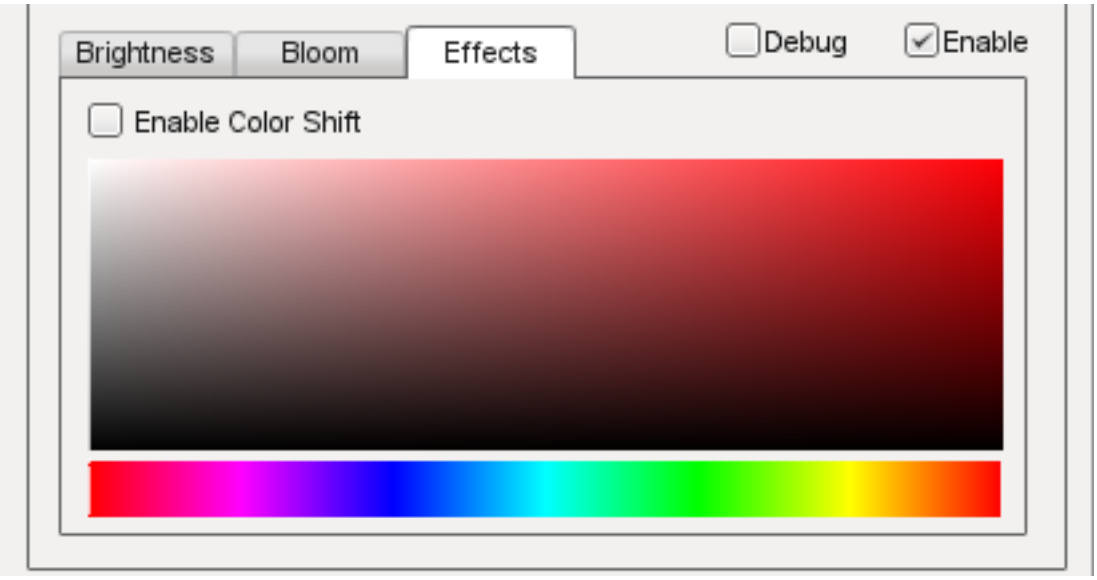
White Cutoff The lowest luminance value which is mapped to white. This is usually set to the highest visible luminance in your scene. By setting this to smaller values you get a contrast enhancement.

Brightness Adapt Rate The rate of adaptation from the previous and new average scene luminance.



Bright Pass Threshold The threshold luminace value for pixels which are considered "bright" and need to be bloomed.

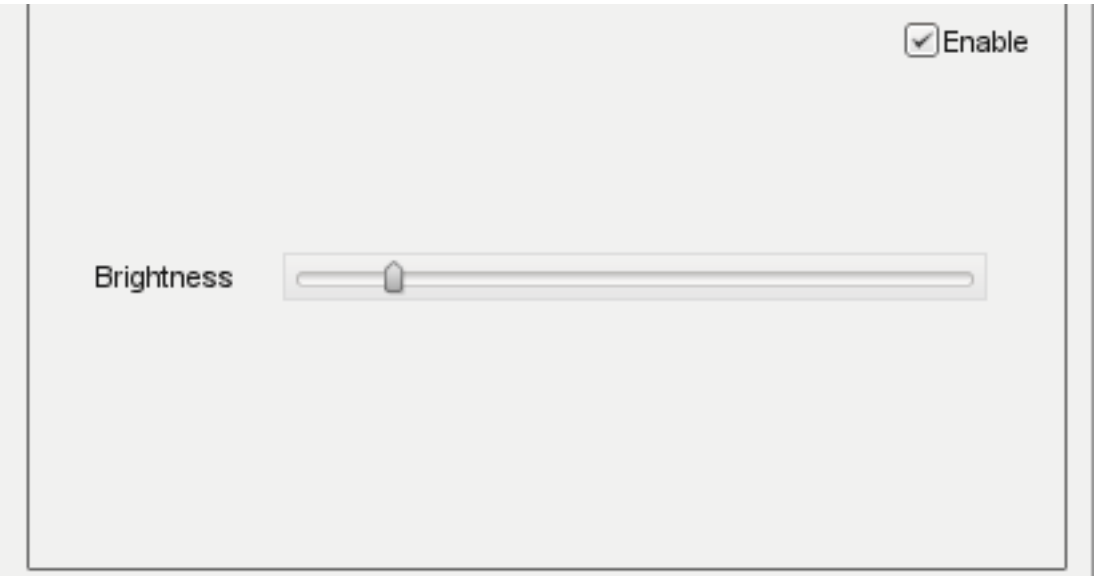
Blur multiplier/mean/Std Dev These control the gaussian blur of the bright pass for the bloom effect.



Enable color shift Enables a scene tinting/blue shift based on the selected color, for a cinematic desaturated night effect.

Light Rays

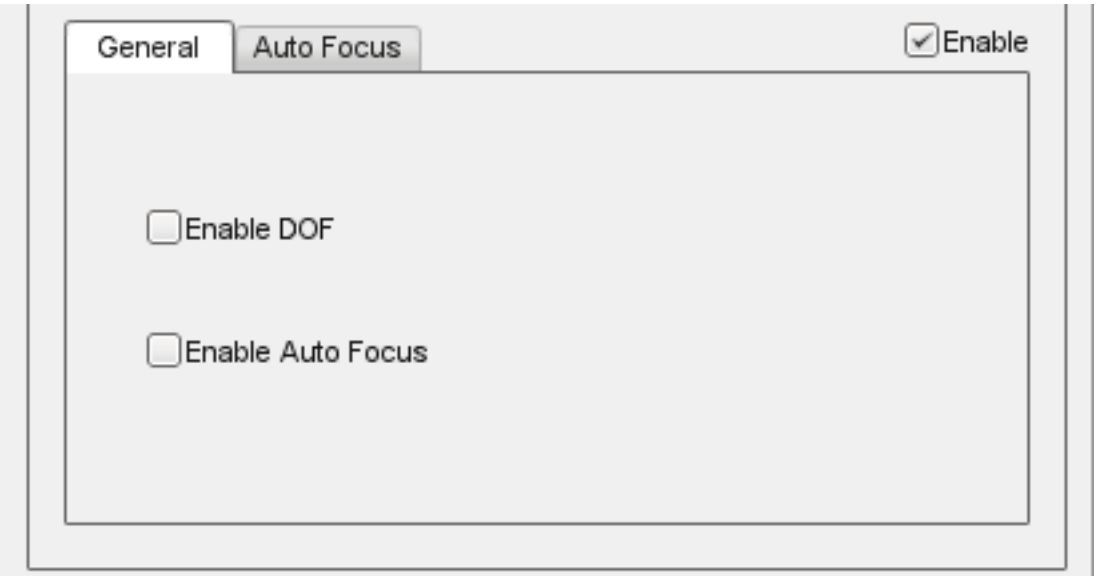
This effect creates radial light scattering (also known as god rays). It works best when the scene contains a very bright light, but even in the example above you should be able to see some scattering occuring around the crystal.



Brightness Intensity of the light ray effect.

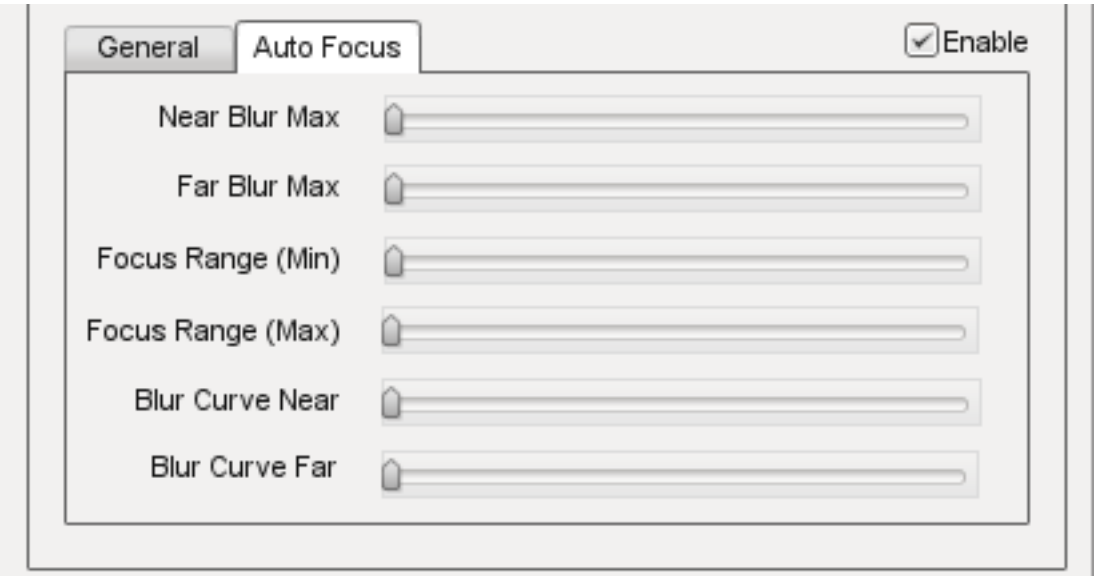
DOF

Depth of Field (DOF) simulates a camera lens, and blurs pixels based on depth from the focal point. DOF is commonly used when zooming in with a weapon.



Enable DOF Enable/disable the DOF effect.

Enable Auto Focus Determines how the focal depth is calculated. When auto-focus is disabled, focal depth is set manually by calling DOFPostEffect::setFocalDist. When auto-focus is enabled, focal depth is calculated automatically by performing a raycast at the screen-center.



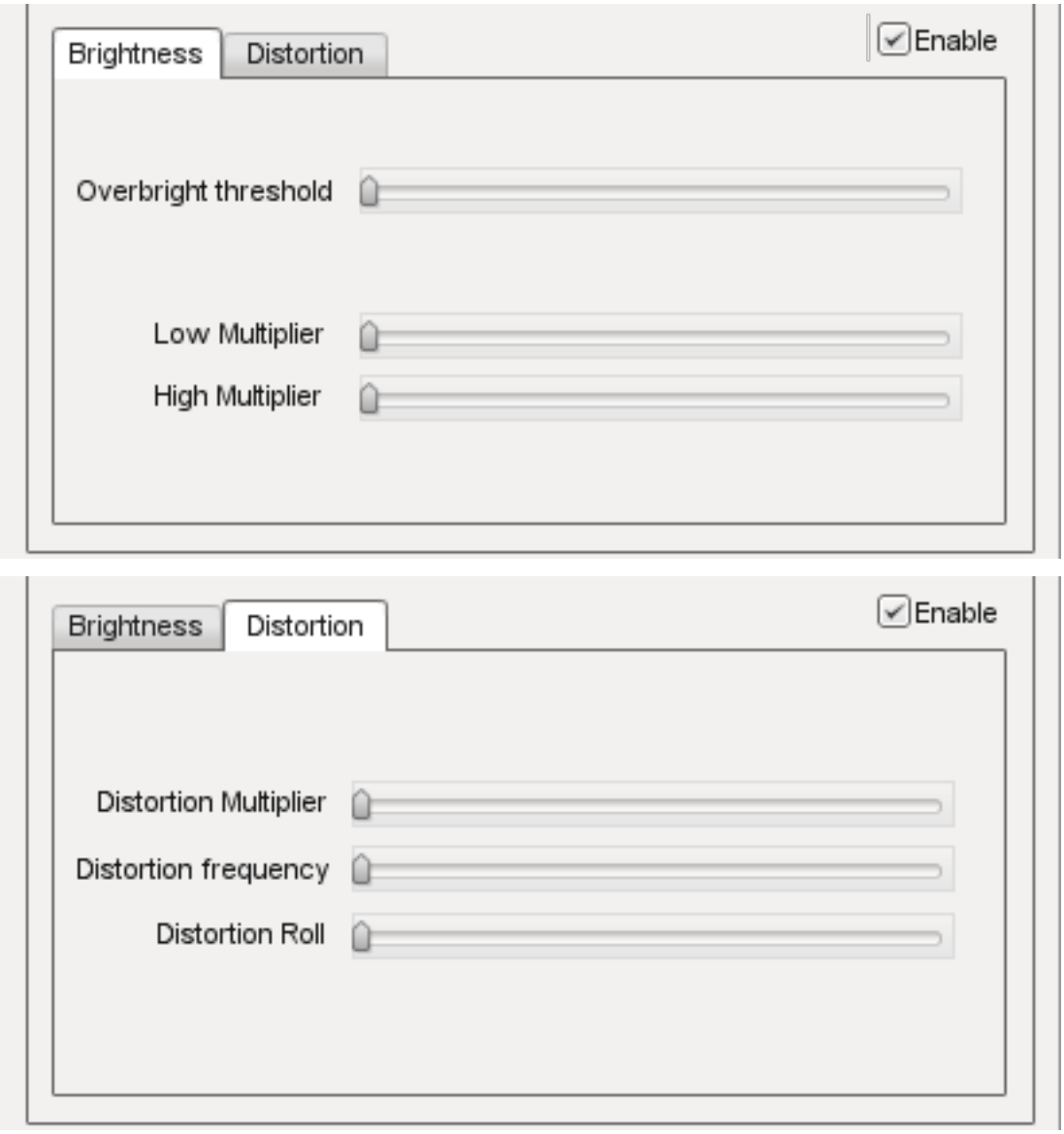
Near/Far Blur Max Sets maximum blur for pixels closer/further than the focal distance.

Focus Range (Min/Max) The min and max range parameters control how much area around the focal distance is completely in focus.

Blur Curve Near/Far Controls the gradient of the near/far blurring curve. A small number causes bluriness to increase gradually at distances closer/further than the focal distance. A large number causes bluriness to increase quickly.

Sharpness

Nightvision



2.2.10 Manipulators

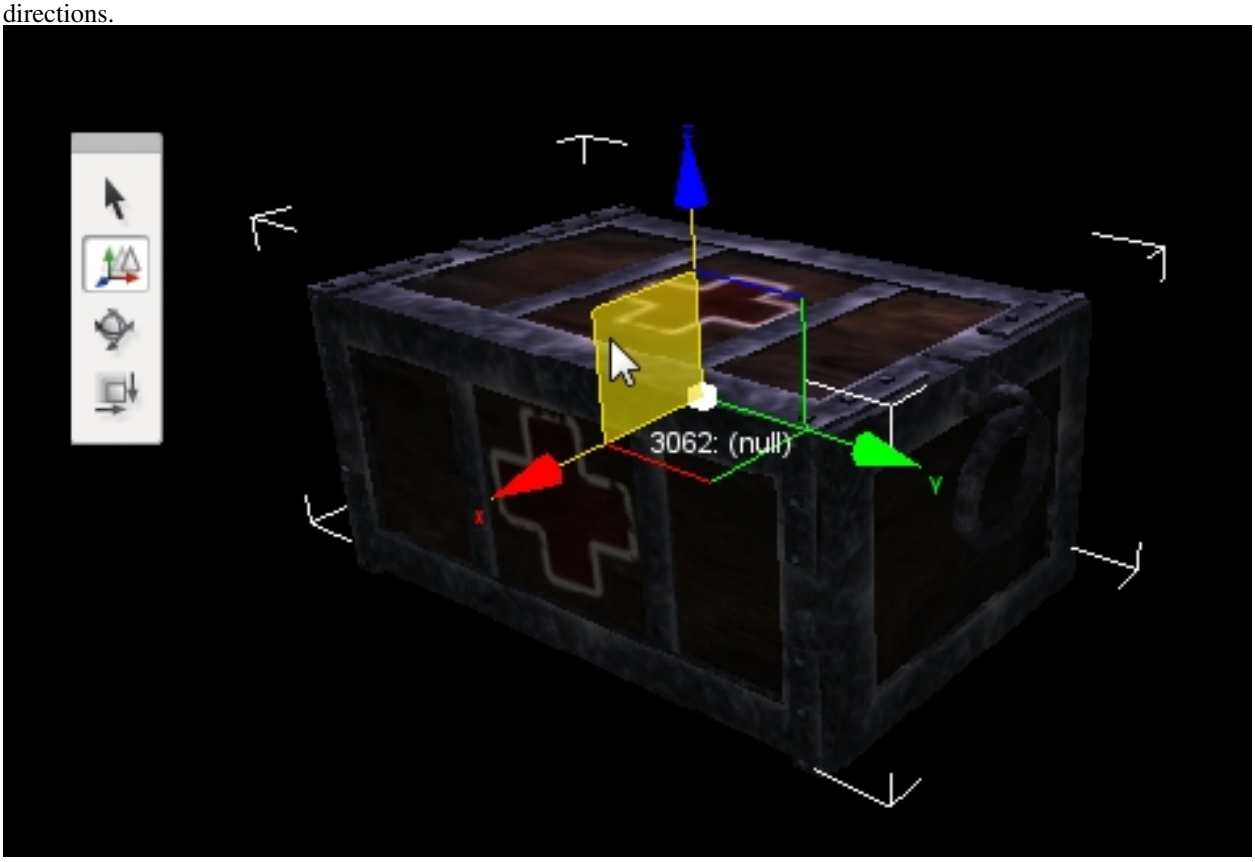
The last World Editor visual we will describe is the gizmo. A gizmo is a three dimensional rendering of an object's transforms. While using the Object Editor tool, you can use a gizmo to adjust an object's location, rotation, and scale without having to manually input number values in the Inspector Panel.

Each gizmo has a unique appearance to notify you of what you are adjusting based upon the tool that you are using.

Move Tool Gizmo

When you wish to move an object from one place to another, you will use the Move Tool. This is represented by a gizmo with arrows pointing toward different axes.

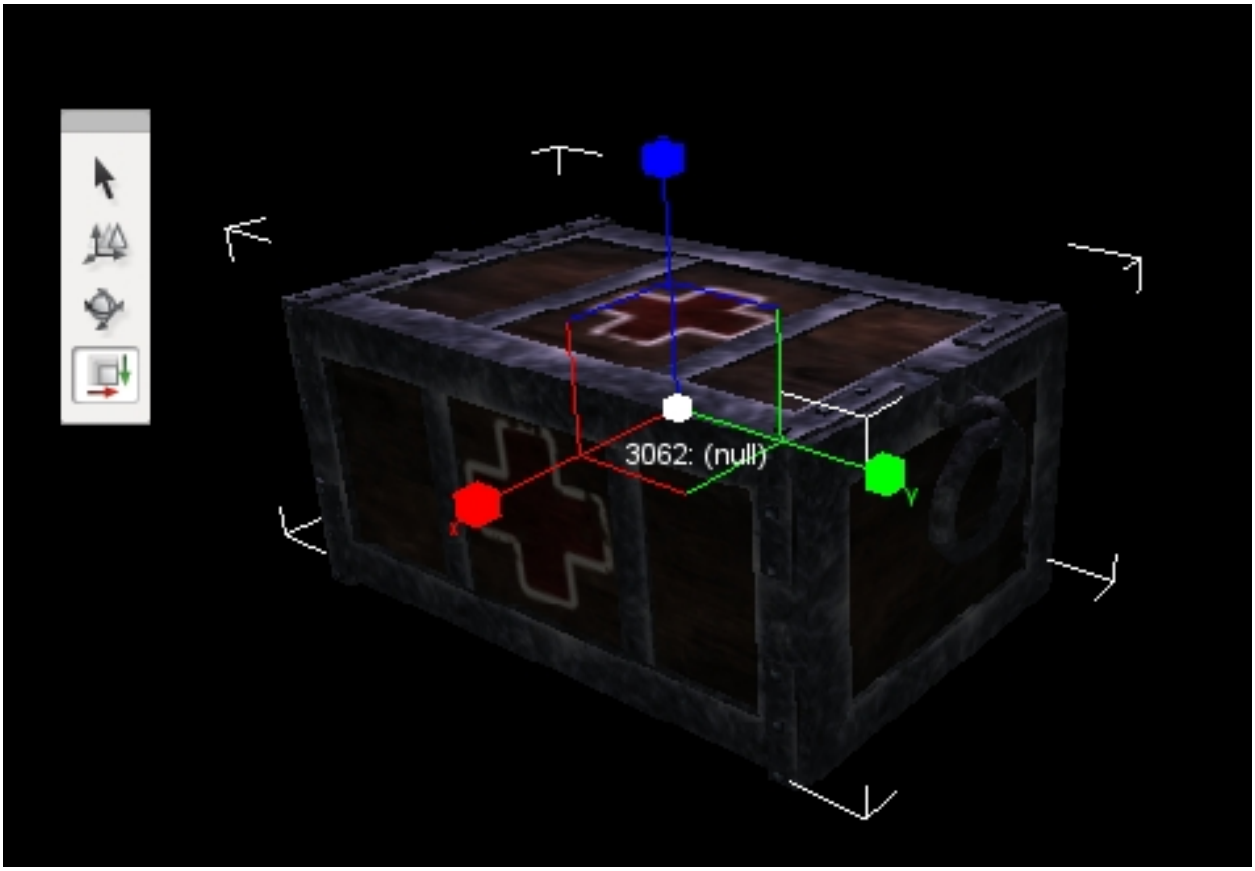
You can grab an arrow to move the object along an axis, or grab a space between two arrows to move it in both directions.



If you look carefully, you should see letters at the end of each arrow. These correspond to Torque 3D's world coordinate system. The engine utilizes the right-handed (or positive) Cartesian coordinate system, where Z is up (top), X is side (right), and Y is front (forward). This applies to the rest of the gizmos.

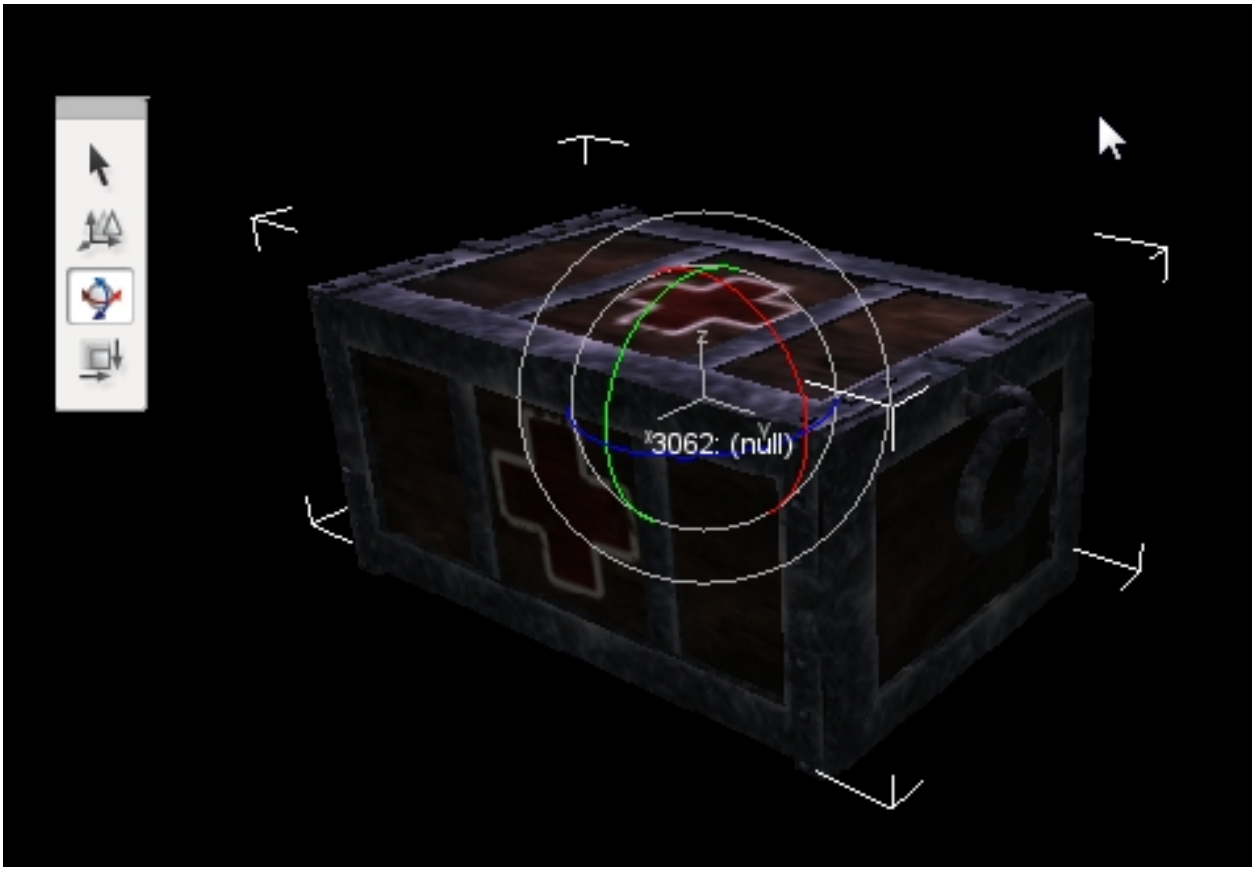
Scaling Tool Gizmo

The Scaling Tool is represented by a gizmo that looks similar to the Translate gizmo. Instead of arrows, there are blocks at the end of the gizmo lines. Dragging one of the boxes in a direction will shrink or grow your object, depending on which direction you move.



Rotation Tool Gizmo

While using the Rotation Tool, the orientation gizmo will be rendered. This gizmo looks and acts much differently than the previous two. Instead of straight lines, multiple circles will surround your object.



Dragging the red circle in a direction will rotate the object along the X-Axis. Green rotates around the Y-Axis. Blue rotates around the Z-axis. The off color circles allow you to rotate an object along multiple axes.

2.3 Level objects

2.3.1 Meshes

Meshes, referred to as shapes in these tutorials, make up most of the objects in your game. This includes players, items, weapons, vehicles, props, buildings, and so on. Currently Torque 3D supports two model formats: DTS and COLLADA.

DTS Short for Dynamix Three Space, this is a proprietary format first developed by a company called Dynamix for its game named Tribes. This has been the primary format used by Torque Technology for importing and rendering 3D model information. The format is binary, which means it is not in a human readable format.

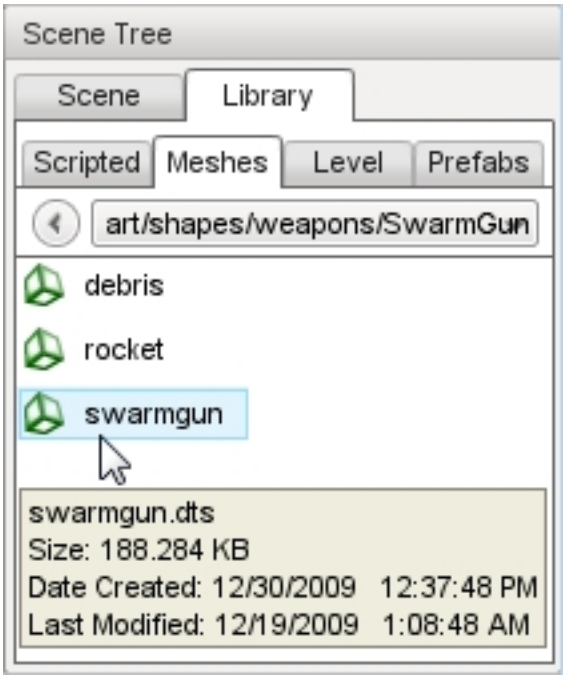
COLLADA Short for COLLAborative Design Activity. COLLADA is emerging as the format for interchanging models between DCC (Digital-Content-Creation) applications. The file extension used to identify COLLADA files is .dae which stand for "digital asset exchange". The COLLADA format has several key benefits: all of the geometry and texture information is readily available in a single file; nearly every major 3D modeling application is able to export directly to the COLLADA format; and the data is stored in an open standard XML schema, which means it can be read and tweaked manually, if need be, in any text editor rather than requiring a specific application.

Adding A DTS Model

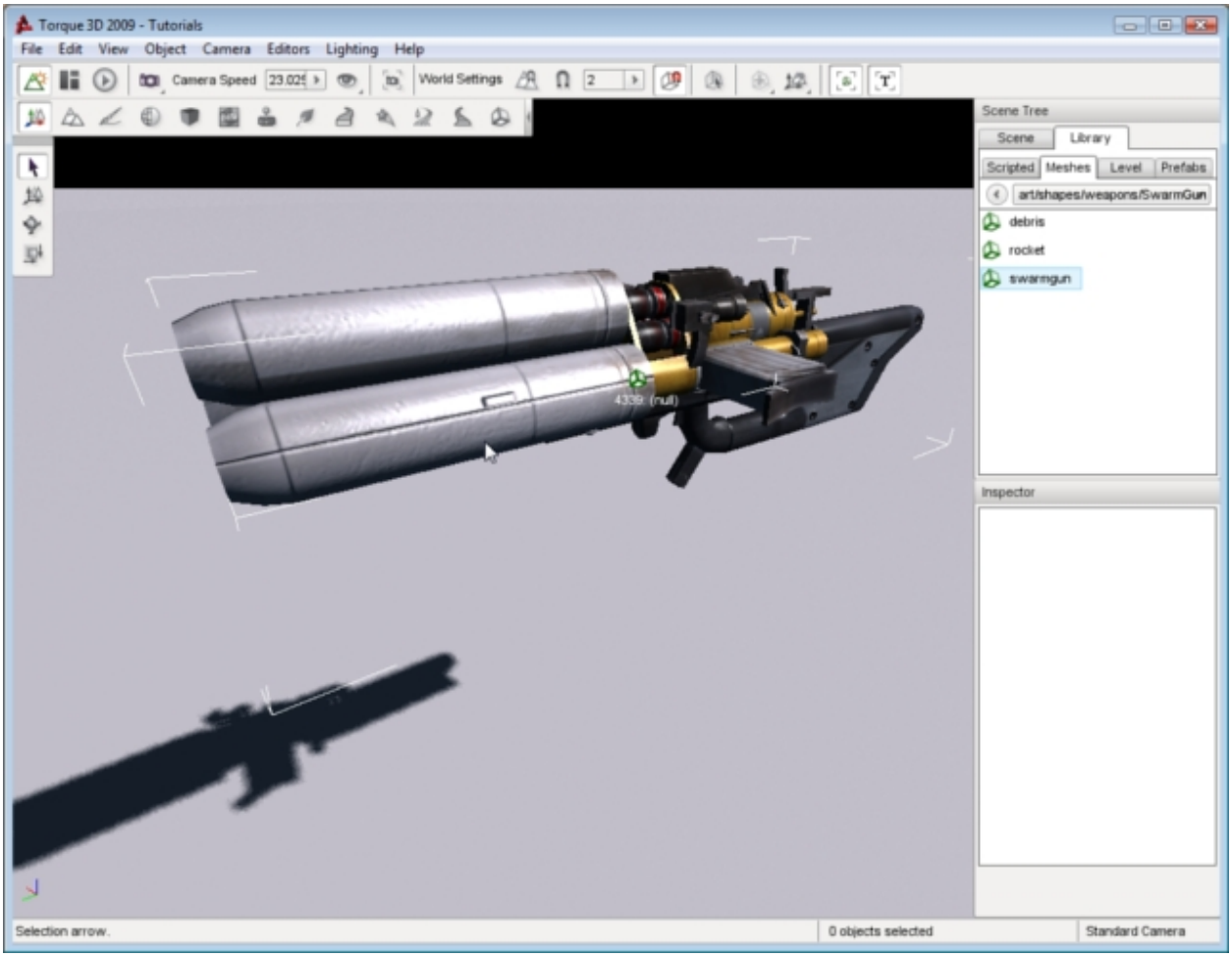
Before you can add a DTS model to World Builder so that it can be placed in a game level, it must be created with an appropriate application. It must then be placed in a folder where the World Builder can find it. When the World Editor is started it searches the game directories for objects and automatically loads any that it finds into the appropriate sub-tabs of the Library based upon the folders they were found in. Placing a model into the /game/core/art/shapes folder of your game project, or any sub-folder that you create, will allow the World Builder to find it and list it in the Library on start-up.

If you've added files or folders after starting World Builder those new entries will not appear until you have navigated out of a folder, or parent folder, and back in again.

Once a model is listed in the Library it is ready to be added to your game level. To add a model to your game level, select the Object Editor tool. Click the Library tab in the Scene Tree panel. Finally, select the Meshes sub-tab. Once the Meshes tab is open, select the entry from the drop down list. This list represents the directory containing your .dts model.



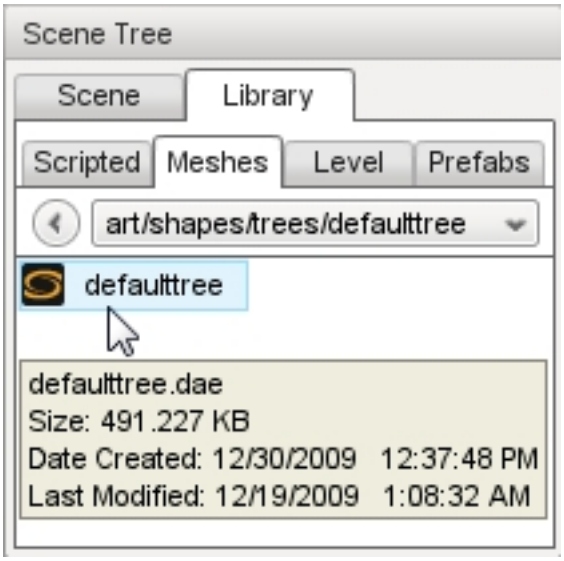
Click on the entry that contains your DTS model name. Hovering over the entry will display information about the model. Double-click the shape to automatically add it to your scene. The file should load extremely fast but you may not be able to see it right away. Where an object is placed in the scene depends upon the current drop location selection which can be set the menus Object > Drop Location command. Move your camera from its current location until the shape is in view.



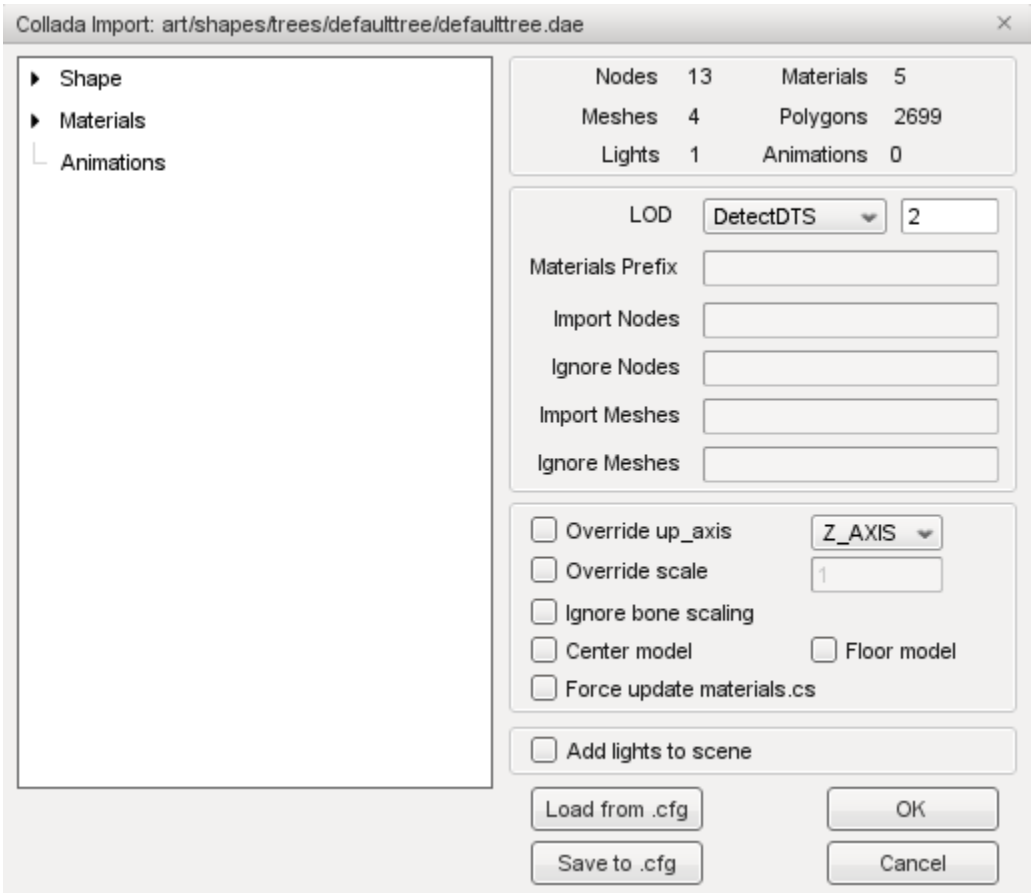
Adding A COLLADA Model

Torque 3D also has the ability to load and render COLLADA models (.dae) . The process of adding a COLLADA shape is identical to adding a DTS. You will first need to create the COLLADA file and place it where the World Editor can find it then you may place it in a level.

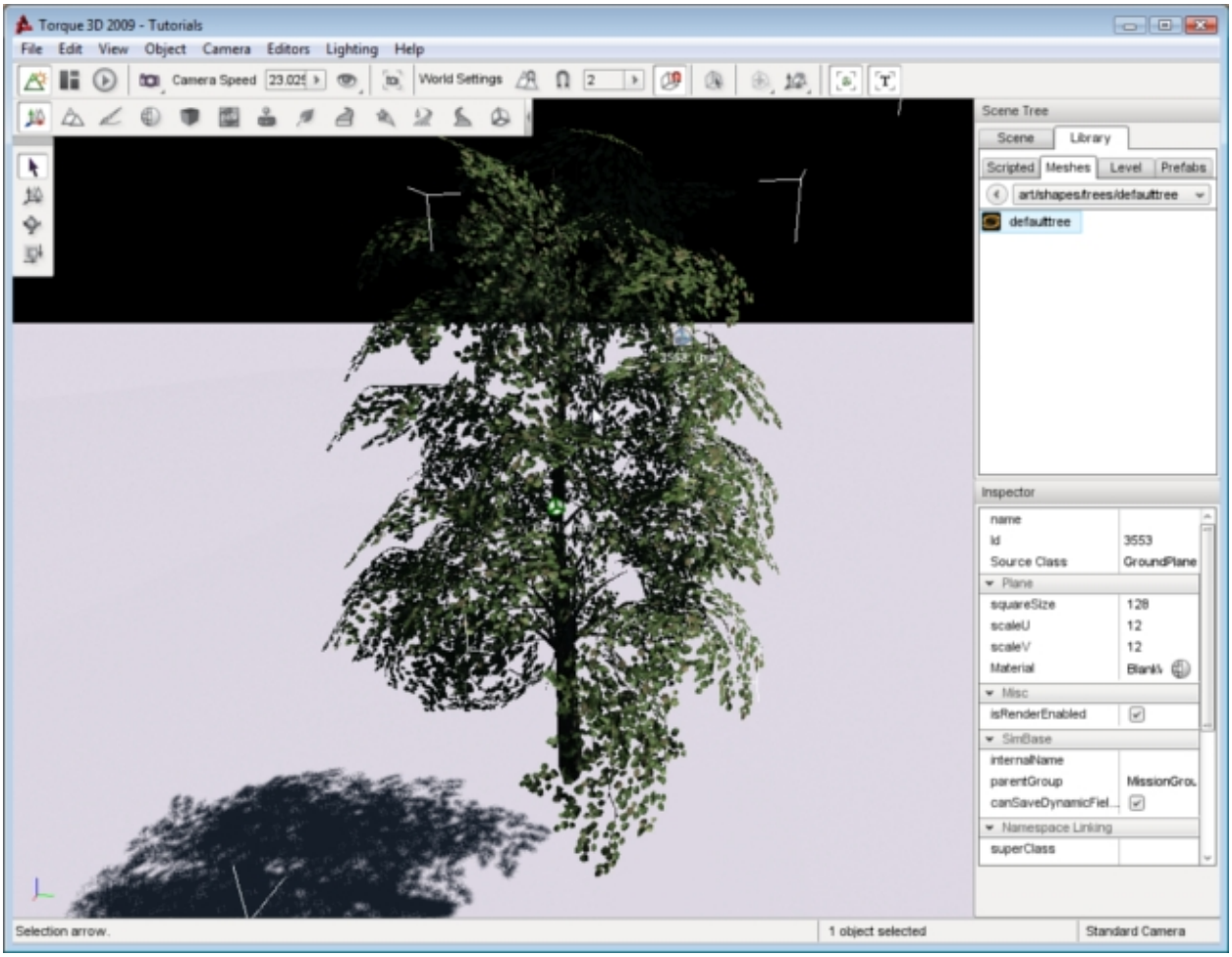
Open the Library > Meshes tab. Navigate to the directory containing your COLLADA model (.dae). If you hover over the item, you will get a brief file description.



Double clicking an object will open the COLLADA import dialog. For the purpose of this example, you can just click OK to load the mesh.



The file should load extremely fast, but you may not be able to see it right away. Where an object is placed in the scene depends upon the current drop location selection which can be set the menus Object > Drop Location command. Pull your camera up and away from its current location until the shape is in view.



Shape Properties

Each shape in a scene has properties which can be set like any other object using the *Object Editor.* Clicking a shape in the scene or selecting it from the Scene Tree will update the Inspector pane with information about that object. Shapes have their own unique set of properties.

Inspector

Name TypeName. Optional global name of this object.

id TypeCaseString. SimObjectId of this object. Read Only.

Source Class TypeCaseString. Source code class of this object. Read Only.

Transform

Position TypeMatrixPosition. Object world position.

Rotation TypeMatrixRotation. Object world orientation.

Scale TypePoint3F. Object world scale.

Media

shapeName TypeFilename. Name and path of model file.

Rendering

playAmbient TypeBool. Play the ambient animation. Animation itself must be named ambient.

meshCulling TypeBool. Enables detailed culling of meshes.

originSort TypeBool. Enables sorting by origin rather than bounds.

Collision

collisionType TypeBool. TypeEnum. The type of mesh data to use for collision queries.

decalType TypeEnum. The type of mesh data to return for decal generation.

allowPlayerStep TypeBool. Allow a player to qalk up sloping polygons on collision.

Debug

renderNormals TypeF32. Debug rendering mode which highlights shape normals.

forceDetail TypeS32. For rendering at a particular detail for debugging.

Editing

isRenderEnabled TypeBool. Only render if true (and if class is render-enabled, too).

isSelectionEnabled TypeBool. Disables editor selection of this object.

hidden TypeBool. Toggle visibility of this object.

locked TypeBool. Toggle whether this object can be edited.

Mounting

mountPID TypeBool. TypePID. PersistentID of the object this one is mounted to.

mountNode TypeS32. Node this object is mounted to.

mountPos TypeBool. Position where this object is mounted.

mountRot TypeBool. Rotation of this object in relation to the mount node.

Object

internalName TypeString. Optional name that may be used to lookup this object within a SimSet.

parentGroup TypeSimObjectPtr. Group hierarchy parent of the object.

class TypeString. Script class of this object.

superClass TypeString. Script superClass of this object.

Persistence

canSave TypeBool. Whether this object can be saved.

canSaveDynamicFields TypeBool. True if dynamic fields added at runtime should be saved. Defaults to true.

persistentID TypePID. Unique identifier for this object.

Dynamic Fields

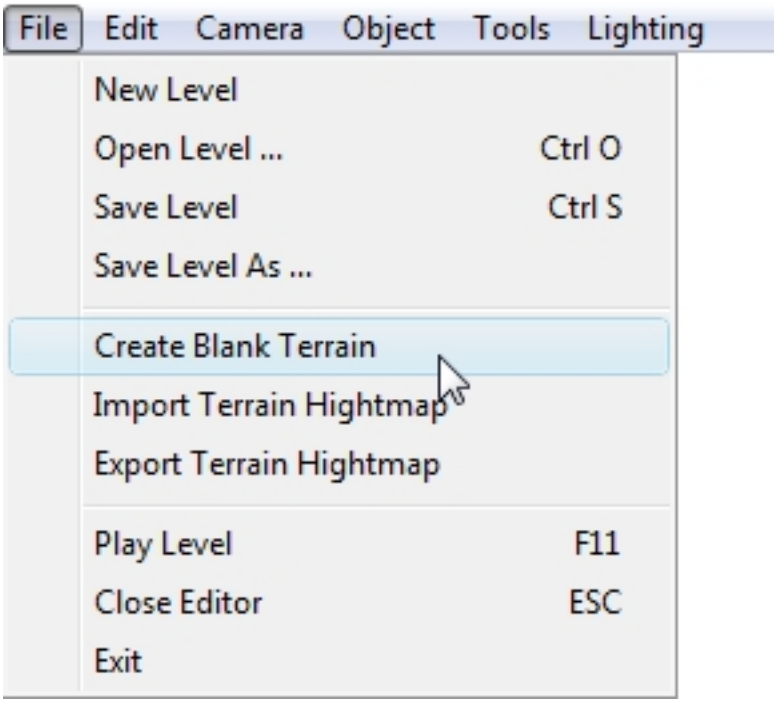
N/A - None by default.

2.3.2 Terrain Block

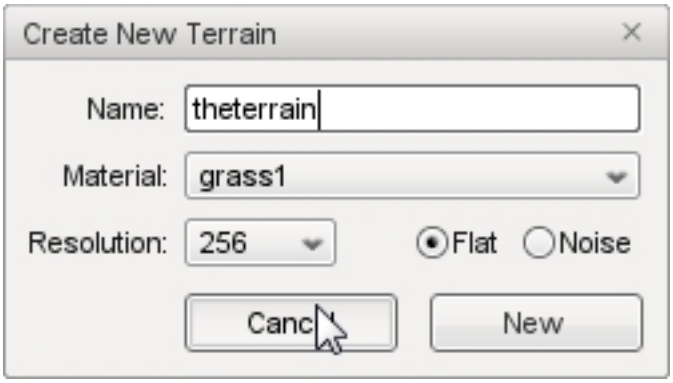
Terrain simulates a land mass in your game which can be occupied, traversed, or flown over by objects in your game world. Terrain is represented in a game level by a Terrain Block. There are three methods to add a Terrain Block to a level: Create a blank terrain, add an existing .ter file or import a heightmap.

Creating Blank Terrain

To create a new blank terrain start from the menu by selecting File>Create Blank Terrain.

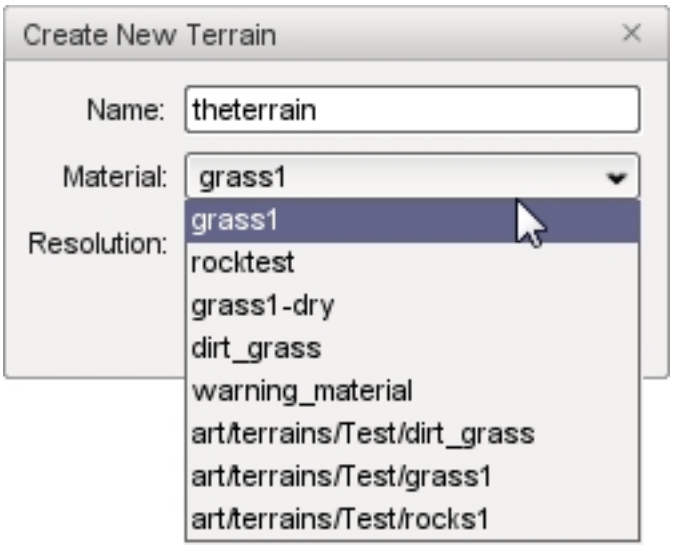


After you click the menu entry, a Create New Terrain Dialog will appear.



The Name field allows you to specify a name for your Terrain Block. This name will appear in the Scene Tree and can be used to reselect your terrain later for editing. Enter a name for the terrain in the text box, in this example *theterrain.*

The Material for the terrain, that is the texture that will be displayed to depict the ground cover, is selected using a drop-down list. This list is populated by the World builder with all the existing materials created specifically for terrains.

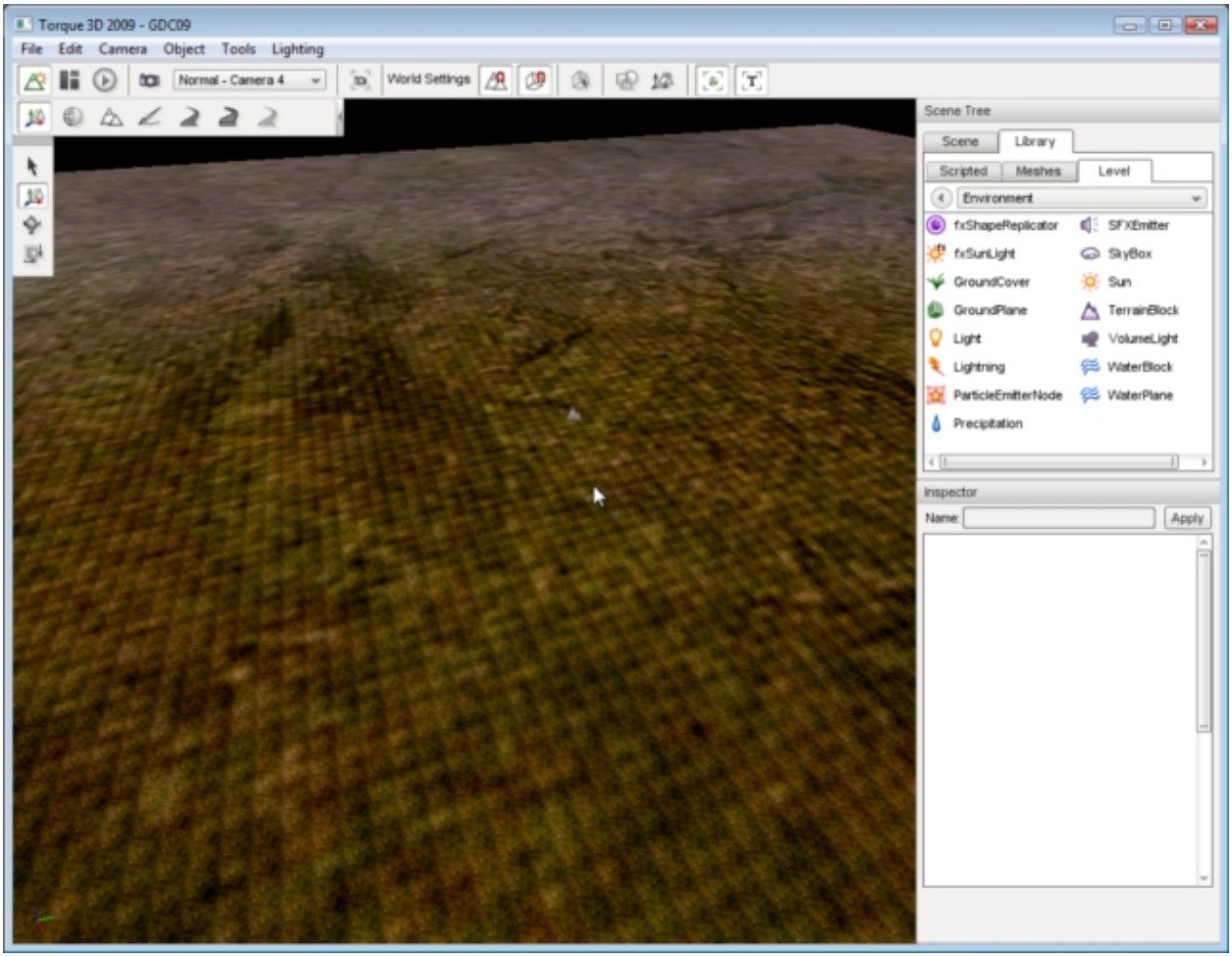


The Resolution that you select from that drop-down list determines the size of the terrain that will be created. The size of the terrain that you choose is largely dependent on the design of your game. You will have to experiment to find the right size that works for each game you create and some combinations of options are not very practical. For example, selecting a terrain size of 256 and using the Noise option will result in a terrain that is so drastically contoured that it will not be of much use.

The radio buttons to the right of the Resolution dropdown determine the smoothness of the terrain that is generated. Selecting Flat will create a relatively smooth terrain and selecting Noise will generate a bumpy terrain.

Creating a Flat Terrain

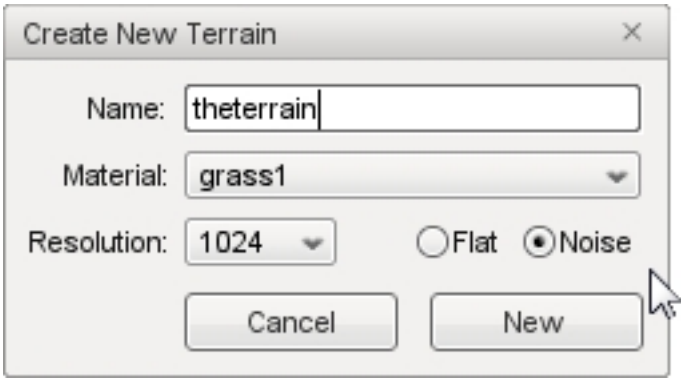
To create a flat terrain: from the main menu select File > Create Blank Terrain; enter a name; select a material; select a size such as 256; and select the Noise radio button, then click the Create New button. A contoured Terrain Block will be generated and automatically loaded into the scene.



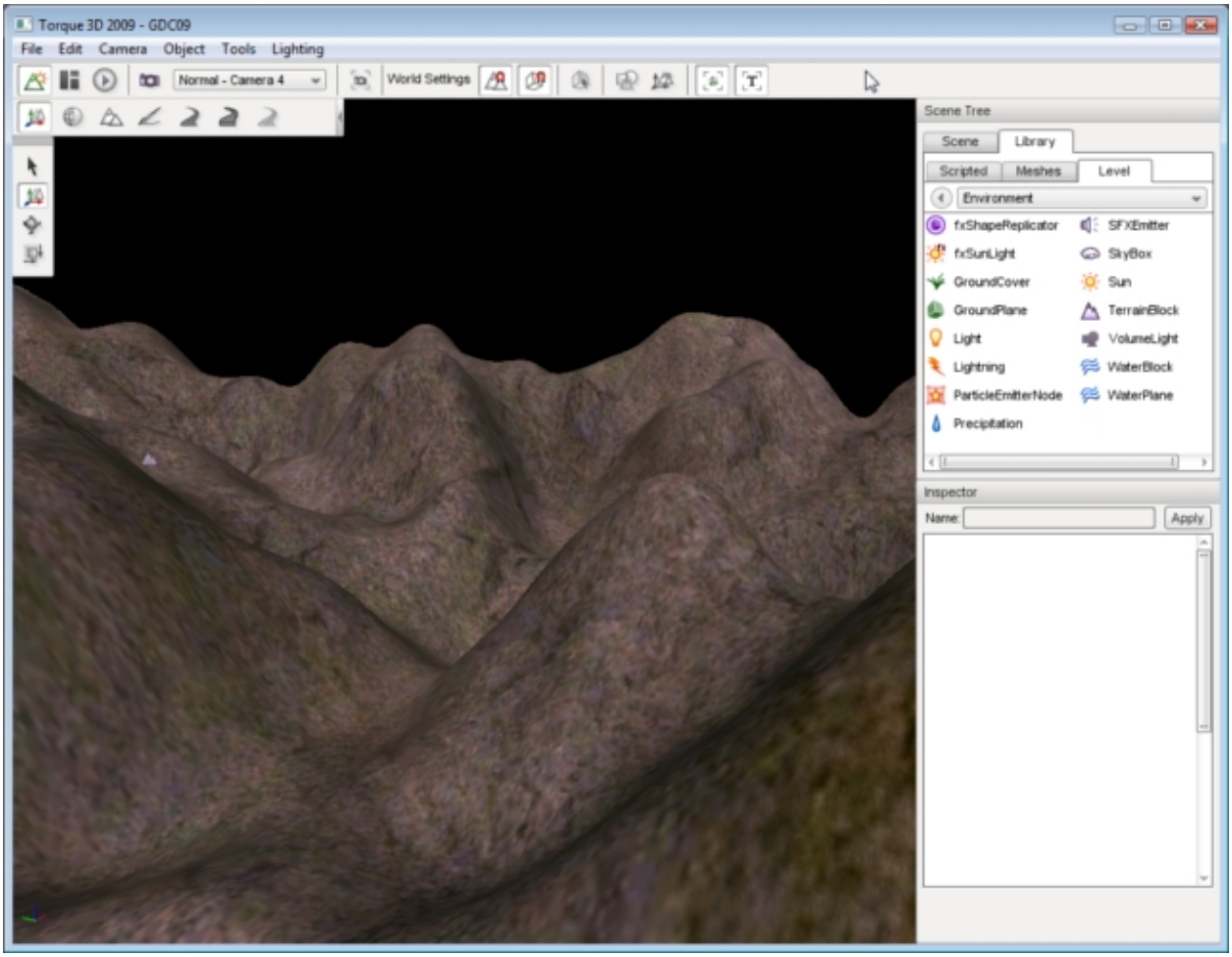
A Flat terrain is a great place to start, but is more suitable for terrains that will remain relatively flat. Using a flat terrain requires you to create all the terrain details yourself using the Terrain Editor.

Creating a Bumpy Terrain

To create a bumpy terrain: from the main menu select File>Create Blank Terrain; enter a name; select a material; select a larger size such as 1024; and select the Noise radio button, then click the Create New button.



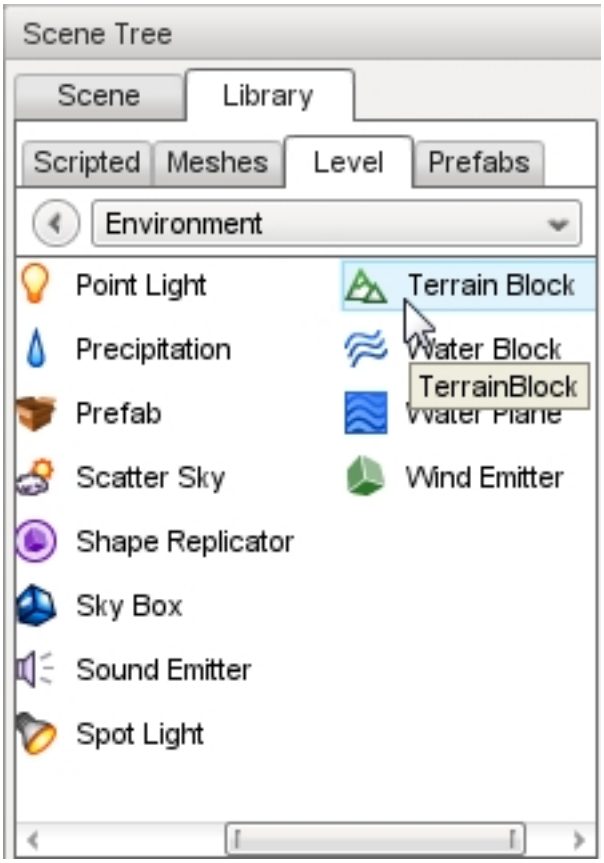
Terrain Block loaded into the scene. A contoured extremely mountainous terrain will be generated and automatically loaded into your scene. The noise algorithm randomly generated the hills and valleys for you.



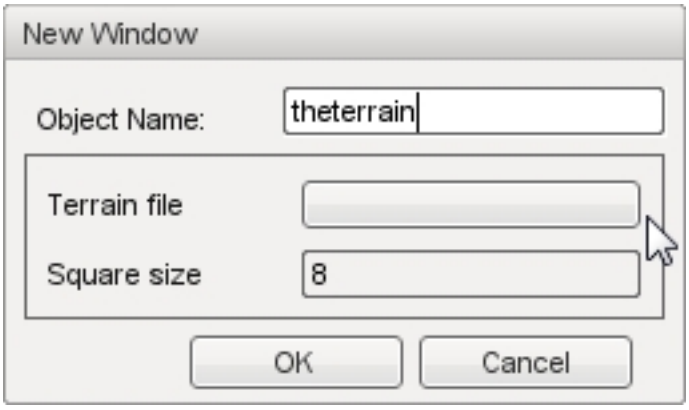
Starting with a contoured terrain this is a decent method of starting from scratch with a little randomness thrown in.

Adding an Existing Terrain File

To add an existing terrain file to a level start by selecting the Object Editor tool. Locate your Library panel and click it. Click on the Level tab then select the Environment folder. Once that is open, locate the Terrain Block entry, and double-click it.



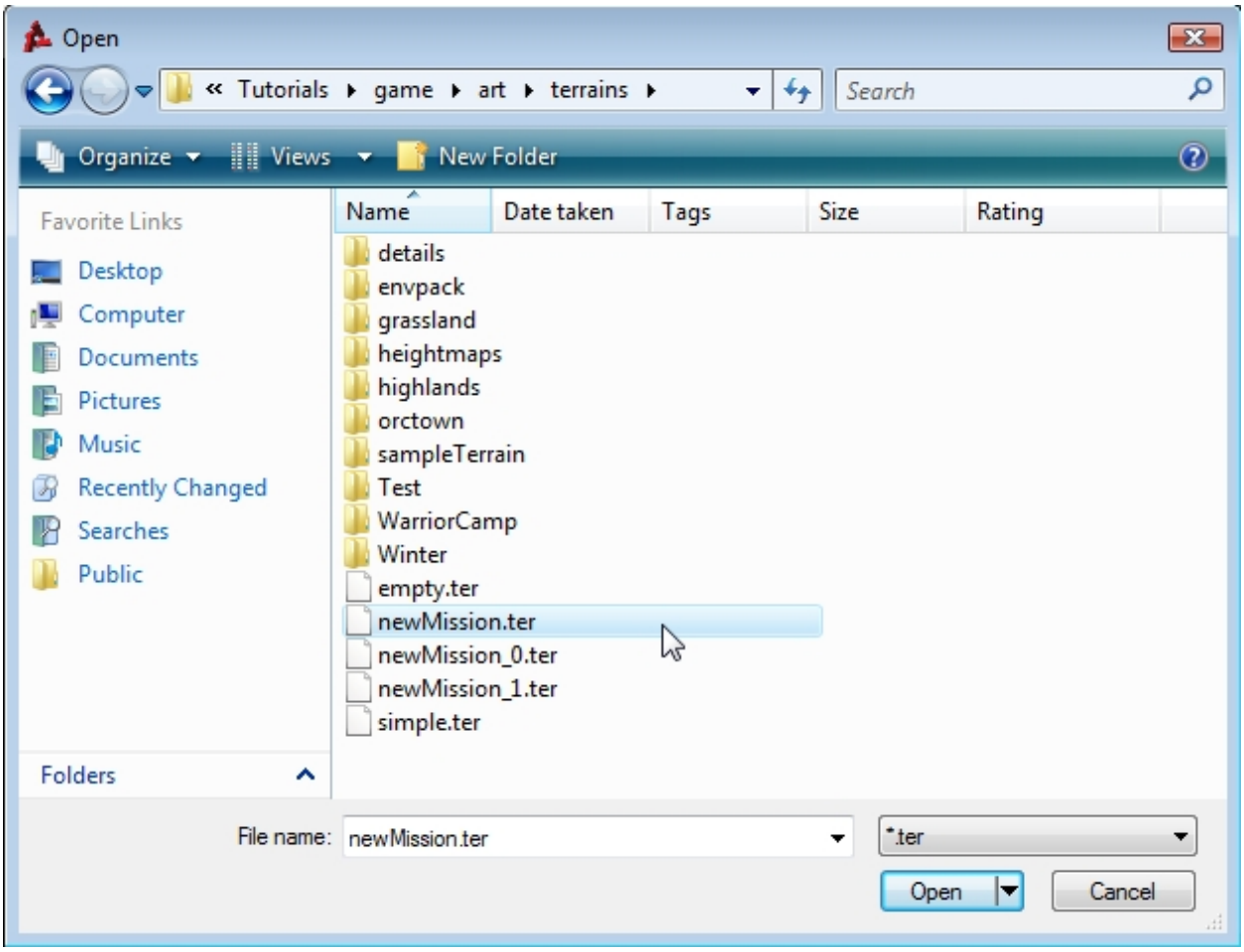
The new terrain dialog will open.



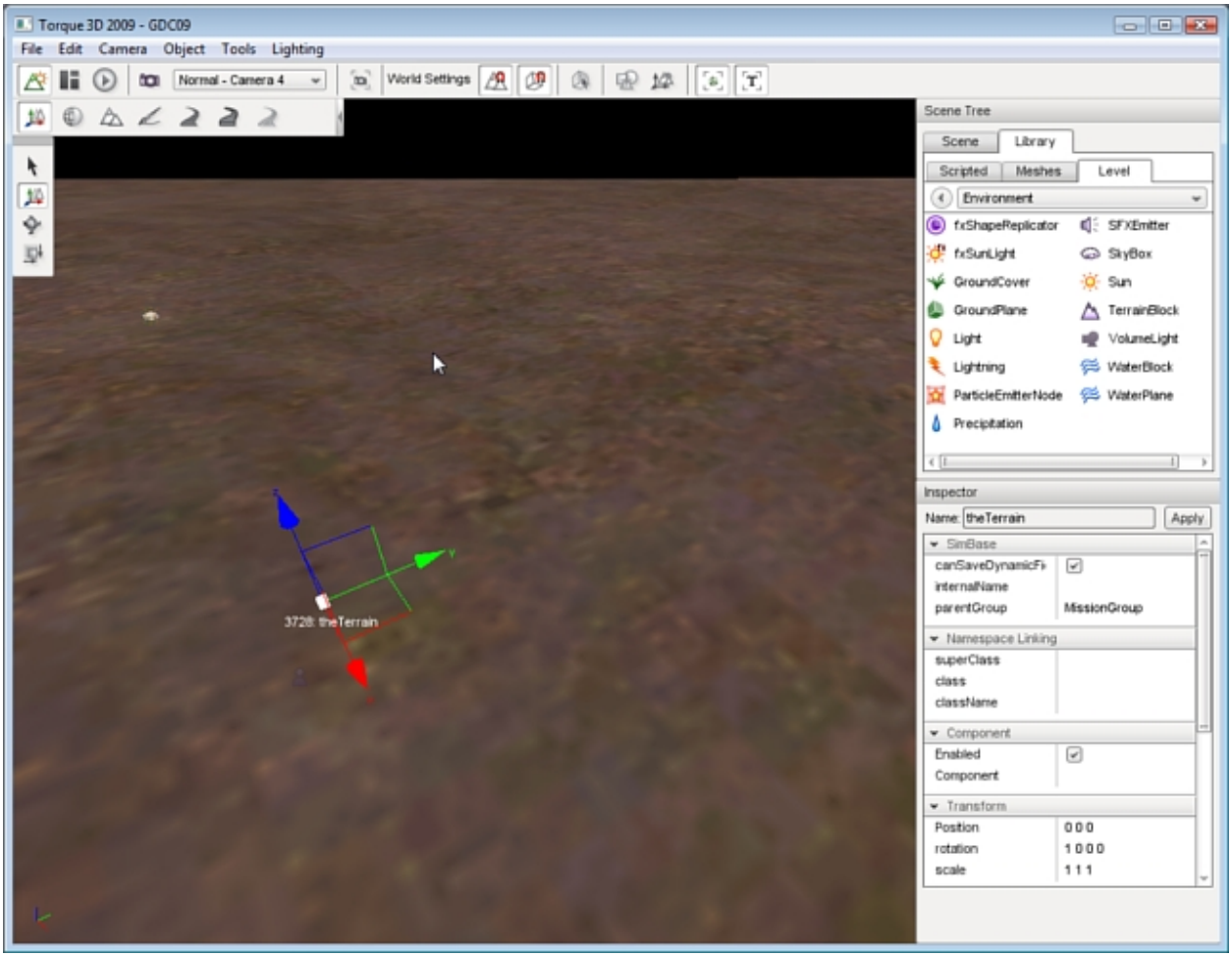
The Object Name field allows you to specify a name for your Terrain Block. This name will appear in the Scene Tree and can be used to reselect your terrain later for editing. Enter a name for the terrain in the text box, in this example theterrain.

The Terrain file box indicates the information file which holds the data describing the terrain to be loaded. Clicking the box loads the OS file browser.

Terrain files are named with a .ter extension. The .ter file type is a proprietary format that contains terrain data understood by Torque 3D. Locating a .ter file then clicking Open/OK will cause it to be selected as the Terrain file to be loaded.



Leave the square size in the dialog set to its default value then click OK. The .ter file will be immediately imported into your scene with both geometry and textures. The sample shown here is a very simple and low detailed terrain file.



Importing a Terrain

The most recommended and effective method to add a Terrain Block to a level is to import the terrain from external data files. However, this method requires the skill and the third-party tools to create those data files. Very high-quality and professional-looking terrain can be created with tools such as L3DT and GeoControl. These tools allow you to generate extremely detailed heightmaps that can be imported by Torque 3D and to generate terrain data.

There are several types of asset required to import and use a terrain in Torque 3D using this method:

- a heightmap
- an opacity map and layers
- texture files.

Heightmaps

The primary asset required is a heightmap. A heightmap is a standard image file which is used to store elevation data rather than a visible picture. This elevation data is then rendered in 3D by the Torque engine to depict the terrain. The heightmap itself needs to be in a 16-bit greyscale image format, the size of which is a power of two, and must be square. The lighter an area of a heightmap is, the higher the elevation will be in that terrain location.

Fig. 2.14: Example Heightmap

Opacity Maps

An opacity map acts as a mask, which is designed to assign opacity layers. Opacity layers need to match the dimensions of the heightmap. For example, a 512x512 heightmap can only use a 512x512 opacity map.

If the opacity map is a RGBA image, four opacity layers will be used for the detailing (one for each channel). If you use an 8-bit greyscale image, only a single channel. You can then assign materials to the layers. This allows us to have up to 255 layers with a single ID texture map, saving memory which we can apply to more painting resolution.

Notice that the following example Opacity Map resembles the original heightmap.

Texture Files

Texture files "paint" the terrain giving it the appearance of real ground materials. When creating a terrain from scratch textures can be manually applied to it using the Terrain Painter, which is built into the World Editor, but that is a time and effort intensive method. Instead of hand painting them, the opacity layer will automatically assign textures to the terrain based upon what channel they are loaded into.

For each type of terrain to be rendered you will want to have three textures: (1) a base texture, also referred to as a diffuse texture, (2) a normal map, and (3) a detail mask.

The base represents the color and flat detail of the texture. The normal map is used to render the bumpiness or depth of the texture, even though the image itself is physically flat. Finally, the detail map provides up-close detail, but it absorbs most of the colors of the base map.

Importing a Heightmap

To import a heightmap for terrain start the World Editor, then from the menu select File > Import Terrain Heightmap:

Fig. 2.15: Example Opacity Map

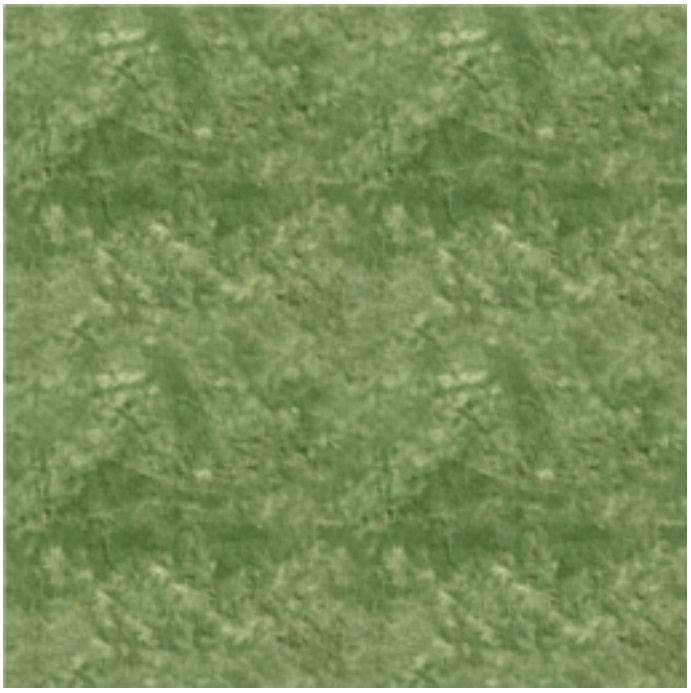


Fig. 2.16: Diffuse

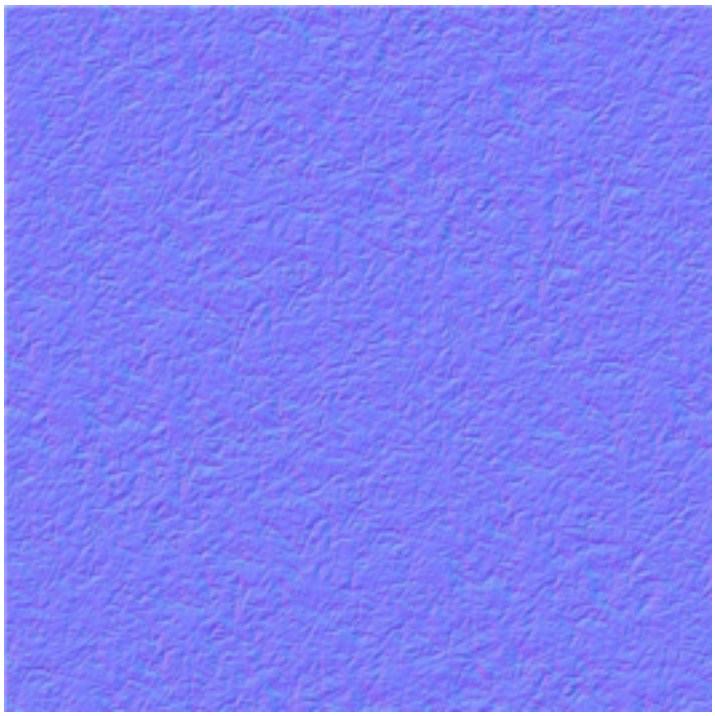
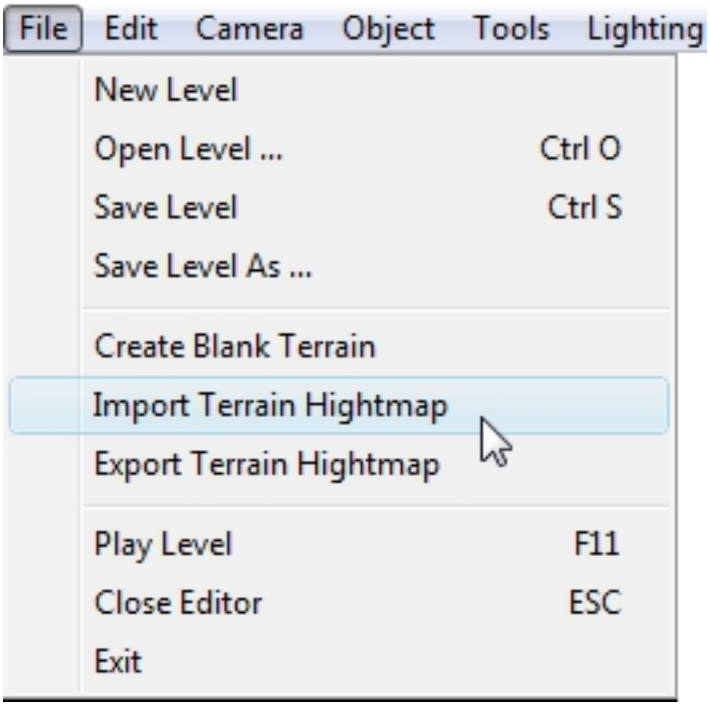
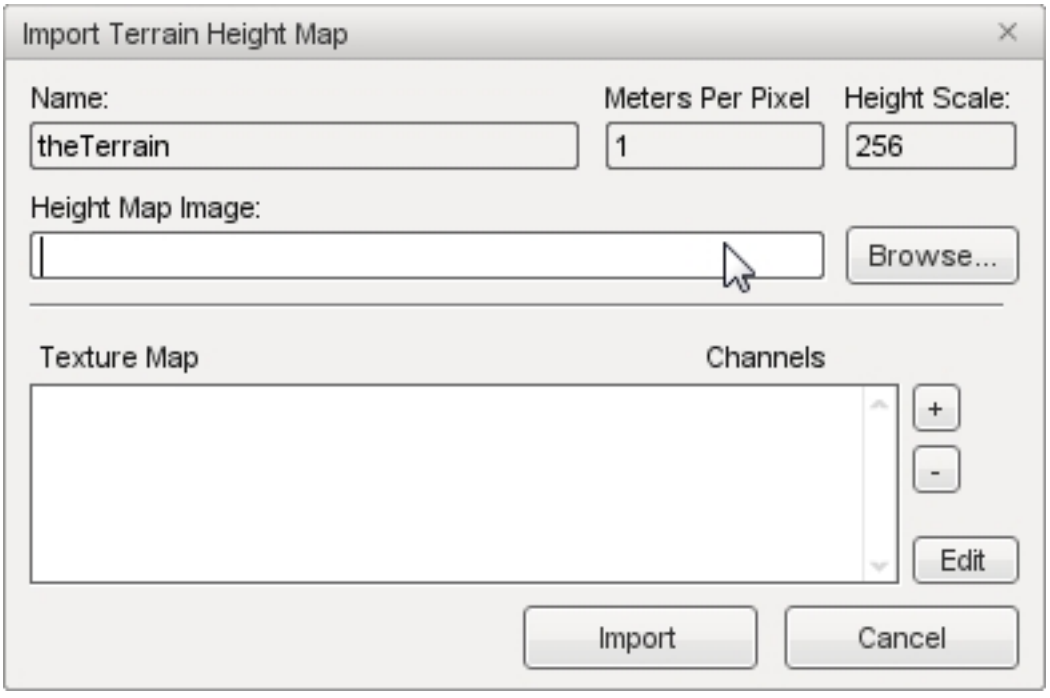


Fig. 2.17: Normal

Fig. 2.18: Detail



The Import Terrain heightmap dialog will appear.



Name If you specify the name of an existing Terrain Block in the dialog it will update that existing Terrain Block and its associated .ter file. Otherwise, a new Terrain Block will be created.

Meters Per Pixel What was the Terrain Block SquareSize (meters per pixel of the heightmap), which is a floating point value. It does not require power of 2 values.

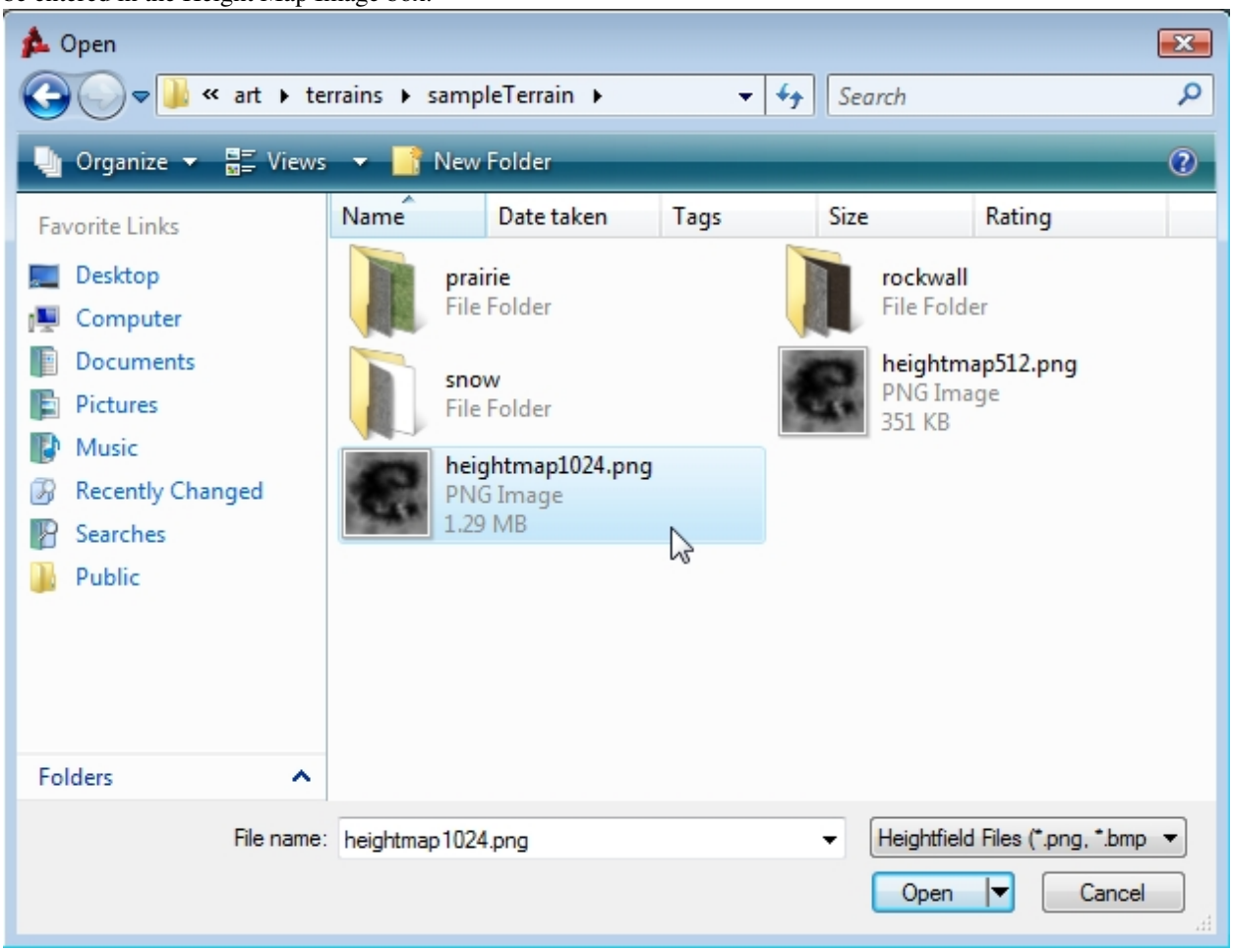
Height Scale The height in meters that you want pure white areas of the heightmap to present.

Height Map Image File path and name of a .png or .bmp file which is the heightmap itself. Remember, this needs to

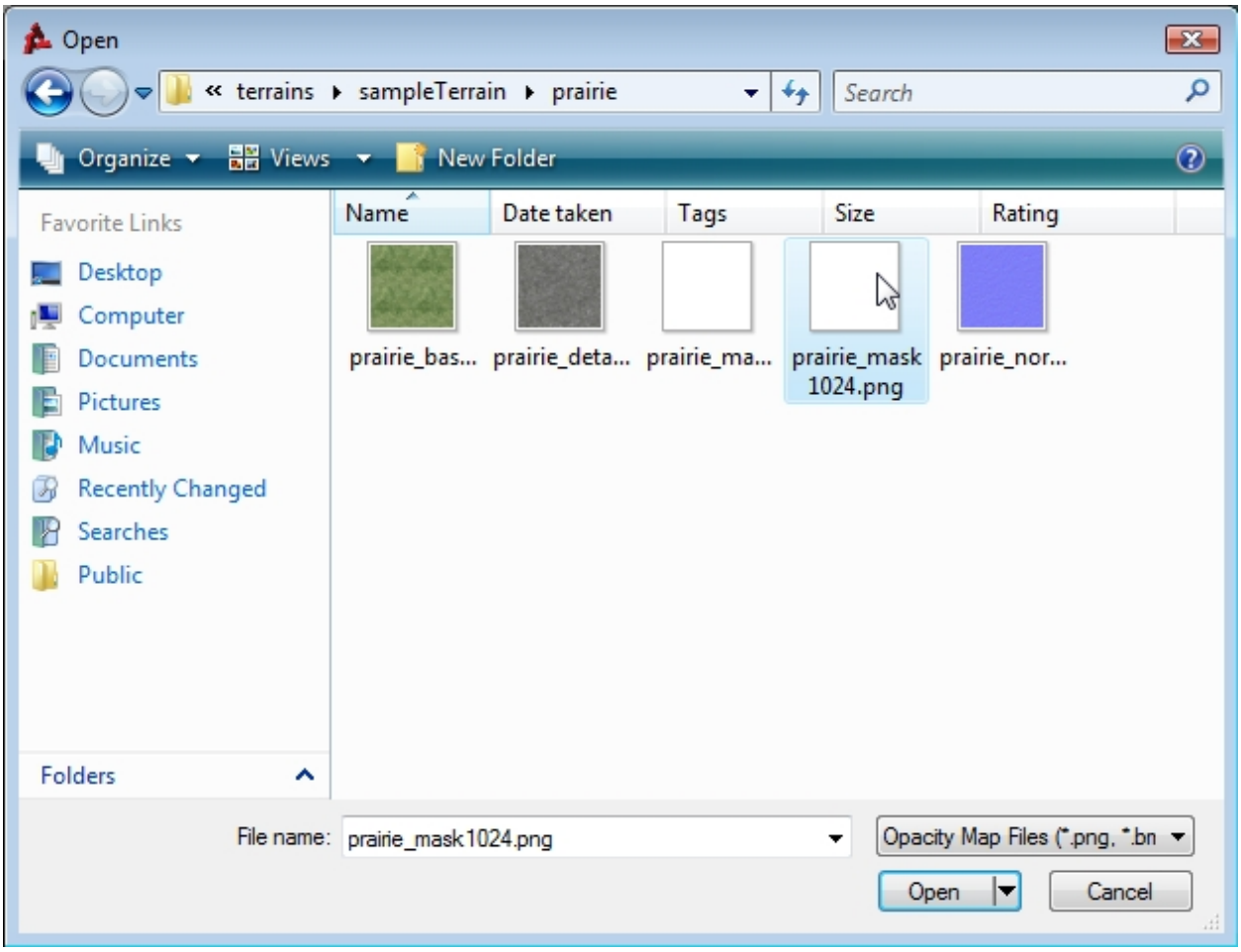
be a 16-bit greyscale image, the size of which is a power of two, and it must be square.

Texture Map This list specifies the opacity layers, which need to match the dimensions of the heightmap image. If you add an RGBA image it will add 4 opacity layers to the list, one for each channel. If you add an 8-bit greyscale image, it will be added as a single channel.You can then assign materials to the layers. If you do not add any layers or do not add materials to the layers, the terrain will be created with just the Warning Material texture.

Click the browse button to the right of the Height Map Image box to open a file browser dialog. Navigate to where your terrain files are located, select the desired heightmap PNG file, then click Open. The selected heightmap file will be entered in the Height Map Image box.

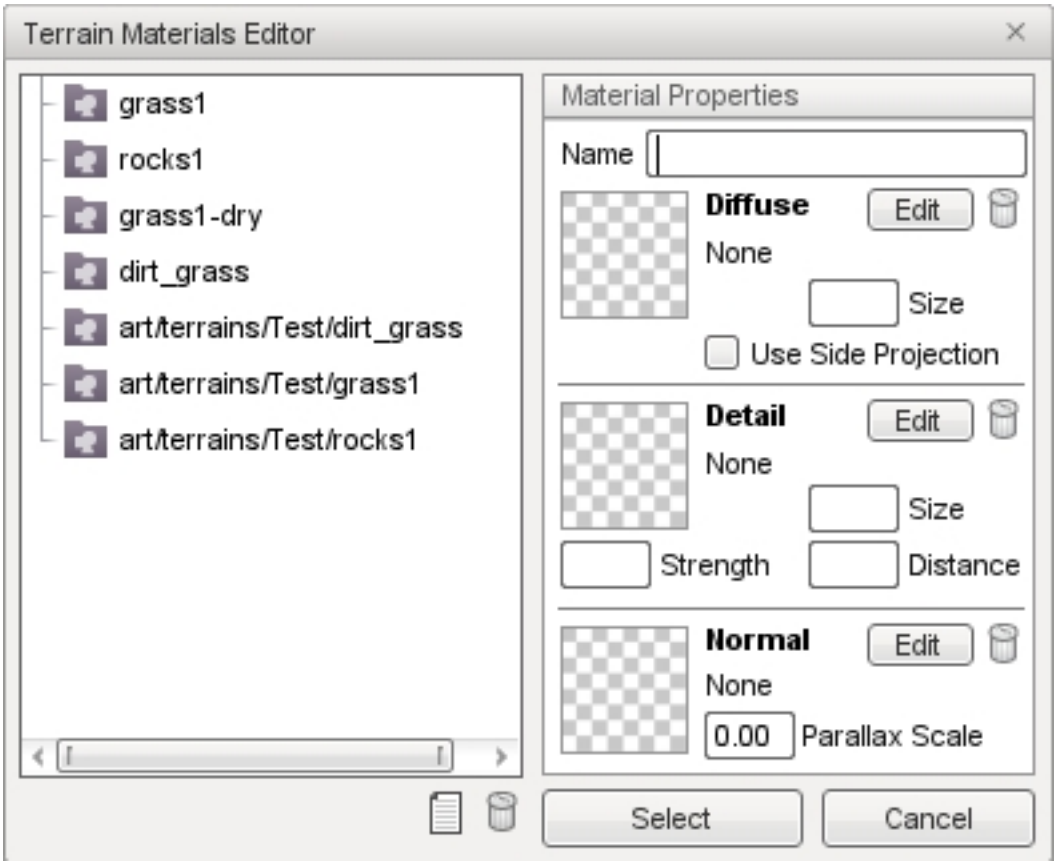


Click on the + button next to Texture Map to open another file browser. This is where you add opacity layers. Start by locating the masks. If you have the right assets, it should resemble something like this:

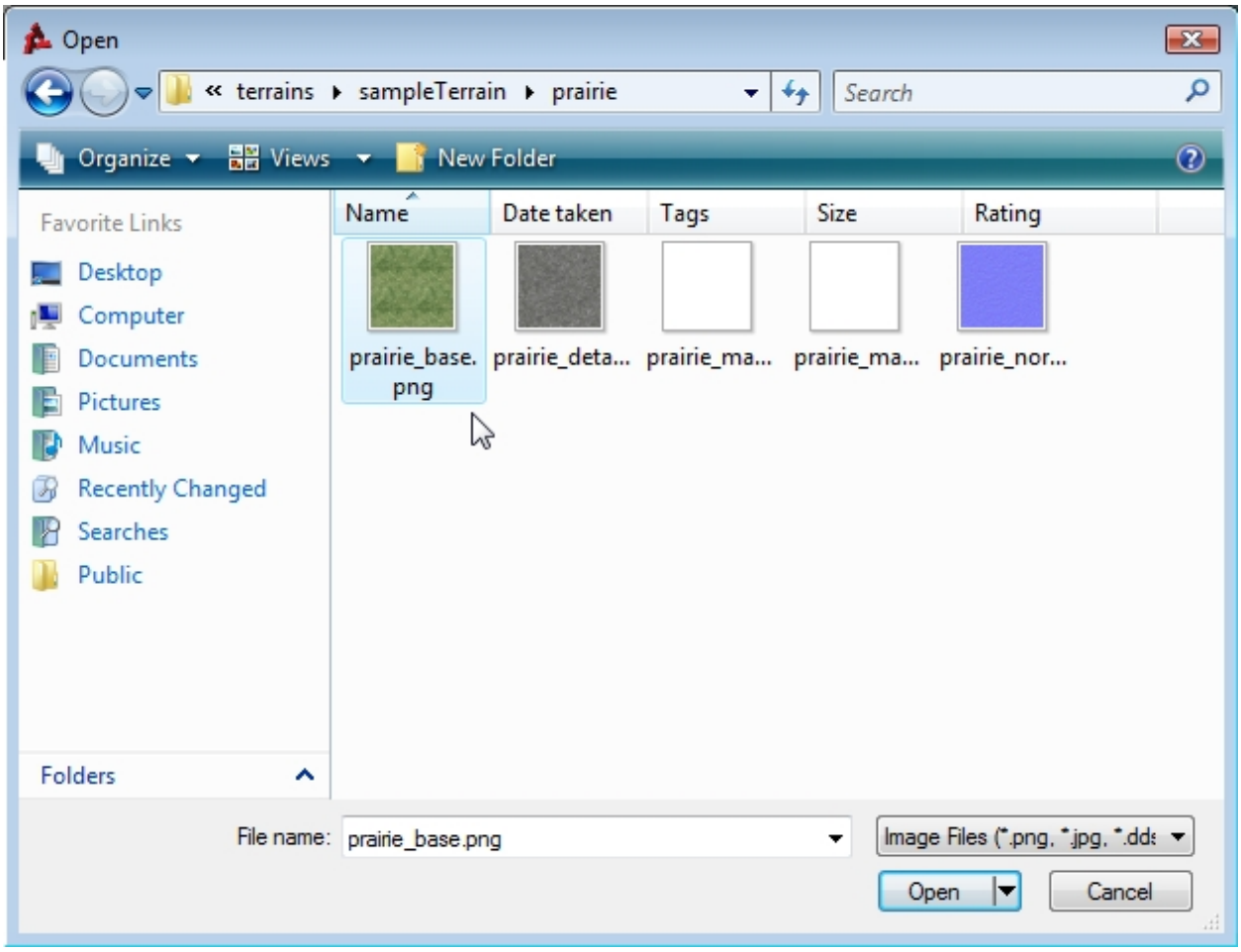


Do not worry if you do not see the detail. The mask is supposed to be solid white. Repeat the process until you have imported all your opacity layers.

Now that our opacity layers have been added, you should assign a material to each one. You can do so by clicking on one of the layers, then clicking the edit button in the bottom right. You will now see the Terrain Materials Editor.

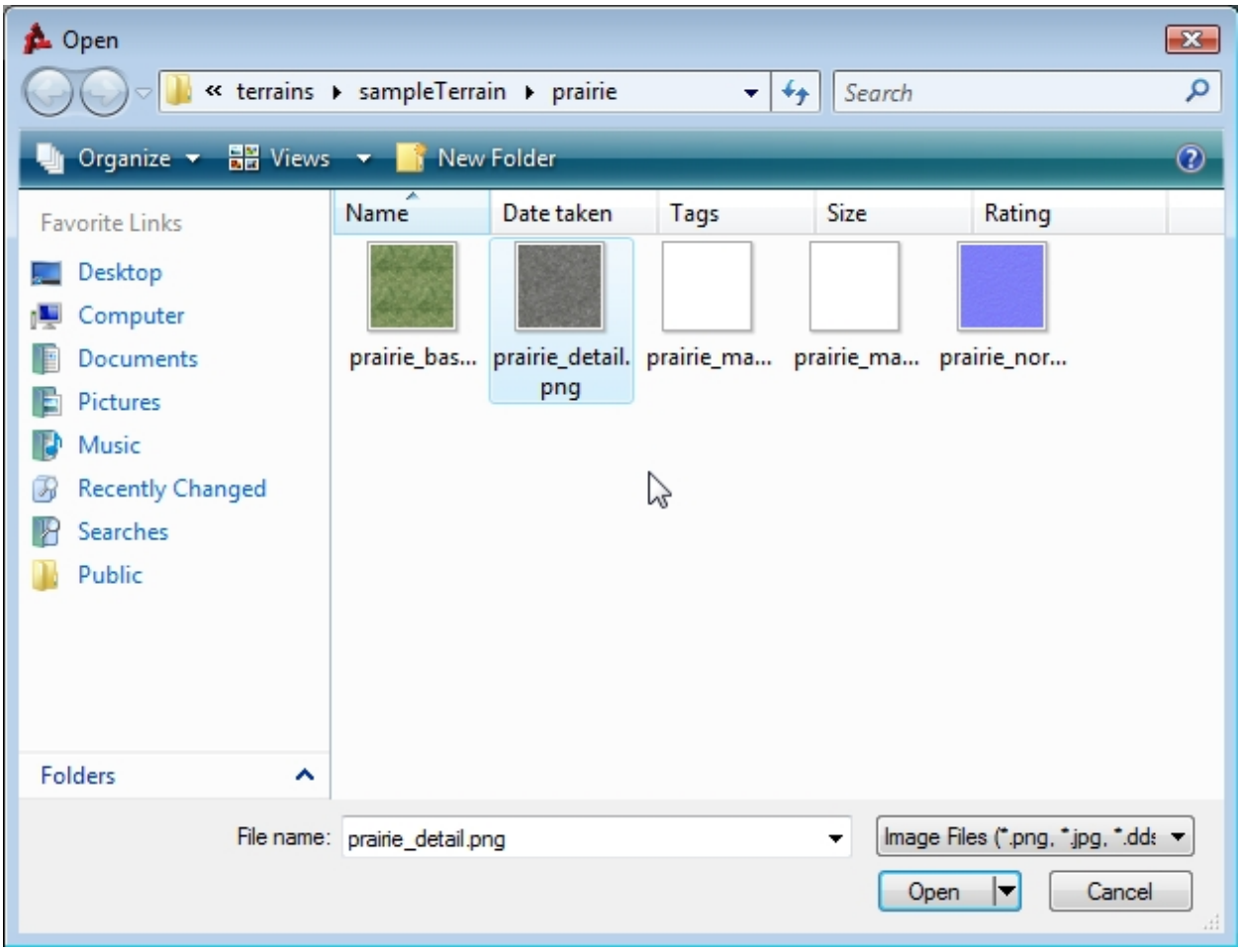


Click the New button, found at the top next to the garbage bin, to add a new material. Type in a name then click the Edit button next to the Diffuse preview box. Again, a file browser will pop up allowing you to open the base texture file for the material. Alternatively, you can click the preview box itself, which is a checkerboard image until you add a texture.

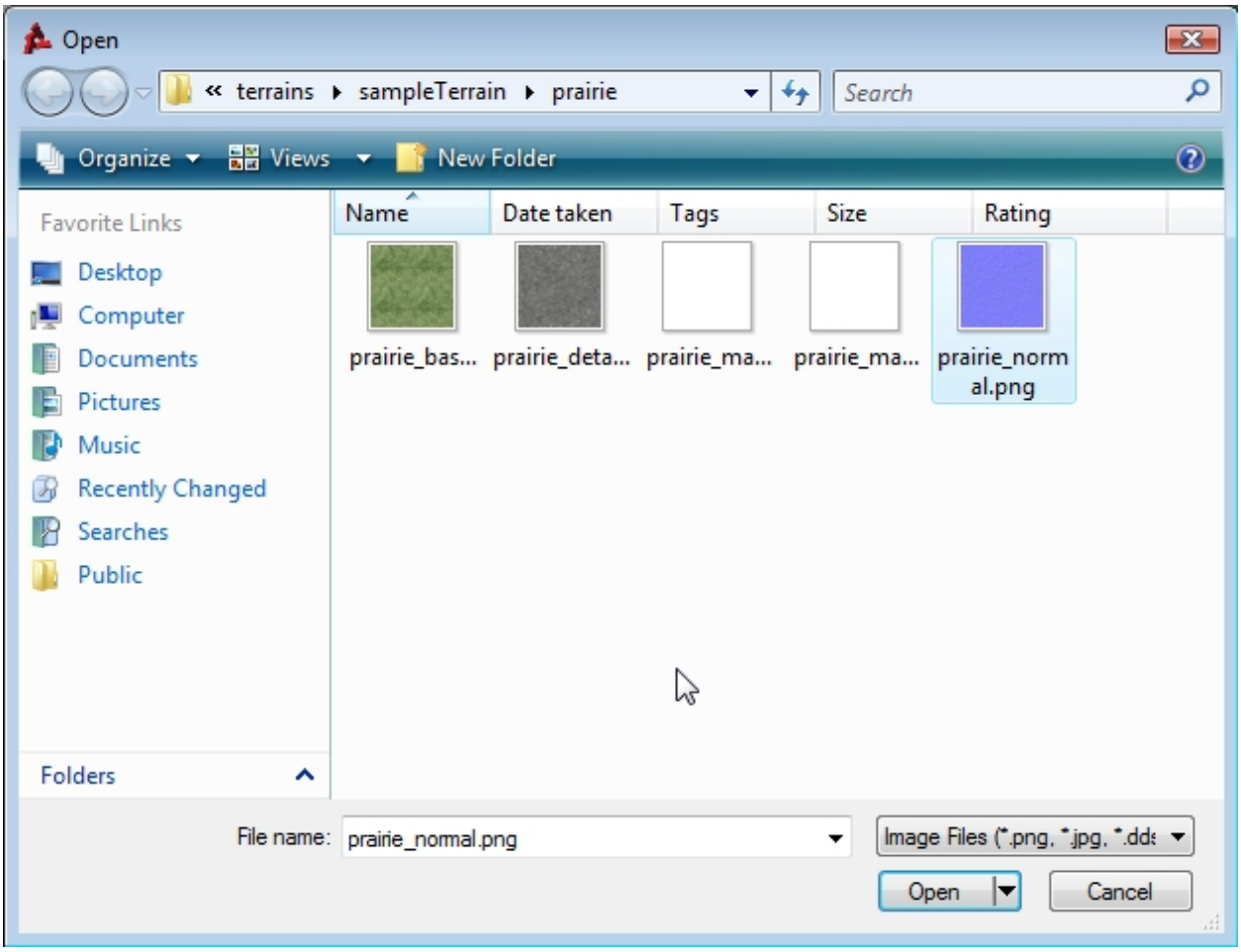


Once you have added the base texture, the preview box will update to show you what you opened. Set the Diffuse size which controls the physical size in meters of the base texture.

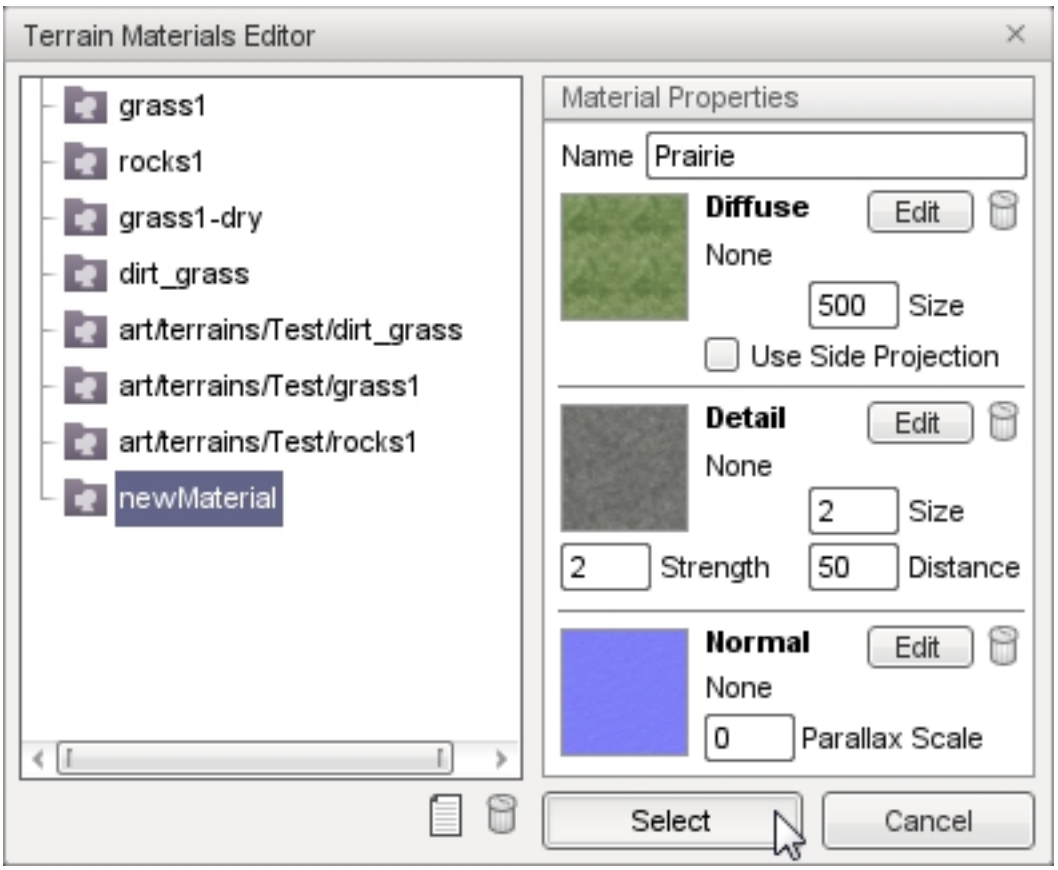
Click on the Edit button next to the Detail Preview box. Using the file browser, load the detail map.



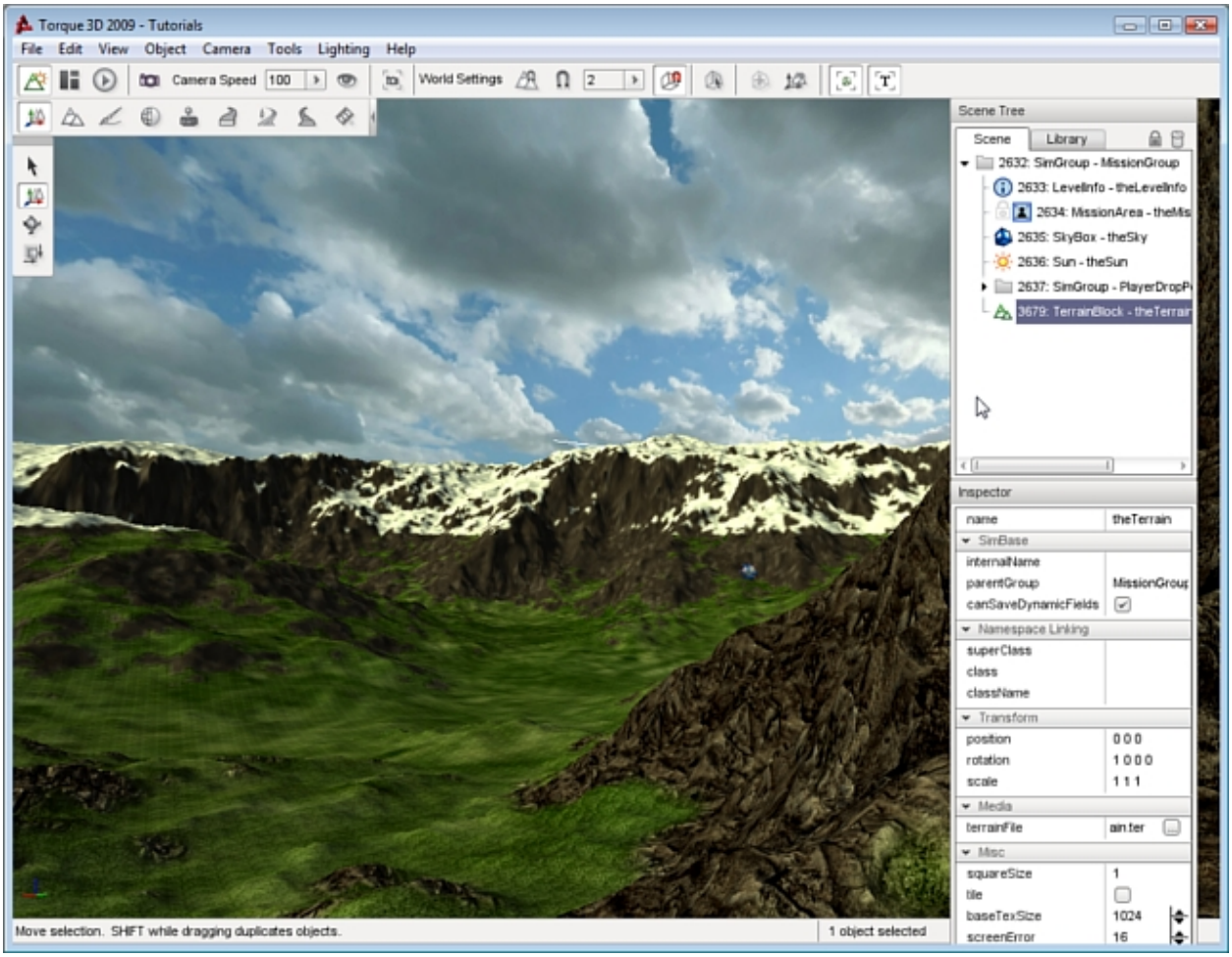
Next, click on the Edit button next to the Normal Preview box. Use the file browser to open the normal map.



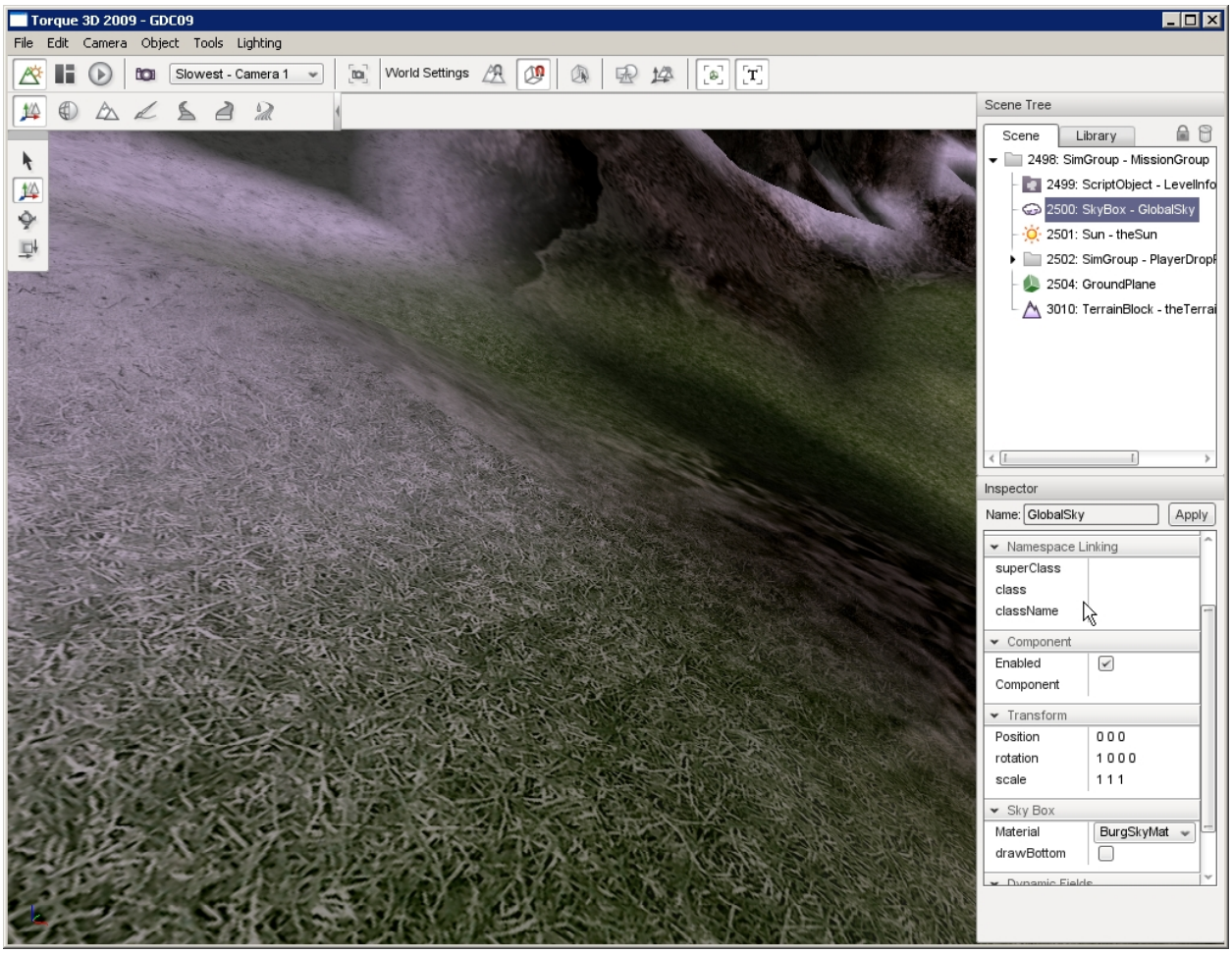
Your final material properties should look like the following:



Repeat this process until each opacity layer has a material assigned to it. Back in the Import Terrain Height Map dialog, click on the import button. It will take a few moments for Torque 3D to generate the terrain data from our various assets. When the import process is complete, the new Terrain Block will be added to your scene (you might need to move your camera back to see it).



If you zoom in close to where materials overlap, you can notice the high quality detail and smooth blending that occurs.



Terrain Block Properties

A Terrain Block has properties which can be set like any other object using the Object Editor. Clicking a Terrain Block in the scene or selecting it from the Scene Tree will update the Inspector pane with information about it. Terrain Blocks have their own unique set of properties.

Inspector

name TypeName. Optional global name of this object.

id TypeCaseString. SimObjectId of this object. Read Only.

Source Class TypeCaseString. Source code class of this object. Read Only.

Transform

position MatrixPosition. Object world position.

rotation MatrixOrientation. Object world orientation.

Media

terrainFile TypeStringFilename. The source terrain data file.

Misc

castShadows TypeBool. Allows the terraint to cast shadows onto itself and other objects.

squareSize TypeF32. Indicates the spacing between points on the XY plane on the terrain.

baseTexSize TypeS32. Size of base texture size per meter.

lightMapSize TypeS32. Lightmap dimensions in pixels.

screenError TypeS32. Not yet implemented.

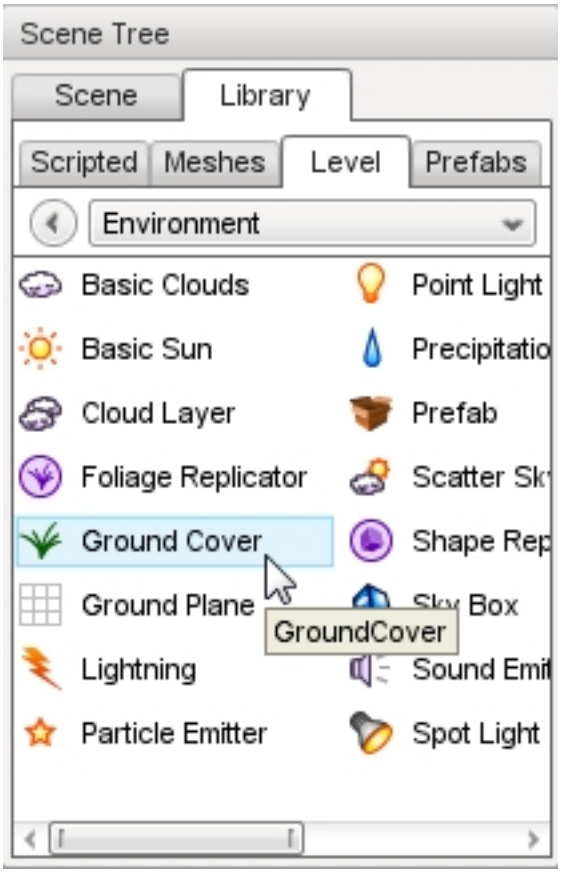
2.3.3 Ground Cover

The Ground Cover system allows you to spread many objects throughout your entire level. This object makes use of the Terrain Material system, applying textures or 3D objects on a per-layer basis. The most practical uses of Ground Cover include:

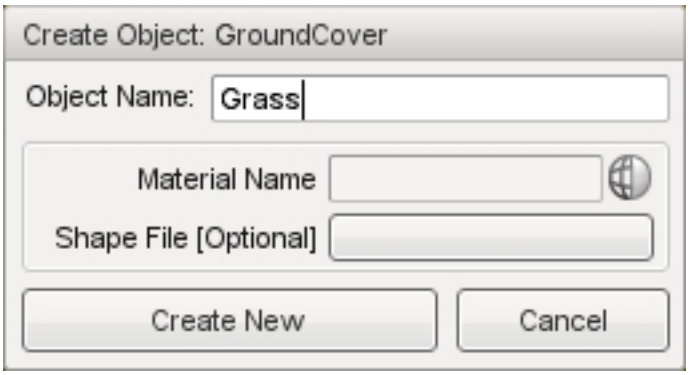
- Creating large fields of foliage (grass, wheat, etc.)
- Automatic placement of shapes and environmental textures on specific terrain types.
- Providing another layer of environmental realism when combined with Forest Editor and Replicators.

Adding Ground Cover

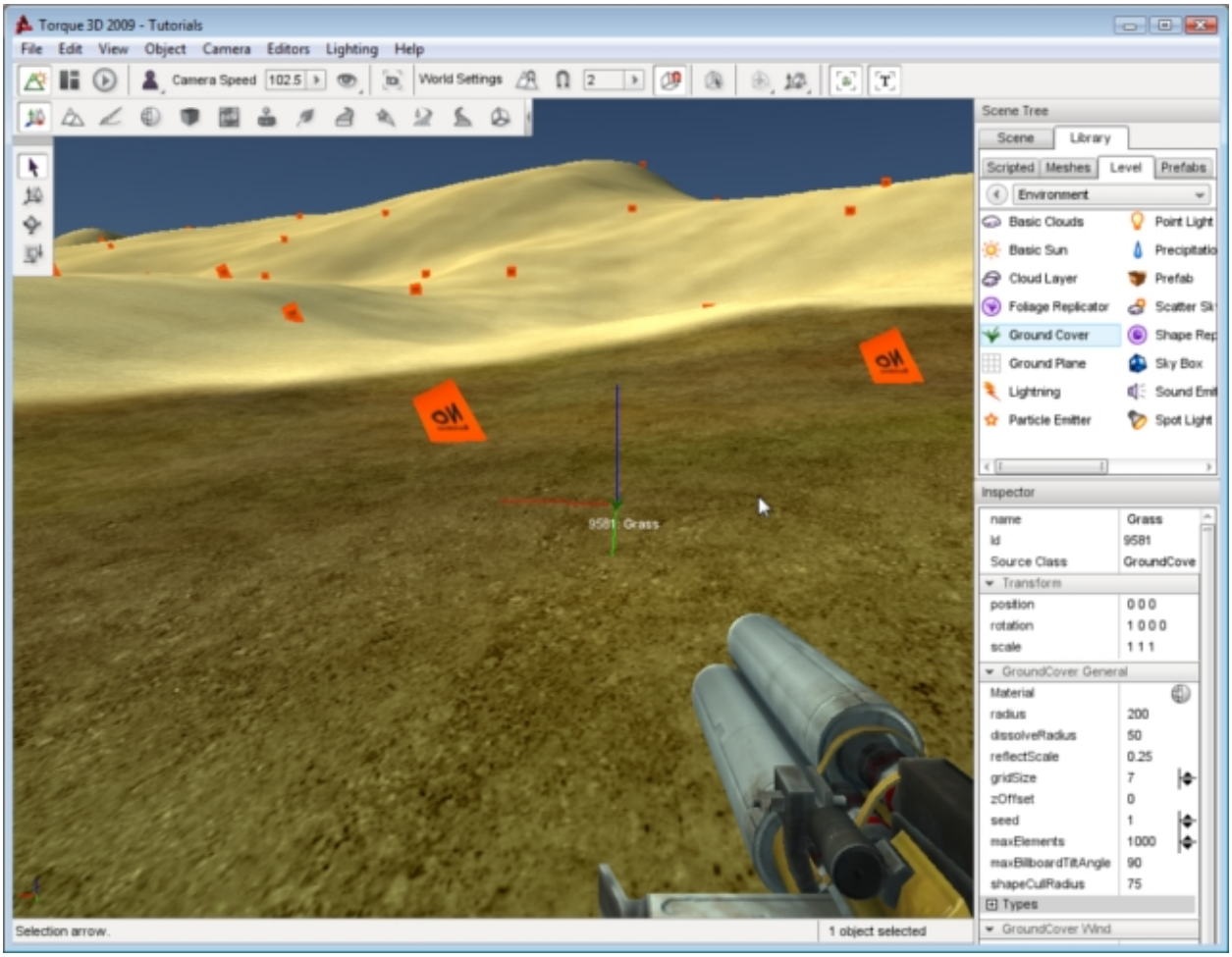
To add a ground Cover object to a level, select the Library tab in the Scene Tree panel. Click on the Level tab and double-click the Environment folder. Locate the Ground Cover entry.



Double-click the Ground Cover entry. The Create Object dialog will appear.



If you already have a material or shape you want to use, you can set them here. Materials are used to paint the ground with textures which can contain transparency so that the underlying ground shows through. A Shape File is used to replicate 3D objects on the ground. Enter a name for you Ground Cover object then click the Create New button. A new Ground Cover object will be added to your level. Without a material, the system will render a pattern on the ground with the default "No Material" texture:



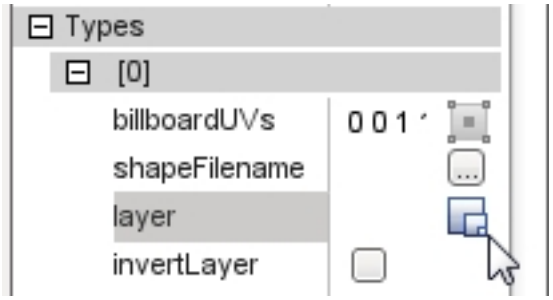
To change the No Material indicators to a real Material scroll through the Ground Cover properties until you get to the Ground Cover General section. In the Material field, click on the globe to open the Material Selector:

Fig. 2.19: Material Field

When the Material Selector appears, you have the option to pick an existing material or create a new one in the Material Editor.

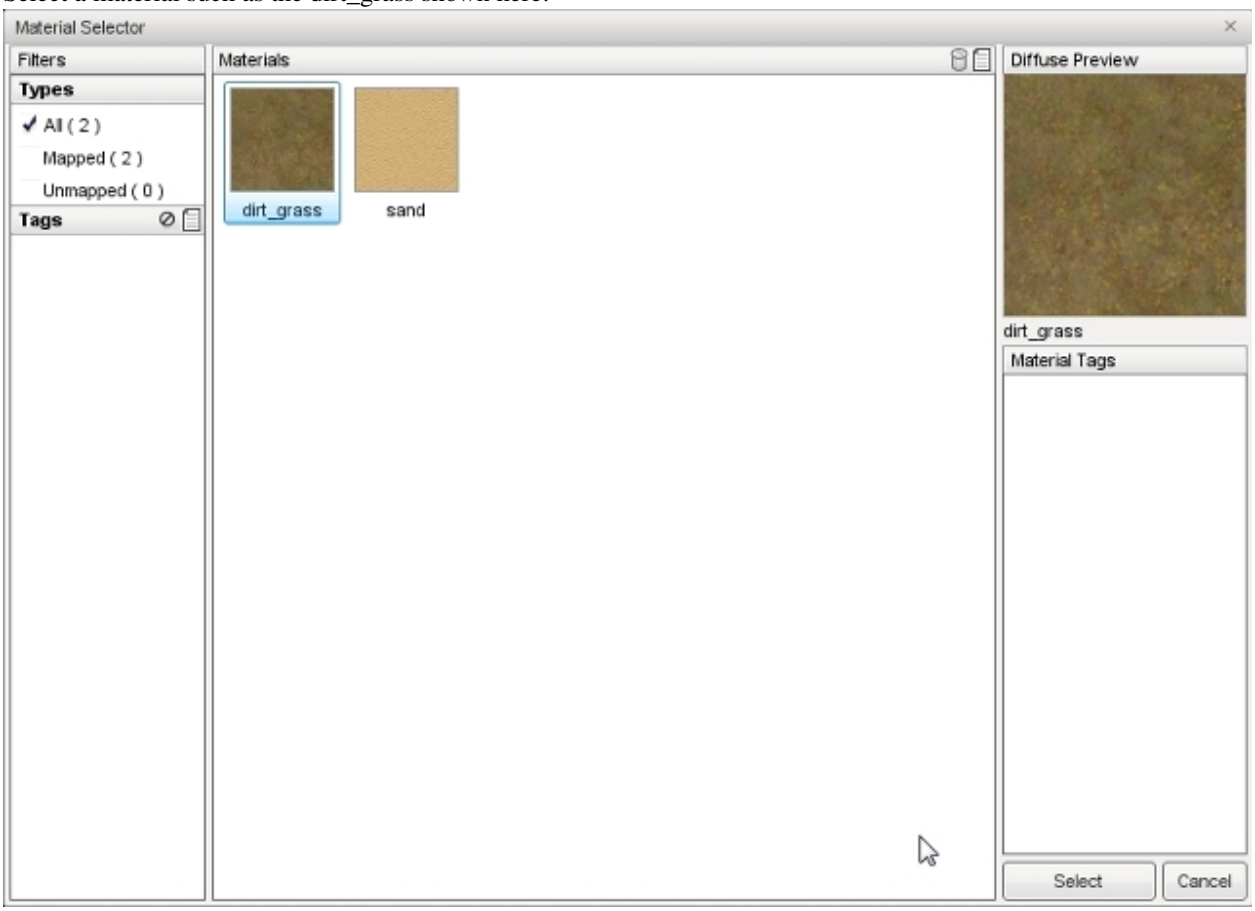
Assigning Terrain Material

When you first add a Ground Cover object, it will place the material or shape on the entire terrain. To limit the placement of Ground Cover, you must set the terrain layer. To set the terrain layer with the Ground Cover selected, scroll down to the Ground Cover General set of fields. Find the Types sub-section.

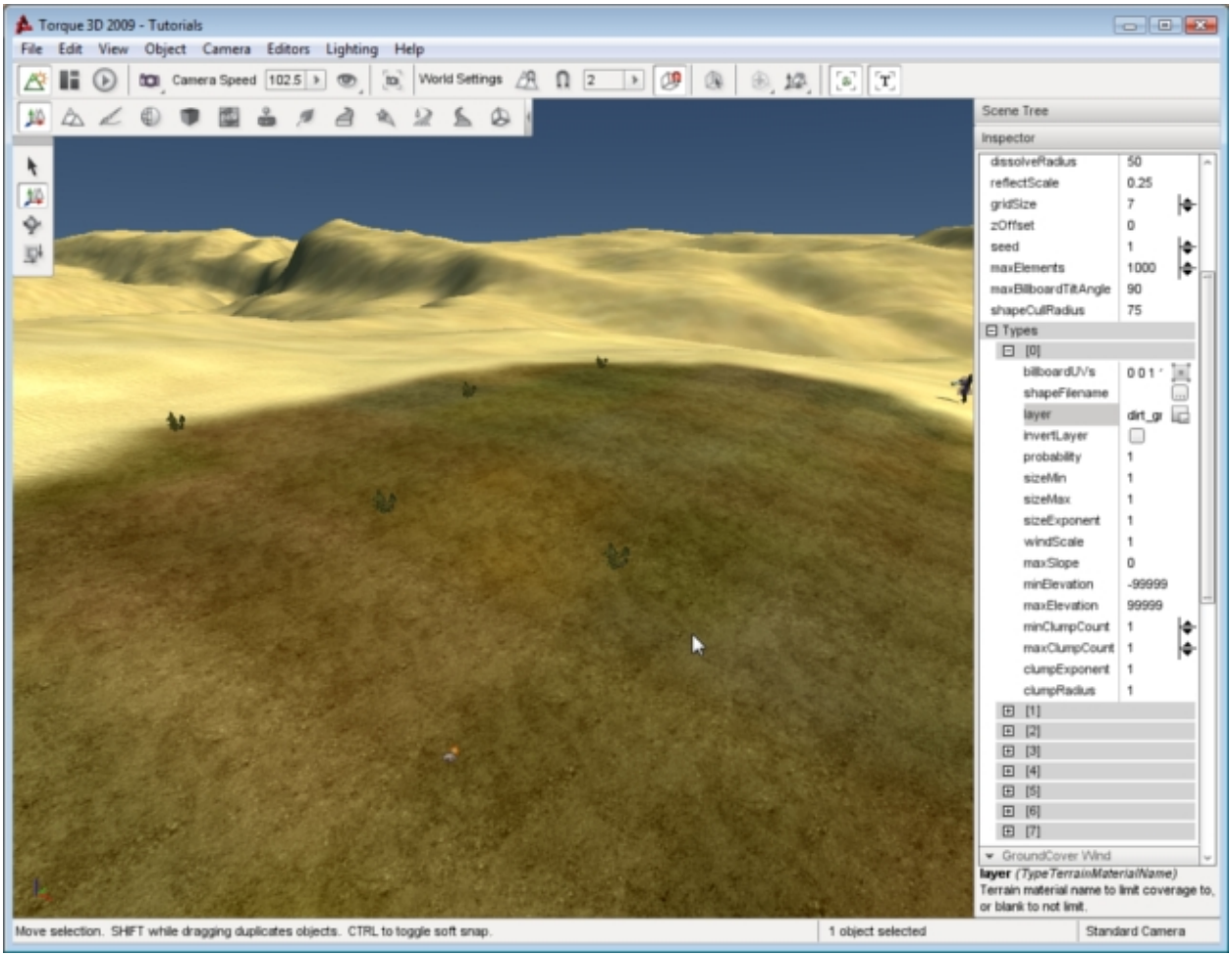


Types is an array with each entry controlling a section of the Ground Cover. If this is confusing, think of it like this as follows. The Ground Cover is a single object that is covering the entire terrain. The object itself is comprised of eight sections, Types[0] through Types[7]. Each section can be told what, where, and how to render a material or shape. You can feasibly have the Ground Cover object rendering simultaneously on eight different terrain layers.

With the above information in mind, you can assign the Ground Cover to terrain materials. Scroll through the properties until you get to Types[0]. Click on the box icon in the layer field. The Material Selector for terrains should appear. Select a material such as the dirt_grass shown here:



After you click the Select button, the Ground Cover will stop placing billboards on the entire terrain. It should now only be placing the foliage on the specific terrain layer you chose.



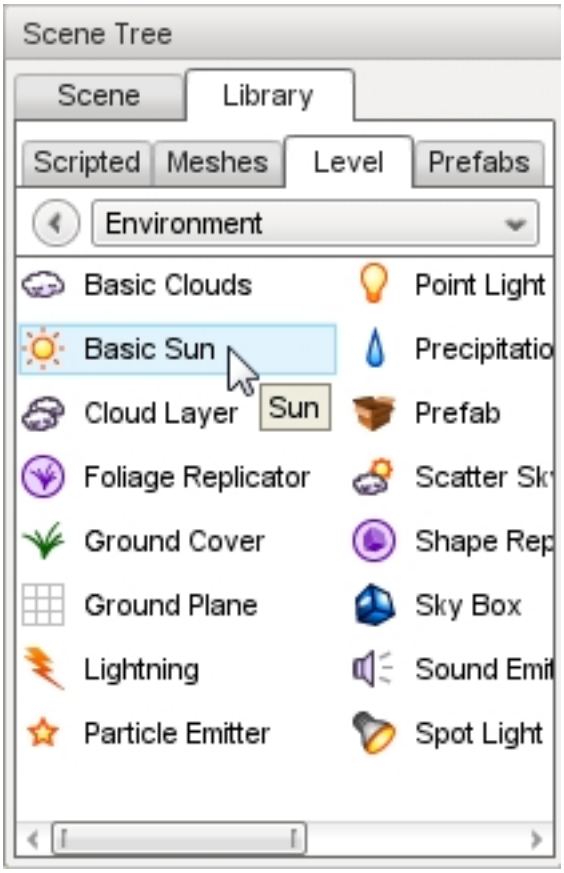
If you are having a difficult seeing this change, locate the maxElements field and increase the value dramatically:

2.3.4 Basic Sun

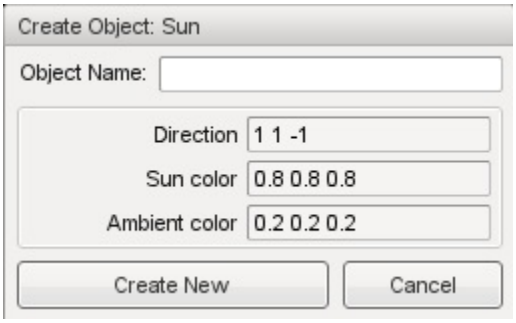
Proper global lighting can dramatically change the appearance of a level, as well as aid in certain game play aspects, such as providing a day and night cycle. The Sun object is used to control the global lighting settings in the level. The main settings include ambient coloring, azimuth, and elevation. The lighting effects produced by the Sun are 100% dynamic, which means as soon as you change a setting it is immediately reflected in the level. In short, the Sun lets you control the day/night cycle of your level.

Adding a Basic Sun

To add a Sun start by opening the Library tab in the Scene Tree dialog. Once the Library tab is active, click on Level sub-tab, then double-click the Environment subcategory.



Double-clicking the Basic Sun object will open the Create Object Dialog. From here, you can change a few basic properties, such as the name, starting color, and location. Enter a name then click the Create New button.



Basic Sun Properties

Additional properties can be changed with the Inspector pane. To change the Sun properties using the Inspector Pane click the Scene tab, then click the name of your new sun object. The Inspector pane will update to display the current properties of your new sun.

Inspector

name TypeName. Optional global name of this object.

id TypeCaseString. SimObjectId of this object. Read Only.

Source Class TypeCaseString. Source code class of this object. Read Only.

Transform

position MatrixPosition. Object world position.

rotation MatrixOrientation. Object world orientation.

scale Point3F. Object world scale.

Transform

azimuth TypeF32. The horizontal angle of the sun measured clockwise from the positive Y world axis.

elevation TypeF32. The elevation angle of the sun above or below the horizon.

Lighting

color TypeColorF. Color shading applied to surfaces in direct contact with light source.

ambient TypeColorF. Color shading applied to surfaces not in direct contact with light source, such as in the shadows or interiors.

brightness TypeF32. Adjust the global Sun contrast/intensity.

castShadows TypeBool. Enables/disables shadows cast by objects due to Sun light.

Corona

coronaEnabled TypeBool. Enable or disable rendering of the corona sprite.

coronaMaterial TypeMaterialName. Material for the corona sprite.

coronaScale TypeF32. Scale the rendered size of the corona (texture size * coronaScale = visible pixel dimensions).

coronaTint TypeColorF. Modulates the corona sprite color (if coronaUseLightColor is false).

coronaUseLightColor TypeBool. Modulate the corona sprite color by the color of the light (overrides coronaTint).

Misc

flareType TypeLightFlareDataPtr. Datablock for the flare and corona produced by the Sun.

flareScale TypeF32. Changes the size and intensity of the flare.

Advanced Lighting

attenuationRatio TypePoint3F. The proportions of constant, linear, and quadratic attenuation to use for the falloff for point and spot lights.

shadowType TypeEnum. The type of shadow to use on this light.

cookie TypeStringFilename. A custom pattern texture which is projected from the light.

texSize TypeS32. The texture size of the shadow map.

overDarkFactor TypePoint4F. The ESM shadow darkening factor.

shadowDistance TypeF32. The distance from the camera to extend the PSSM shadow.

shadowSoftness TypeF32. Adjusts shadow edge clarity.

numSplits TypeF32. The logrithmic PSSM split distance factor.

logWeight TypeF32. The logrithmic PSSM split distance factor.

fadeStartDistance TypeF32. Start fading shadows out at this distance. 0 equates to auto calculate this distance.

lastSplitTerrainOnly TypeBool. This toggles only terrain being rendered to the last split of a PSSM shadow map.

Advanced Lighting Lightmap

representedInLightmap TypeBool. This light is represented in lightmaps (static light, default: false).

shadowDarkenColor TypeColorF. The color that should be used to multiply-blend dynamic shadows onto lightmapped geometry (ignored if representedInLightmap is false).

includeLightmappedGeometryInShadow TypeBool. This light should render lightmapped geometry during its shadow-map update (ignored if representedInLightmap is false).

Azimuth and Elevation

The Azimuth and Elevation fields are very important to determining the global position of the sun, which affects the lighting intensity and shadow casting for every object in your level. You cannot think of these two fields as numbers that simply move your sun or make it higher. Azimuth and Elevation are actually angles:

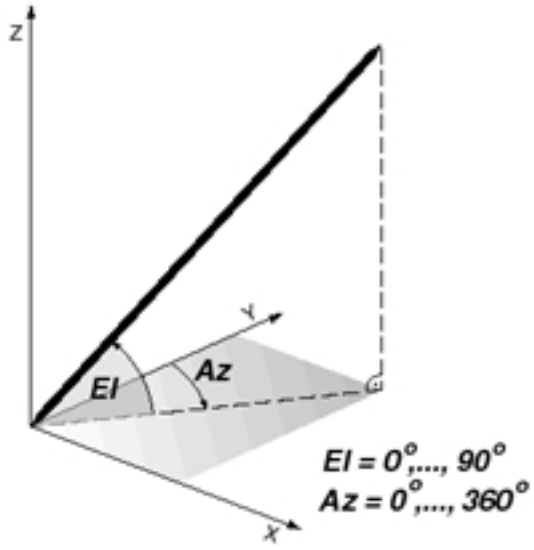


Fig. 2.20: Azimuth

Elevation (El) is measured between 0 and 180 degrees. It refers to the vertical angle measured from the geometric horizon (0°) towards the zenith (+90°).

- 0° will place the Sun at one end of the horizon as though it were just about to rise or set.

- 90° will place the Sun directly over the level, shining straight down.
- 180° will place the Sun at the opposite end of the horizon as though it were just about to rise or set.

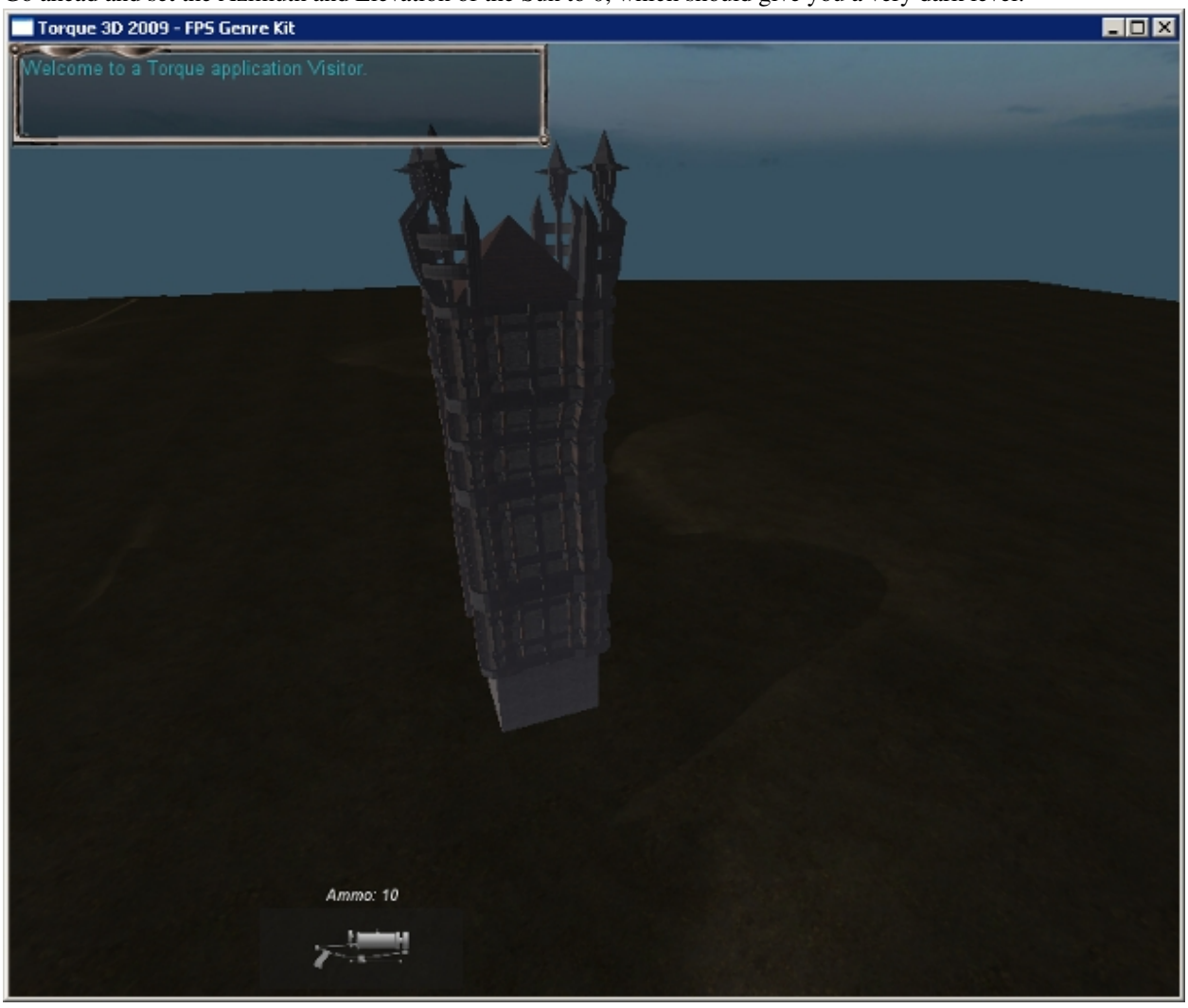
Azimuth ranges between 0 and 360 degrees, and refers to a horizontal angle which determines the direction the Sun is facing in the level.

- 0° is true North.
- 90° is due east.
- 180° is due south.
- 270 is due west.

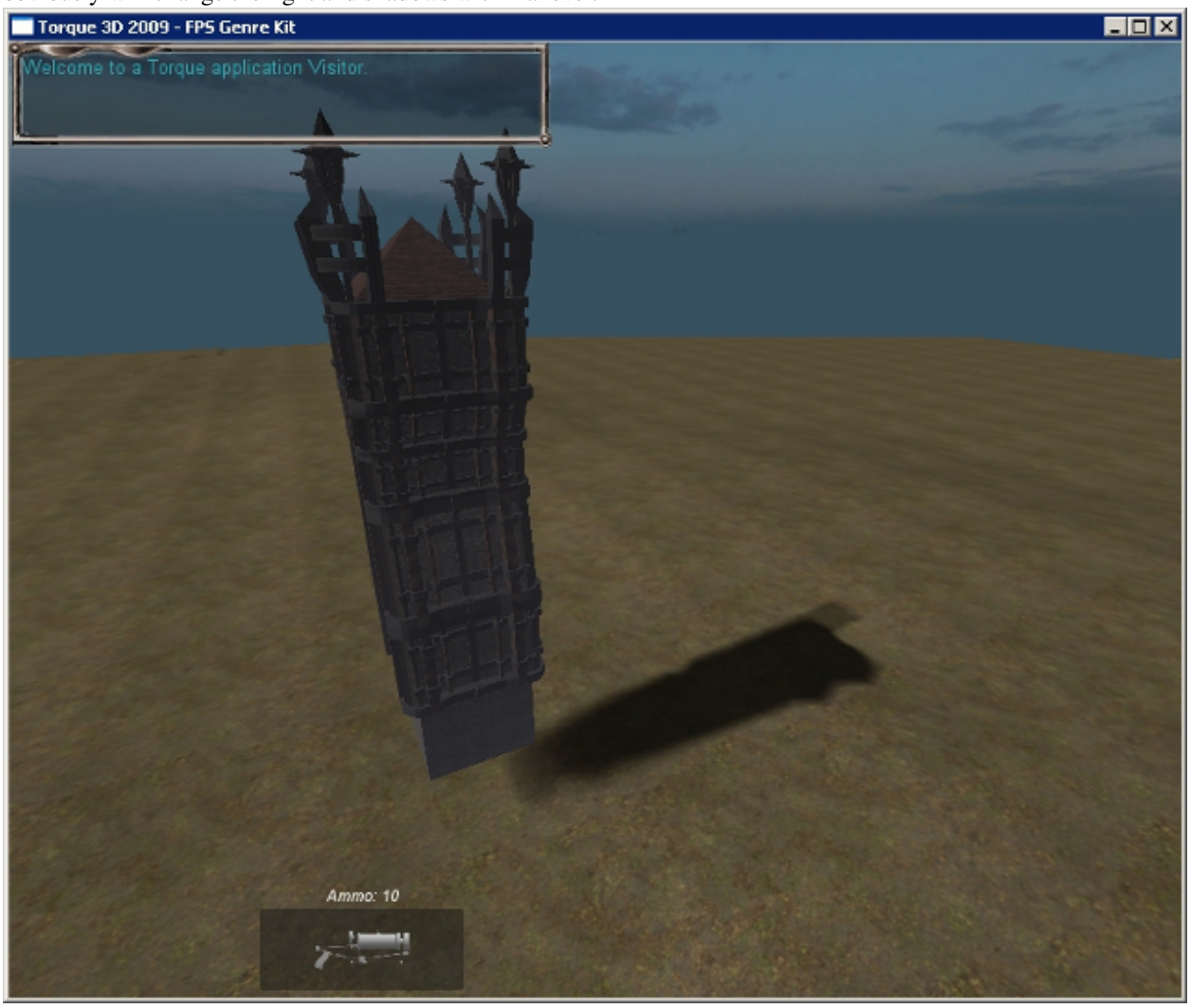
If you have a completely flat terrain with no objects, it will be difficult for you to visually measure the position of the Sun. You can use any object you want as a reference, but make sure you have your camera fixed on it to see the changes that you are making.

Adjusting Elevation

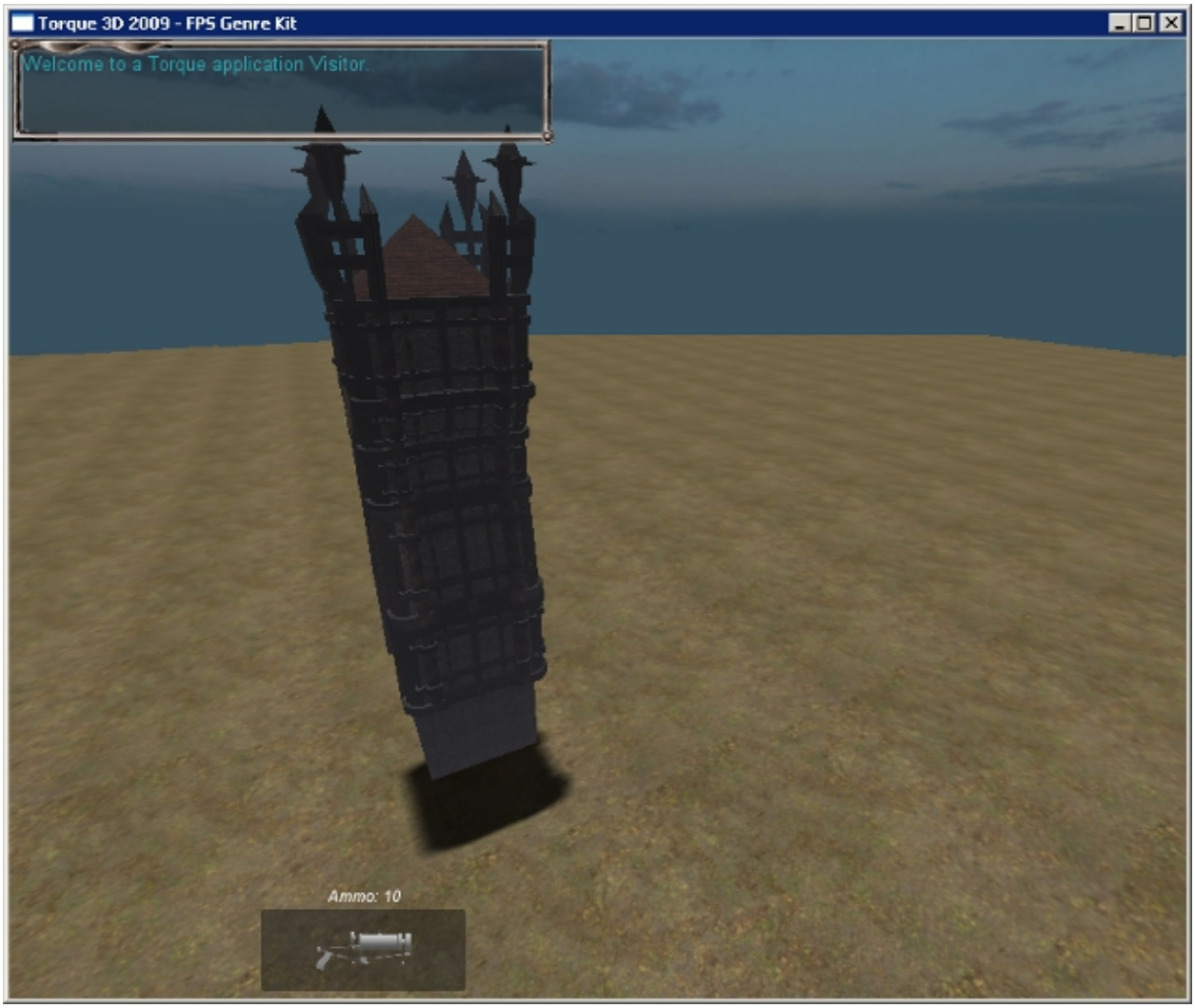
Go ahead and set the Azimuth and Elevation of the Sun to 0, which should give you a very dark level.



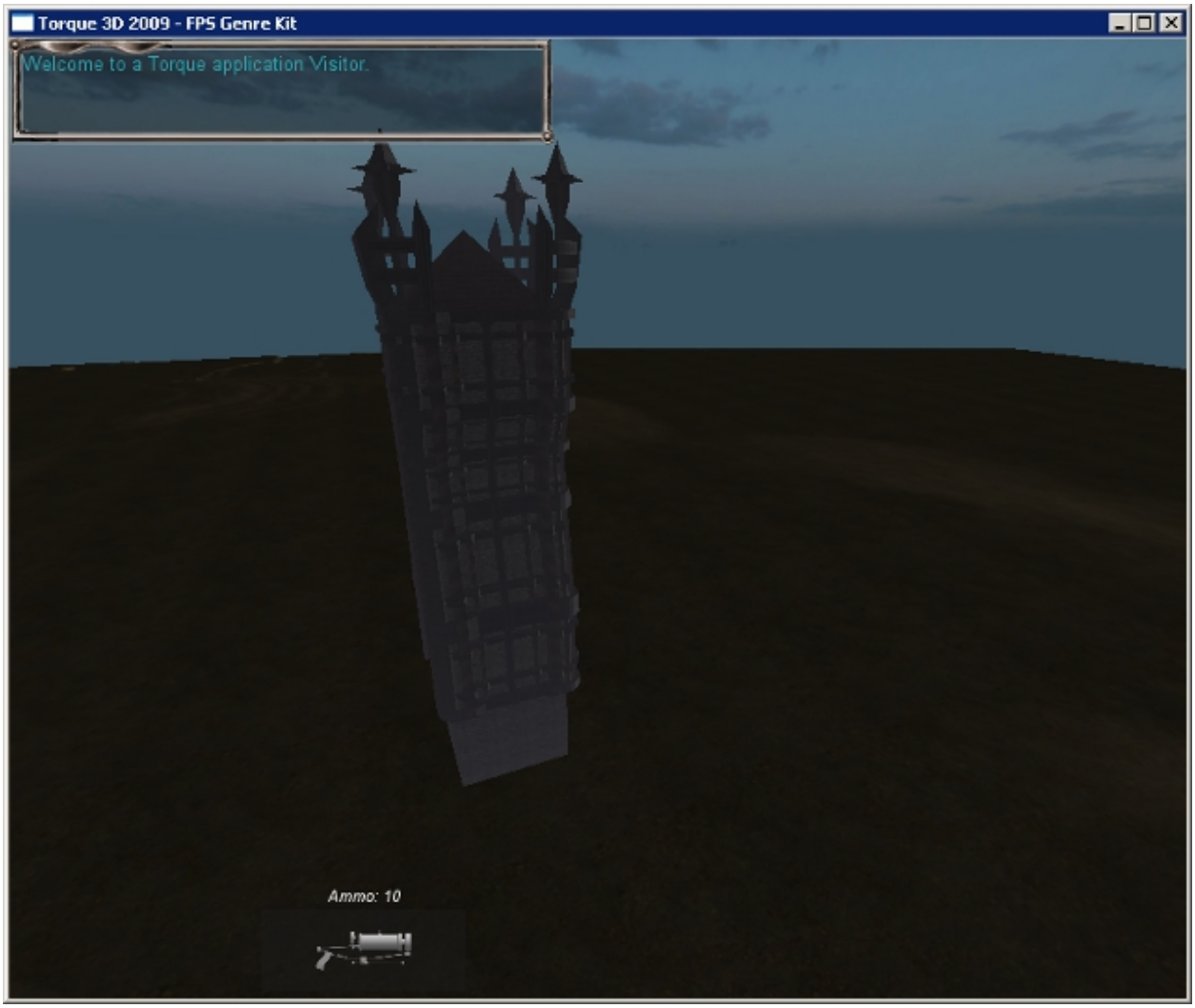
At this point, the angle of the Sun matches the horizon of your level perfectly. By increasing the elevation to 45, and you will see the objects in your level begin to cast short shadows. If you dont see the shadows changing make sure that you do not have more than one sun in your scene. The World Builder allows more than one sun in a scene, which obviously will change the light and shadows within a level.



Thinking back to angles, if 0° is parallel with the horizon, then 90° degrees will be directly overhead. Change the elevation to 90. You will see all of the shadows for the objects are directly below, just as in real life when the sun is sitting at zenith (straight overhead).



Setting the elevation to 180 will place the Sun at the opposite end of the horizon, once again resulting in a dark level. If you really focus, there is a slight change in shadow direction than when the elevation was 0°.



Adjusting Azimuth

The Azimuth of the Sun is measured clockwise from a fixed overhead perspective. To help you understand this rotation, we are going to adjust the Aazimuth of the Sun so that shadows of an object rotate like a sun dial or hands on a clock.

If you set the elevation to 45 and azimuth to 0, it will look like the shadow is pointing at 12 o'clock (if viewed from overhead).

Now, increase the azimuth by 45. At a sharp 45° angle, the shadow looks like it is pointing at 1 o'clock.

If you set the azimuth property to 90, you will notice a very familiar angle. The object and its shadow are forming a perfect right angle.

Half of a full rotation is 180°. After Setting the azimuth to this value, the shadow will now be pointing in the opposite direction from its original state.

Now, set the azimuth property to 270 and watch as the shadow points to 9 O'clock. The shadow should be pointing directly opposite from the 90° setting.

Finally, set the azimuth to 360. We have achieved full rotation. Careful examination will show that even though your shadows are pointing in the same direction as the 0° setting, they have been flipped.

Standard Lighting

The last topic we are going to touch on that is specific to the Sun object is standard lighting. Under the Lighting properties of the sun object there are three variables to adjust. Checking the castShadows box will cause surfaces to project shadows based on the direction of the sunlight. Removing the check will disable any shadows cast due to the Sun. If you uncheck that box you'll see that the shadows you have been observing will not be displayed at all.

In addition to creating shadows, the light from the Sun will also affect the color shading of all surfaces in the level. There is a subtle, yet important difference between the color and ambient fields. If you want realistic lighting color, you will need to tweak both values.

The value of the color field will shade all surfaces that are in direct contact with the sunlight. A completely black color will make it seem like there is no light at all. Using the color picker to choose an orange hue will result in a sunset appearance for your level.

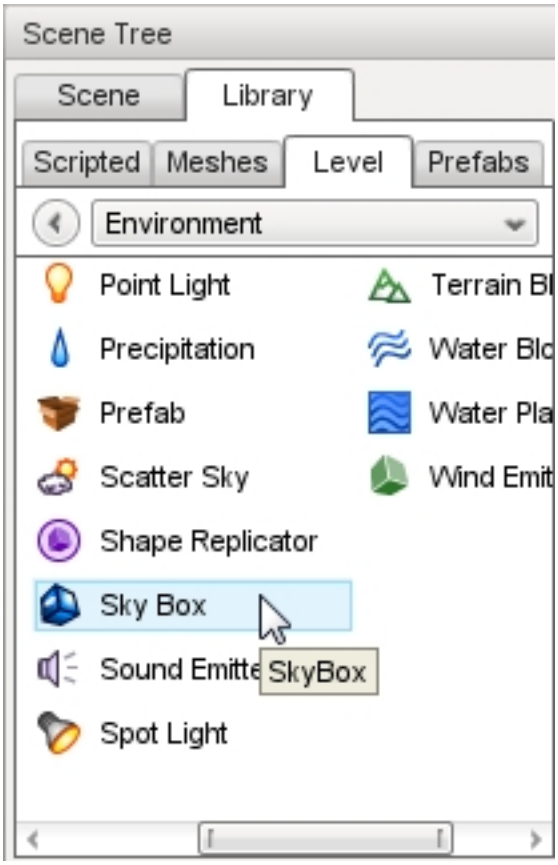
Ambient light is the available light in a space, whether from natural or mechanical sources. It is applied to everything in the world and also contributes to the direct lighting of the sun. The ambient field will lighten dark shadows and brighten well lit surfaces based on the color value.

2.3.5 Sky Box

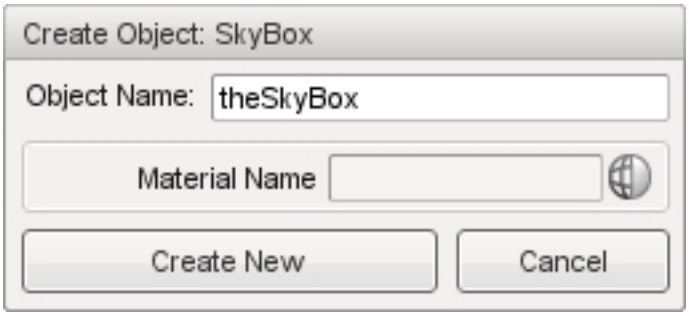
Torque 3D uses a cubemap to produce the appearance of a sky in a level. The cubemap method will allow you to create realistic looking skies that provide a sense of depth to your mission.

Adding a Sky Box

To add a new Sky Box to a level change to the Library tab in the Scene Tree panel and double-click the Environment folder, locate the Sky Box entry then double-click it.



The Object Name is what you want your Sky Box to be called and will be displayed in the Scene Tree after it is created. The Material box allows you to select the starting material to use when creating the object.



Sky Box Properties

Additional properties can be changed with the Inspector pane. To change the Skybox properties using the Inspector Pane, click the Scene tab, then click the name of your new Sky Box object. The Inspector pane will update to display the current properties of your new Sky Box.

Inspector

name TypeName. Optional global name of this object.

id TypeCaseString. SimObjectId of this object. Read Only.

Source Class TypeCaseString. Source code class of this object. Read Only.

Transform

position MatrixPosition. Object world position.

rotation MatrixOrientation. Object world orientation.

scale Point3F. Object world scale.

Sky Box

material TypeMaterialName. The name of a cubemap material for the sky box.

drawBottom TypeBool. If false the bottom of the skybox is not rendered.

fogBandHeight TypeF32. The height (0-1) of the fog band from the horizon to the top of the SkyBox.

2.3.6 Scatter Sky

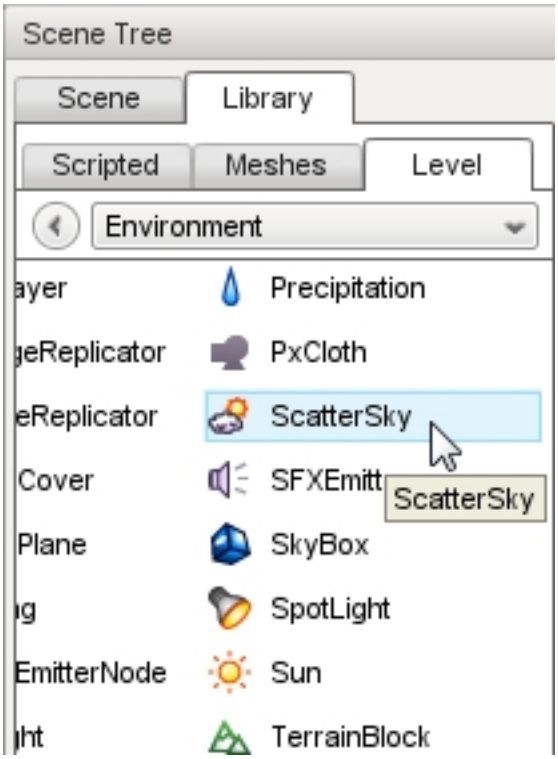
Torque 3D includes an object called a Scatter Sky which uses a dynamic sky coloring system to create more vibrant varying skies than the simple Skybox object.

As the name implies, the Scatter Sky object produces the sky. Additionally, it includes level lighting, sun positioning, and a hook for time of day manipulation. This can be used for a fully functioning day/night system.

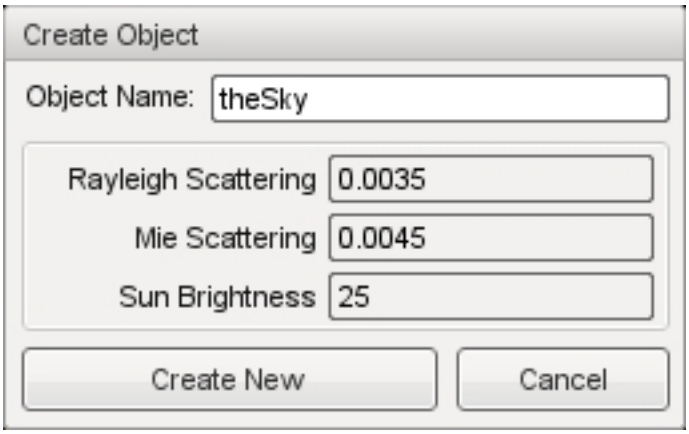
Adding a Scatter Sky

Every new mission starts with both Sky Box and Sun objects. Since the Scatter Sky object contains the functionality of those two objects embedded within it they must be removed in order to use a Scatter Sky in the level.

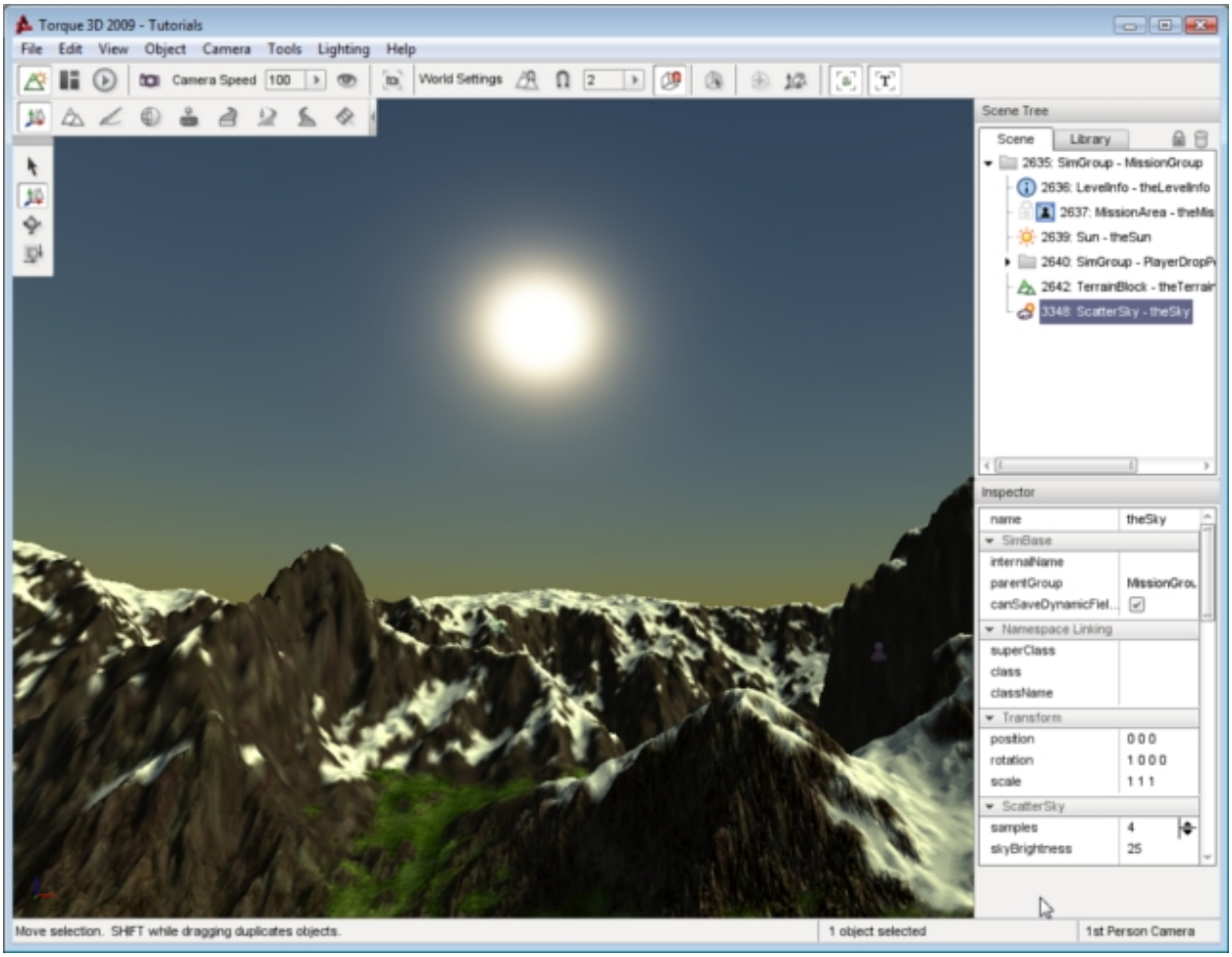
To create a new Scatter Sky, change to the Library tab in the Scene Tree panel. Click on the Level tab and select the Level folder. Locate the Scatter Sky entry and double-click it.



The Create Object dialog will appear. The Object Name is what you want your Scatter Sky to be named. It will appear in the Mission Group of the Scene Tree. Enter theSky as the name, leave the rest of the values at their defaults, then click Create New.



A new Scatter Sky object will be created and automatically added to your level. Therefore, it will once again have a sky. Since the Scatter Sky supplies a sun, the level should now be lit.



Scatter Sky Properties

Additional properties can be changed with the Inspector pane. To change the Scatter Sky properties using the Inspector Pane, click the Scene tab. Then click the name of your new Scatter Sky object. The Inspector pane will update to display the current properties of your new Scatter Sky.

Inspector

name TypeName. Optional global name of this object.

id TypeCaseString. SimObjectId of this object. Read Only.

Source Class TypeCaseString. Source code class of this object. Read Only.

Transform

position MatrixPosition. Object world position.

rotation MatrixOrientation. Object world orientation.

scale Point3F. Object world scale.

Scatter Sky

skyBrightness TypeF32. Global brightness and intensity applied to the sky and objects in the level.

sunSize TypeF32. Affects the size of the sun's disc.

colorizeAmount TypeF32. Controls how much the the alpha component of colorize brigthens the sky. Setting to 0 returns default behavior.

colorize TypeColorF. Tints the sky the color specified, the alpha controls the brigthness. The brightness is multipled by the value of colorizeAmount.

rayleighScattering TypeF32. Controls how blue the atmosphere is during the day.

sunScale TypeColorF. The color shading applied to objects in direct sun light.

ambientScale TypeColorF. The color shading applied to objects not in direct sun light, such as in the shadows.

fogScale TypeColorF. Modulates the fog color. Note that this overrides the LevelInfo.fogColor property, so you should not use LevelInfo.fogColor if the level contains a ScatterSky object.

exposure TypeF32. Controls the contrast of the sky and sun.

Orbit

azimuth TypeF32. The horizontal angle of the sun measured clockwise from the positive Y world axis.

elevation TypeF32. The elevation angle of the sun above or below the horizon.

moonAzimuth TypeF32. The horizontal angle of the moon measured clockwise from the positive Y world axis.

moonElevation TypeF32. The elevation angle of the moon above or below the horizon.

Lighting

castShadows TypeBool. Enables/disables shadows cast by objects due to Sun light.

brightness TypeF32. Adjust the global Sun contrast/intensity.

Misc

flareType TypeLightFlareDataPtr. Datablock for the flare and corona produced by the Sun.

flareScale TypeF32. Changes the size and intensity of the flare.

Night

nightColor TypeColorF. The ambient color during night. Also used for the sky color if useNightCubemap is false.

nightFogColor TypeColorF. Color shading of fog present during night scenes.

moonEnabled TypeBool. Toggles rendering of moon image during night.

moonMat TypeMaterialName. Material for the moon sprite.

moonScale TypeF32. Controls size the moon sprite renders, specified as a fractional amount of the screen height.

moonLightColor TypeColorF. Color of light cast by the directional light during night.

useNightCubemap TypeBool. Toggles rendering of star cubemap during night scenes, similar to Sky Box.

nightCubemap TypeCubemapName. Cube map used to render stars in the sky during night scene.

Advanced Lighting

attenuationRatio TypePoint3F. The proportions of constant, linear, and quadratic attenuation to use for the falloff for point and spot lights.

shadowType TypeEnum. The type of shadow to use on this light.

cookie TypeStringFilename. A custom pattern texture which is projected from the light.

texSize TypeS32. The texture size of the shadow map.

overDarkFactor TypePoint4F. The ESM shadow darkening factor.

shadowDistance TypeF32. The distance from the camera to extend the PSSM shadow.

shadowSoftness TypeF32. Adjusts shadow edge clarity.

numSplits TypeF32. The logrithmic PSSM split distance factor.

logWeight TypeF32. The logrithmic PSSM split distance factor.

fadeStartDistance TypeF32. Start fading shadows out at this distance. 0 equates to auto calculate this distance.

lastSplitTerrainOnly TypeBool. This toggles only terrain being rendered to the last split of a PSSM shadow map.

Advanced Lighting Lightmap

representedInLightmap TypeBool. This light is represented in lightmaps (static light, default: false).

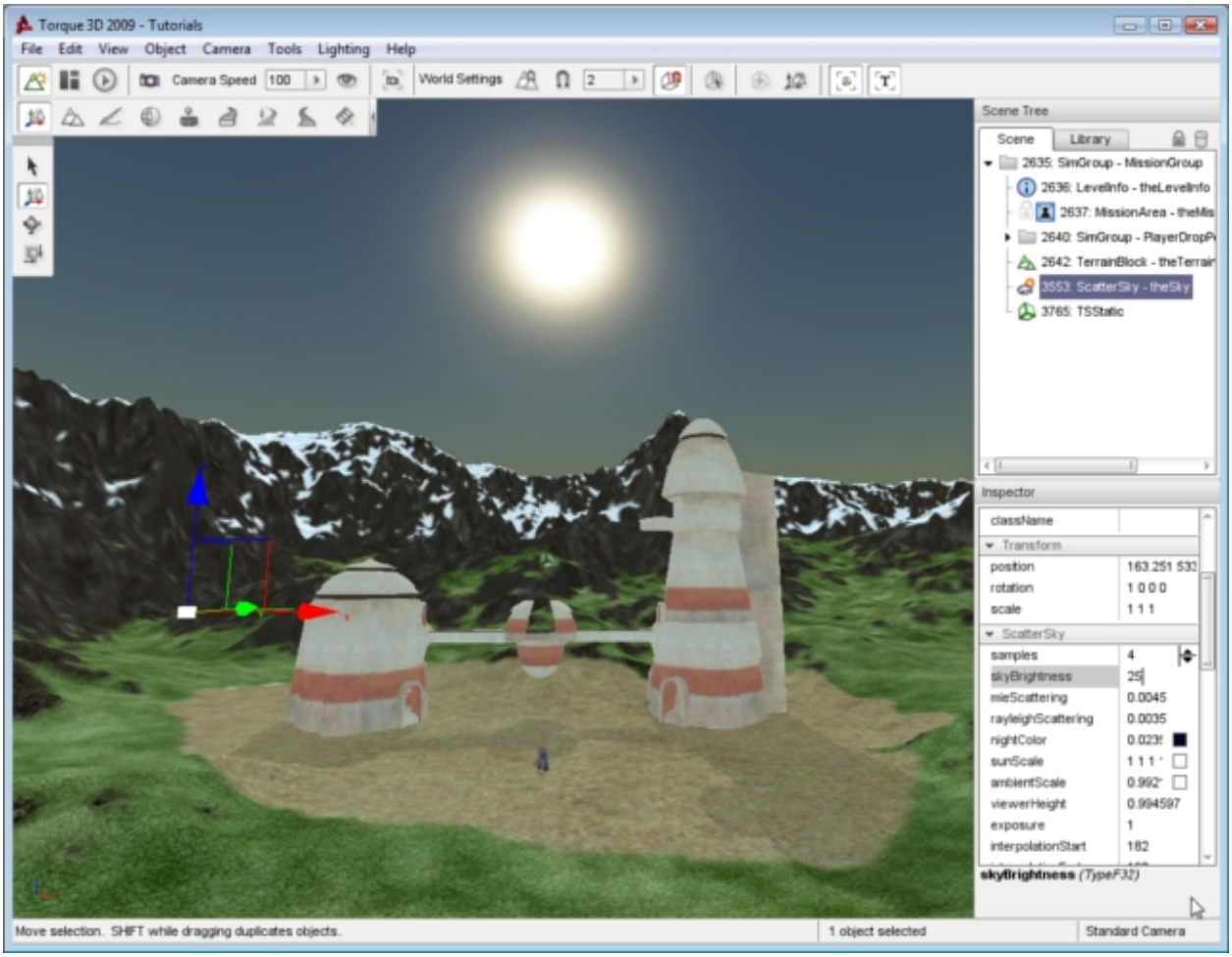
shadowDarkenColor TypeColorF. The color that should be used to multiply-blend dynamic shadows onto lightmapped geometry (ignored if representedInLightmap is false).

includeLightmappedGeometryInShadow TypeBool. This light should render lightmapped geometry during its shadow-map update (ignored if representedInLightmap is false).

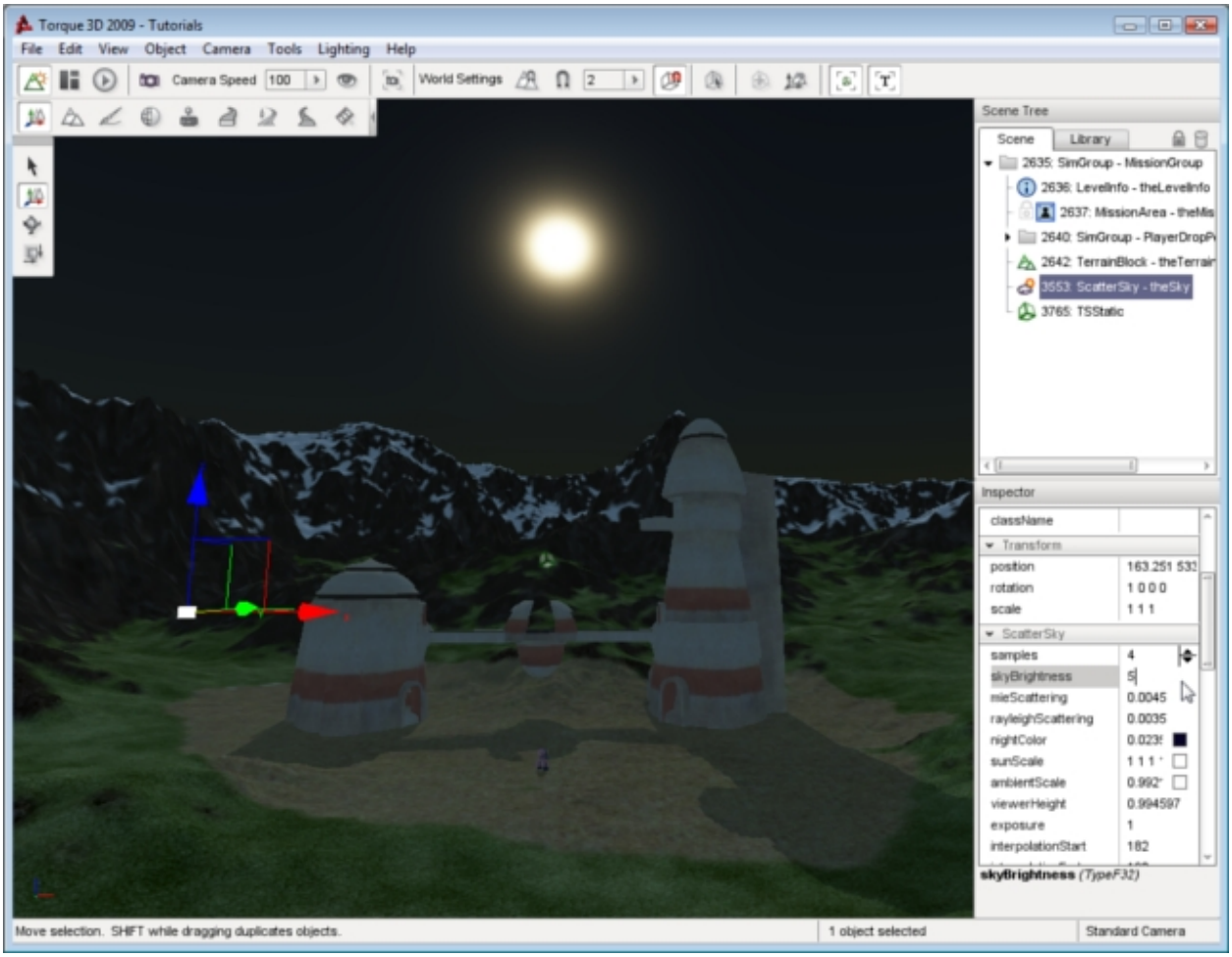
Modifying Brightness

Remember to refer back to the properties as you proceed through the rest of this guide. It is time to modify some of the more important fields of the current ScatterSky object. After each change is demonstrated, you will be reverting back to the stock values to show how these modifications affect the object.

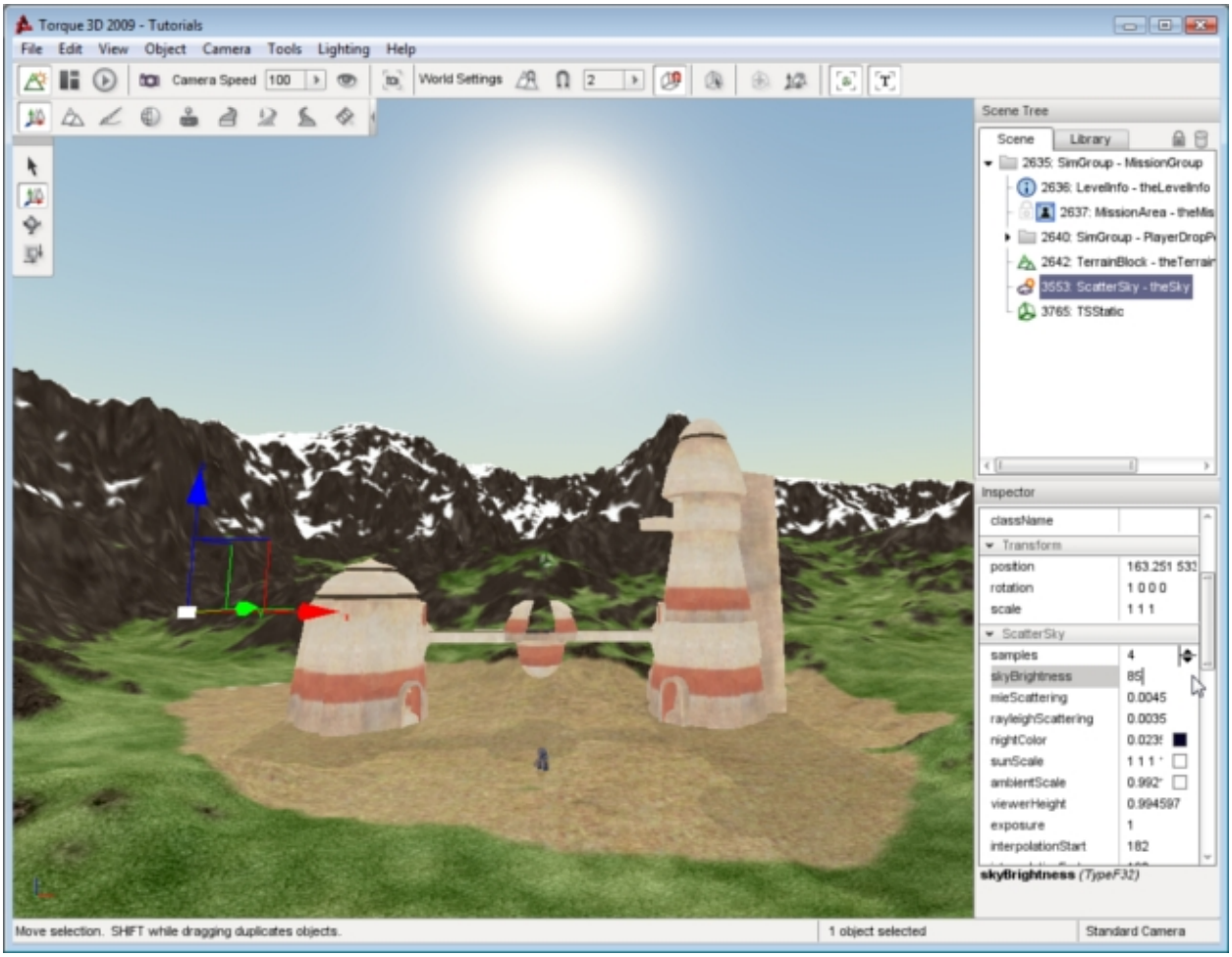
Now will start with adjusting the brightness of the sky and atmosphere. Under the ScatterSky section of the properties, look for the skyBrightness field. The default value is 25.



The skyBrightness field acts a global modifier of your brightness in the scene. Changing this value is similar to adjusting the contrast of a camera or monitor. Reduce the value to 5. This reduction will dramatically change the appearance of your level.



Now, greatly increase the value of skyBrightness to around 85. The view of your level should be extremely bright, as if the scene takes place in a desert at high noon.



If you have not already done so, revert your default brightness back to 25. You should change each value back to the default in this manner between each section of the remaining guide to see the effects of the next property.

Modifying Scattering

The scientific concept of scattering and how it affects your level is somewhat complex. The rayleighScattering and mieScattering values are extremely sensitive, and it is important that you have an understanding of how they work. The simplest way to explain scattering is to answer a question children often ask: "Why is the sky blue?"

In reality, beyond the atmosphere of the sky is blank space, that is, blackness. When you look up at the night sky, you can see the black of space and the stars it contains. However, during the day you see a blue sky. The blue color is due to the light rays from the sun being scattered by the molecules of the atmosphere as it passes through. Light appears to be white but it is actually composed of many different colors. The sky is usually blue because blue light scatters more easily than the other colors due to its physical properties.

The sky at the zenith is a darker blue than the sky near the horizon for two reasons. First, the atmosphere at this altitude is composed of much smaller particles, which is only capable of scattering the darker shades of blue light. Second, the light has had less opportunity to be scattered since it has not passed through as much atmosphere yet. The more times the same light is scattered, the paler the blue will become.

However, blue is not the only color that is scattered by the atmosphere, it is just the most common. Other colors, such as the reds at sunset, are due to how much atmosphere the light has passed through to get to your eyes. In this case, reds and blues are both being scattered but blue has been dissipated so much that it is no longer visible. The result is a red sky.

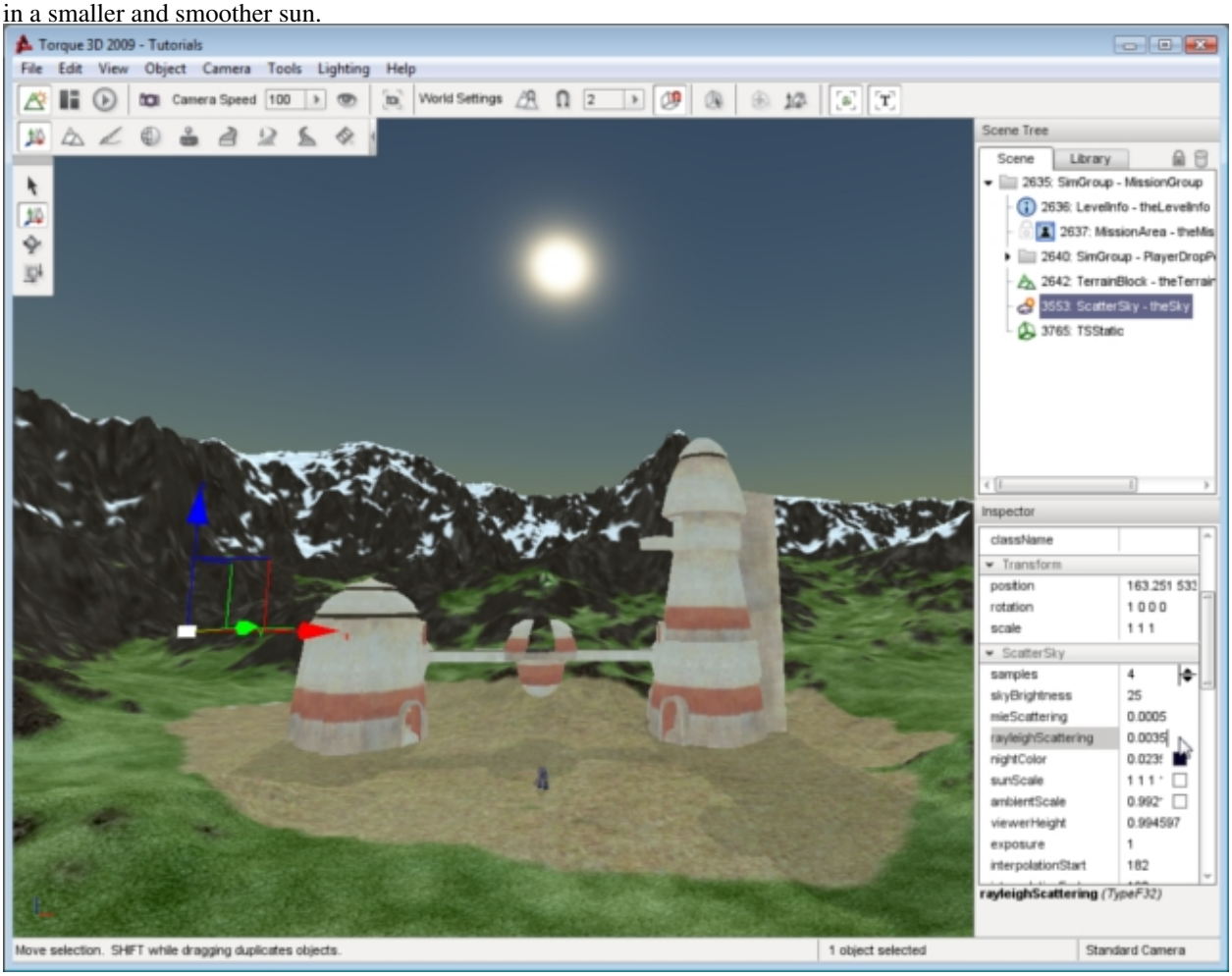
The scientific term for this light scattering effect is called Rayleigh Scattering, thus the rayleighScattering property of Torque 3D controls the color and darkness of the ScatterSky object.

The size and composition of particles in the atmosphere, such as dust and water, also has an effect on how light appears. Larger particles tend to scatter all colors of light approximately the same. This effect makes clouds, which are made of water vapour, appear to be white or grey. This is because the colors of the light are being scattered the same so that what you see resembles the original white form.

This type of scattering is also responsible for the clarity of a bright object and how the light rays are projected from it. The scientific term for this type of scattering is Mie Scattering, thus the mieScattering property of Torque 3D controls the clarity of light from bright objects such as the sun.

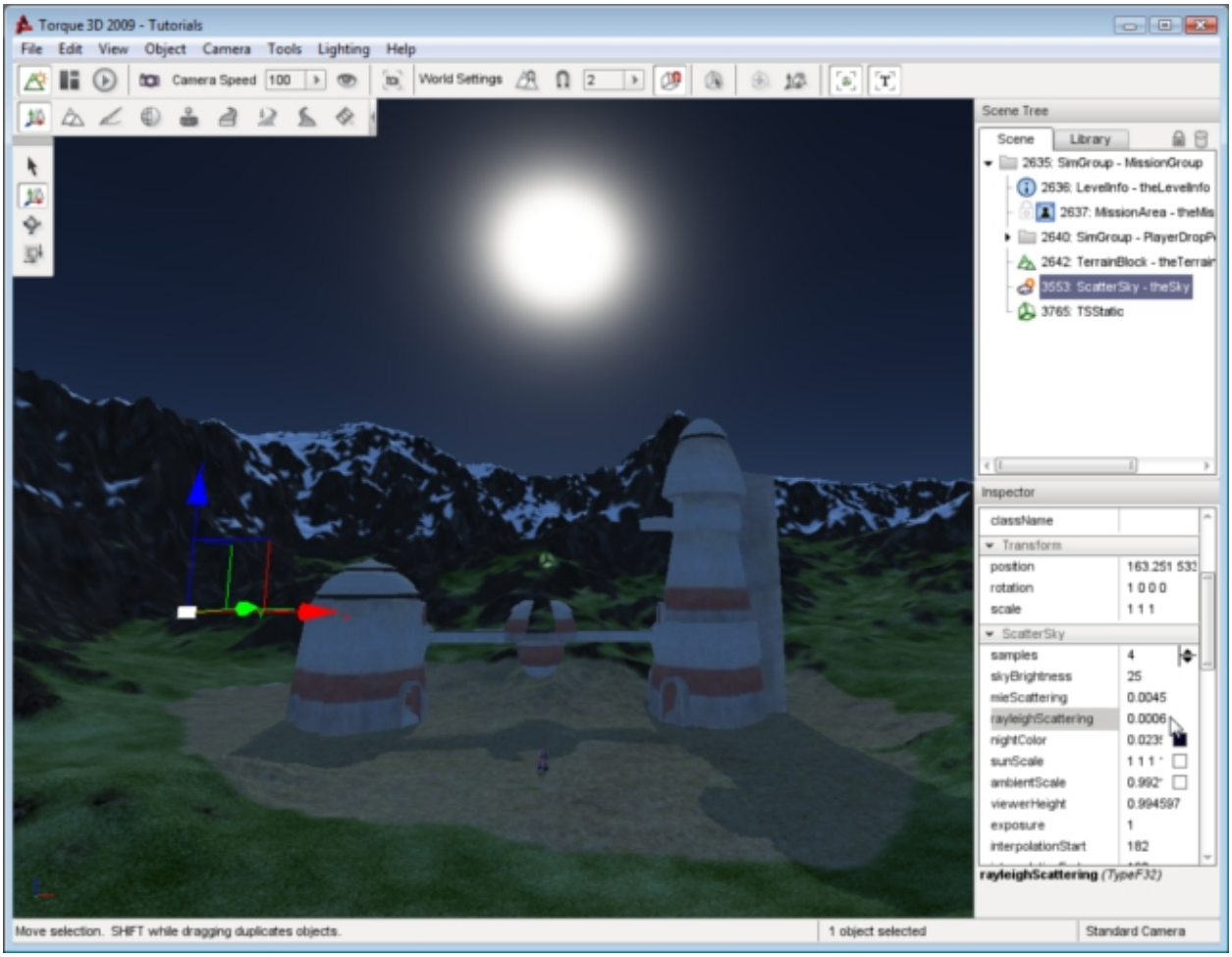
In summary, the scattering properties in Torque 3D are used to emulate the affects of nature. The mieScattering property affects the appearance of how light waves are projected from the sun object and the rayleighScattering property affects the color of the sky including how blue it will be.

Proceed to see the adjustment of these properties in action. Reduce the mieScattering field to a small value, such as 0.0005. You should notice that the scattering of the light around the sun object has been drastically reduced, resulting in a smaller and smoother sun.



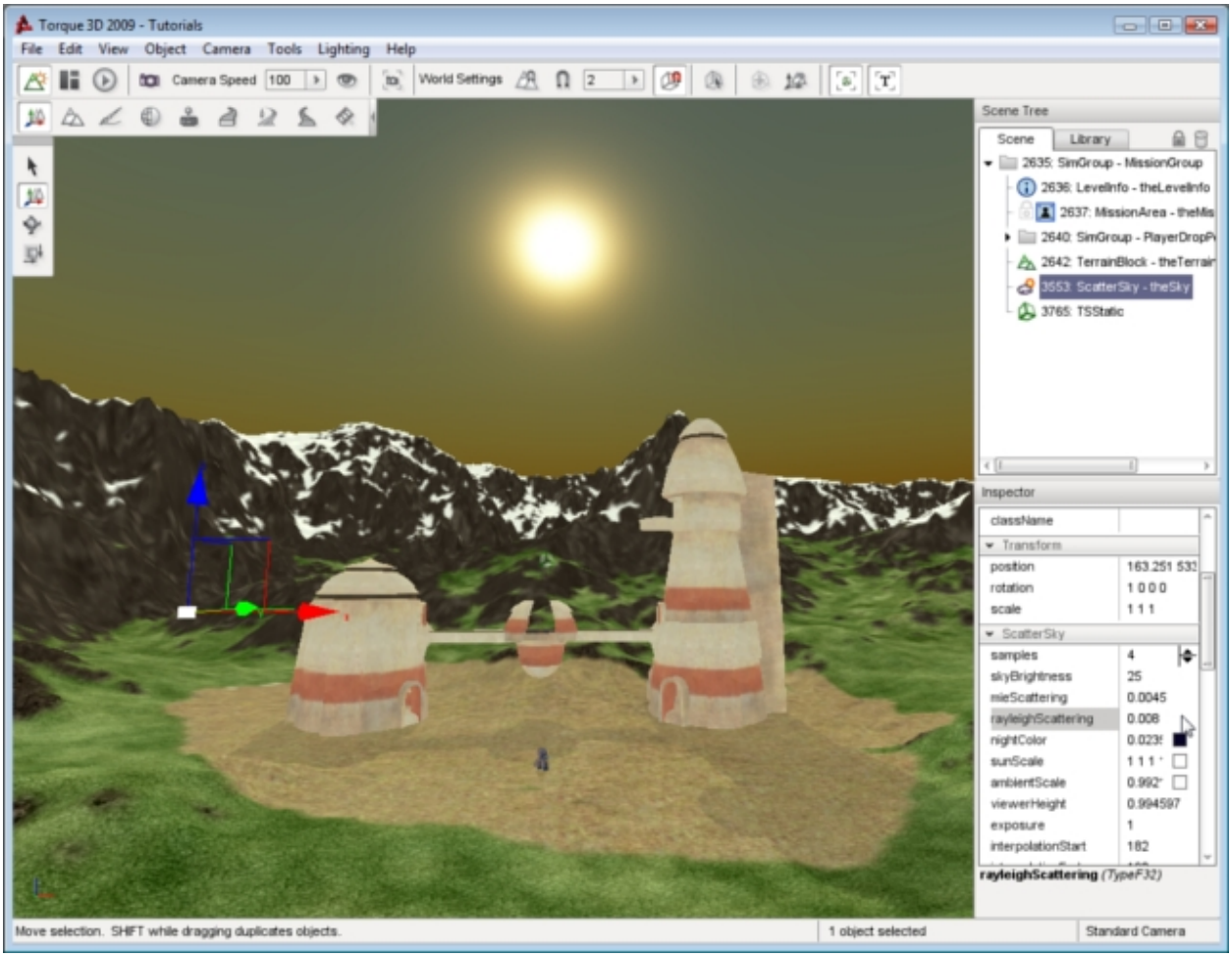
Reset the mieScattering back to the default value (approximately 0.0045). Lower the rayleighScattering field to 0.0006.

The atmosphere of the sky should now be a darker shade of blue. Reducing the rayleighScattering value simulates two things. First, it simulates an atmosphere which reduces the colors of light that will be scattered limiting it to the darker blues. Second, it simulates an atmosphere that has had less opportunity to dissipate the light leaving the darker shade of blue intact:



At some point, you can reduce the value only so far before you hit a shade of blue that is almost completely black. This does not mean you are actually seeing the black of space, rather you are seeing the darkest shade of blue light which has not been dissipated at all.

Go in the opposite direction. Begin increasing the rayleighScattering until you hit a value of 0.008. This simulates two things. First, it simulates an atmosphere which allows more colors of light to be scattered. Second, it simulates an atmosphere that has had less opportunity to dissipate the light leaving paler shades of the light. The result in your level is a broader range of colors in your sky.



If you go too high with the value, your sky will eventually become black. This is due to the allowance of all wave lengths to interact with the atmosphere. The effect is known as the subtractive rule of colors: white is the complete lack of color (light interaction) and black is the presence of all colors. In other words, the atmosphere is absorbing all colors so you see black.

If you have become confused, there are quite a few resources in your local library and on the Internet you can look up to learn more. If you have gotten this far, but wish to keep it simple, remember the following:

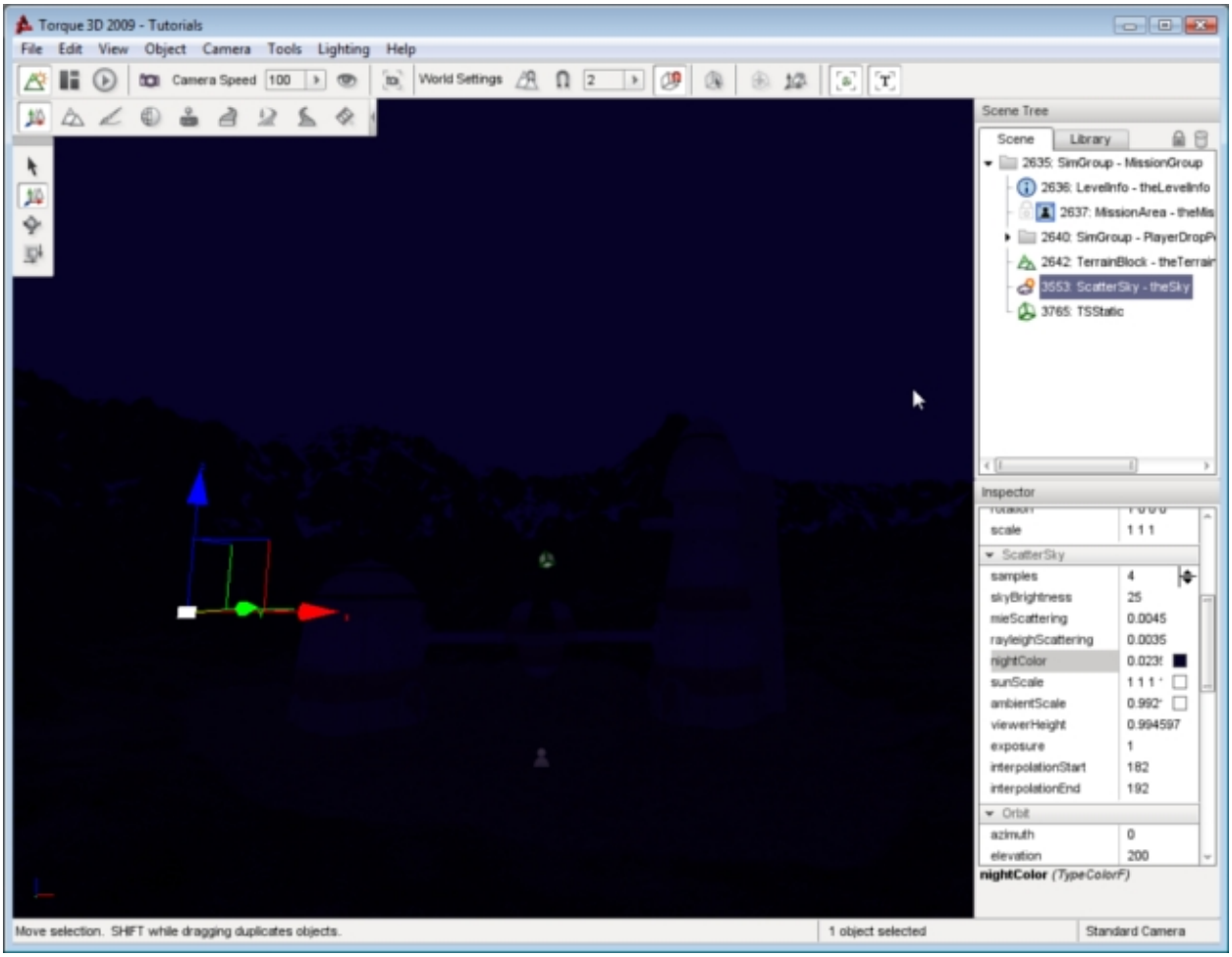
mieScattering Higher equals bigger and more scattered Sun. Lower equals smaller, smoother Sun.

rayleighScattering Higher equals less blue sky. Too high equals black sky. Lower equals more blue sky. Too low equals black sky.

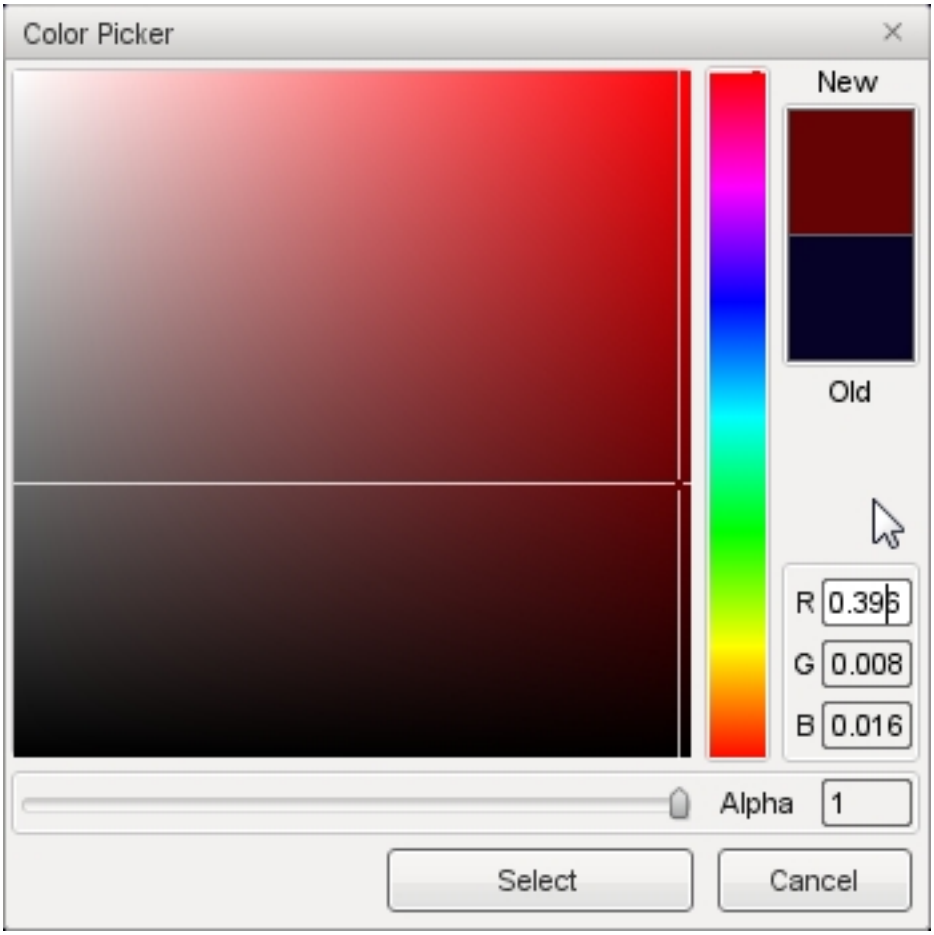
Modifying Colors

Move on to simpler concepts and property adjustments. The nightColor is a conditional property, as it only affects the scene during certain lighting conditions. As explained in the Sun documentation, modifying the azimuth and elevation will change the "time of day" for your level.

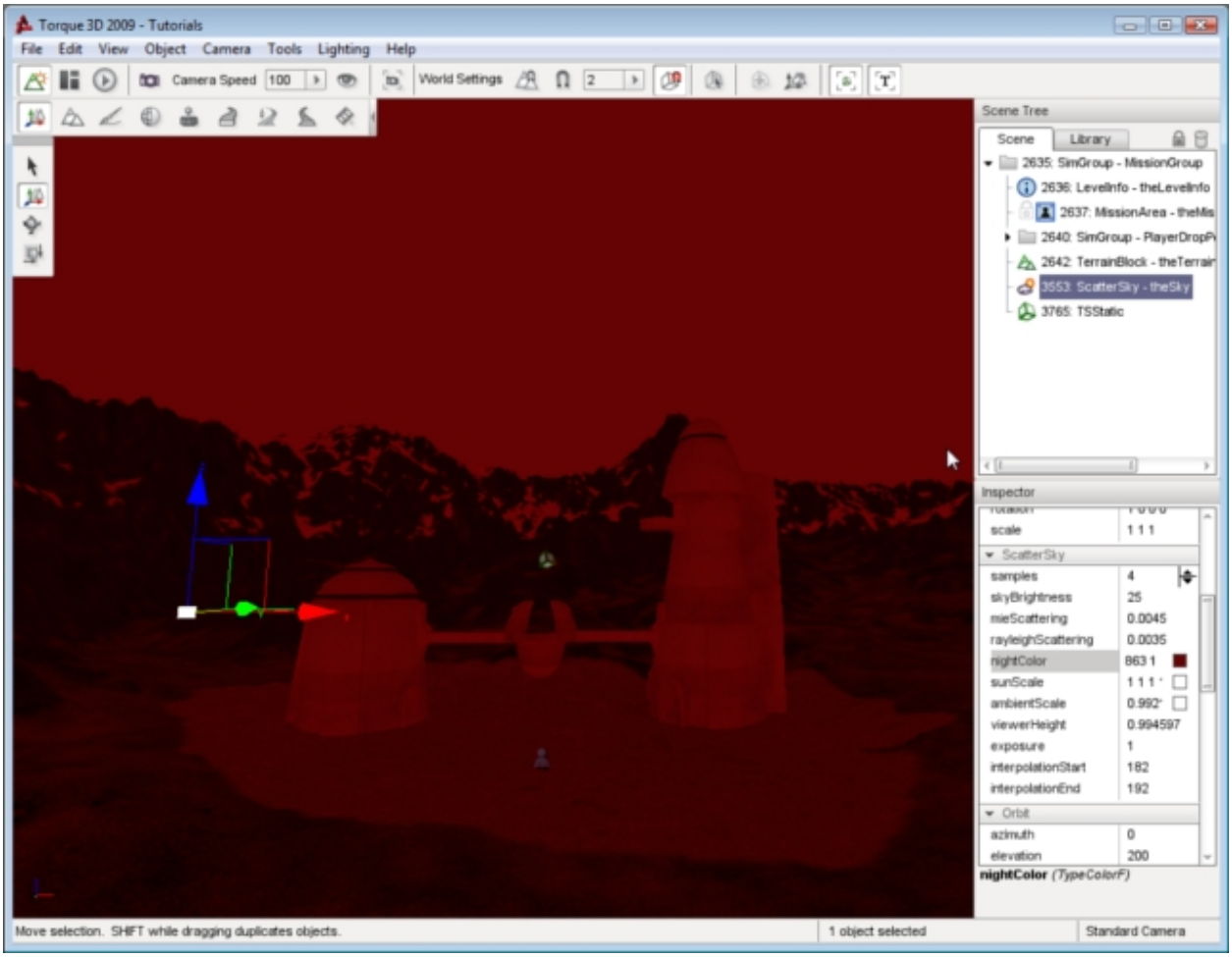
Go ahead and set the Elevation property in the Orbit section to 200, which will place the sun below the horizon. When the sun is no longer shining on your level, it is night time.



Scroll to the Night section in the Inspector Pane. Instead of manually guessing color values, click on the colored box next to the nightColor property. This action will open the Color Picker dialog. The dialog allows you to visually adjust the shade of your night time color. For an intense effect, go with an unnatural color such as red.



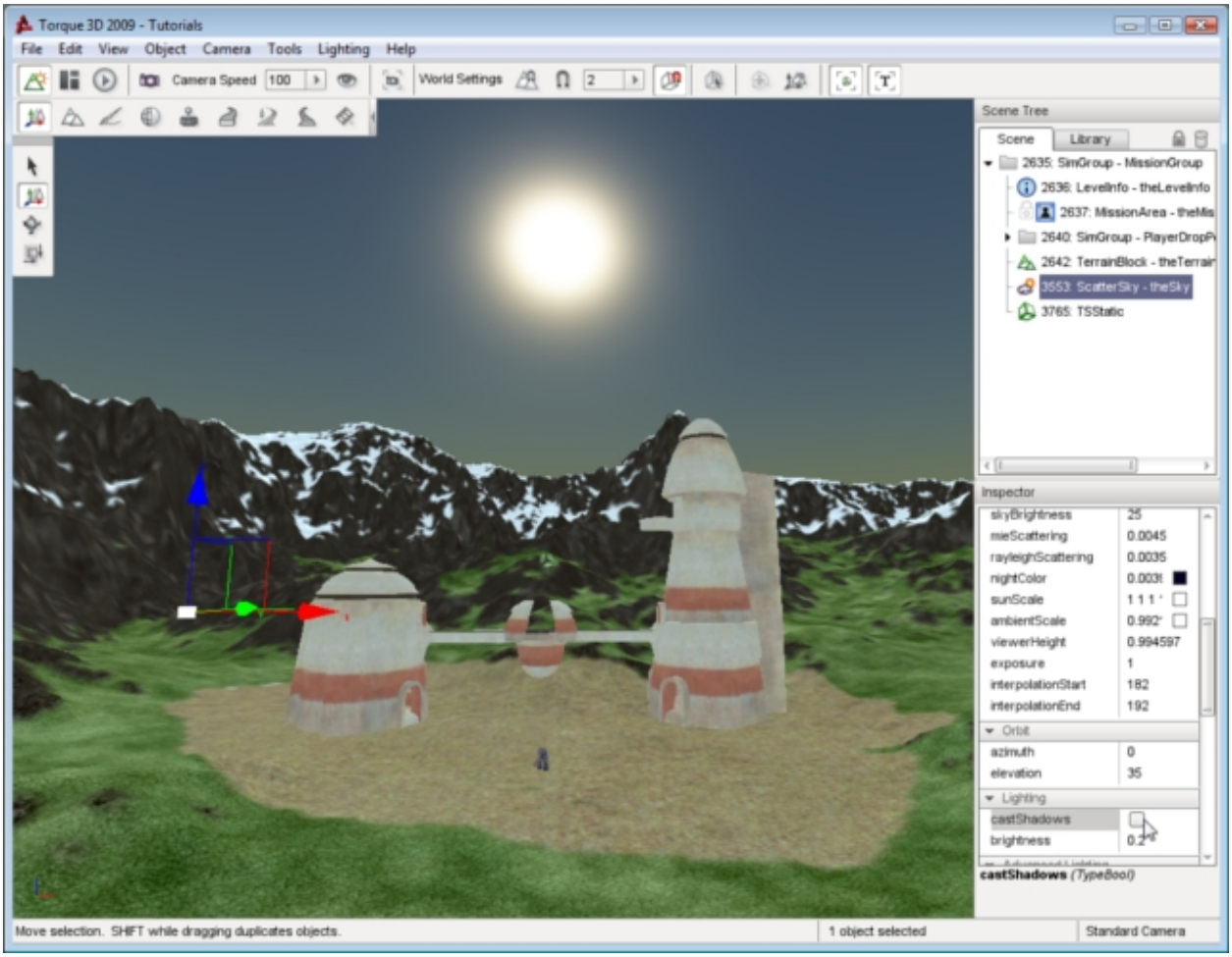
Click the select button when you are ready. Your level should immediately reflect the nightColor change you have made. Very creepy...



Definitely change the value back to something more suitable, such as deep blue/black color. Change the Elevation property in the Orbit section back to a number between 0 and 90 such as 45 to bring the sun back above the horizon and relight your scene.

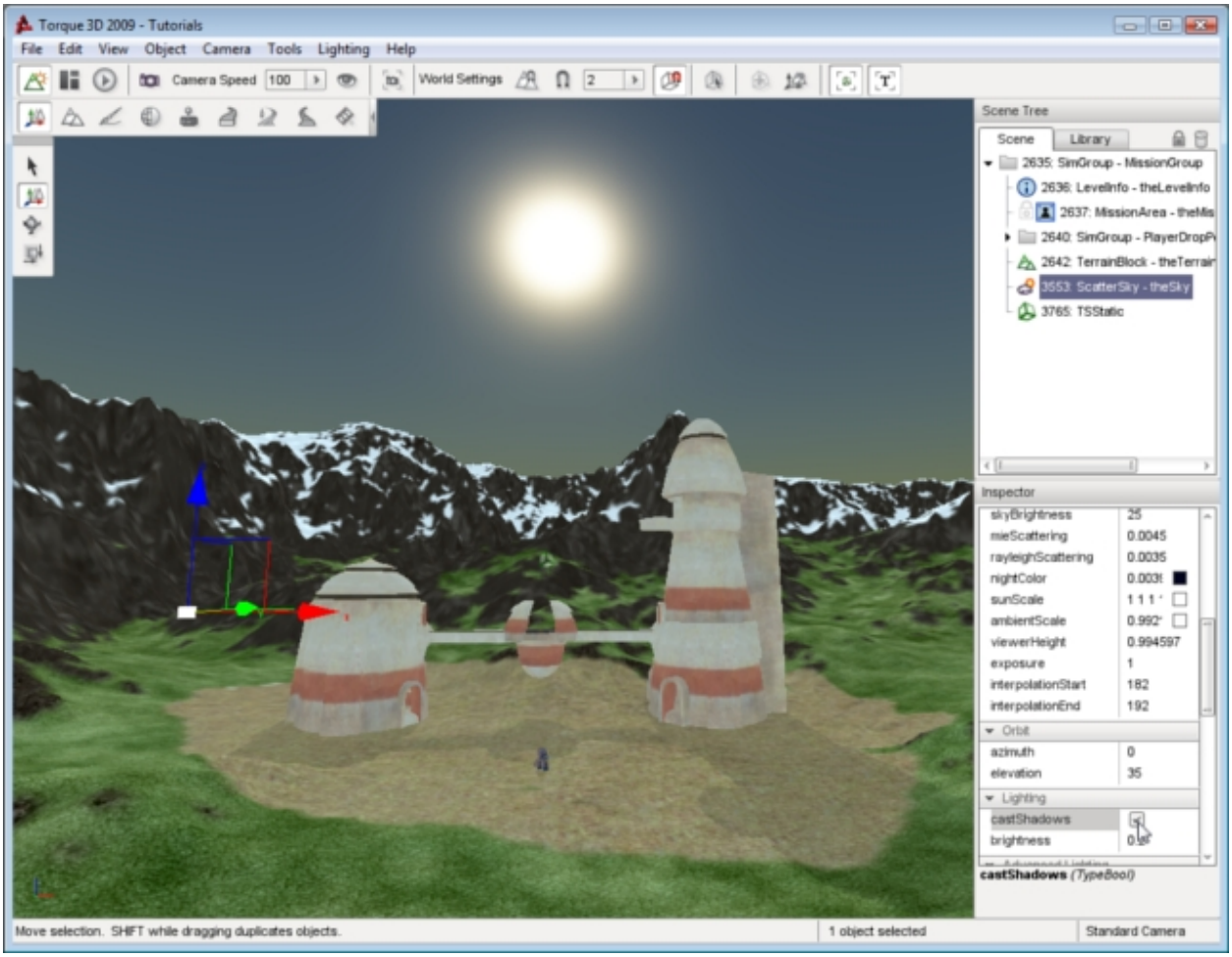
Modifying Shadows and Light Intensity

There are two fields under the Lighting section that strongly influence how your scene appears. The first property, castShadows, can be toggled on or off. Clicking on the property to toggle it off will result in a blank box. With castShadows disabled, nothing in your scene will cast a shadow: objects, terrain, etc.



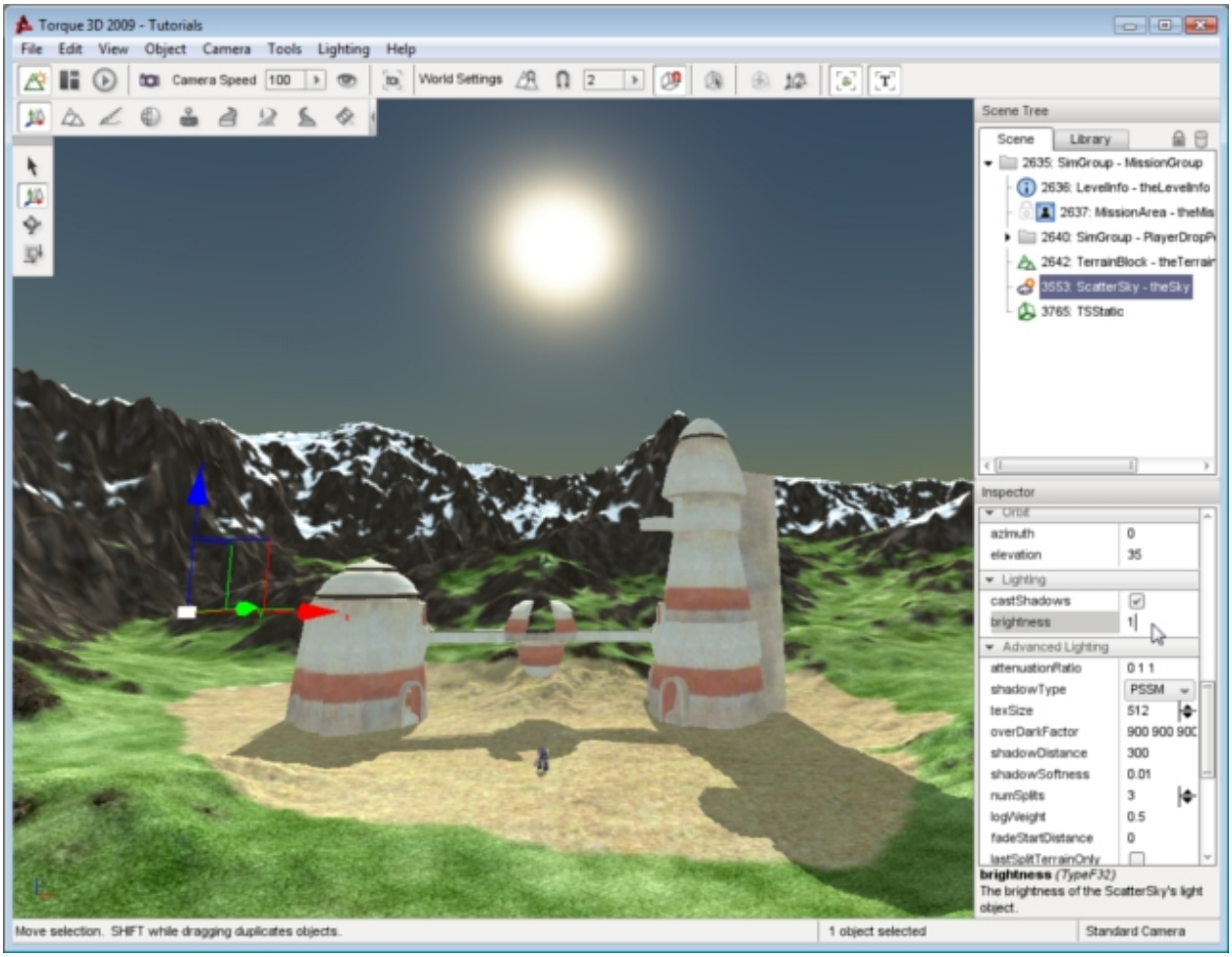
You can re-enable the shadows in your scene by clicking the box again, which will produce a check mark informing you that it has been enabled.

If you are using Advanced Lighting, the objects in your level will immediately begin casting shadows. If you are using Basic Lighting, you will need to relight the scene. Either way, the shadows will update according to the position of the sun.

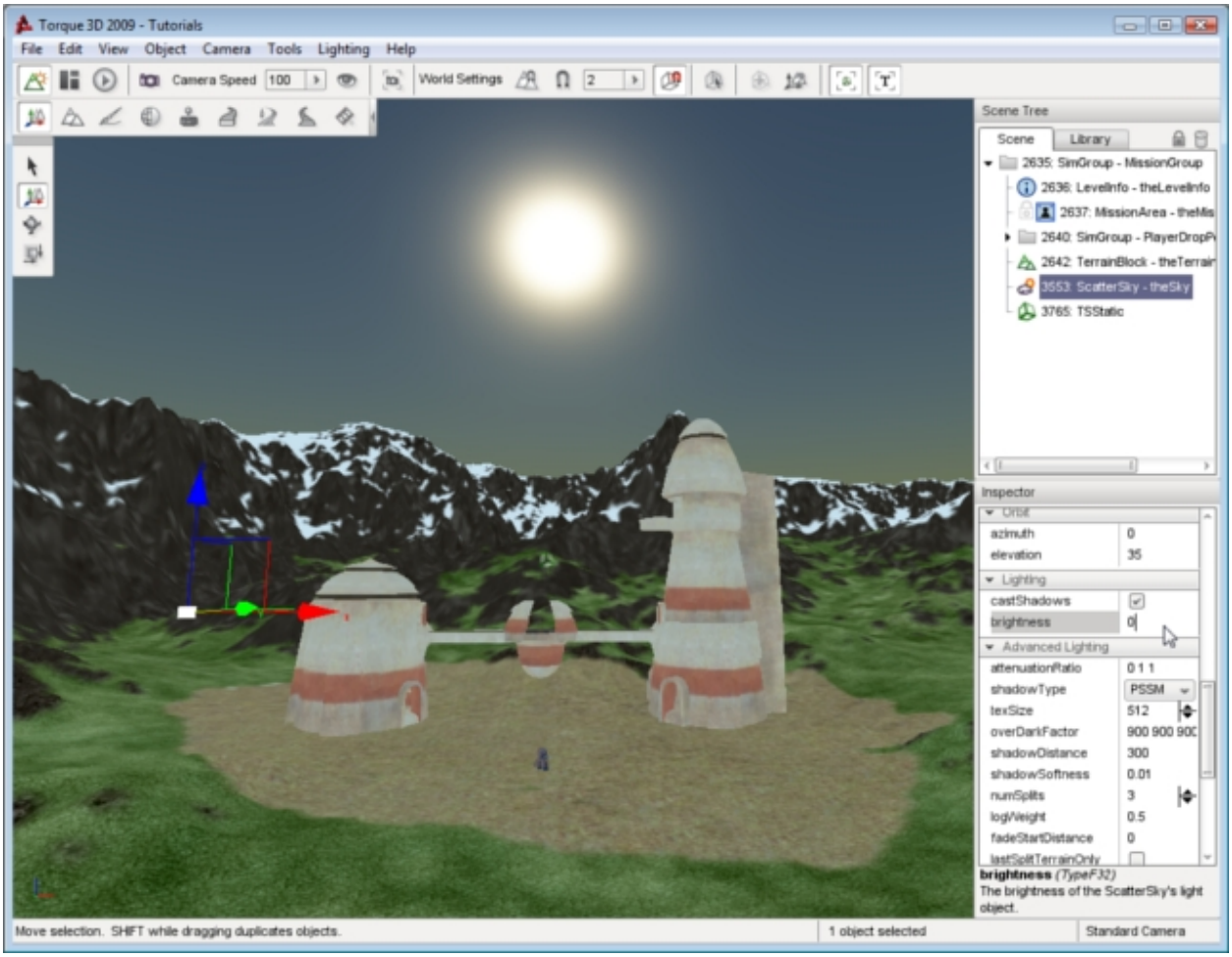


The brightness field under the Lighting section is completely separate from the skyBrightness property in the ScatterSky section. Unlike skyBrightness, which changes the contrast of your entire scene (particularly the sky itself), the brightness property under Lighting directly affects your objects in the scene.

You can see how this property functions by adjusting the value. Increase the brightness to 1. The lighting in your scene should be much brighter. Additionally, the shadows in your scene will be much darker and more defined.



Notice how your atmosphere (sky and sun) did not change. Every other object in your scene should be better lit. You can remove the additional brightness by setting the value of the property to 0. The result is the additional, global brightness factor has been completely removed. Your lighting should now be minimal, and your shadows nearly invisible.



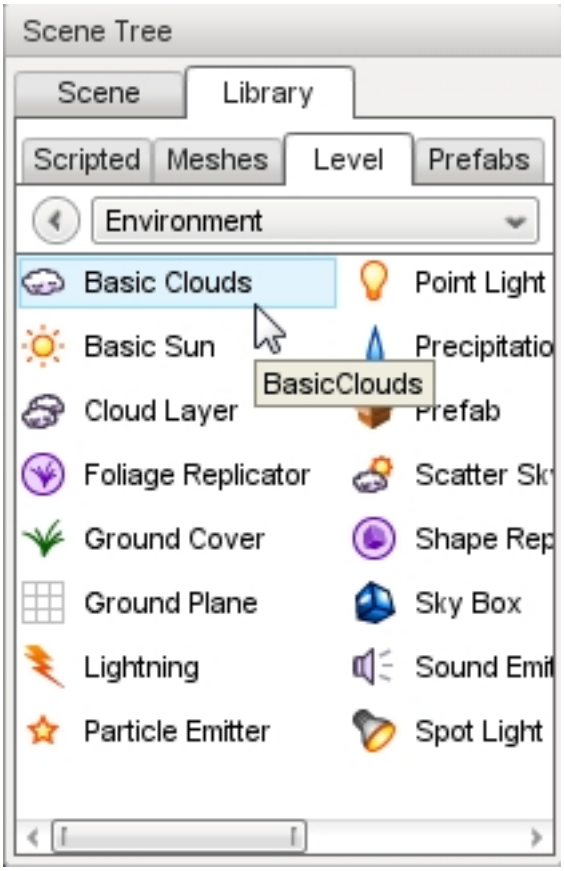
2.3.7 Basic Clouds

Once you have decided on your level's sky, whether it's a Skybox or ScatterSky, you can continue to customize the scene with clouds. There are two cloud objects you can choose between: Basic and Advanced Clouds. This guide covers the Basic Clouds object.

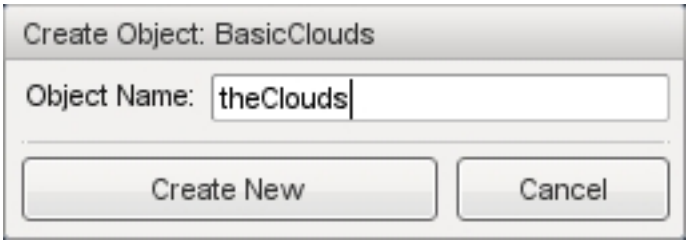
The Basic Cloud system renders up to three textures to separate layers, at varying heights, detail levels, and speeds. Though basic and less memory intensive, you can build very detailed and realistic clouds for your level.

Adding a Basic Clouds Object

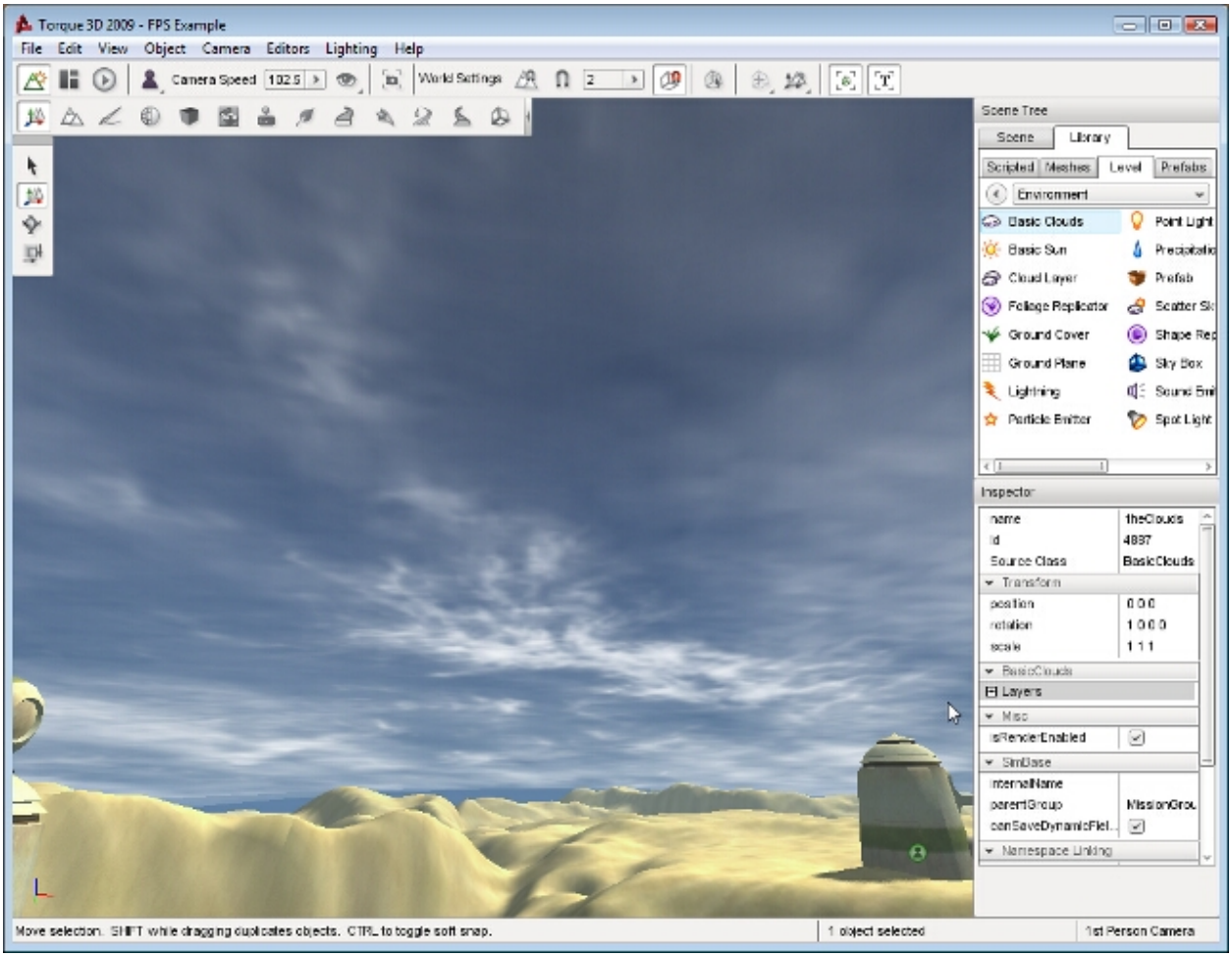
To add a Basic Clouds object: select the Library tab in the Scene Tree panel. Click on the Level tab and then double-click the Environment folder. Locate the Basic Clouds entry.



Double-click the Basic Clouds entry and a dialog will appear:



Enter a name for your clouds object then click the Create New button. A Basic Clouds object will be added to your level. Three separate cloud layers will be rendering and moving across the sky slowly:



Basic Cloud Properties

Additional properties can be changed with the Inspector pane. To change a Basic Clouds properties using the Inspector Pane click the Scene tab, then click the name of your new Basic Cloud object. The Inspector pane will update to display the current properties of your new sun.

Inspector

name TypeName. Optional global name of this object.

id TypeCaseString. SimObjectId of this object. Read Only.

Source Class TypeCaseString. Source code class of this object. Read Only.

Transform

position MatrixPosition. Object world position.

rotation MatrixOrientation. Object world orientation.

scale Point3F. Object world scale.

BasicClouds

layerEnabled TypeBool. Enable or disable rendering of this layer.

texture TypeImageFilename. Texture for this layer.

texScale TypeF32. Texture repeat for this layer.

texDirection TypePoint2F. Direction texture scrolls for this layer.

texSpeed TypeF32. Speed texture scrolls for this layer.

texOffset TypePoint2F. UV offset for this layer.

height TypeF32. Abstract number which controls the curvature and height of the dome mesh.

Editing

isRenderEnabled TypeBool. Only render if true (and if class is render-enabled, too

isSelectionEnabled TypeBool. Determine if the object may be selected from within the Tools.

Object

internalName TypeString. Non-unique name used by child objects of a group.

parentGroup TypeString. Group object belongs to.

class TypeString. Links object to script class namespace.

superClass TypeString. Links object to script super class (parent) namespace.

Cloud Layers

While editing your Basic Clouds object, you may discover the need to view and edit individual layers. Open the BasicClouds section of the Inspector pane. Under the Layers sub-section you will find three layers labeled by an index. Each index refers to a layer and determines rendering order. The *layer[0]* will be rendered first, *layer[1]* next, and finally *layer[2].* In simpler terms:

- layer[0] is drawn on top of the sky
- layer[1] is drawn on top of layer[0]
- layer[2] is drawn on top of layer[1].

You can adjust the visibility of each layer by toggling the layerEnabled property. If all three layers are disabled the Basic Clouds object will not be visible at all:

Regarding Movement

Unfortunately, static images cannot properly show how the remaining fields affect the Basic Cloud layers, since they all pertain to the motion of the clouds. Clouds can only move horizontally, they can not move up and down. This horizontal movement is described in the texDirection property.

The texDirection property takes two values, separated by a space: “X Y”. Each value corresponds to the axis a texture should scroll on as well as the direction of movement on that axis.. The range of each value is -1.0 to 1.0. For example: A value of “1 0” will scroll the texture directly along the X axis in the positive direction with no movement along the Y axis.

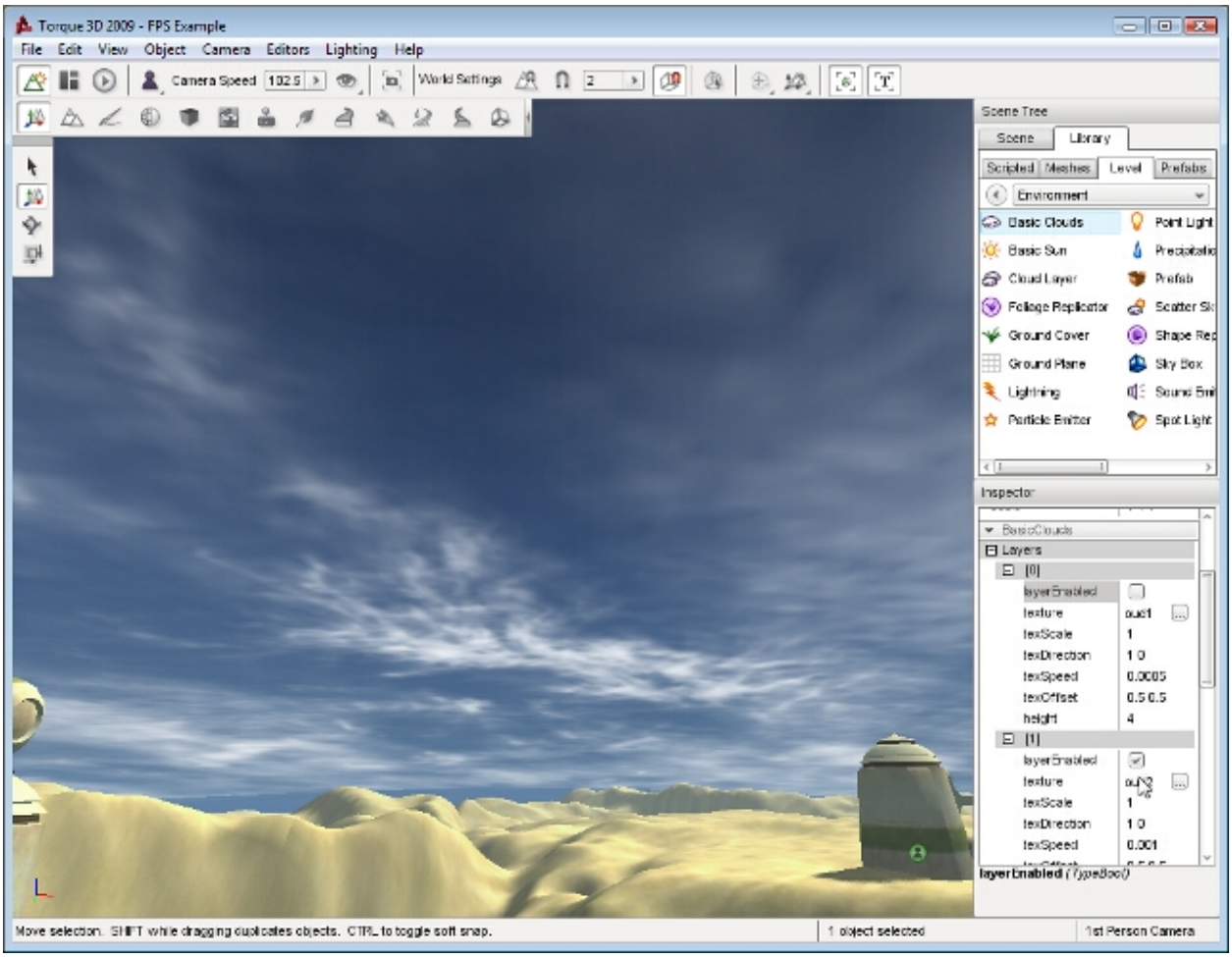


Fig. 2.21: Layer 0 Disabled

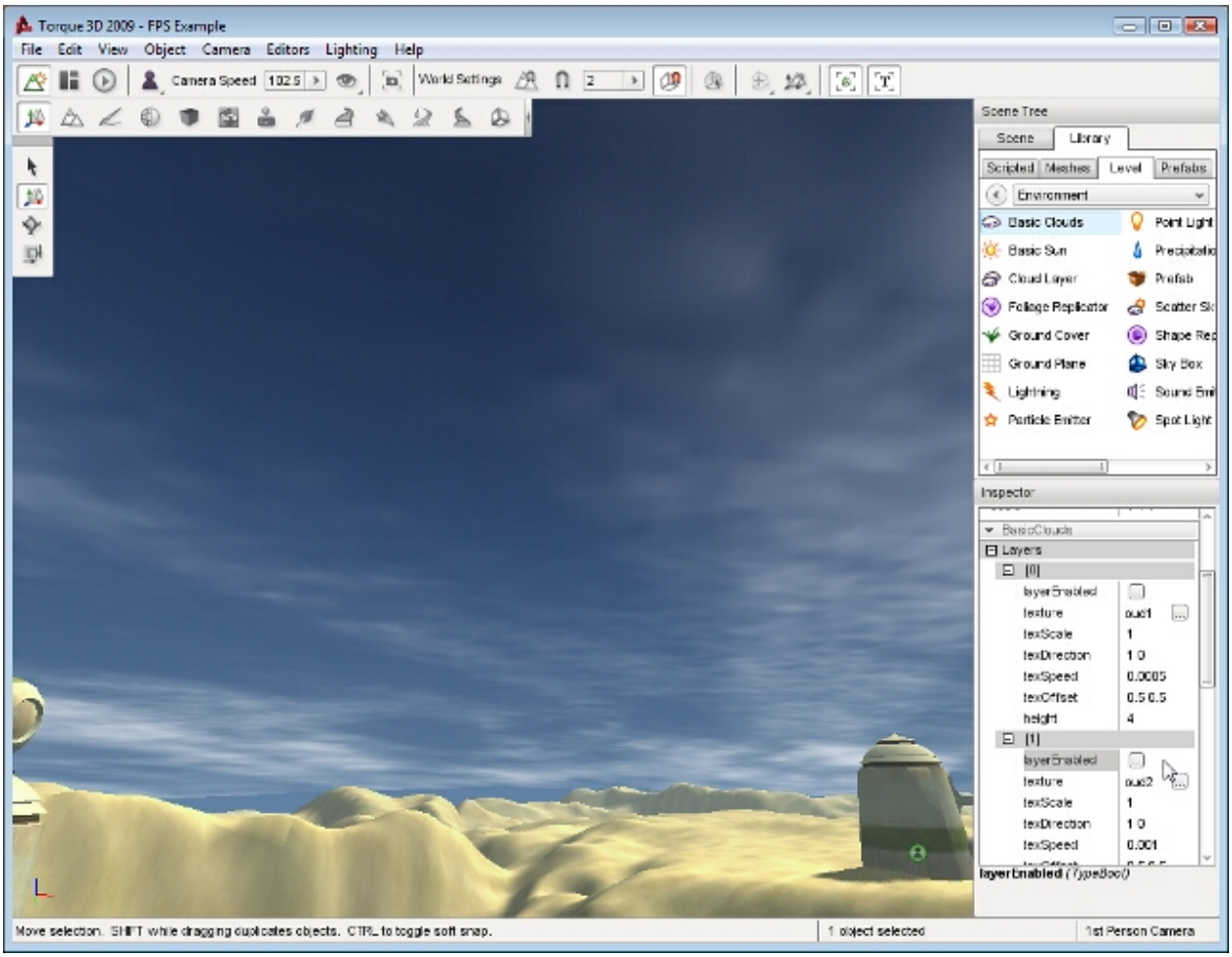


Fig. 2.22: Layer 1 Disabled

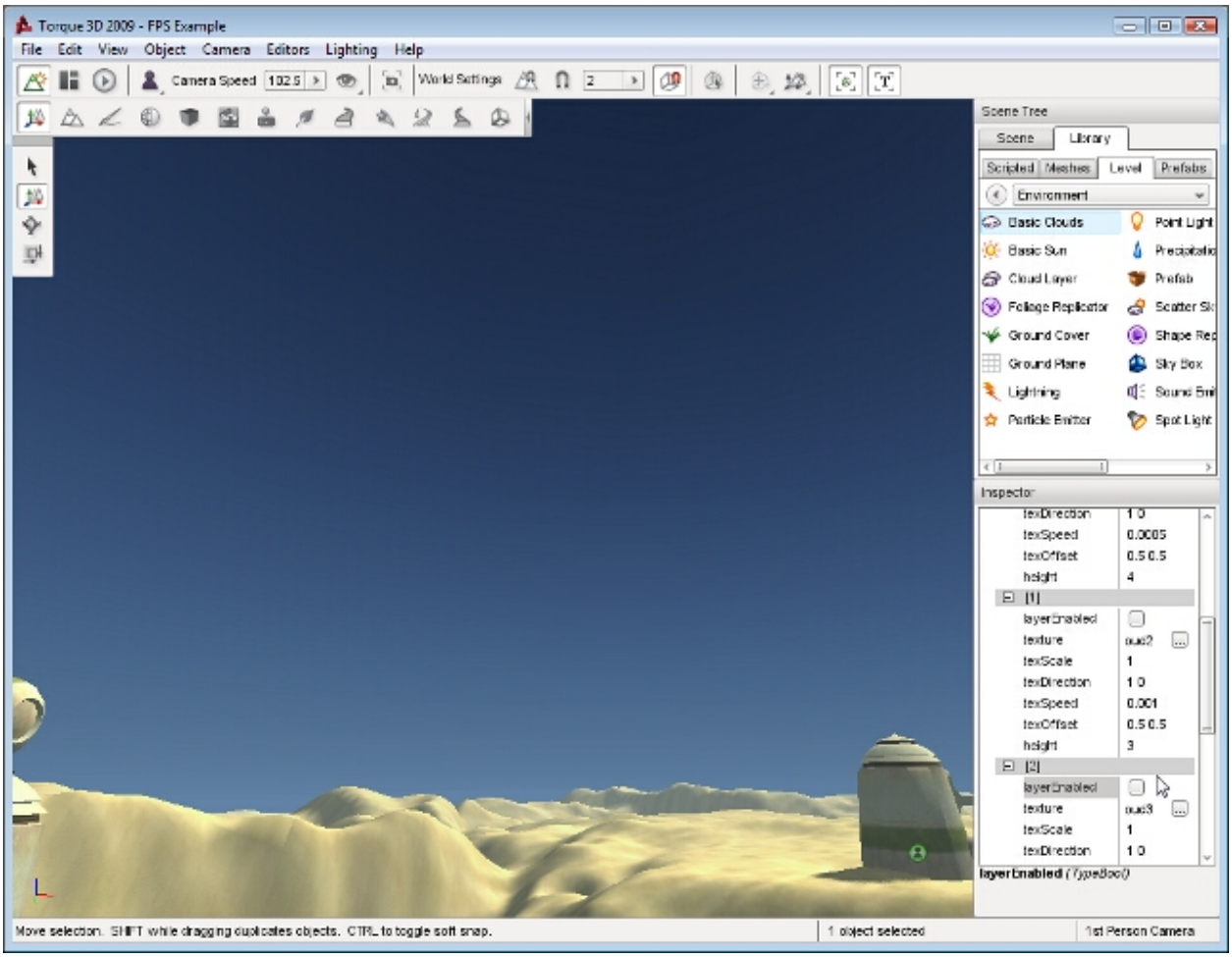


Fig. 2.23: Layer 2 Disabled

A single property, texSpeed, controls how fast the cloud layer moves. If the property is set to 0, the cloud layer will not move. The higher the number, the faster your cloud texture will scroll across the sky.

With the texOffset property you can displace how the multiple textures line up or overlap with each based upon whatever looks visually best. For example, at the seam where the texture repeats, you might want that to be on the horizon rather than directly overhead. Adjusting the texOffset helps you visually adjust this. If you have a grasp of UV animation, this will come naturally.

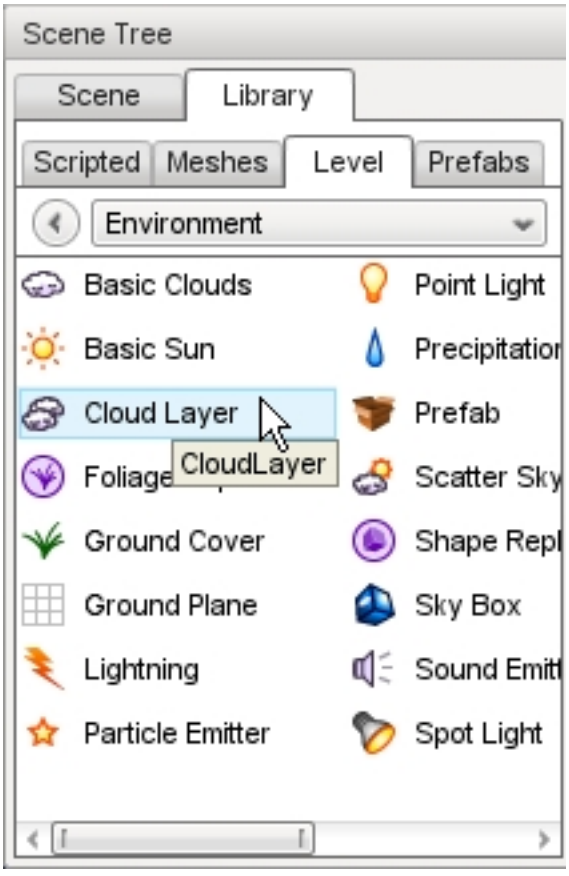
2.3.8 Cloud Layer

Once you have decided on your level's sky, whether it's a Sky Box or Scatter Sky, you can continue to customize the scene with clouds. There are two cloud objects you can choose between. This guide covers the Cloud Layer object.

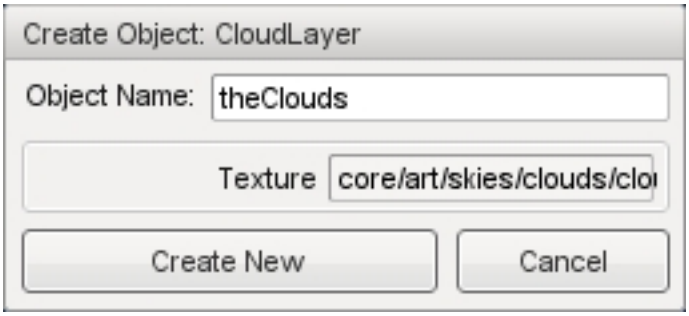
The Cloud Layer object uses the material system and procedurally generates clouds in the atmosphere. This cloud layer is extremely powerful and flexible. For the most realistic simulation of an atmosphere, the Cloud Layer is highly recommended.

Adding a Cloud Layer

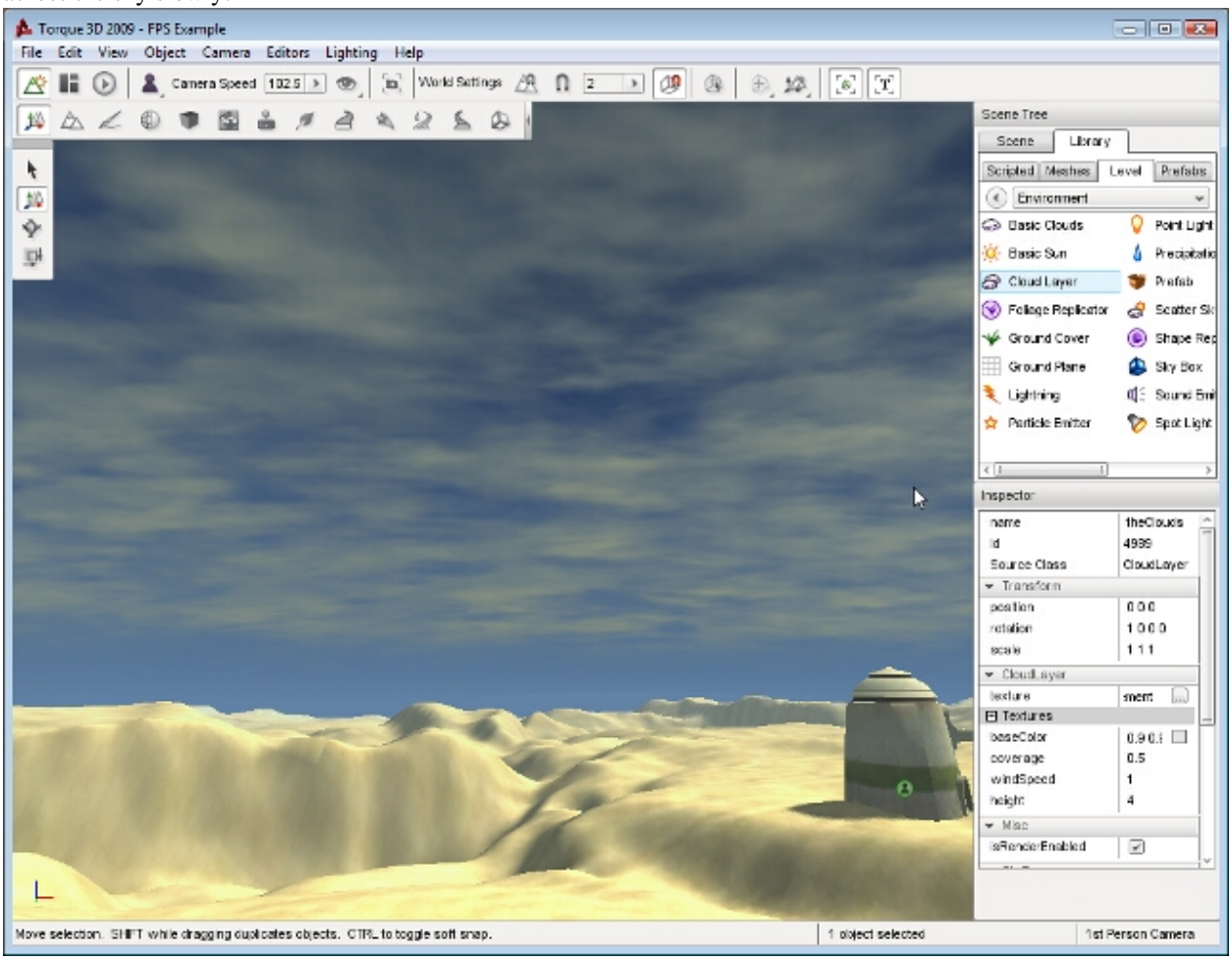
To add a Cloud Layer object, select the Library tab in the Scene Tree panel. Click on the Level tab and double-click the Environment folder. Locate the Cloud Layer entry:



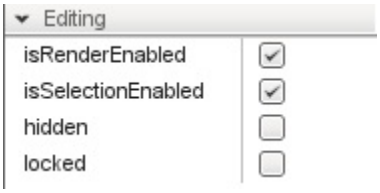
Double-click the Cloud Layer entry. The Create Object dialog box will appear:



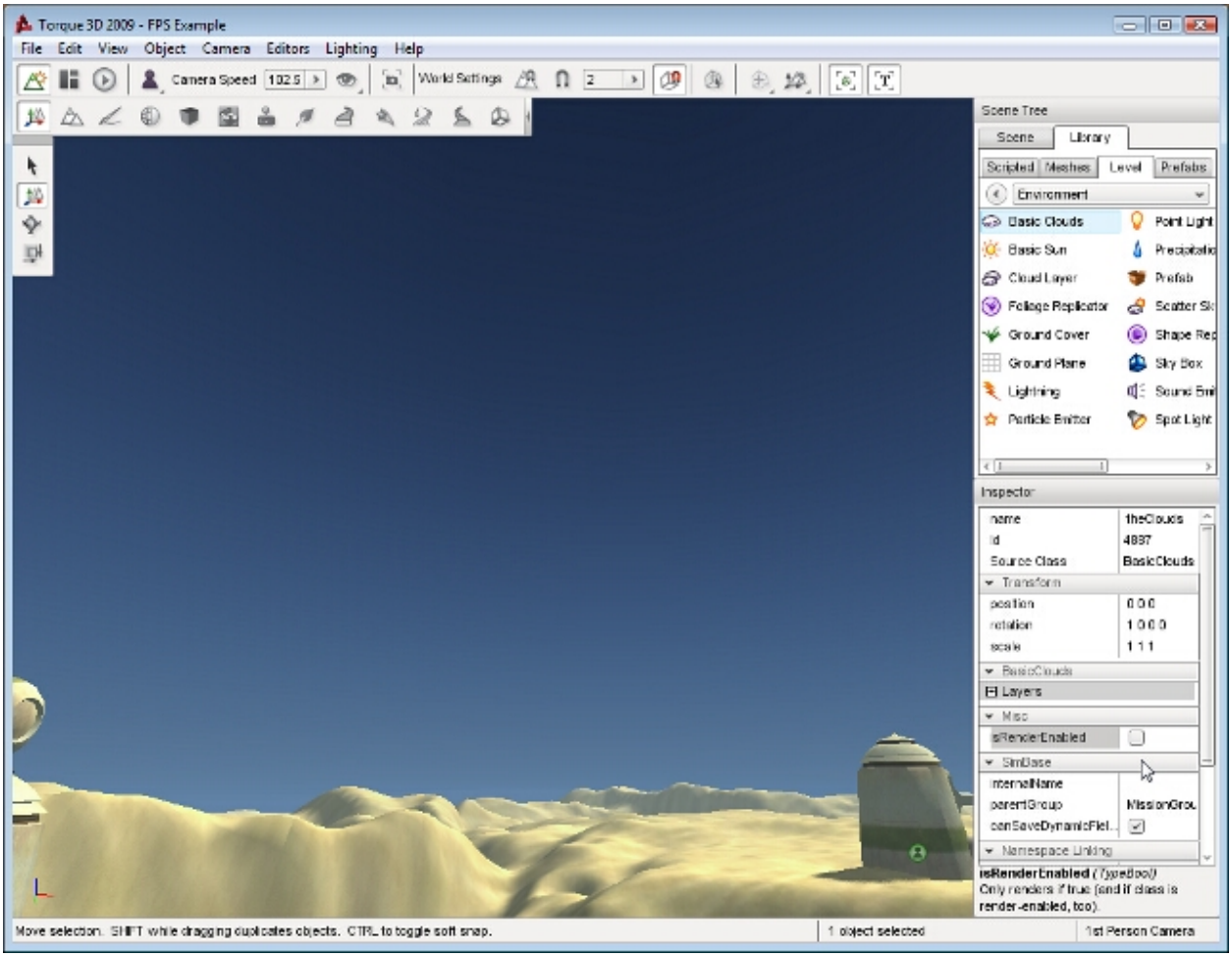
Enter a name for your Cloud Layer object then click the Create New button. A Cloud Layer object will be added to your level. The three procedural cloud layers generated by the Cloud Layer object will be rendering and moving across the sky slowly:



At any time you can toggle the visibility of the cloud layer by locating the isRenderEnabled field under the Editing properties:



When you toggle this property, the clouds will render according to the status of the check box:



Cloud Layer Properties

Additional properties can be changed with the Inspector pane. To change a Cloud Layers properties using the Inspector Pane, click the Scene tab, and then click the name of your new Cloud Layer object. The Inspector pane will update to display the current properties of your new sun.

Inspector

Name TypeName. Optional global name of this object.

id TypeCaseString. SimObjectId of this object. Read Only.

Source Class TypeCaseString. Source code class of this object. Read Only.

Transform

position MatrixPosition. Object world position.

rotation MatrixOrientation. Object world orientation.

scale Point3F. Object world scale.

CloudLayer

texture TypeImageFilename. An RGBA texture which should contain normals and opacity/height.

baseColor TypeColorF. Base cloud color.

exposure Brightness scale so CloudLayer can be overblown if desired.

coverage TypeF32. Fraction of sky covered by clouds 0 to 1.

windSpeed Wind speed.

height TypeF32. Abstract number which controls the curvature and height of the dome mesh.

Editing

isRenderEnabled TypeBool. Toggle whether rendering of this object is enabled.

isSelectionEnabled TypeBool. Toggle whether to allow selection in the editor.

hidden TypeBool. Toggle whether this object is visible.

locked TypeBool. Toggle whether this object can be changed.

Mounting

mountPID Unique identifier of the mount.

mountNode Node where the mount occurs.

mountPos Offset for positioning the node.

mountRot Rotation of this object in relation to the mount node.

Object

internalName TypeString. Non-unique name used by child objects of a group.

parentGroup TypeString. Group object belongs to.

class TypeString. Links object to script class namespace.

superClass TypeString. Links object to script super class (parent) namespace.

Persistence

canSave TypeBool Toggle whether the object can be saved in the editor.

canSaveDynamicFields TypeBool. True if dynamic fields (added at runtime) should be saved, defaults to true.

persistentId Unique ID of this object.

Cloud Layers

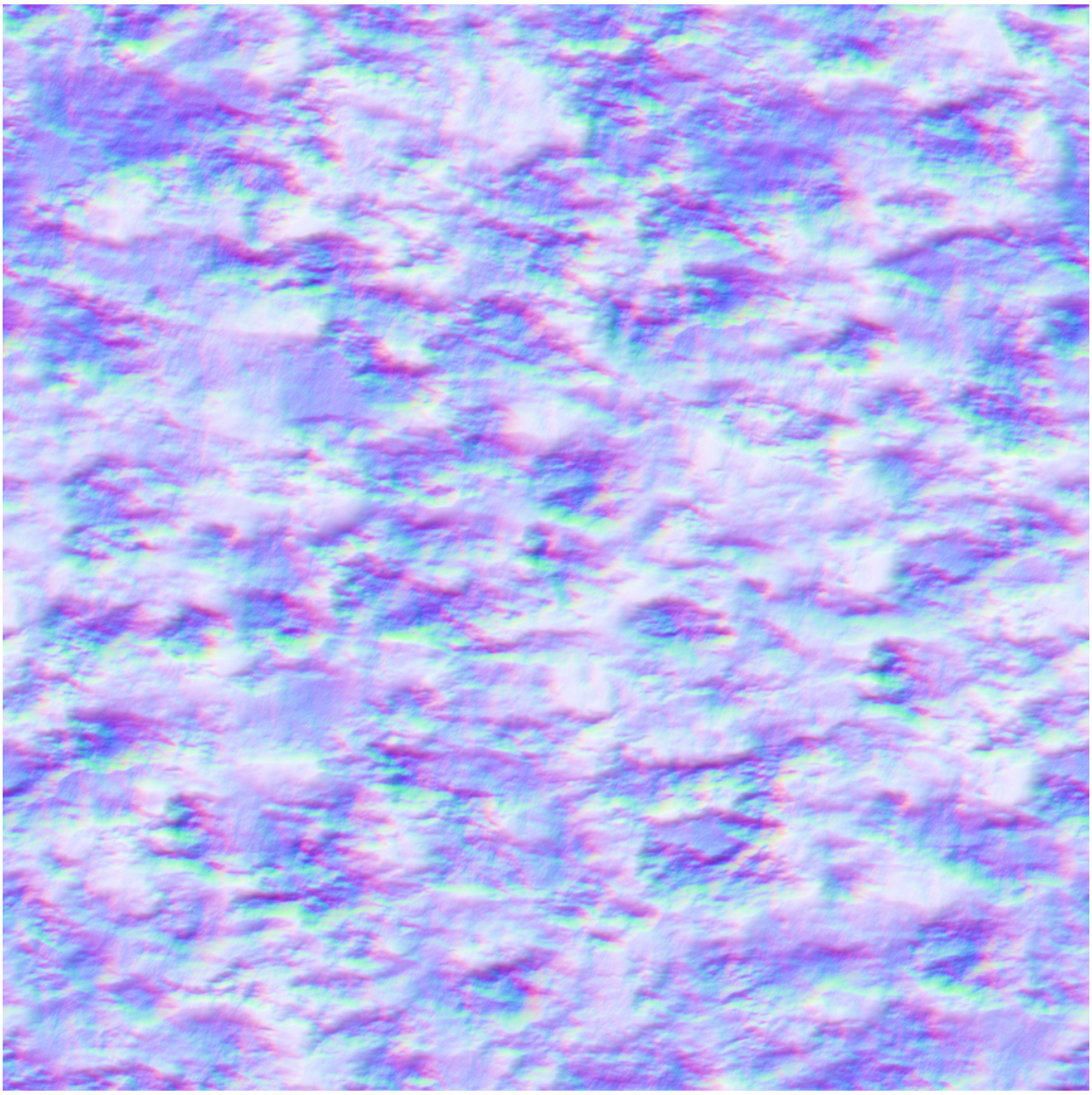
While editing your Cloud Layer object, you may discover the need to edit individual layers. Under the Textures property section you will find layers labeled by an index. Each index refers to a layer and determines rendering order. The *layer[0]* will be rendered first, *layer[1]* next, and finally *layer[2].* In simpler terms:

- layer[0] is drawn on top of the sky.
- layer[1] is drawn on top of layer[0].
- layer[2] is drawn on top of layer[1].

Unlike the Basic Clouds object, you cannot toggle the visibility of each layer. All three work together for procedural generation.

Editing Cloud Texture

The very first visual modification you can make is selecting a texture. This is the first field under the CloudLayer properties. The stock Cloud Layer uses the following normal map:



Cloud Layer does not use a diffuse texture. Color is calculated per-pixel based on the normal map using the sun/ambient/fog colors. It is designed to work with the ScatterSky and TimeOfDay where simple/constant diffuseMap based color will not work. For the procedural layer to work, the texture needs to be 4-channel. RGB (red blue green) is a normal map and A (alpha) is the transparency.

Regarding Movement

Unfortunately, static images cannot properly show how the remaining fields affect the Cloud Layer since they all pertain to the motion of the clouds. Clouds can only move horizontally, they can not move up and down. This horizontal movement is described in the texDirection property.

The texDirection property takes two values, separated by a space: "X Y". Each value corresponds to the axis a texture should scroll on as well as the direction of movement on that axis. The range of each value is -1.0 to 1.0. For example, a value of "1 0" will scroll the texture directly along the X axis in the positive direction with no movement along the

Y axis.

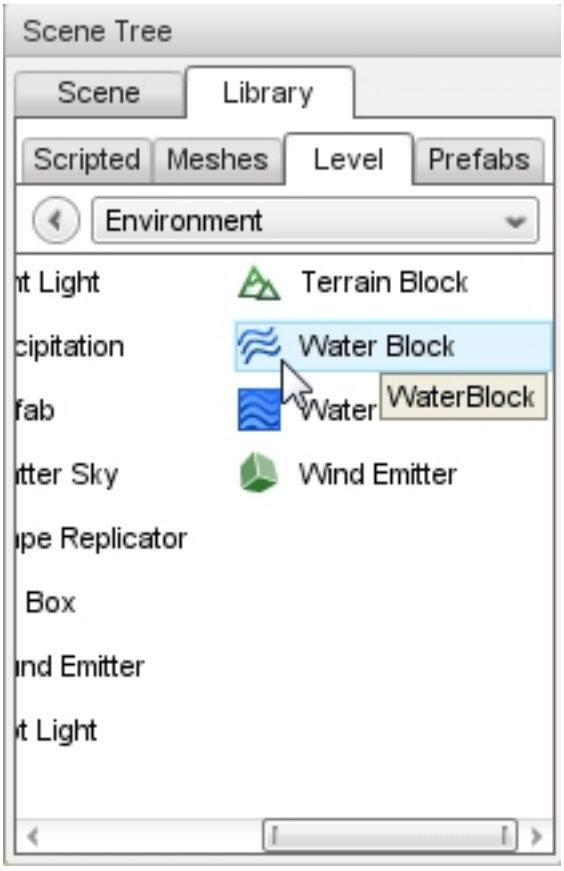
Two properties control how fast the cloud layer moves: texSpeed and windSpeed. The windSpeed property is a global modifier, whereas texSpeed will affect a single layer. The two are added to each other to determine a layer's final speed. If either is set to 0, the cloud layer will not move. The higher the number, the faster your cloud texture will scroll across the sky.

2.3.9 Water Block

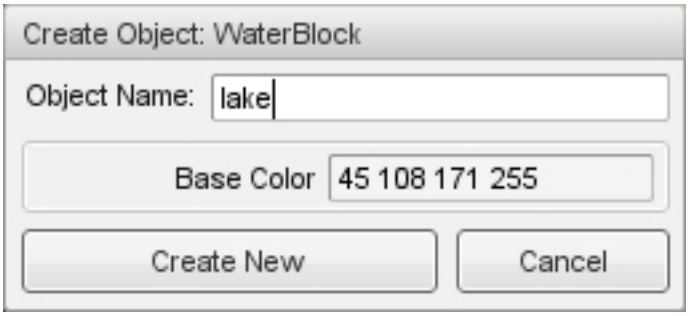
The Water Block object can add a lot of realism to your level's environment. Primarily, you should use a WaterBlock to simulate isolated bodies of water with a limited size. They do not necessarily need to be small, but a WaterPlane can simulate a massive/endless body of water.

Adding a Water Block

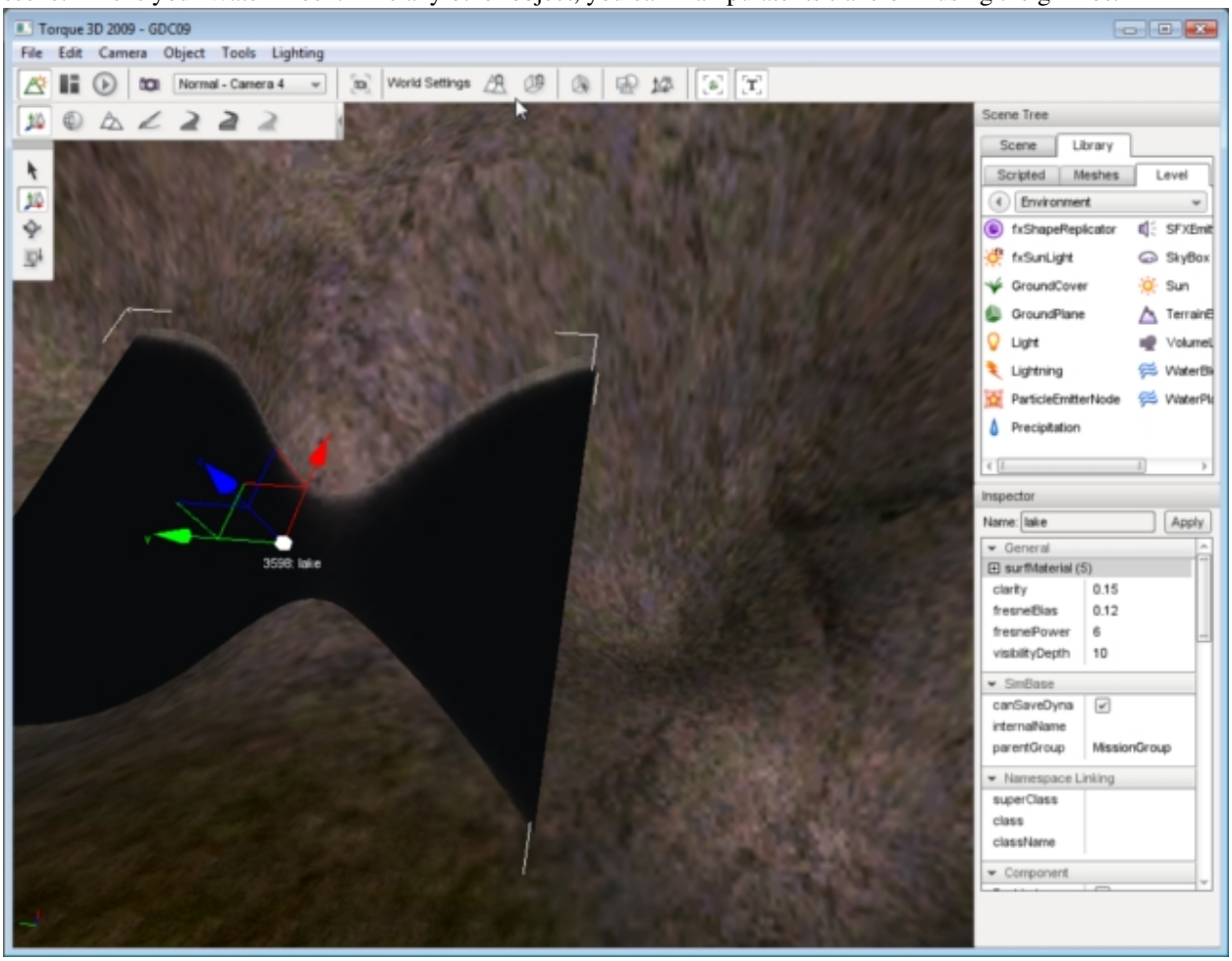
To add a water block, switch to the Object Editor tool. Locate the Library panel and click it. Click on the Level tab and then double-click the Environment folder. Locate the Water Block entry.



Double-click on the Water Block entry. A dialog box will appear:



Enter a name for your Water Block then click the Create New button. A square body of water will be added to the scene. This is your Water Block. Like any other object, you can manipulate its transform using the gizmos.



Water Block Properties

Additional properties can be accessed with the Inspector pane. To change a Water Blocks properties using the Inspector Pane click the Scene tab, then click the name of your new Water Block object. The Inspector pane will update to display the current properties of your new sun.

Inspector

Name TypeName. Optional global name of this object.

id TypeCaseString. SimObjectId of this object. Read Only.

Source Class TypeCaseString. Source code class of this object. Read Only.

Transform

position MatrixPosition. Object world position.

WaterBlock

gridSize TypeF32. Duplicate of gridElementSize for backwards compatibility.

gridElementSize TypeF32. Spacing between vertices in the WaterBlock.

WaterObject

density TypeF32. Affects buoyancy of an object, thus affecting the Z velocity of a player (jumping, falling, etc).

viscosity TypeF32. Affects drag force applied to an object submerged in this container.

liquidType TypeRealString. Liquid type of WaterBlock, such as water, ocean, lava. Currently only Water is defined and used.

baseColor TypeColorI. Changes color of water fog, which is what gives the water its color appearance.

fresnelBias TypeF32. Extent of fresnel affecting reflection fogging.

fresnelPower TypeF32. Measures intensity of affect on reflection based on fogging.

specularPower TypeName. Power used for specularity on the water surface (sun only).

specularColor TypeColorF. Color used for specularity on the water surface (sun only).

emissive TypeBool. When true, the water colors do not react to changes in environmental lighting.

Waves (vertex undulation)

overallWaveMagnitude TypeF32. Master variable affecting entire body of water undulation.

rippleTex TypeImageFilename. Normal map used to simulate small surface ripples.

Ripples (texture undulation)

overallRippleMagnitude TypeF32. Master variable affecting the entire surface of the WaterBlock.

foamTex TypeImage Filename. Diffuse texture for foam in shallow water (advanced lighting only).

Foam

overalFoamOpacity TypeF32. Opacity of foam texture.

foamMaxDepth TypeF32. Maximum depth for foam texture display.

foamAmbientLerp TypeF32. Interpolation for foam settings.

foamRippleInfluence TypeF32. Intensity of the ripples.

Reflect

cubemap TypeCubemapName. Cubemap is used instead of reflection texture if fullReflect is off.

fullReflect TypeBool. Enables dynamic reflection rendering.

reflectivity TypeF32. Overall reflectivity of the water surface.

reflectPriority TypeF32. Affects the sort order of reflected objects.

reflectMaxRateMs TypeF32. Affects the sort time of reflected objects.

reflectDetailAdjust TypeF32. Scale up or down the detail level for objects rendered in a reflection.

reflectNormalUp TypeBool. Always use Z up as the reflection normal.

useOcclusionQuery TypeBool. Turn off reflection rendering when occluded (delayed).

reflectTexSize TypeF32. Texure size used for reflections (square).

Underwater Fogging

waterFogDensity TypeF32. Intensity of underwater fogging.

waterFogDensityOffset TypeF32. Delta, or limit, applied to waterFogDensity.

wetDepth TypeF32. The depth in world units at which full darkening will be received giving a wet appearance.

wetDarkening TypeF32. The refract color intensity scaled at wetDepth.

Misc

depthGradientTex TypeImage filename. 1D texture defining the base water color.

depthGradientMax TypeF32. Depth in world units, the max range of the color gradient texture.

Distortion

distortStartDist TypeF32. Determines start of distortion effect where water surface intersects.

distortEndDist TypeF32. Max distance that distortion algorithm is performed.

distortFullDepth TypeF32. Determines the scaling down of distortion in shallow water.

Basic Lighting

clarity TypeF32. Relative opacity or transparency of the water surface.

underwaterColor TypeColor. Changes the color shading of objects beneath the water surface.

Sound

soundAmbience TypeSFXAmbienceName. Ambient sound environment when listener is active.

Editing

isRenderEnabled TypeBool. Toggles whether the object is rendered.

isSelectionEnabled TypeBool. Toggle whether this object can be selected in the editor.

hidden TypeBool.Toggle visibility in editor.

locked TypeBool. Toggle whether the object can be edited.

Mounting

mountPID TypePID. Unique identifier of the mount.

mountNode TypeS32. Node where the mount occurs.

mountPos TypeS32. Offset for positioning the node.

mountRot TypeS32. Rotation of this object in relation to the mount node.

Object

internalName TypeString. Non-unique name used by child objects of a group.

parentGroup TypeString. Group object belongs to.

class TypeString. Links object to script class namespace.

superClass TypeString. Links object to script super class (parent) namespace.

Persistence

canSave TypeBool. Toggle whether the object can be saved in the editor.

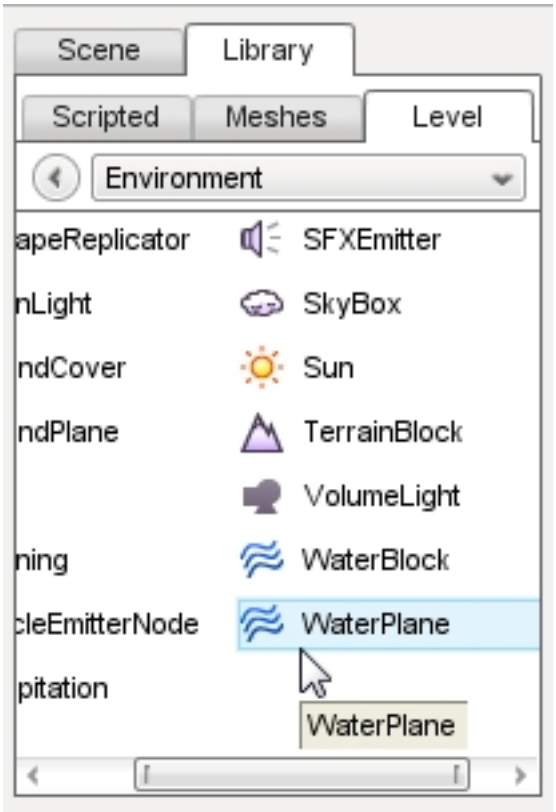
canSaveDynamicFields TypeBool. True if dynamic fields (added at runtime) should be saved, defaults to true.

2.3.10 Water Plane

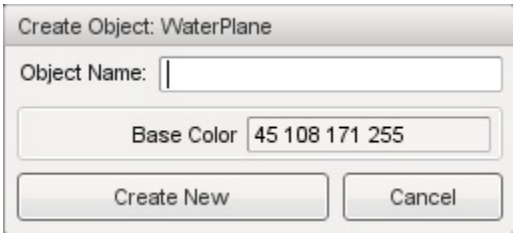
Like a Water Block, the Water Plane object can add realism to the environment of your level. However, the Water Plane is an infinite body of water with an adjustable height. The moment you add this object to your scene, everything below the height of the Water Plane will be submerged, similarly to a Water Block, but the Water Plane has no edges.

Adding a Water Plane

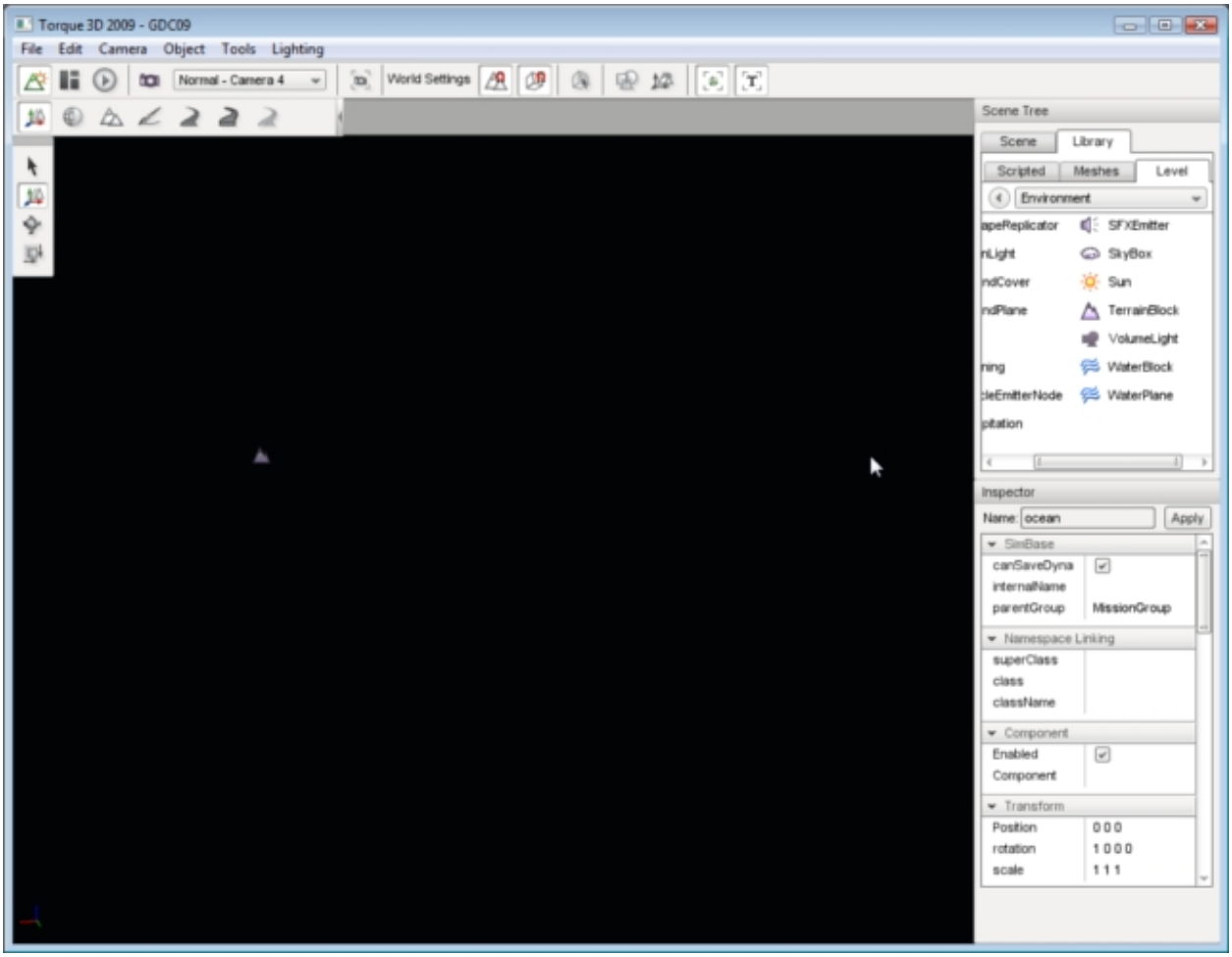
To add a Water Plane, select the Library tab in the Scene Tree Panel. Select the Level tab and double-click the Environment folder. Locate the WaterPlane object.



Double-click the WaterPlane entry. the Create Object dialog window will appear.



Enter a name for your Water Plane, then click the Create New button. A new Water Plane will be added to your scene. If your entire screen fills up with a dark blue or black color, this means the WaterPlane was placed above your camera so that your camera is underwater. Just move your camera up to get out of the water.



This is typical if your objects are set to drop at or above your camera in the Object > Drop Location sub-menu.

Water Plane Properties

Additional properties can be changed with the Inspector pane and are identical to what you will find in a WaterBlock. To change a Water Planes properties using the Inspector Pane, click the Scene tab, then click the name of your new Water Plane object. The Inspector pane will update to display the current properties of your new Water Plane.

Inspector

Name TypeName. Optional global name of this object.

id TypeCaseString. SimObjectId of this object. Read Only.

Source Class TypeCaseString. Source code class of this object. Read Only.

Transform

position MatrixPosition. Object world position.

WaterPlane

gridSize TypeF32. Duplicate of gridElementSize for backwards compatibility.

gridElementSize TypeF32. Spacing between vertices in the WaterPlane.

WaterObject

density TypeF32. Affects buoyancy of an object, thus affecting the Z velocity of a player (jumping, falling, etc).

viscosity TypeF32. Affects drag force applied to an object submerged in this container.

liquidType TypeRealString. Liquid type of WaterPlane, such as water, ocean, lava. Currently only Water is defined and used.

baseColor TypeColorI. Changes color of water fog, which is what gives the water its color appearance.

fresnelBias TypeF32. Extent of fresnel affecting reflection fogging.

fresnelPower TypeF32. Measures intensity of affect on reflection based on fogging.

specularPower TypeName. Power used for specularity on the water surface (sun only).

specularColor TypeColorF. Color used for specularity on the water surface (sun only).

emissive TypeBool. When true, the water colors do not react to changes in environmental lighting.

Waves (vertex undulation)

overallWaveMagnitude TypeF32. Master variable affecting entire body of water undulation.

rippleTex TypeImageFilename. Normal map used to simulate small surface ripples.

Ripples (texture undulation)

overallRippleMagnitude TypeF32. Master variable affecting the entire surface of the waterplane.

foamTex TypeImage Filename. Diffuse texture for foam in shallow water (advanced lighting only).

Foam

overalFoamOpacity TypeF32. Opacity of foam texture.

foamMaxDepth TypeF32. Maximum depth for foam texture display.

foamAmbientLerp TypeF32. Interpolation for foam settings.

foamRippleInfluence TypeF32. Intensity of the ripples.

Reflect

cubemap TypeCubemapName. Cubemap is used instead of reflection texture if fullReflect is off.

fullReflect TypeBool. Enables dynamic reflection rendering.

reflectivity TypeF32. Overall reflectivity of the water surface.

reflectPriority TypeF32. Affects the sort order of reflected objects.

reflectMaxRateMs TypeF32. Affects the sort time of reflected objects.

reflectDetailAdjust TypeF32. Scale up or down the detail level for objects rendered in a reflection.

reflectNormalUp TypeBool. Always use Z up as the reflection normal.

useOcclusionQuery TypeBool. Turn off reflection rendering when occluded (delayed).

reflectTexSize TypeF32. Texure size used for reflections (square).

Underwater Fogging

waterFogDensity TypeF32. Intensity of underwater fogging.

waterFogDensityOffset TypeF32. Delta, or limit, applied to waterFogDensity.

wetDepth TypeF32. The depth in world units at which full darkening will be received giving a wet appearance.

wetDarkening TypeF32. The refract color intensity scaled at wetDepth.

Misc

depthGradientTex TypeImage filename. 1D texture defining the base water color.

depthGradientMax TypeF32. Depth in world units, the max range of the color gradient texture.

Distortion

distortStartDist TypeF32. Determines start of distortion effect where water surface intersects.

distortEndDist TypeF32. Max distance that distortion algorithm is performed.

distortFullDepth TypeF32. Determines the scaling down of distortion in shallow water.

Basic Lighting

clarity TypeF32. Relative opacity or transparency of the water surface.

underwaterColor TypeColor. Changes the color shading of objects beneath the water surface.

Sound

soundAmbience TypeSFXAmbienceName. Ambient sound environment when listener is active.

Editing

isRenderEnabled TypeBool. Toggles whether the object is rendered.

isSelectionEnabled TypeBool. Toggle whether this object can be selected in the editor.

hidden TypeBool.Toggle visibility in editor.

locked TypeBool. Toggle whether the object can be edited.

Mounting

mountPID TypePID. Unique identifier of the mount.

mountNode TypeS32. Node where the mount occurs.

mountPos TypeS32. Offset for positioning the node.

mountRot TypeS32. Rotation of this object in relation to the mount node.

Object

internalName TypeString. Non-unique name used by child objects of a group.

parentGroup TypeString. Group object belongs to.

class TypeString. Links object to script class namespace.

superClass TypeString. Links object to script super class (parent) namespace.

Persistence

canSave TypeBool. Toggle whether the object can be saved in the editor.

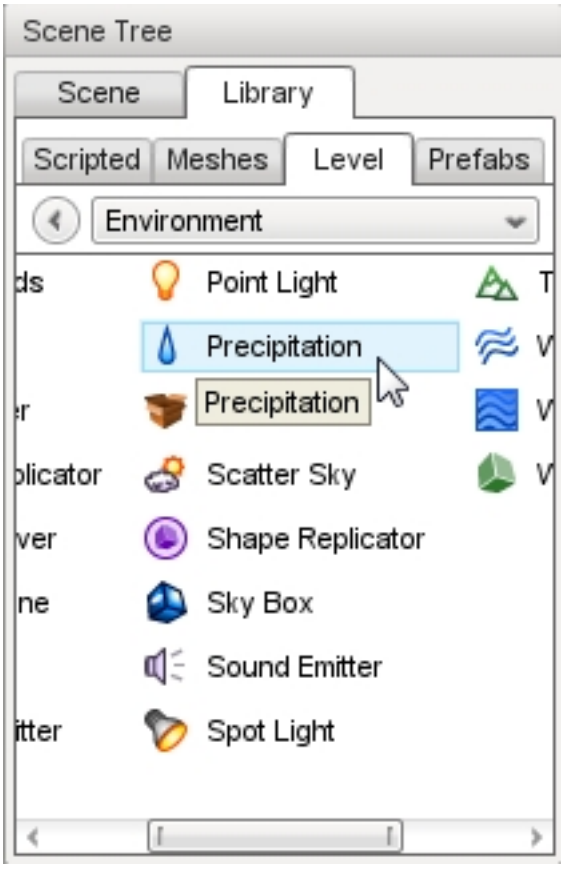
canSaveDynamicFields TypeBool. True if dynamic fields (added at runtime) should be saved, defaults to true.

2.3.11 Precipitation

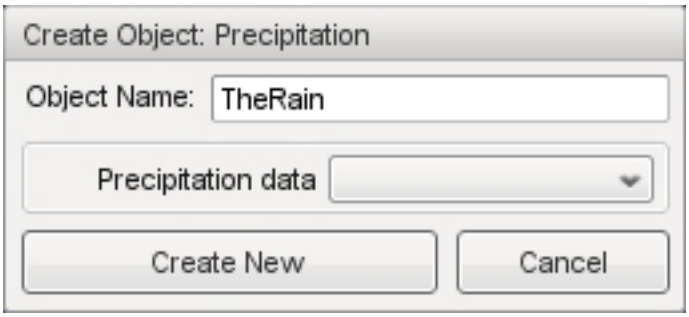
The Torque 3D World Editor allows you to quickly add different types of precipitation to your level. However, Precipitation is used as a general term meaning any type of particle moving downward. The ability to quickly add rain, snow, or even a sandstorm to your level is built into the editor.

Adding Precipitation

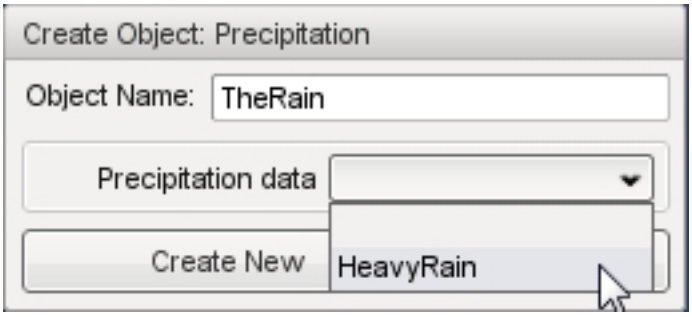
To add Precipitation to a level, switch to the Library tab in the Scene Tree panel. Click on the Level tab amd double-click the Environment folder. Locate the Precipitation entry.



Double-click the Precipitation entry.The Create Object dialog will appear.

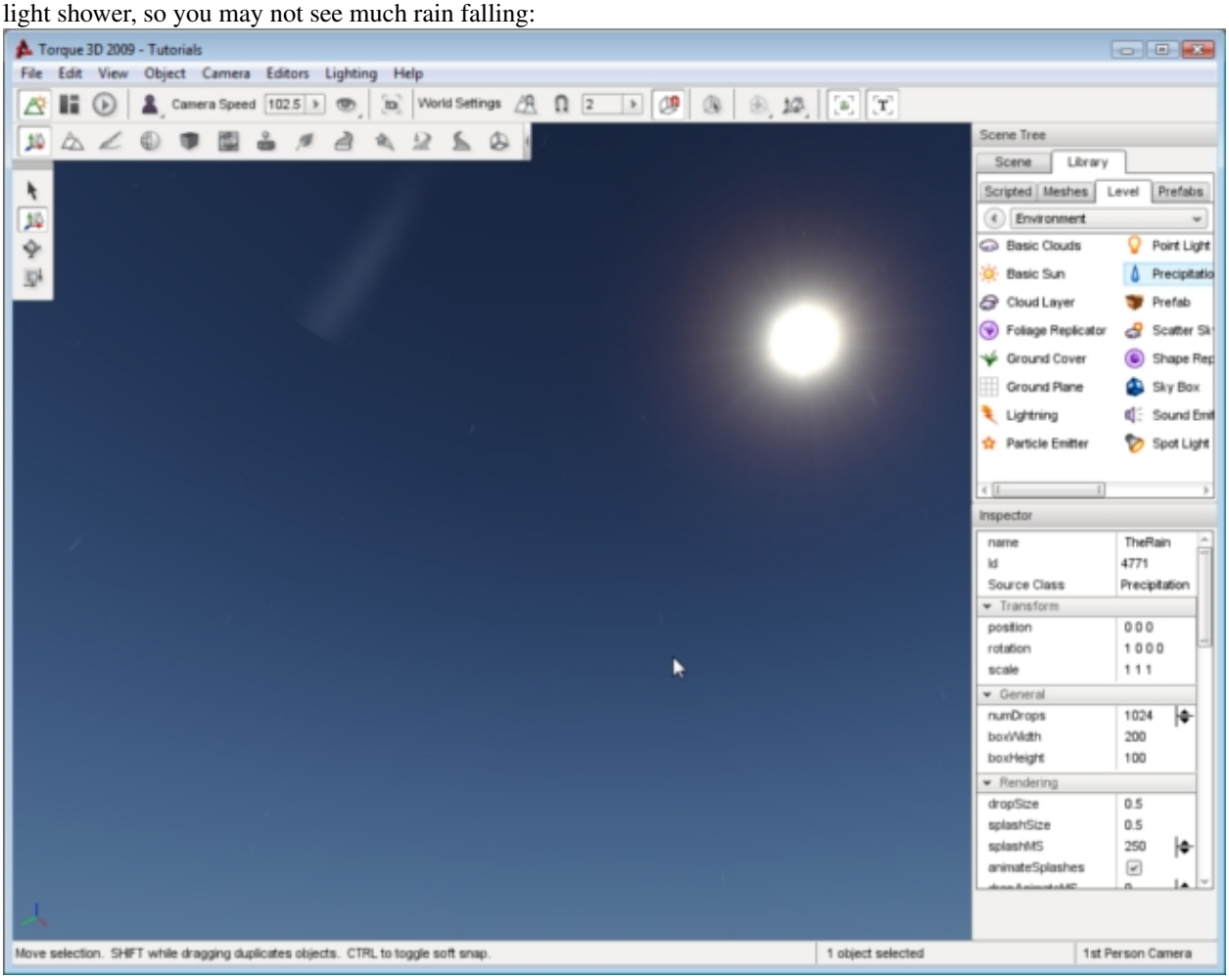


Enter a name for your Precipitation object. The Precipitation data field allows you to choose a datablock to start with as the basis for your new object. Click the drop down box for a list of available datablocks.



For the Full template, your only choice is HeavyRain so select it then click Create New. Your new Precipitation object will be added to your level, and rain will start falling automatically. The stock datablock for HeavyRain simulates a

light shower, so you may not see much rain falling:



The HeavyRain datablock is located in the game/art/datablocks/environment.cs file. Its initial data contains the following:

```
datablock PrecipitationData(HeavyRain)
{
   soundProfile = "HeavyRainSound";

   dropTexture = "art/environment/precipitation/rain";
   splashTexture = "art/environment/precipitation/water_splash";
   dropSize = 0.35;
   splashSize = 0.1;
   useTrueBillboards = false;
   splashMS = 500;
};
```

We will get into manipulating the datablock later.

Precipitation Properties

Additional properties can be changed with the Inspector pane. To change a Precipitation objects properties using the Inspector Pane click the Scene tab, then click the name of your new Precipitation object. The Inspector pane will update to display the current properties of your new sun.

Inspector

name TypeName. Optional global name of this object.

id TypeCaseString. SimObjectId of this object. Read Only.

Source Class TypeCaseString. Source code class of this object. Read Only.

Transform

position MatrixPosition. Object world position.

rotation MatrixOrientation. Object world orientation.

scale Point3F. Object world scale.

Precipitation

numDrops TypeS32. Number of drops allowed to exists in the precipitation box at any one time.

boxWidth TypeF32. Width of precipitation box.

boxHeight TypeF32. Height of precipitation box.

Rendering

dropSize TypeF32. Size of each drop of precipitation. This will scale the texture.

splashSize TypeF32. Size of each splash animation for when a drop collides.

splashMS TypeS32. Life of splashes in milliseconds.

animateSplashes TypeS32. Check to enable splash animation on collision.

dropAnimateMS TypeS32. If greater than zero, will animate the drops from the frames in the texture.

fadeDist TypeF32. The distance at which fading of the drops begins.

fadeDistEnd TypeF32. The distance at which fading of the particles ends.

useTrueBillboards TypeBool. Check to make drops true (non axis-aligned) billboards.

useLighting TypeBool. Check to enable shading of the drops and splashes by the sun color.

glowIntensity TypeColorF. Set to 0 to disable the glow or or use it to control the intensity of each channel.

reflect TypeBool. This enables the precipitation to be rendered during reflection passes. This is expensive.

rotateWithCamVel TypeBool. Enables drops to rotate to face camera.

Collision

doCollision TypeBool. Allow collision with world objects.

hitPlayers TypeBool. Allow collision on player objects.

hitVehicles TypeBool. Allow collision on vechiles.

Movement

followCam TypeBool. Enables system to follow the camera or stay where it is placed.

useWind TypeBool. Check to have the Sky property windSpeed affect precipitation.

minSpeed TypeF32. Minimum speed that a drop will fall.

maxSpeed TypeF32. Maximum speed that a drop will fall.

minMass TypeF32. Minimum mass of a drop.

mMaxMass TypeF32. Maximum mass of a drop.

Turbulence

useTurbulence TypeBool. Check to enable turubulence. This causes precipitation drops to spiral while falling.

maxTurbulence TypeF32. Radius at which precipitation drops spiral when turbulence is enabled.

turbulenceSpeed TypeF32. Speed at which precipitation drops spiral when turbulence is enabled.

Game

dataBlock TypeGameBaseData. Script datablock used for game objects.

2.3.12 Lightning

TODO

Adding Lightning

TODO

Lightning Properties

TODO

2.3.13 Wind Emitter

TODO

Adding Wind Emitter

TODO

Wind Emitter Properties

TODO

2.3.14 Point Light

TODO

Adding Point Light

TODO

Point Light Properties

TODO

2.3.15 Spot Light

TODO

Adding Spot Light

TODO

Spot Light Properties

TODO

2.3.16 Particle Emitter

TODO

Adding Particle Emitter

TODO

Particle Emitter Properties

TODO

2.3.17 Sound Emitter

TODO

Adding Sound Emitter

TODO

Sound Emitter Properties

TODO

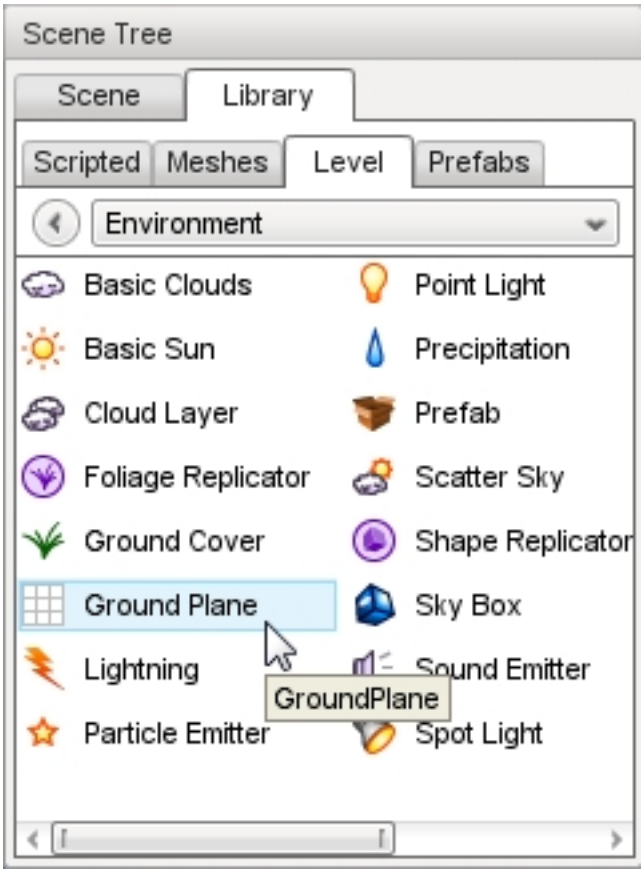
2.3.18 Ground Plane

The Ground Plane object provides you with a solid piece of geometry that acts as floor for a level, while avoiding the use of a terrain object or big plane mesh. You can assign a material to the Ground Plane so that it has a unique appearance. Unlike terrain, a Ground Plane cannot be manipulated like a normal ground surface raising or lowering areas of it.

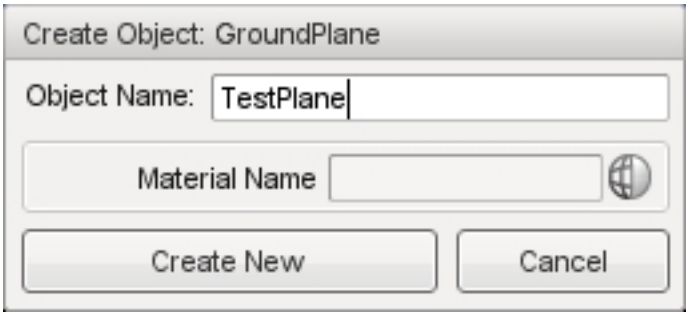
From a practical perspective, the Ground Plane object is most useful for creating prototype missions or experimenting with models or other objects. An example would be creating a demo to show a model or material, when you do not care about the surface you are displaying them on. This type of situation only requires a flat surface within the level to drop them on. A Ground Plane object is a perfect candidate to supply that surface. For actual game play levels, you will most likely want to use a TerrainBlock.

Adding a Ground Plane

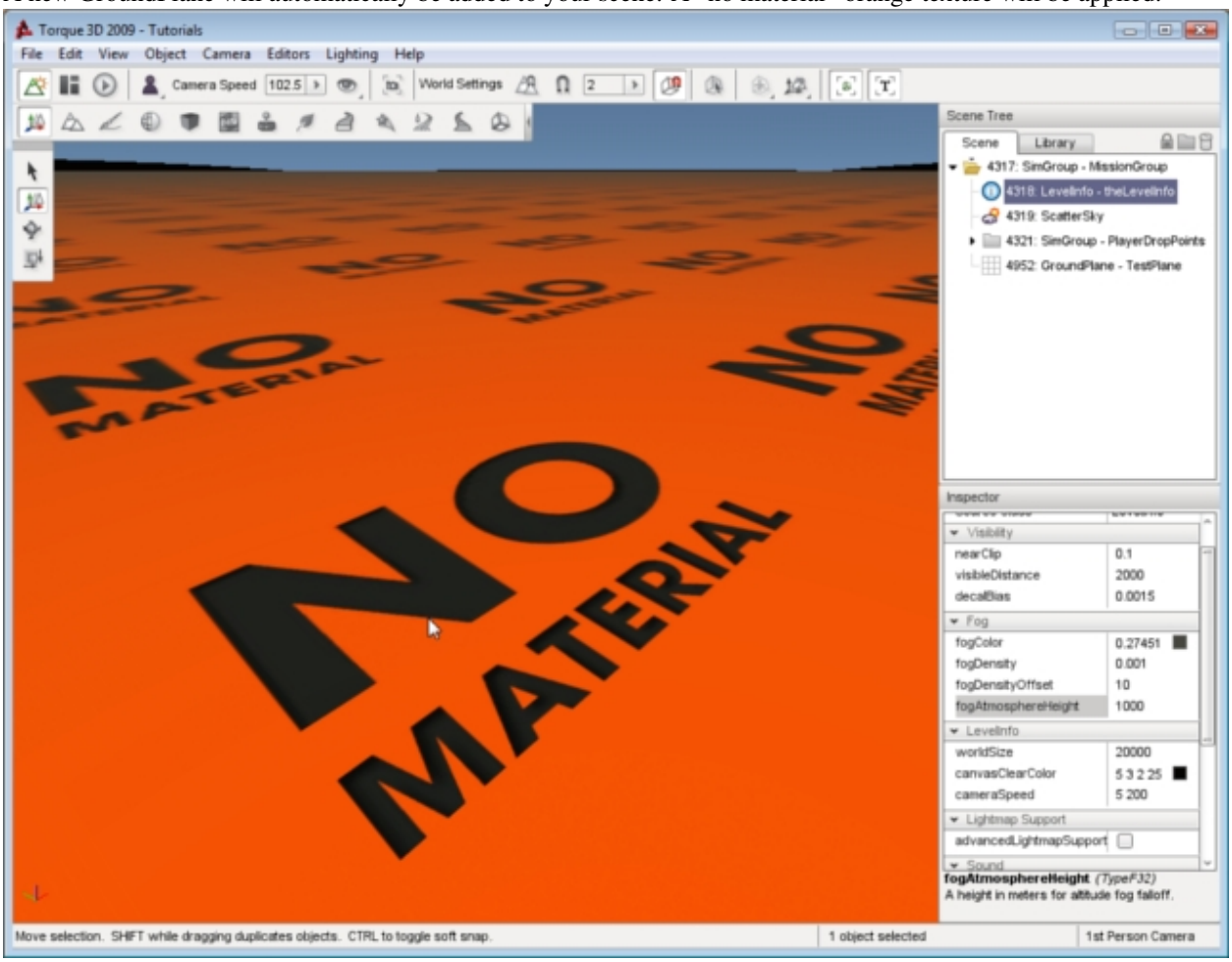
To add a Ground Plane to a level, switch to the Object Editor tool and select the Library tab. Click on the Level tab. Double-click the Environment folder and locate the Ground Plane entry.



Double-click the GroundPlane entry.



A new GroundPlane will automatically be added to your scene. A "no material" orange texture will be applied.



Ground Plane Properties

Properties can be changed with the Inspector pane. To change a Ground Planes properties using the Inspector Pane click the Scene tab, then click the name of your new Ground Plane object. The Inspector pane will update to display the current properties of your new Ground Plane.

Inspector

Name TypeName. Optional global name of this object.

id TypeCaseString. SimObjectId of this object. Read Only.

Source Class TypeCaseString. Source code class of this object. Read Only.

Plane

squareSize F32. World units per grid cell edge.

scaleU F32. Scale factor for U texture coordinates.

scaleV F32. Scale factor for V texture coordinates.

Material TypeMaterialName. Instantiated material based on given material name.

Editing

isRenderEnabled TypeBool. Toggles whether the object is rendered.

isSelectionEnabled TypeBool. Toggles whether the object is selectable in the editor.

hidden TypeBool. Toggles whether the object is visible.

locked TypeBool. Toggles whether the object is editable.

Mounting

mountPID TypeName. PersistentID of the object we are mounted to.

mountNode TypeName. Node object is mounted to.

mountPos TypeName. Position in relation to the mounted node.

mountRot TypeName. Rotation in relation to the mounted node.

Object

internalName typeString. Non-unique name used by child objects of a group.

parentGroup typeString. Group object belongs to.

class TypeString. Links object to a script class namepsace.

superClass TypeString. Links object to a script super class (parent) namespace.

Persistence

canSave TypeName. Optional global name of this object.

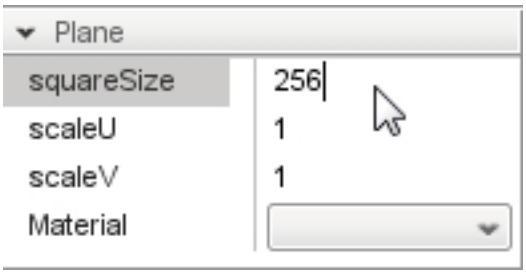
canSaveDynamicFields typeBool. True if dynamic fields (added at runtime) should be saved, defaults to true.

persistentId TypeName. Optional global name of this object.

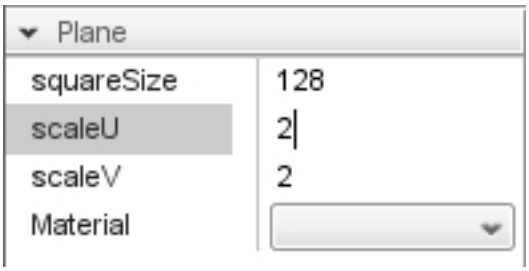
Modifying Scale

The material currently displayed on the object is a general warning texture:

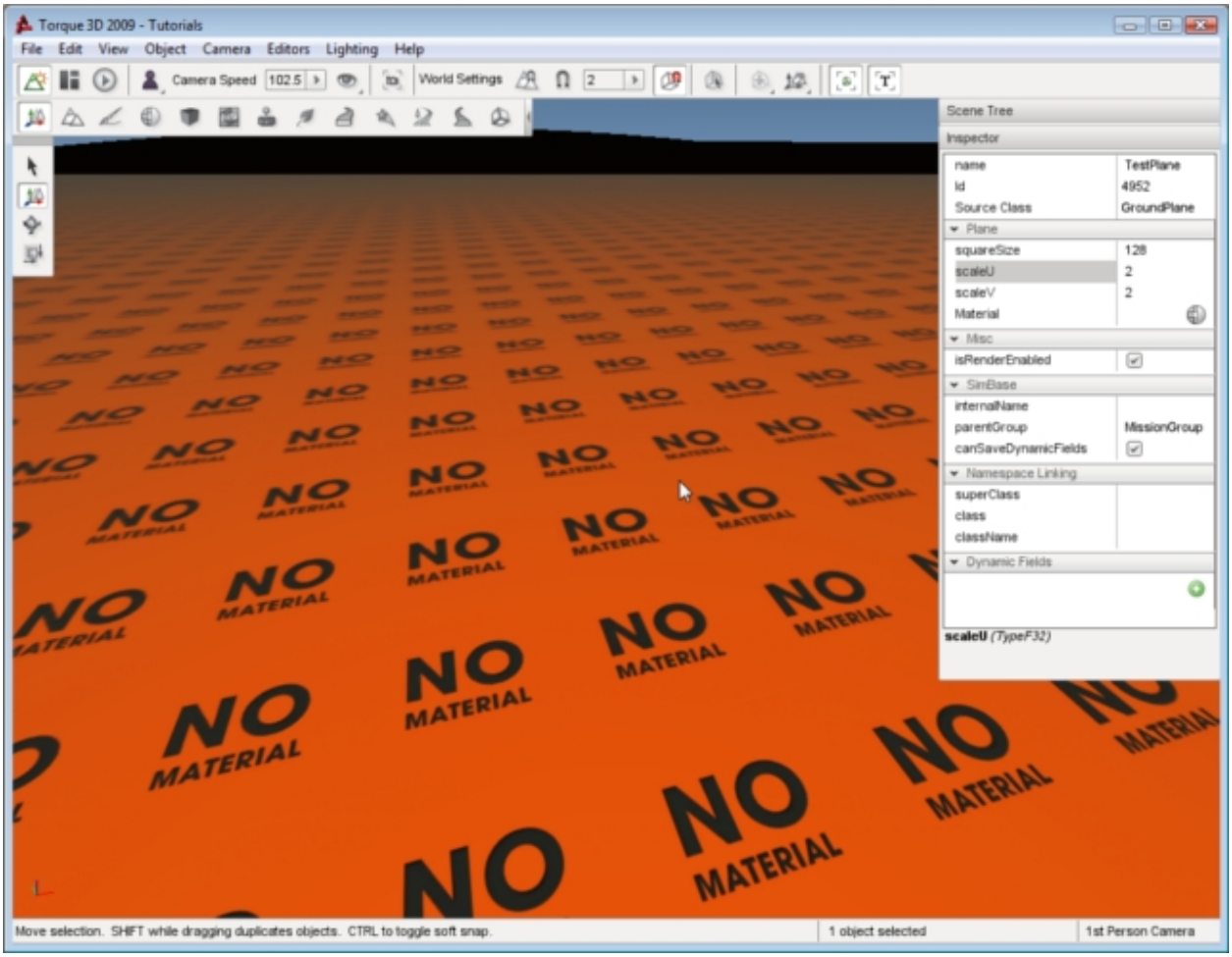
You can change the way this material is tiled across the plane by adjusting the square size and UV scale. Scroll through the properties until you get to the Plane set.



Start by observing the squareSize. At 256, you will notice that each tile is large and stretching the material further. We can push more tiles per meter with tighter UV scaling. Set the squareSize to 128, then set scaleU and scaleV to 2.

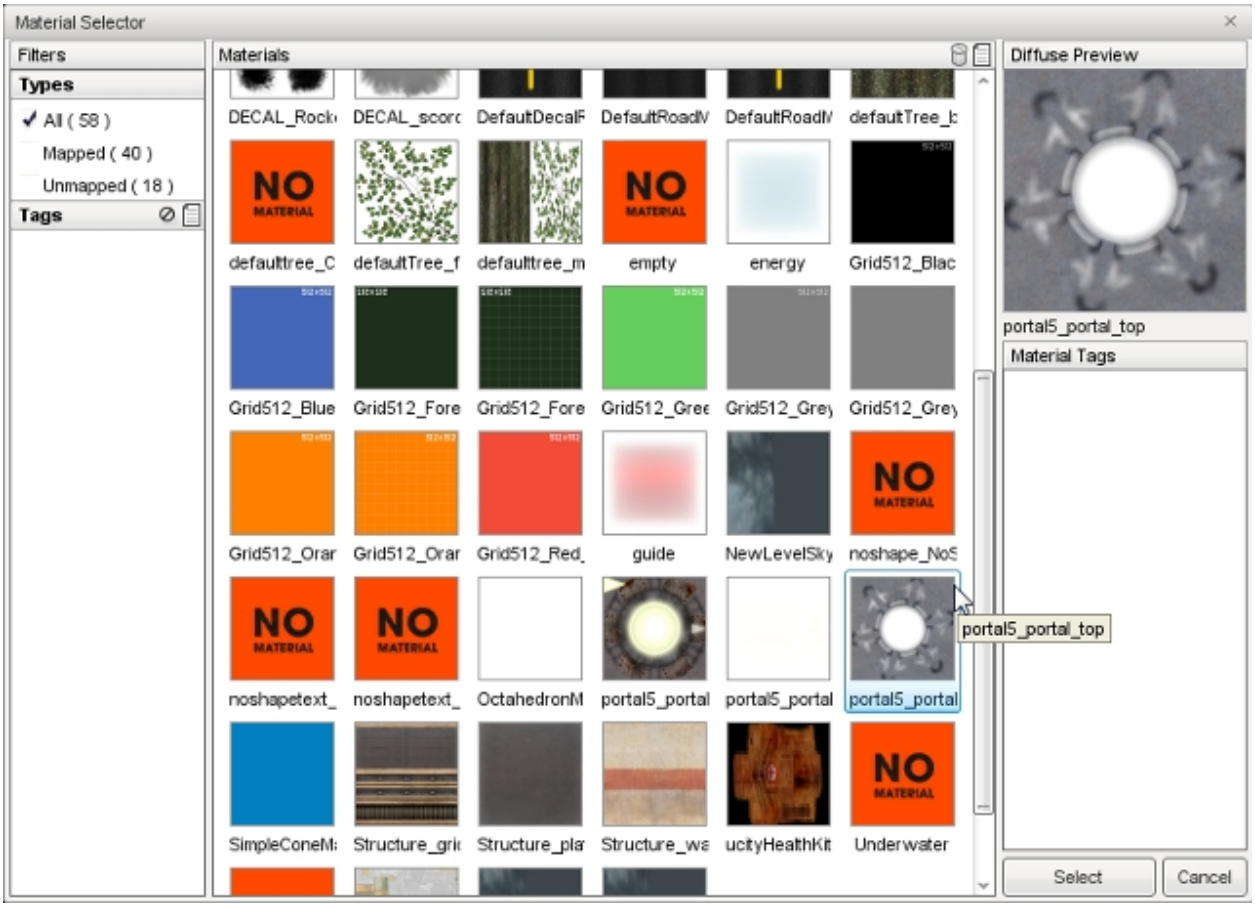


The words on the material are much closer and appear to have been shrunken.

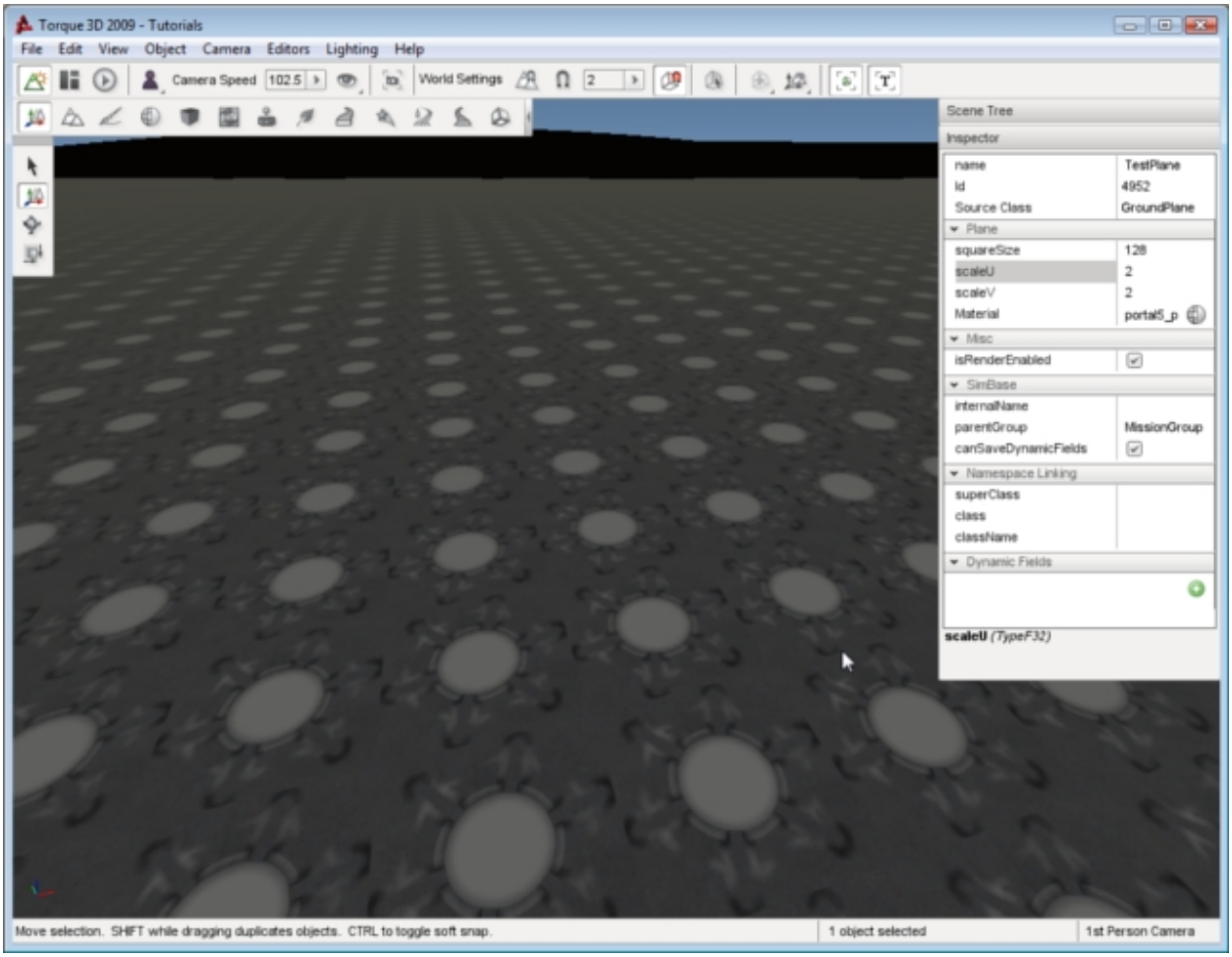


Changing Material

The warning material is a bit of an eyesore, so we will change that now. Click on the Material property in the Plane section of properties to bring up a list of available materials.



Select a material then click the Select button. Your GroundPlane will automatically be updated to use the new material you have selected.



That is the extent of your control over the material displayed on a GroundPlane. If you are using an extremely large texture, you could increase the squareSize and UV scale to make the tiling less blatant.

2.4 Editors

2.4.1 Terrain Editor

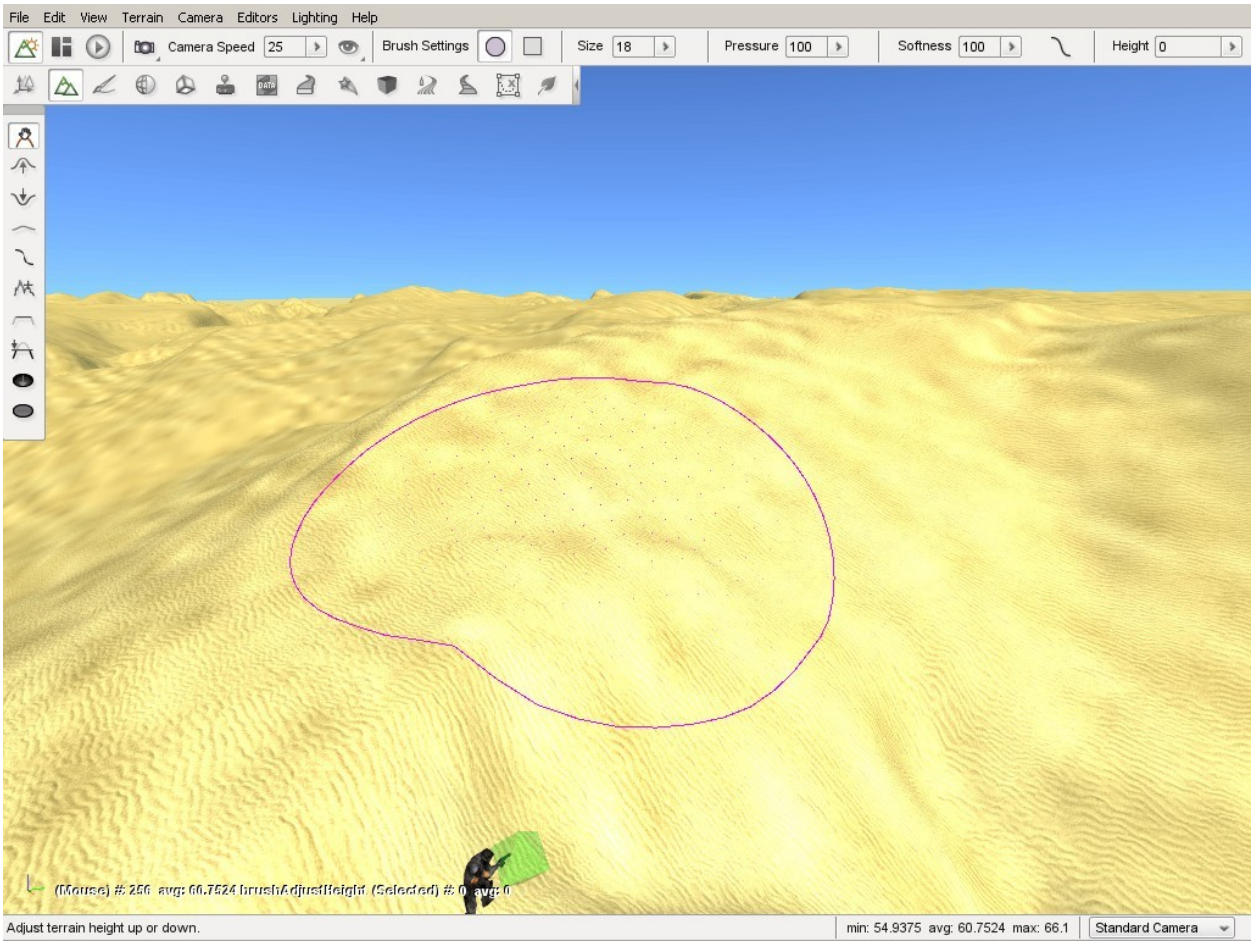
The Terrain Editor is used to modify a TerrainBlock's surface in real time within the World Editor.

With the Terrain Editor, you can elevate, excavate, smooth, flatten, and randomize sections of your terrain as if you were painting on the ground with a simple set of brushes. This is a great way to add the final details and polish to your level before populating it with objects.

The Terrain Editor is a powerful tool and allows for more than just adding hills or holes. You can cut channels for rivers, generate valleys for a mountainous region, turn a rocky mountain chain into a series of smooth hills, and other similar advanced operations.

Interface

To switch to the Terrain Editor press the F2 key or from the main menu select Editors > Terrain Editor.



There are three main areas of the interface. On the far left is the tool palette, which is used to select what kind of modification you wish to make.

Grab Terrain Allows you to manually raise or lower terrain under brush.

Raise Height Adds dirt to terrain beneath brush, thus elevating it.

Lower Height Excavates terrain below brush, thus lowering it and creating holes.

Smooth Smooths jagged terrain beneath brush, creating more rounded elevation.

Smooth Slope Smooth slopes in terrain.

Paint Noise Creates random divots, elevation, and depressions under brush. Used for adding detail.

Flatten Flattens terrain, elevated or excavated, to the level of brush's starting point.

Set Height Set terrain to fixed height.

Clear Terrain Make holes in terrain.

Restore Terrain Cover holes in terrain made with the Clear Terrain tool.

At the top, the Toolbar has updated to show various brush options. The brush options will change the intensity and pattern of your terrain modification.

Shape Toggles between a round or square brush.

Size Changes the size of the grid that makes up the brush, increasing or decreasing the amount of terrain being modified.

Pressure Determines the amount of modification being applied to the terrain.

Softness Determines how much of the brush is affected by the pressure and intensity.

Softness Curve Customize how softness is applied.

Height The brush starting height.

Finally, while moving the mouse over the terrain, you will see a circle drawn around the mouse cursor. This is the Terrain Editor brush and is controlled by your mouse. You can use this brush to "paint" your terrain adjustments by left clicking and dragging the cursor.

Brush Settings

Now we will go into deeper explanation of how the brush options can affect your terrain editing, and how the editor lets you know what you are doing. On the toolbar at the top of the screen, find the Brush Settings section.

Shape

You should see a circular icon and square icon. Toggling between the two will change the shape of your brush.

Size

In the Size section you will find a box with a number in it. When you click on the arrow, a slider will appear. This slider goes from 0 to 100, and it changes the size of your brush allowing it to modify larger sections of the terrain.

Pressure

The Pressure setting is a decimal number ranging from 0.0 to 100.0, and determines how much change is applied to the terrain under the brush.

The easiest way to understand brush pressure is to think of using a real paintbrush on a piece of paper. The harder you press as you stroke, the more paint the brush will leave on the paper. Pressing lightly but going over and over the same spot again and again will also leave more paint behind.

Since you can not press harder or softer on the terrain with a mouse button the Pressure value lets you simulate how much affect the brush has when you are "painting" terrain changes. The higher the Pressure setting, the more dramatic the change will be under the brush over time.

For example: select the Raise Height tool; set the Pressure to 1; place the mouse over the terrain; then quickly press the mouse button and release it. You will see the terrain slightly rise under the brush. Now change the Pressure to 100 and click elsewhere on the terrain in the same manner. You will see the terrain under the brush rise much more quickly and higher than during the first operation.

Like a real paintbrush, our terrain brush left more change behind because its pressure was greater. The same change can be accomplished by clicking the same spot on the terrain multiple times with a lower setting. The lower the setting, the more control you will have to make accurate changes to the terrain.

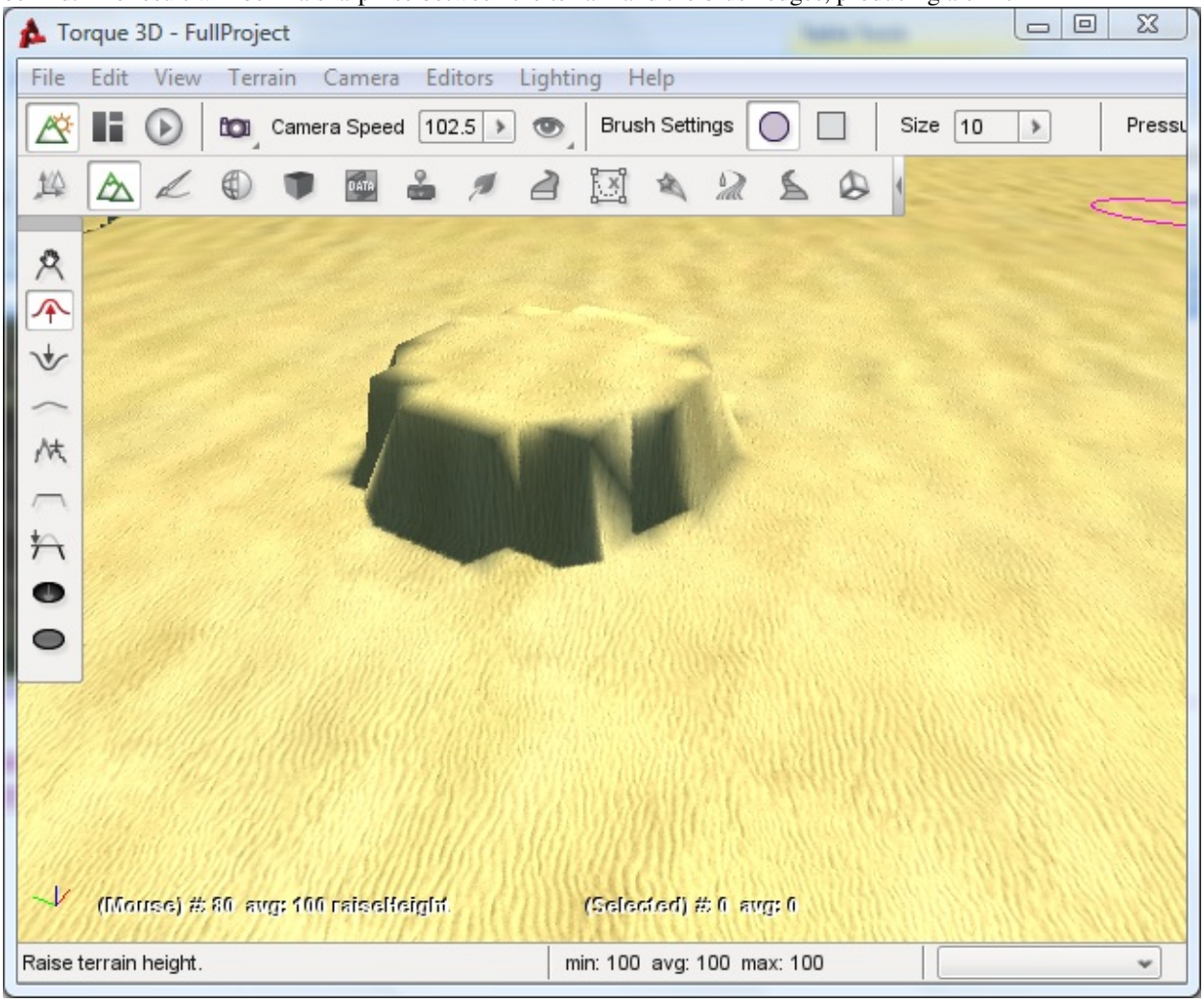
You will also notice that the ground rose higher in the center than around the edges of the brush. Again, this mimics the real world where the pressure around the edges of a paintbrush will be less because there are less bristles, which makes the edges of the brush softer than the center. Therefore the edges will leave less paint behind than the center.

Softness

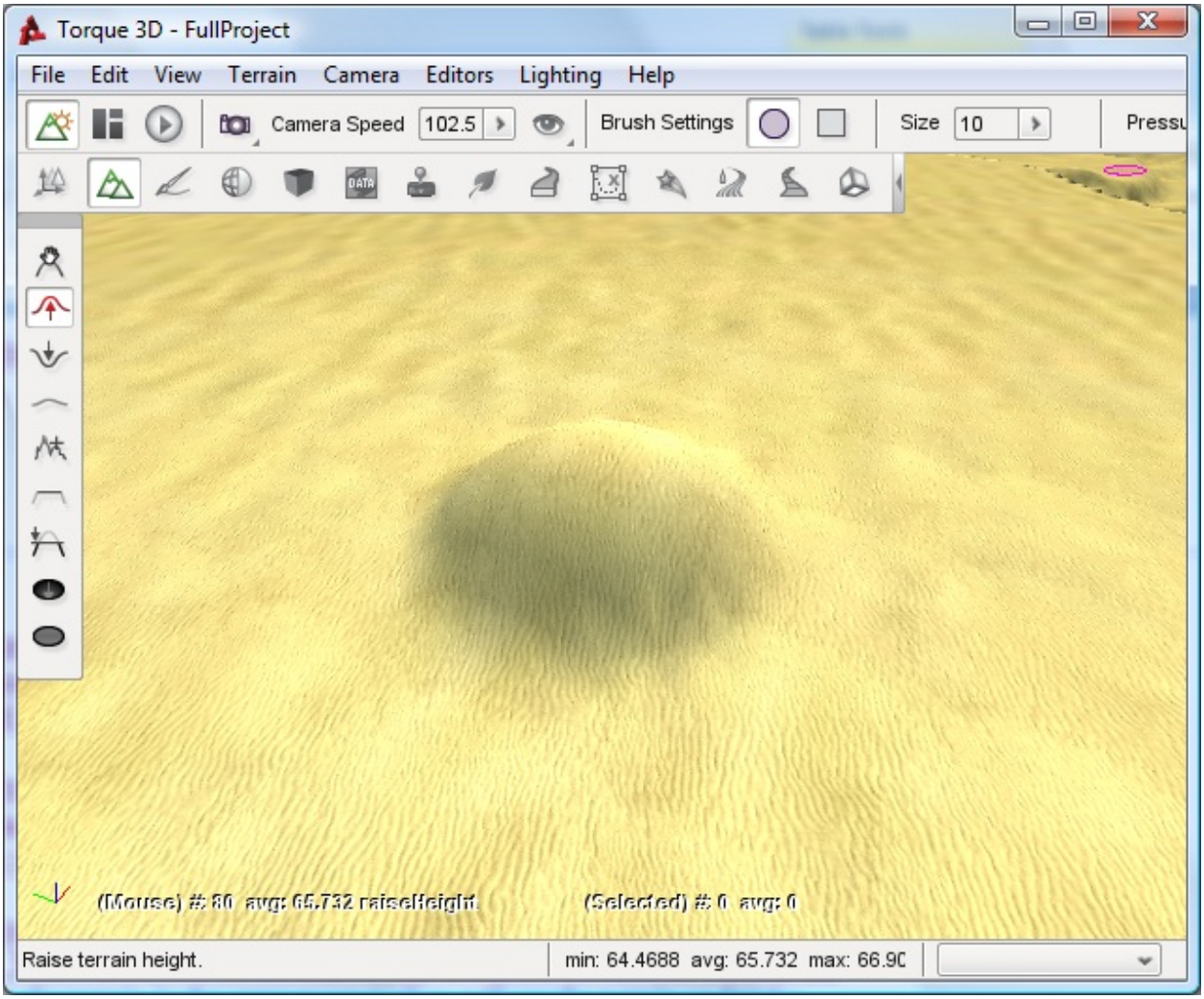
The Softness setting directly affects the intensity of the entire brush's surface. Like the others, it is represented by a slider. The number is a decimal value ranging between 0.0 and 100.0, with 100.0 being the softest and 0.0 the hardest.

The harder the brush is, the more dramatically the terrain under the brush will change. Think of a paint brush that has been dipped in paint and allowed to somewhat dry. The entire brush will be harder and thus will leave more paint behind than the same brush which has not dried and hardened. Since the entire brush has hardened the pattern that it leaves will be the same, that is more paint in the center and less at the edges, but the overall amount of paint will increase across the whole surface. If you allow the brush to completely dry then the entire brush will be the same hardness and thus will leave the same amount of paint across its entire surface.

Because you can not change the softness of a mouse cursor, the Terrain Editor provides the Softness settings to emulate these characteristics. Setting the softness to 0.0, meaning the brush has no softness at all, will result in the entire brush being hard. The edges will be just as hard as the center and so the entire brush will leave the same amount of change behind. The result will be in a sharp rise between the terrain and the brush edges, producing a cliff.



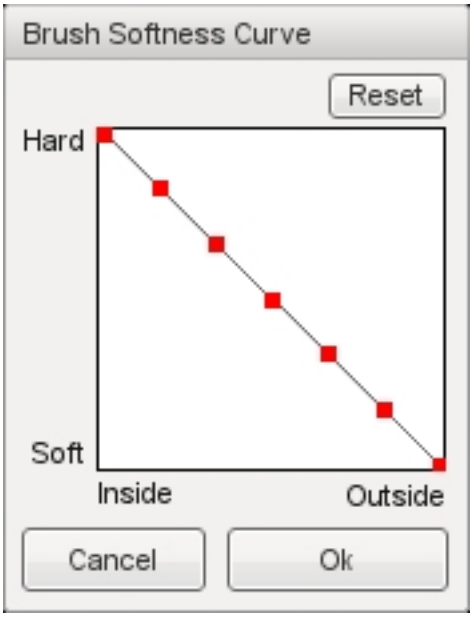
Conversely, if you set the brush to 100.0, meaning maximum softness the brush will exhibit its natural behaviour returning to very soft edges and a harder center. Setting the Softness to 100.00 (maximum softness) will cause the change at the edges to be much less dramatic than the change in the center and will result in a gentle rise from the edges to the center, producing a rolling hill.



Softness Curve

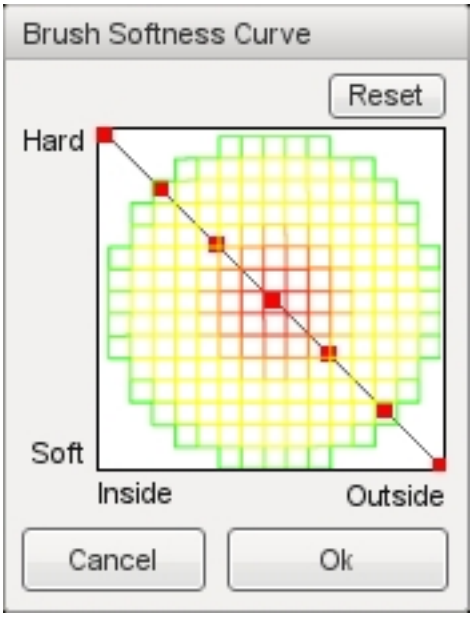
The previous section discussed how changing the Softness affects the brush over its entire surface mimicking the natural effects of a brush which is harder in the center and softer at the edges due to the distribution of its bristles. The Brush Softness Curve allows you to customize this behaviour further by changing the way softness and hardness is distributed within the brush.

Click the curved line next to the brush Softness slider. The Brush Softness Curve dialog box will appear.



The graph contains multiple nodes which can be moved by clicking and dragging them up or down. Modifying the nodes will determine which parts of your brush are hard or soft. As the graph shows, going from left to right will determine where in the grid you are changing the hardness.

Left nodes are closer to the center of the brush, and each node moving to the right will move toward the outer edges of the brush. The higher a node is situated, the harder it is. The following image is a visual of how this system works:

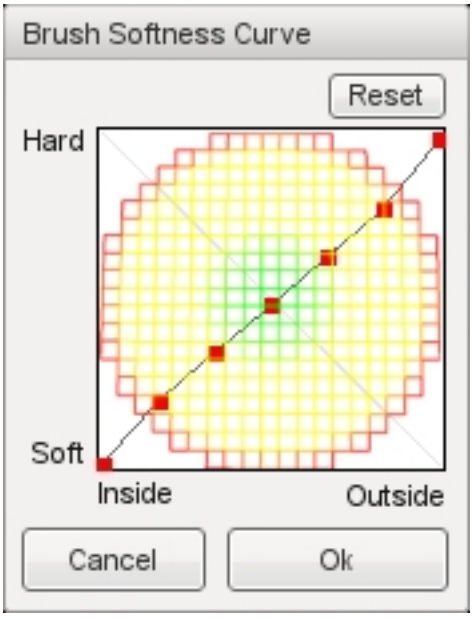


The circular pattern represents the shape of the brush looking straight on at its tip. Hardness of the brush is represented by red, softness by green, and yellow indicates variations in between. The node in the upper left represents the very center of the brush, since it is at the far left on the Inside-Outside axis. Because it is also at the very top of the Hard-Soft axis, it means that the brush is at its hardest at that location. So the combination of these two node positions indicates that the brush is at its hardest (indicated by red) in the very center.

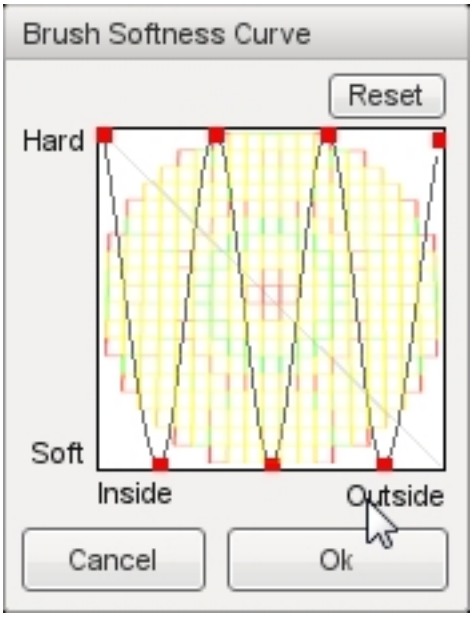
On the other end of the graph line, the node in the lower right represents the very edge of the brush, since it is at the far right on the Inside-Outside axis. Because it is also at the very bottom of the Hard-Soft axis it means that the brush is at its softest at that location. So the combination of these two node positions indicates that the brush is at its softest

(indicated by green) all around the edge.

If you were to drag each node so that the line is reversed, the brush will be softer toward the center and harder toward the edges.



To get an unusual setting, you can create a "wavy" version of the curve. Alternating the nodes in extremes from top to bottom, will result in rings of softness with the brush.

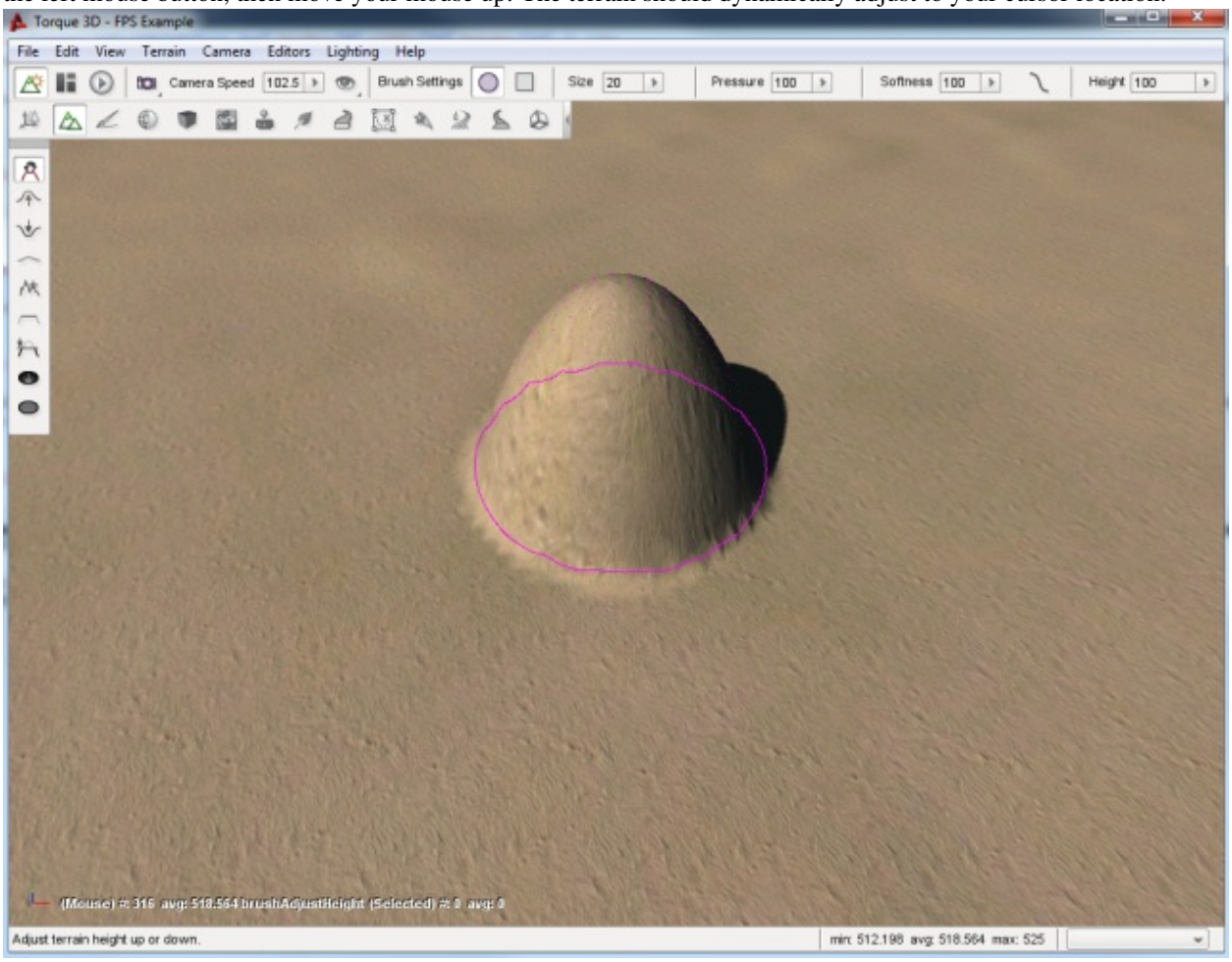


Now that you are familiar with the interface, it is time to edit the terrain.

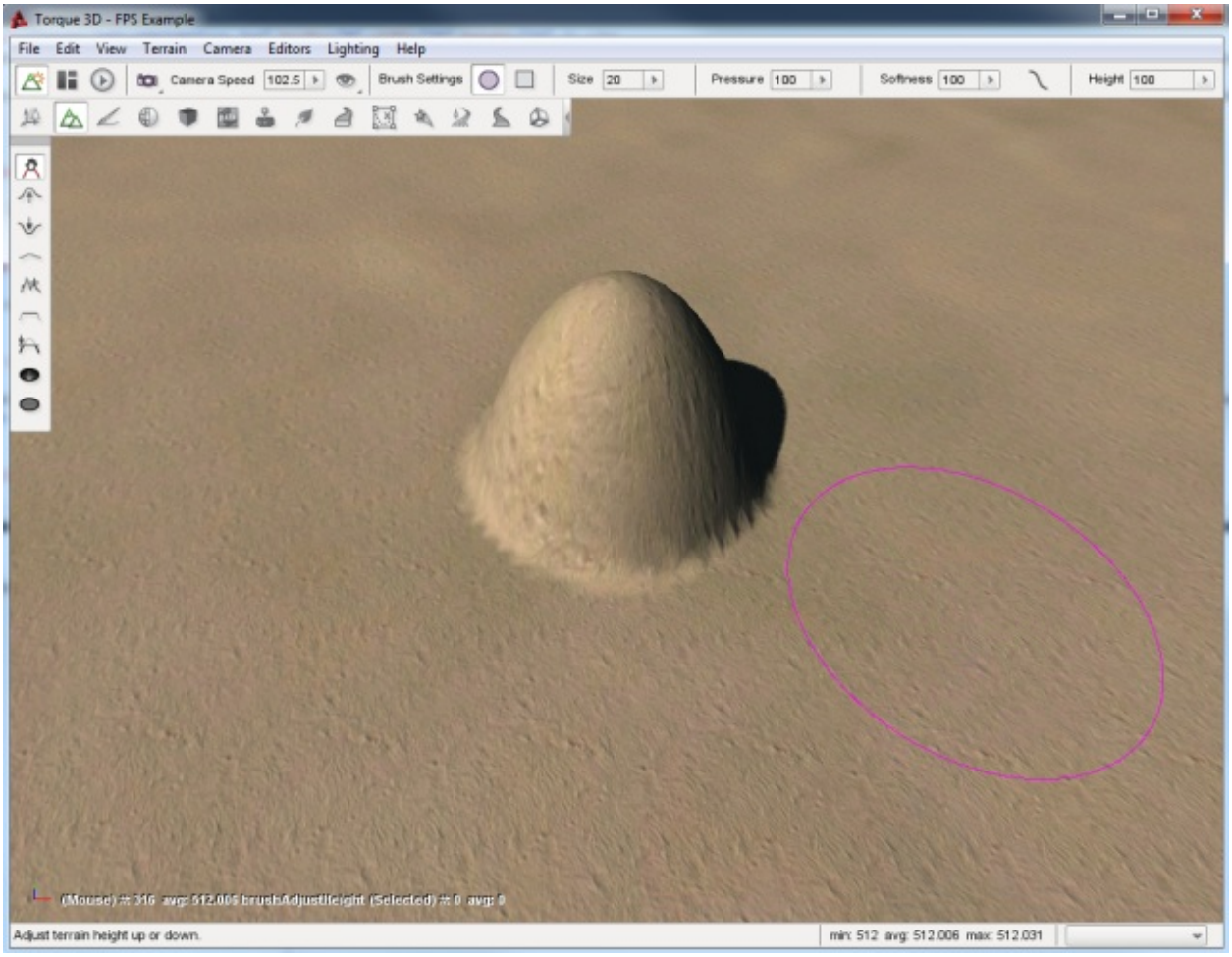
Grab Terrain Tool

Let's start by selecting the Grab Terrain tool from the palette. With the Grab Terrain tool, you can move a section of terrain up or down depending on which direction you are dragging your mouse.

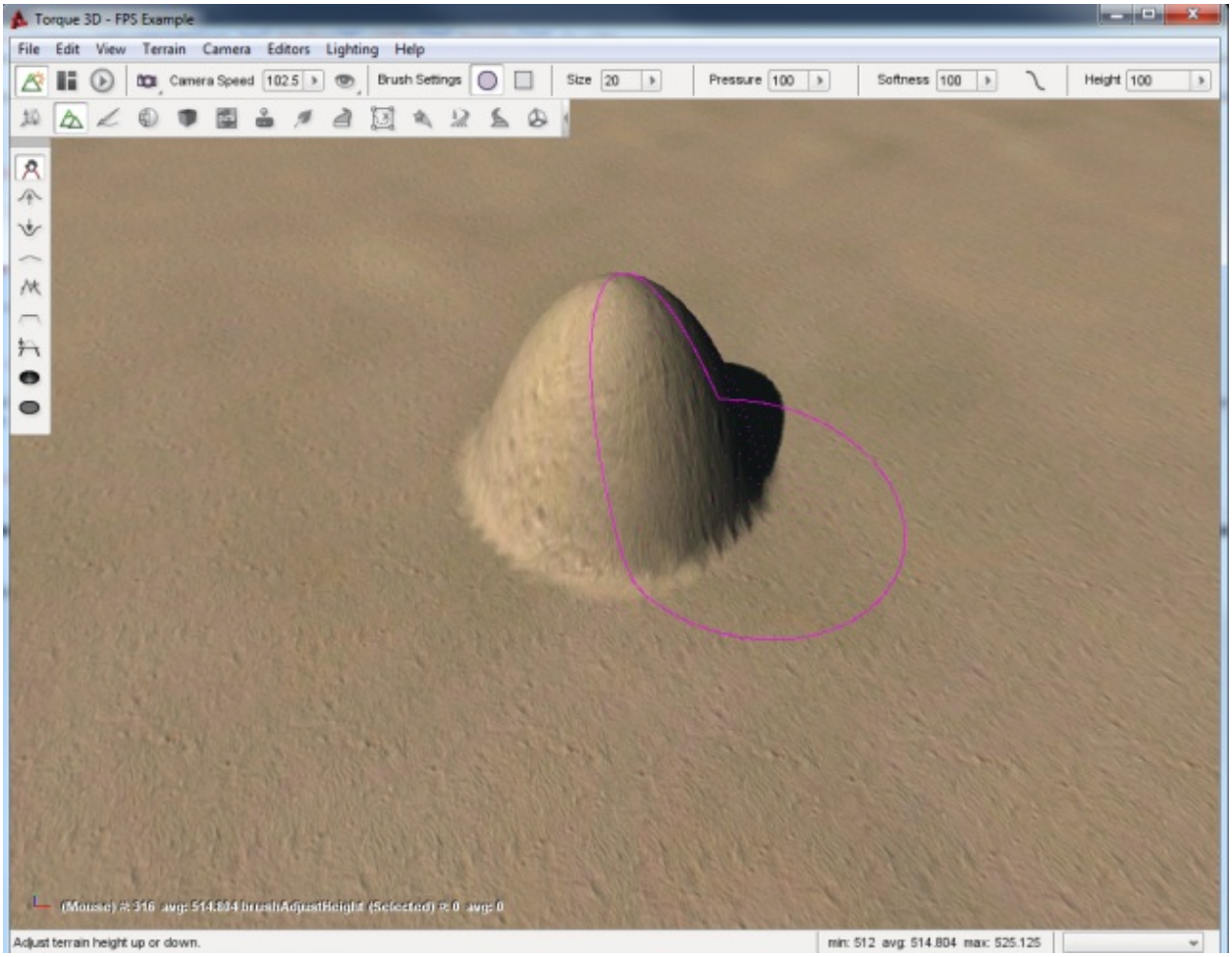
Use the circle brush, size 20, 100 pressure, and 100 softness. Hover your brush over a section of terrain, hold down the left mouse button, then move your mouse up. The terrain should dynamically adjust to your cursor location.



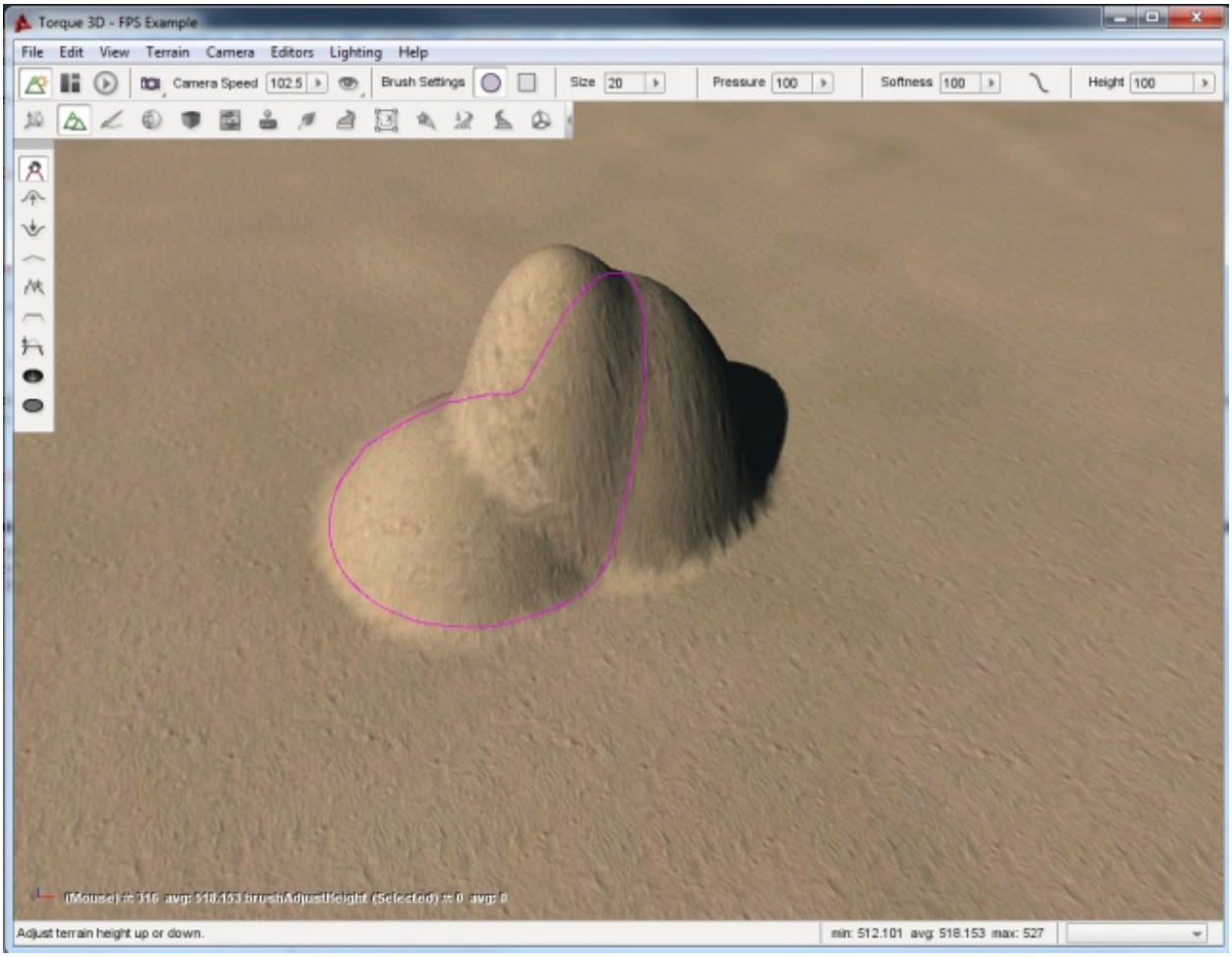
When you are satisfied with the height, let go of the mouse to see your terrain modification.



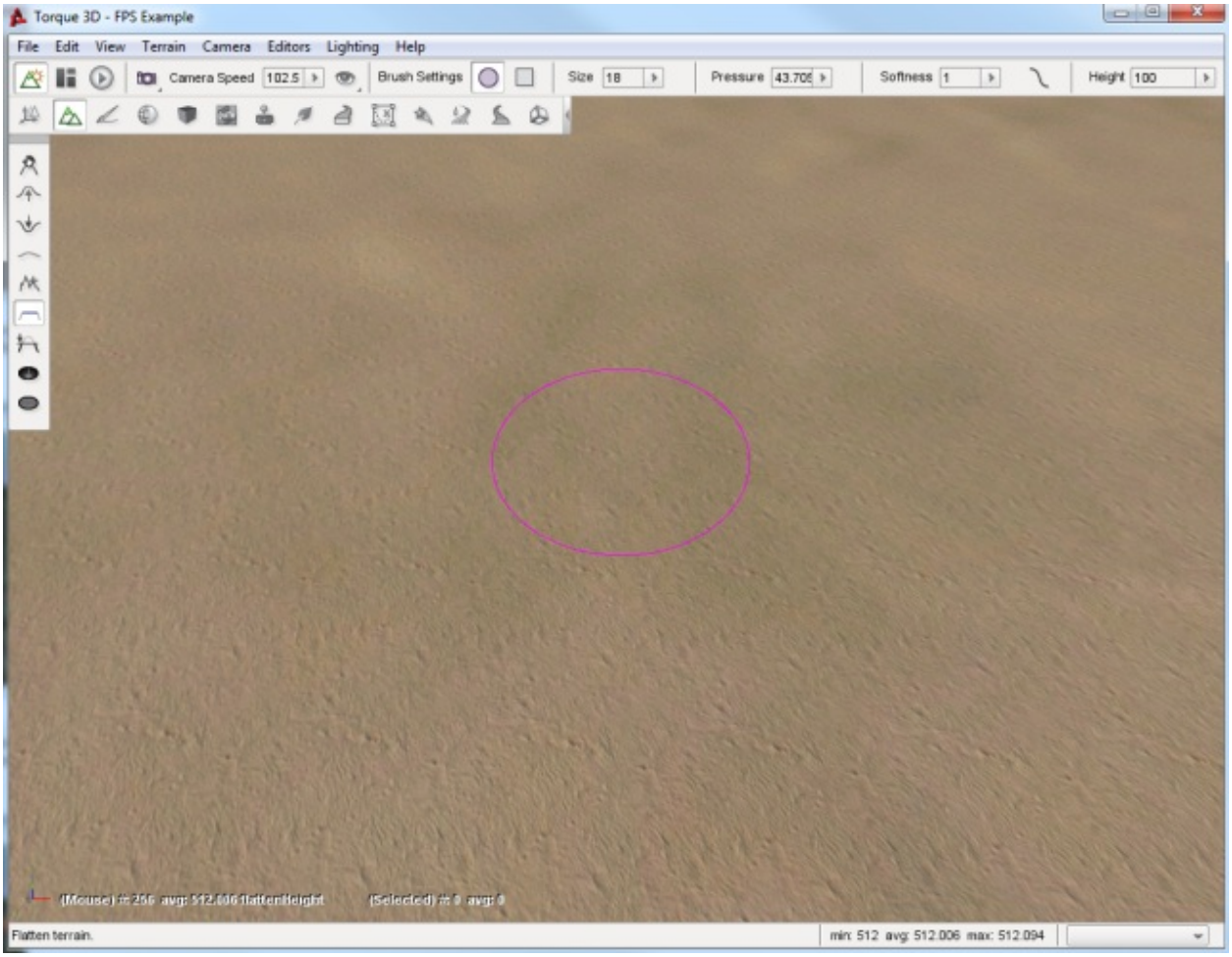
Use the mouse to cover part of your new adjustment with the brush. Notice how the brush clamps to the terrain, maintaining the shape you are using while still selecting a section.



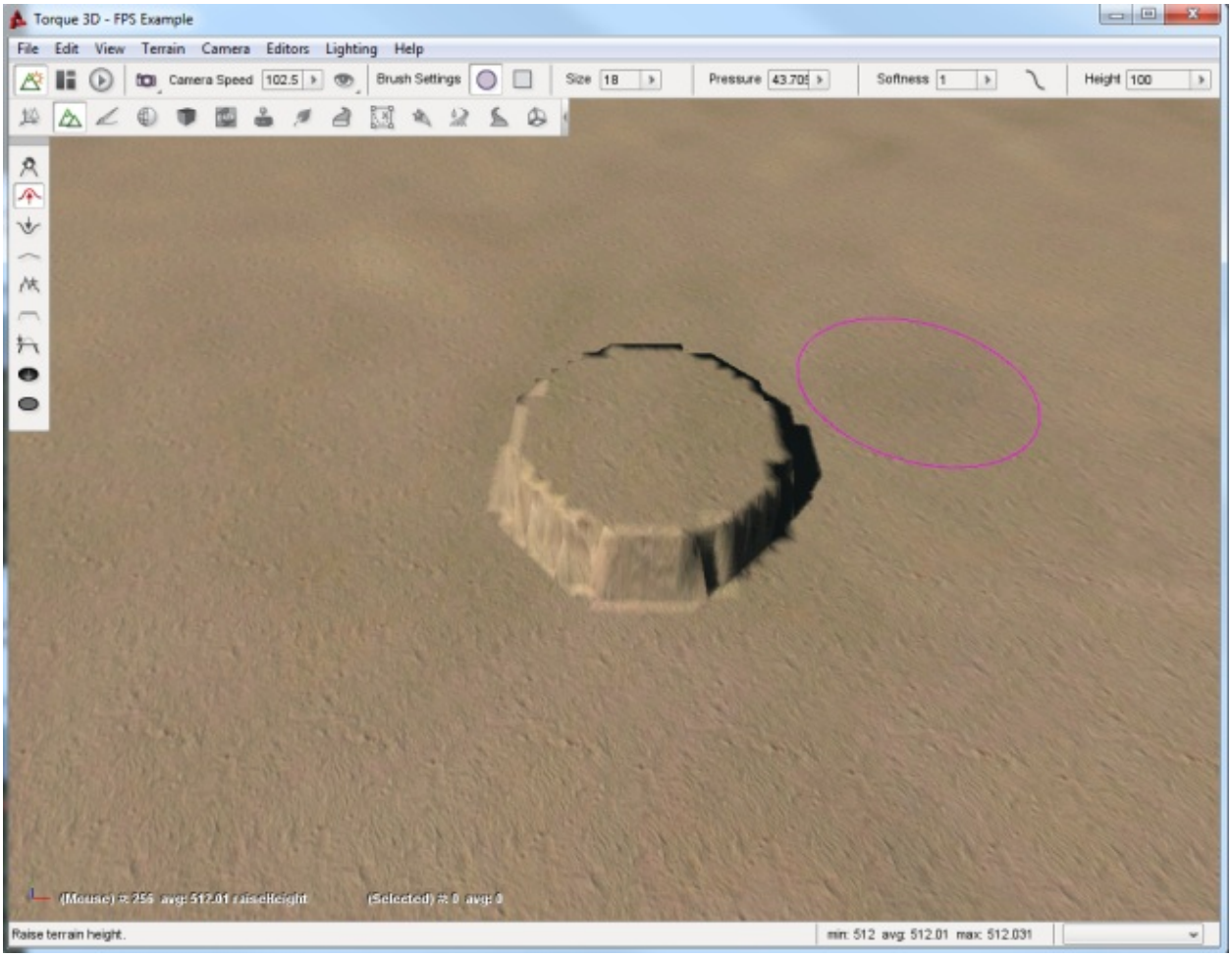
Despite the elevation of your current selection, the section under the hardest part of the brush will still adjust more dramatically. Using the default Softness Curve, if the center of the brush is just to the edge of a hill, you can adjust nearby terrain to match elevation. Terrain under the softer part of the brush will still elevate, but not as much.



Before moving on to the next tool, we will experiment with the softness value. Set the softness of your current brush to 1 (very hard). Move the brush over a flat section of the terrain.



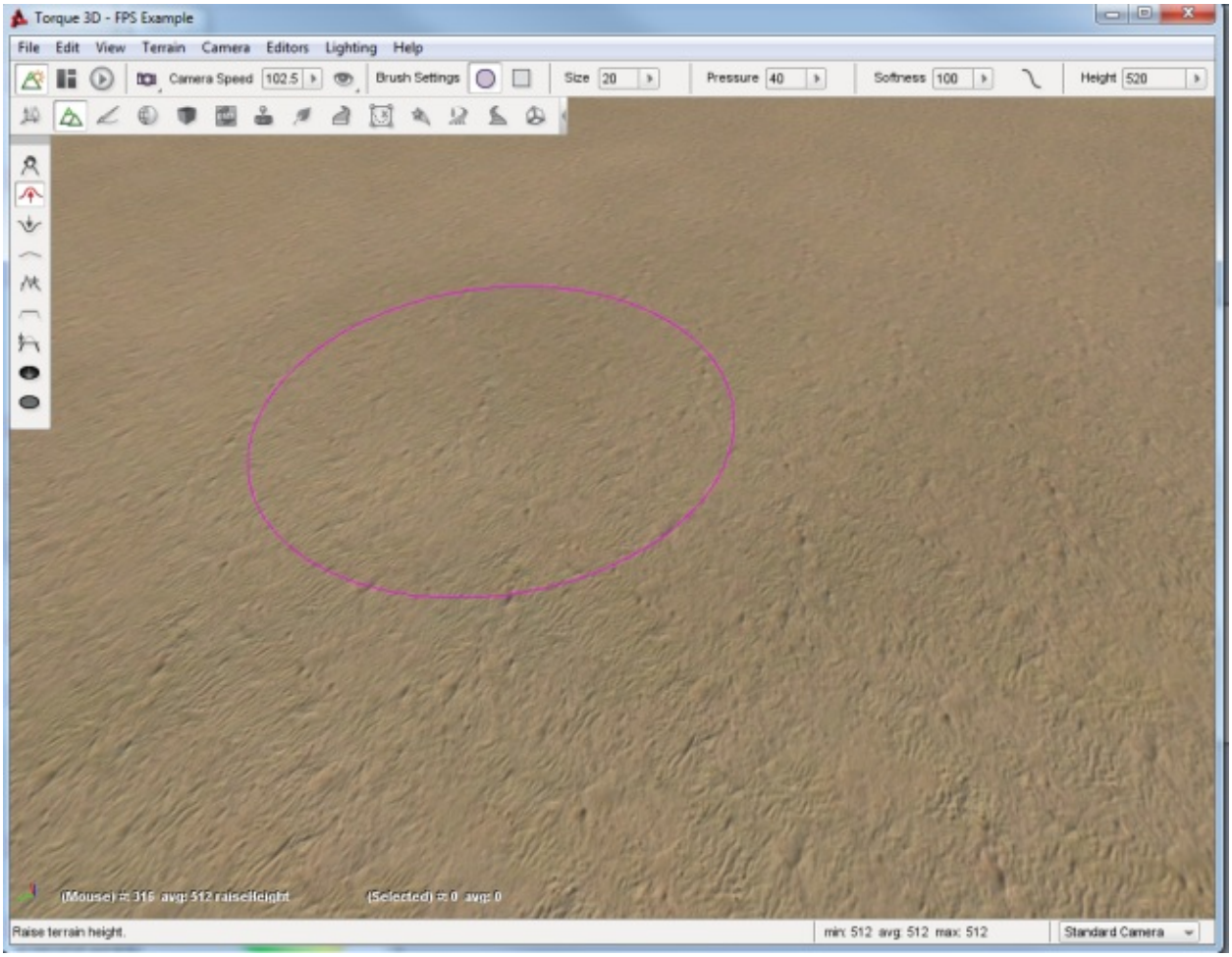
Click on the terrain and drag your mouse up. Instead of an elevated hill with a smooth slope, your brush should have created a flat plateau with completely vertical sides. With a softness of 1, your brush's shape will be used to extrude the terrain in a sharp manner.



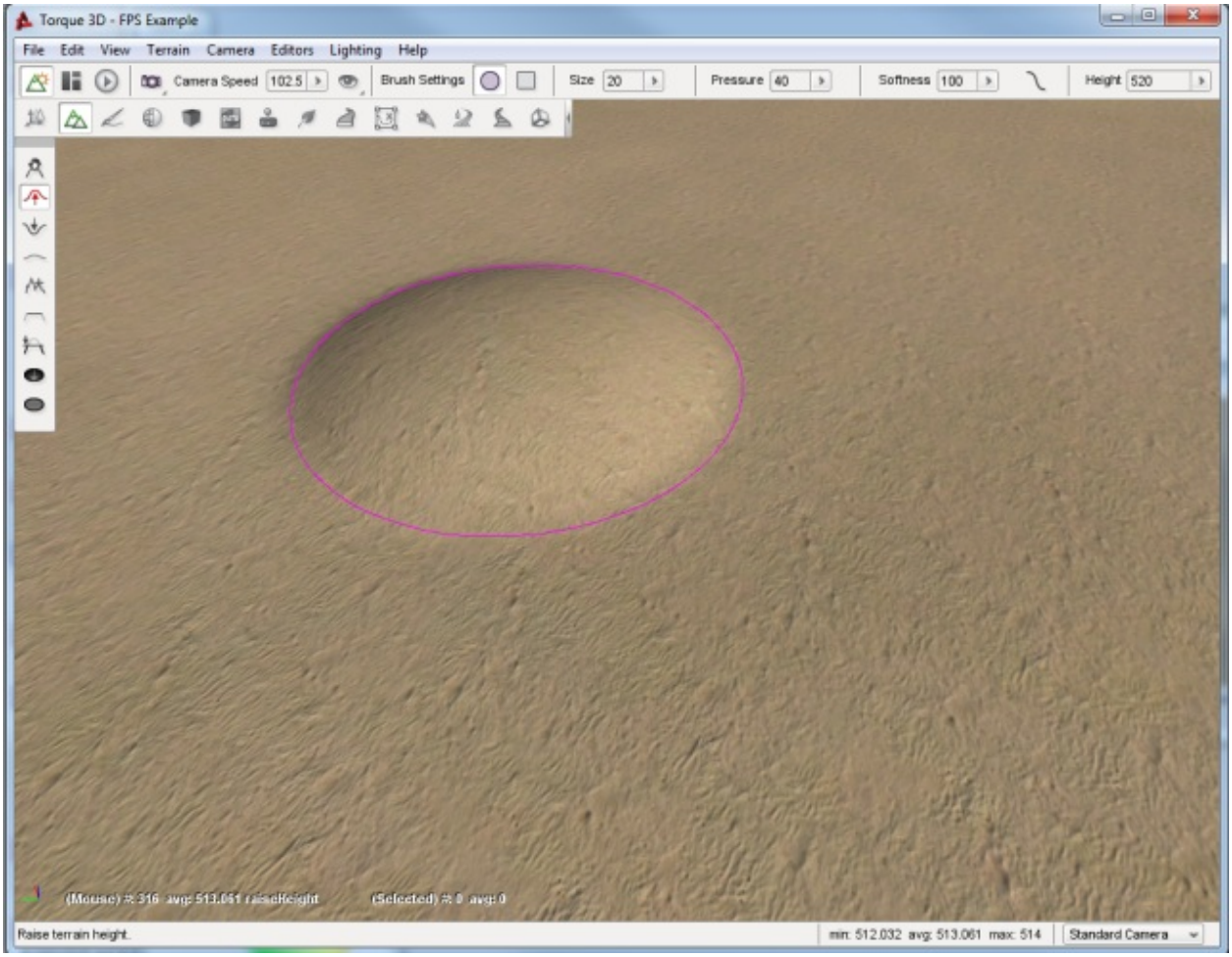
Raise Height Tool

The Raise Height tool can only elevate the terrain, but it does so in a very controlled manner. Instead of manually lifting, you can "paint" the terrain in a sweeping motion by dragging the mouse while holding the left button. The longer you keep the brush in one location, the higher that section will be and the higher the Pressure setting, which was reviewed earlier, the faster it will change.

Set your brush size to 20, pressure to 40, and softness to 100. Find a flat section of the terrain and move your mouse cursor to that location to hover the brush.



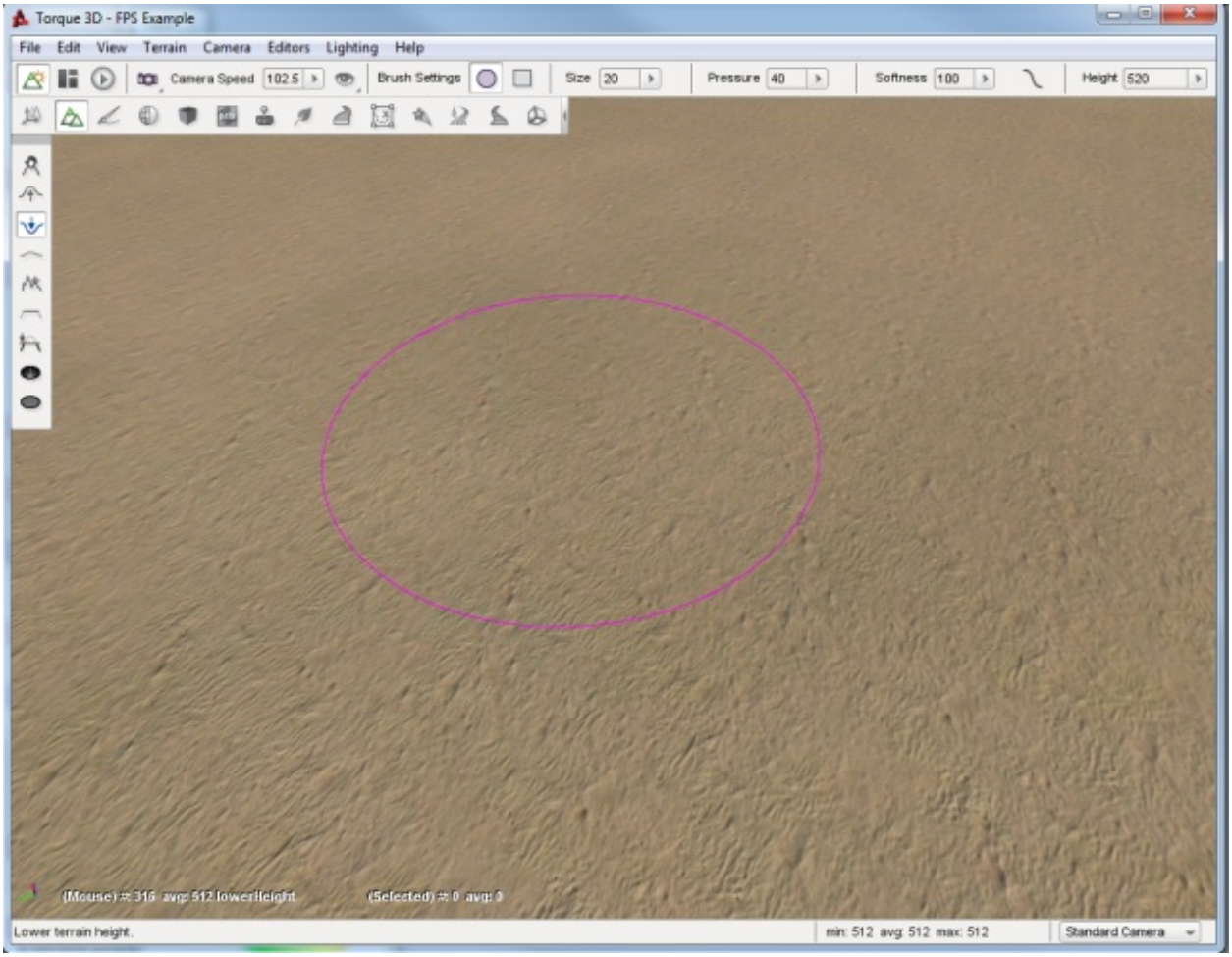
When you are ready, click and hold the left mouse button and begin dragging your brush in a direction. The terrain should elevate wherever your brush passes over. You can use this to create a hill over a long section of terrain.



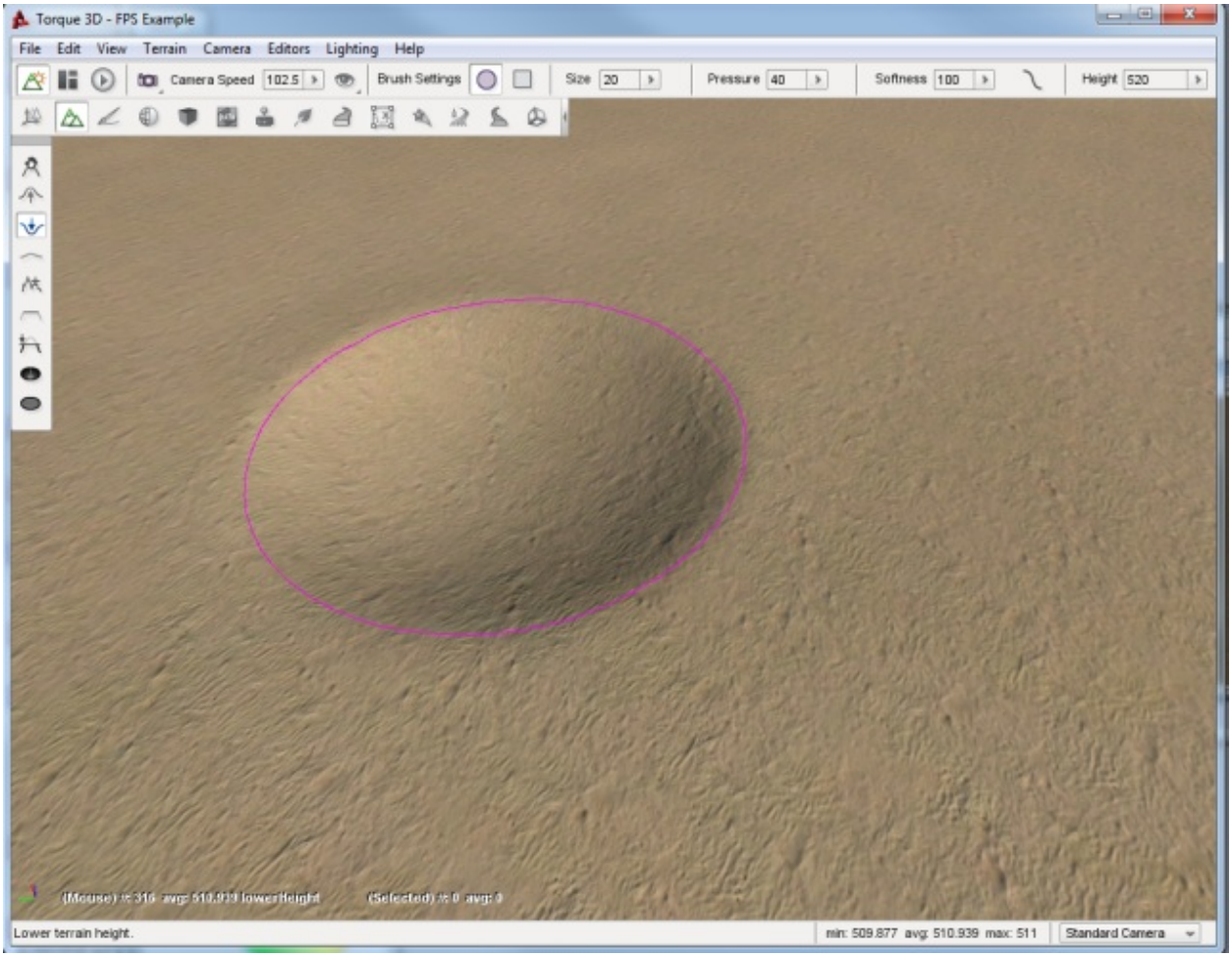
Using a lower brush pressure results in less dramatic terrain elevation as you "paint". This allows you to be more exact in cases where you need to.

Lower Height Tool

The Lower Height tool functions completely opposite of the Raise Height tool. Instead of elevating, you can dig holes in the terrain with this tool. Again, use a circular brush with 20 size, 40 pressure, and 100 softness. With the Lower Height tool selected, locate a flat section of the terrain and hover your brush over it.



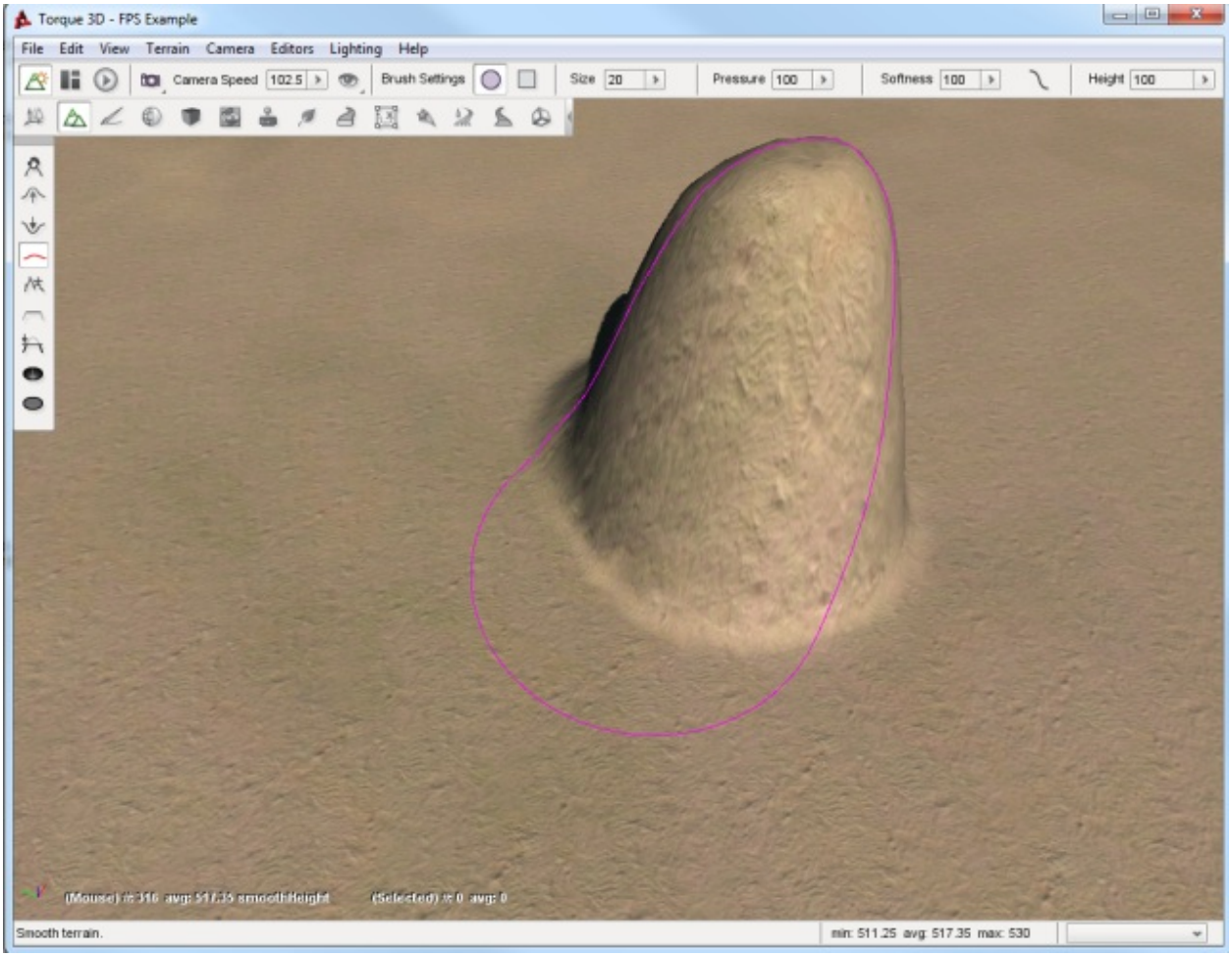
Click and hold down the left mouse button. As you do so, the terrain will sink down below the brush. If you sweep your mouse as if you are painting, you will create a path of lowered terrain. The longer you hold the mouse in a single location, the deeper the hole will be.



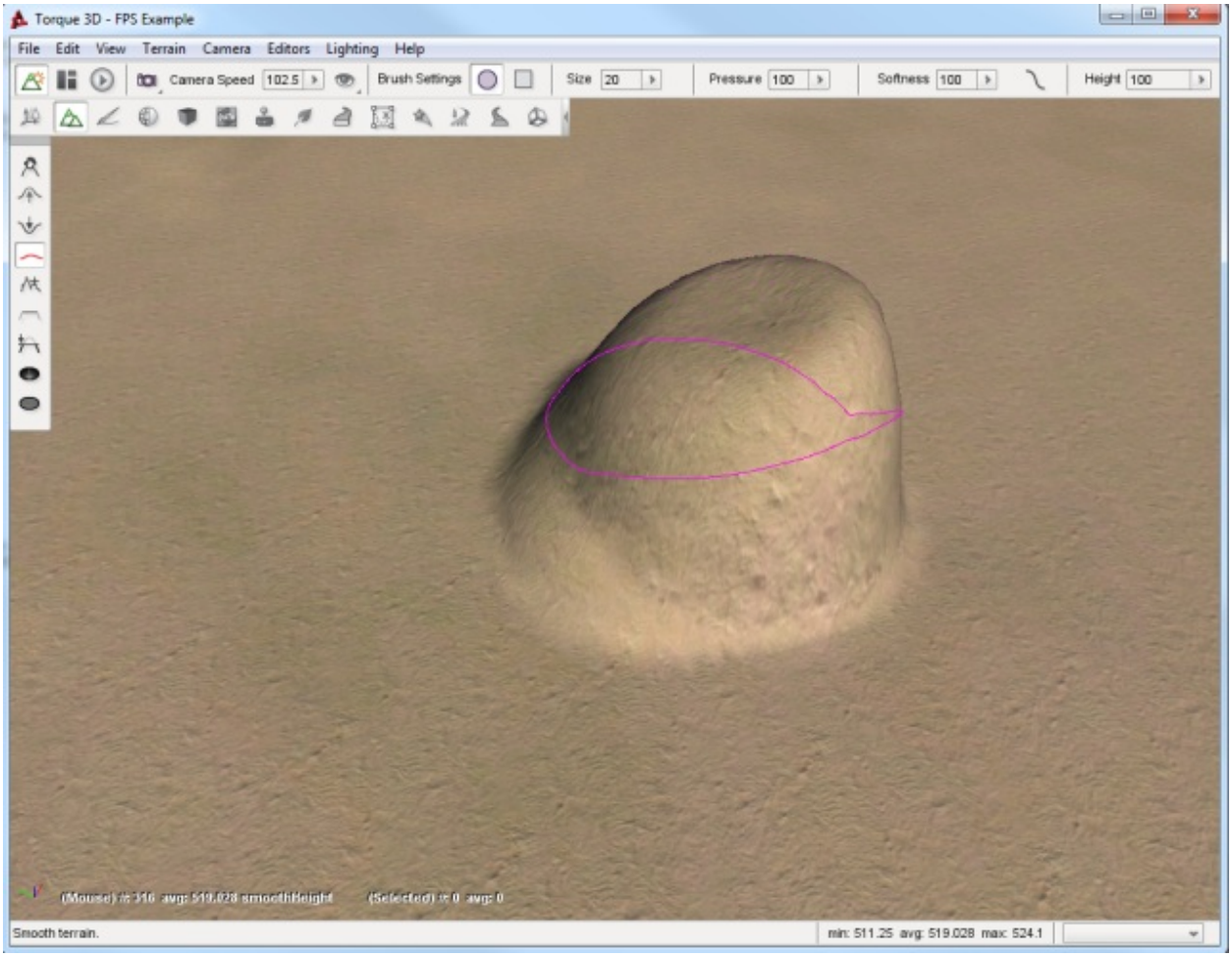
Smooth Tool

The Smooth tool erodes jagged terrain sections under the brush to create a smoother surface. This tool will only work if you sweep the brush across a surface. Simply holding down the left mouse button will have little to no effect.

Keeping the same settings we have been working, locate a jagged section of terrain. If you have to, create one with the Raise Height tool first. Make sure the elevation difference is significant. Select the Smooth tool then hover the brush over the applicable terrain.



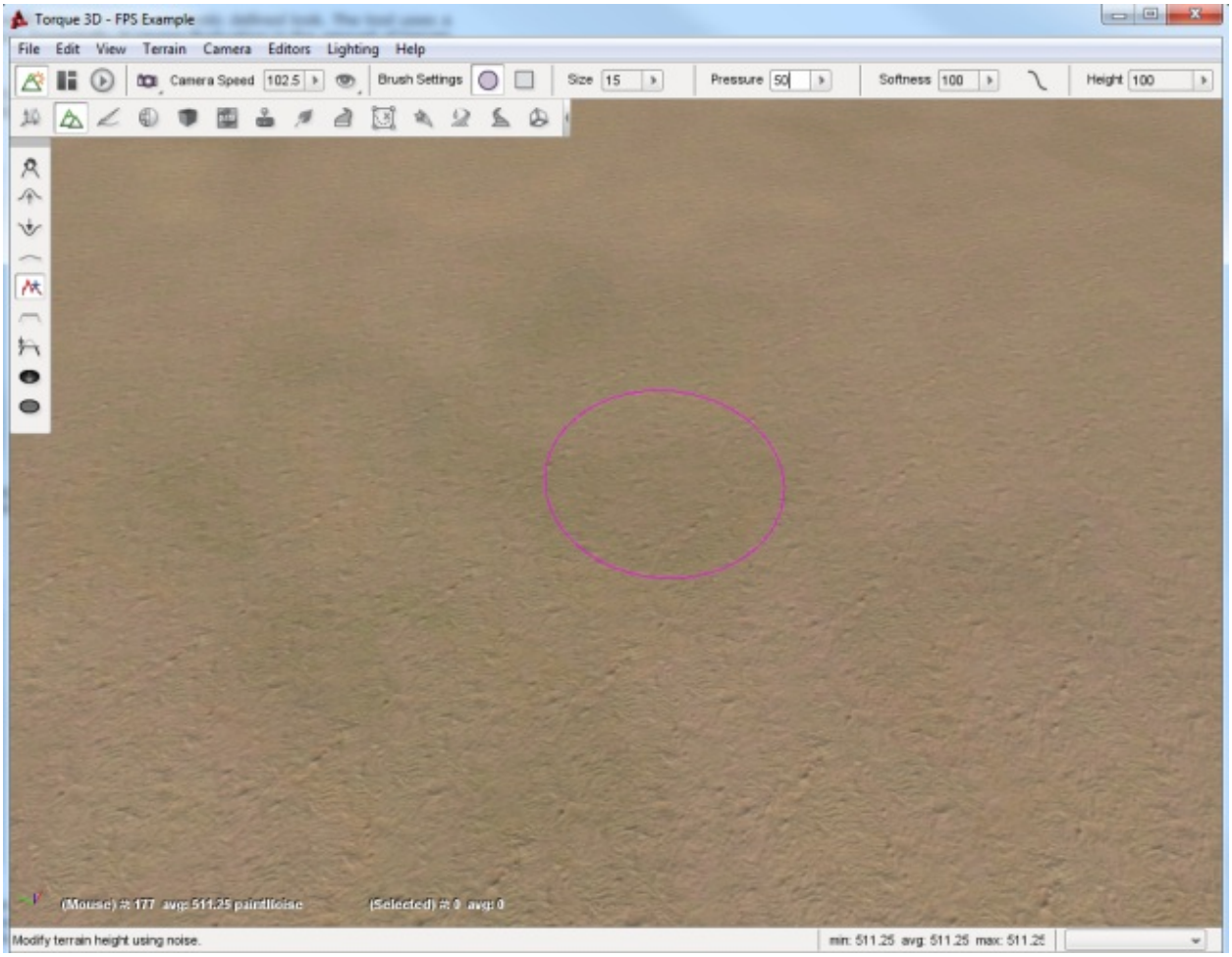
Click and hold the left mouse button, then make small circles around the peak of the terrain section. The tip should lower and have a broader surface. The broader your sweep, the more terrain is affected by the smoothing process.



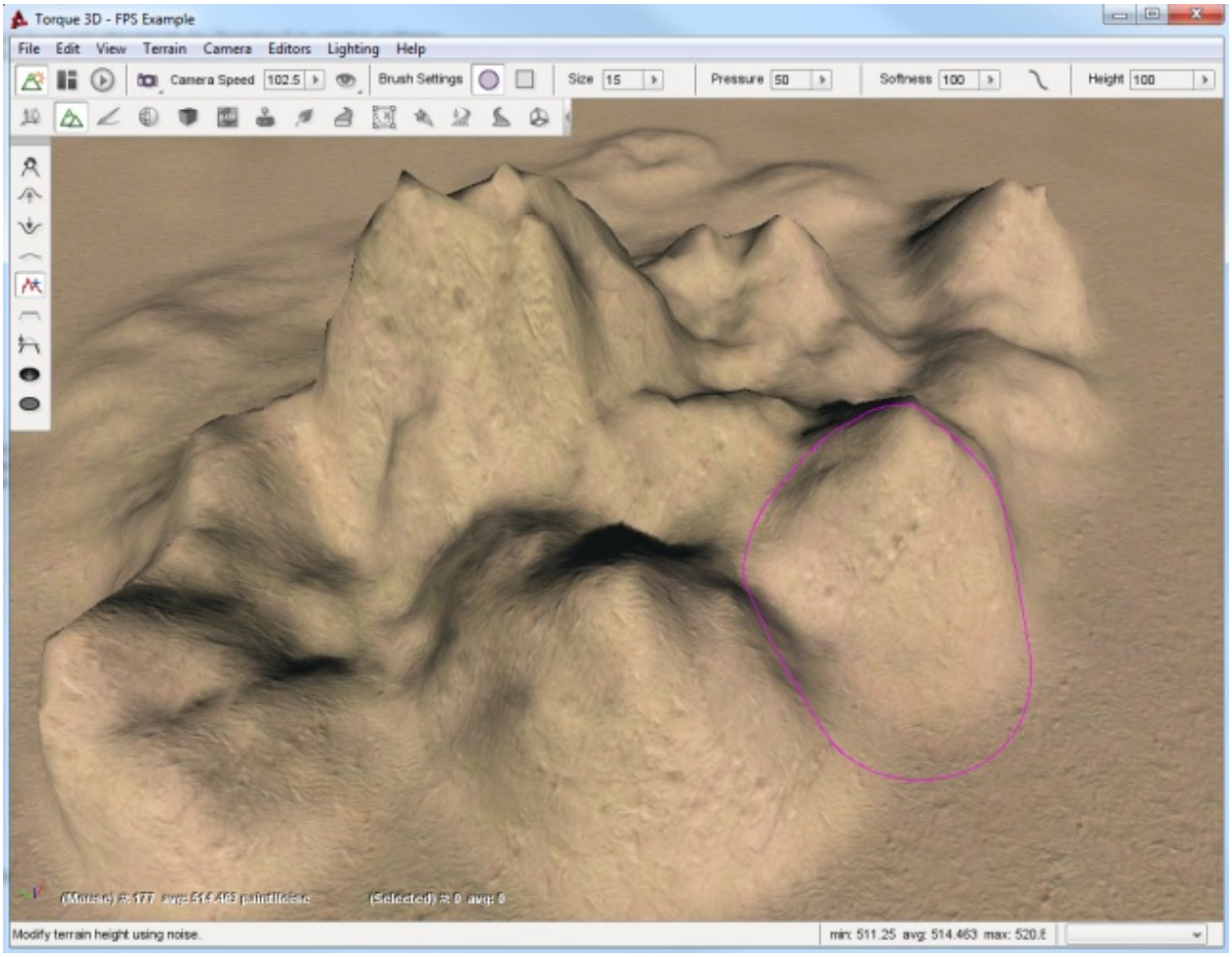
Paint Noise Tool

The Paint Noise tool is used to give your terrain modifications a more randomly defined look. The tool uses a noise algorithm for sporadic elevation and excavation. Essentially, it causes fluctuation in the amount of terrain it modifies and how intensely it changes.

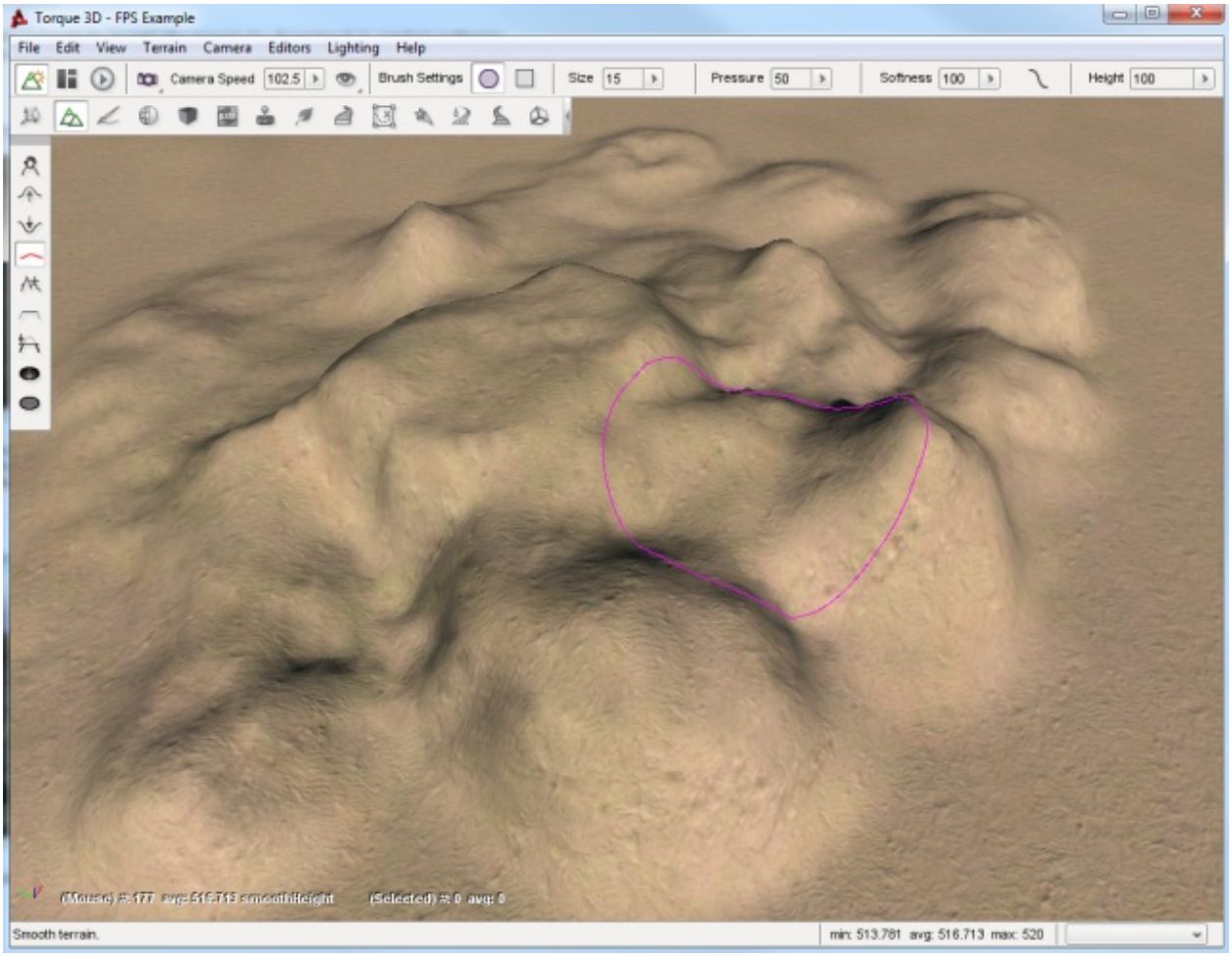
Select the Paint Noise tool, then set your brush to size 15, 50 pressure, and 100 softness. Locate a large section of flat terrain and move your camera to a high elevation.



Click and hold the left mouse button down then begin to "paint" the terrain by dragging it in random patterns. Try making several concentric circles, varying spirals, zig-zag motions, etc. You should eventually see some definition forming.



When you are finished with the tool, fly your camera around the section of the terrain to see how the terrain was affected. Keep in mind that most of these changes were random, which can add much needed detail to your terrain but can cause some weird effects. The Smooth tool can be used to go back and blend out any such effects that do not look natural.

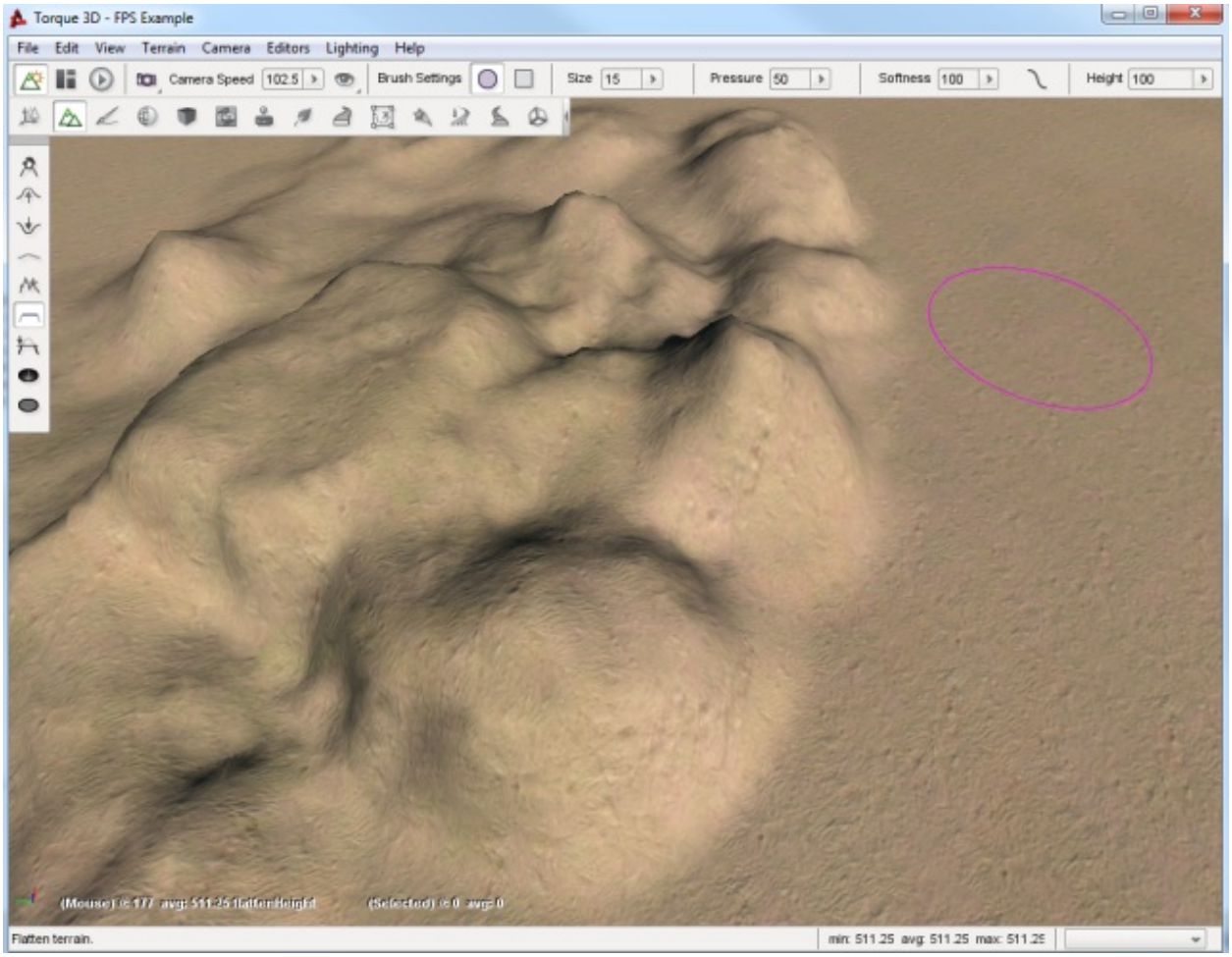


You can use this tool on terrain that has already been modified to remove unrealistic adjustments, such as perfectly smooth or flat slopes.

Flatten Tool

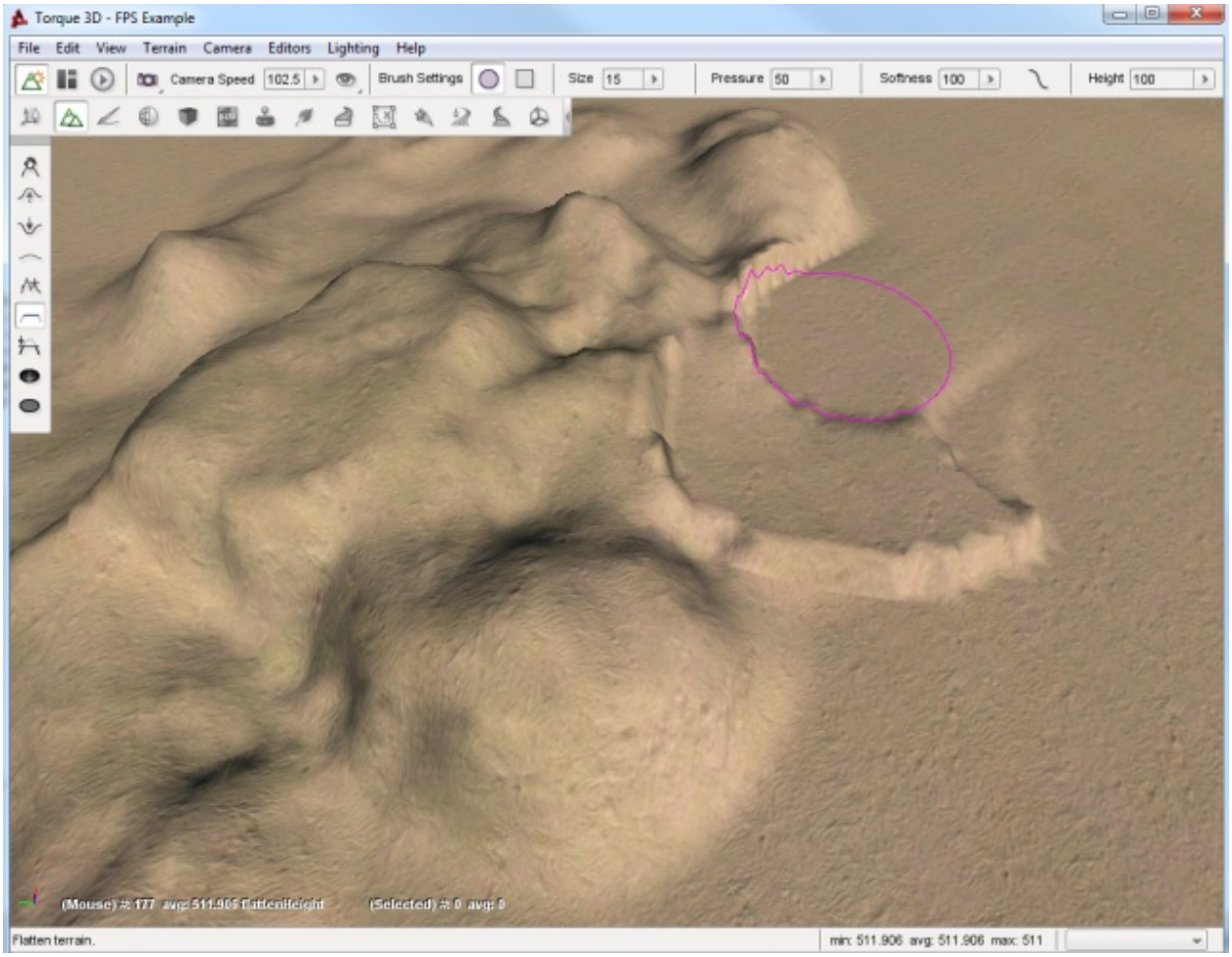
The Flatten tool is used to make the terrain surrounding the brush's starting point be equal to that points elevation. In other words, this either lower or raise your terrain to the same elevation as that starting point.

Use a circular brush with 15 size, 50 pressure, and 100 softness. Find a section of terrain that is elevated. Position your brush near it, but on a flatter section of the terrain.



Click and hold your left mouse button, then drag it toward and over the elevated terrain until you have swept over most of it. You should see that the tool has flattened a strip of terrain, based on the brush's location as it swept. The flattening process will become weaker the further you take the brush into the higher terrain such that it will not cut a path that is exactly the elevation of the starting point but rather relative to it and the terrain you are crossing. If you sweep the Flatten tool slowly across a hilly terrain, you will see that it is well suited for specialized tasks such as creating road and rail beds or mountain passes. Creating these types of features can be accomplished using the other tools but this tool in particular makes that job much easier.

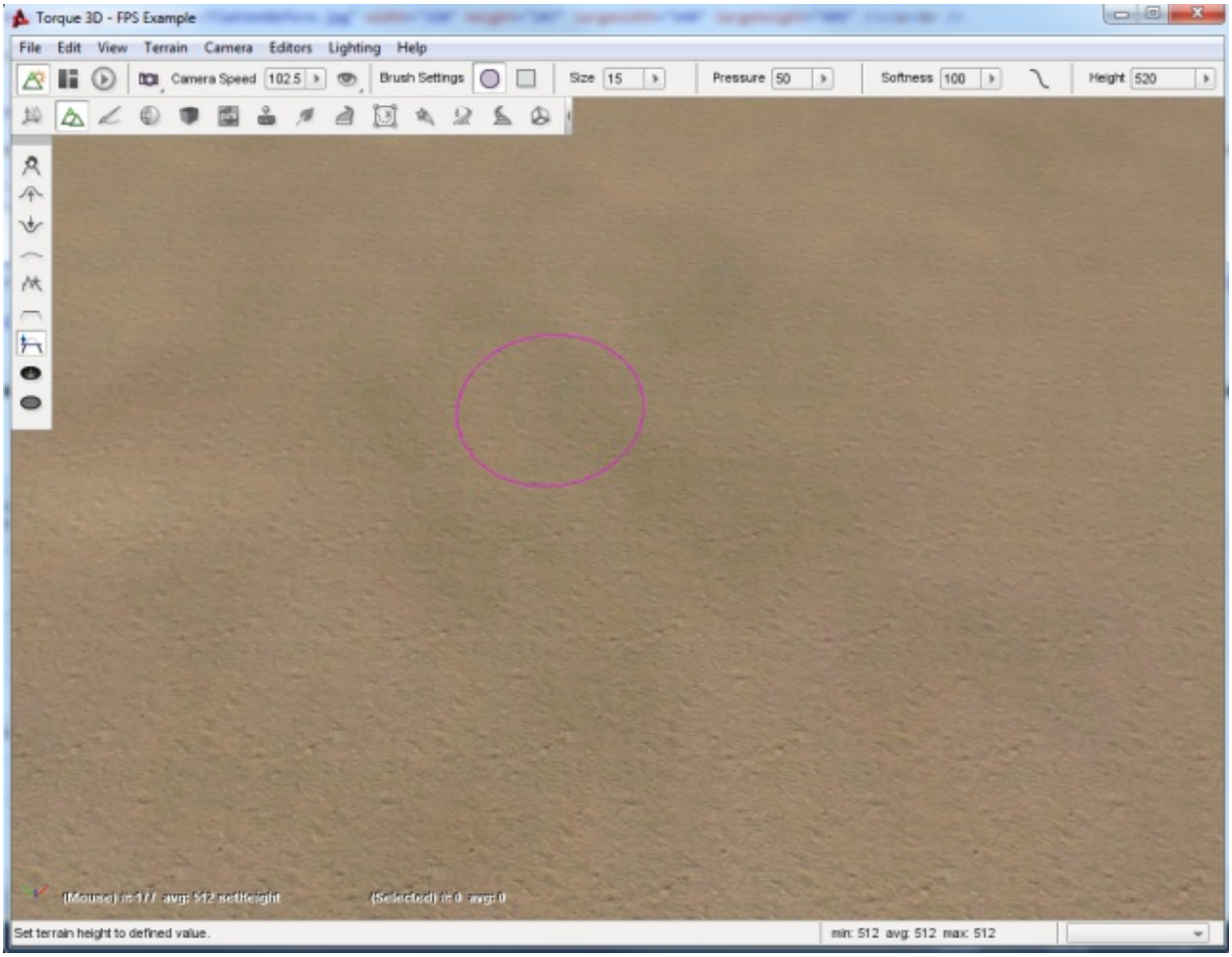
If you make several sweeps in the same direction, from the same starting point, your terrain will eventually smooth out into a flat plateau almost level with your original starting point.



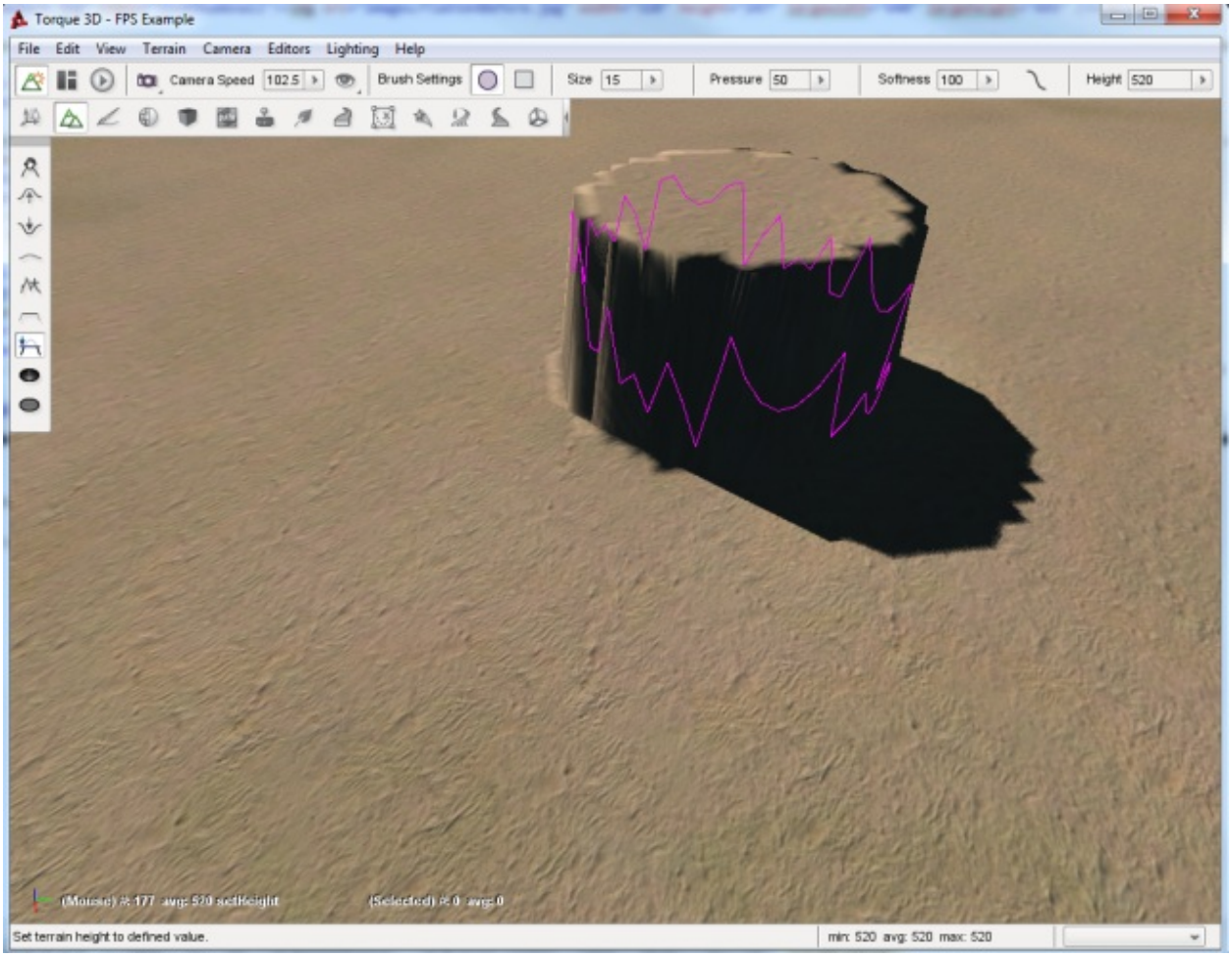
This is generally handy for clearing a smooth path from one elevation to another. However, this is not the optimal approach for flattening huge sections of terrain. The other tools can perform that process much faster and more efficiently.

Set Height Tool

The Set Height tool will allow you to determine the exact height for the terrain brush. Use a circular brush with a size of 15, pressure of 50 and softness of 100, and a height of 520.

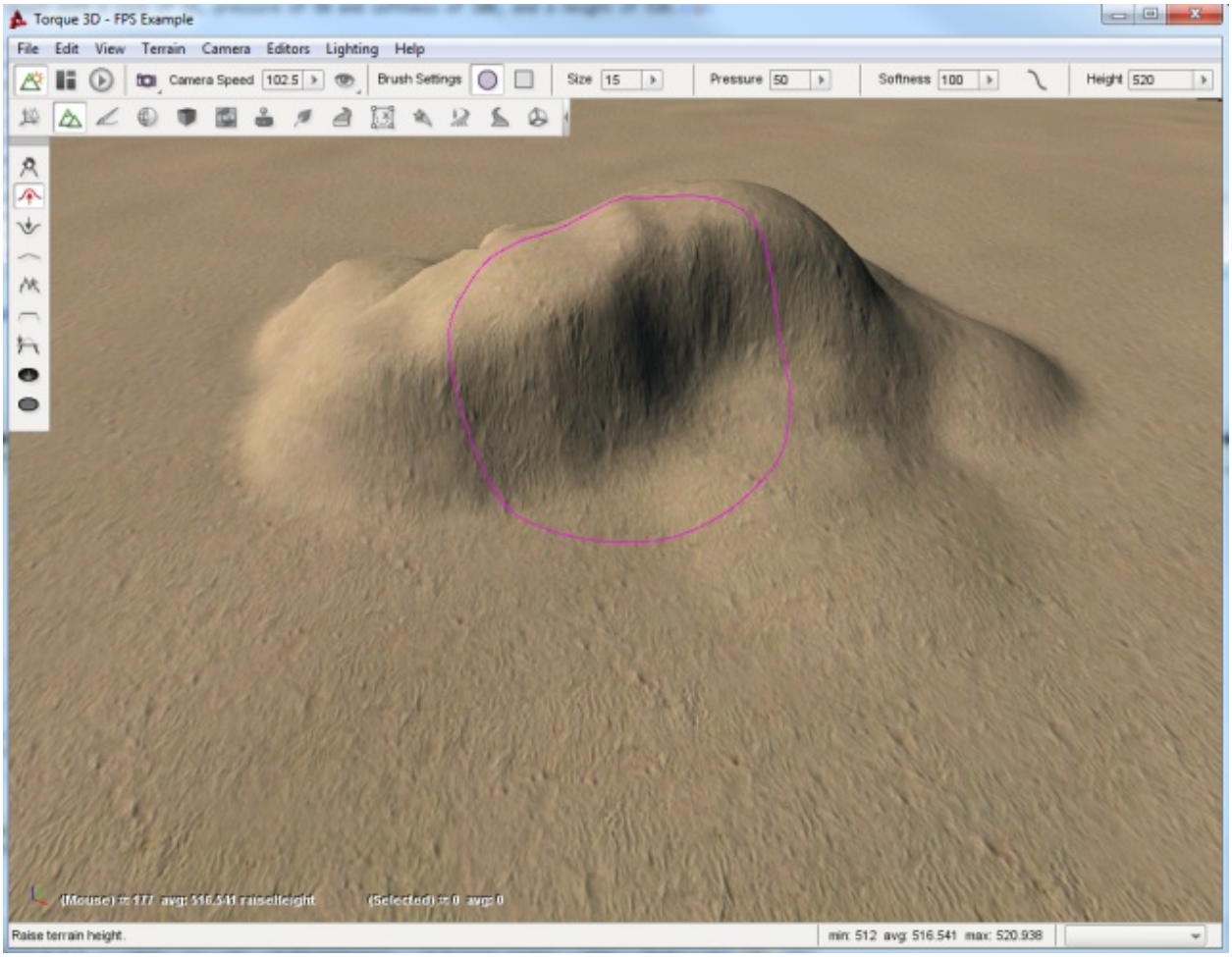


Now, when you press the left mouse button, it will create a plateau at exactly that height.

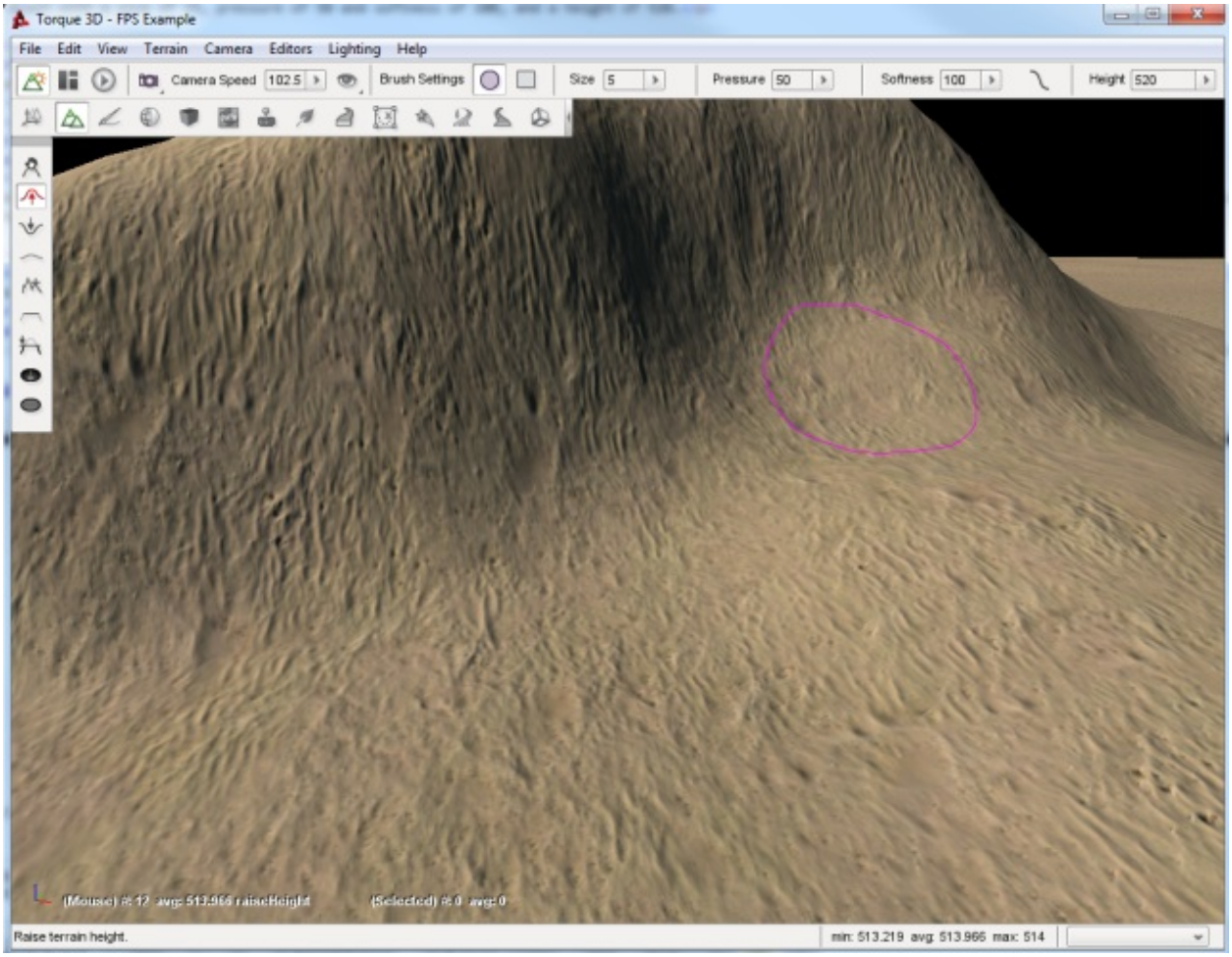


Clear Terrain Tool

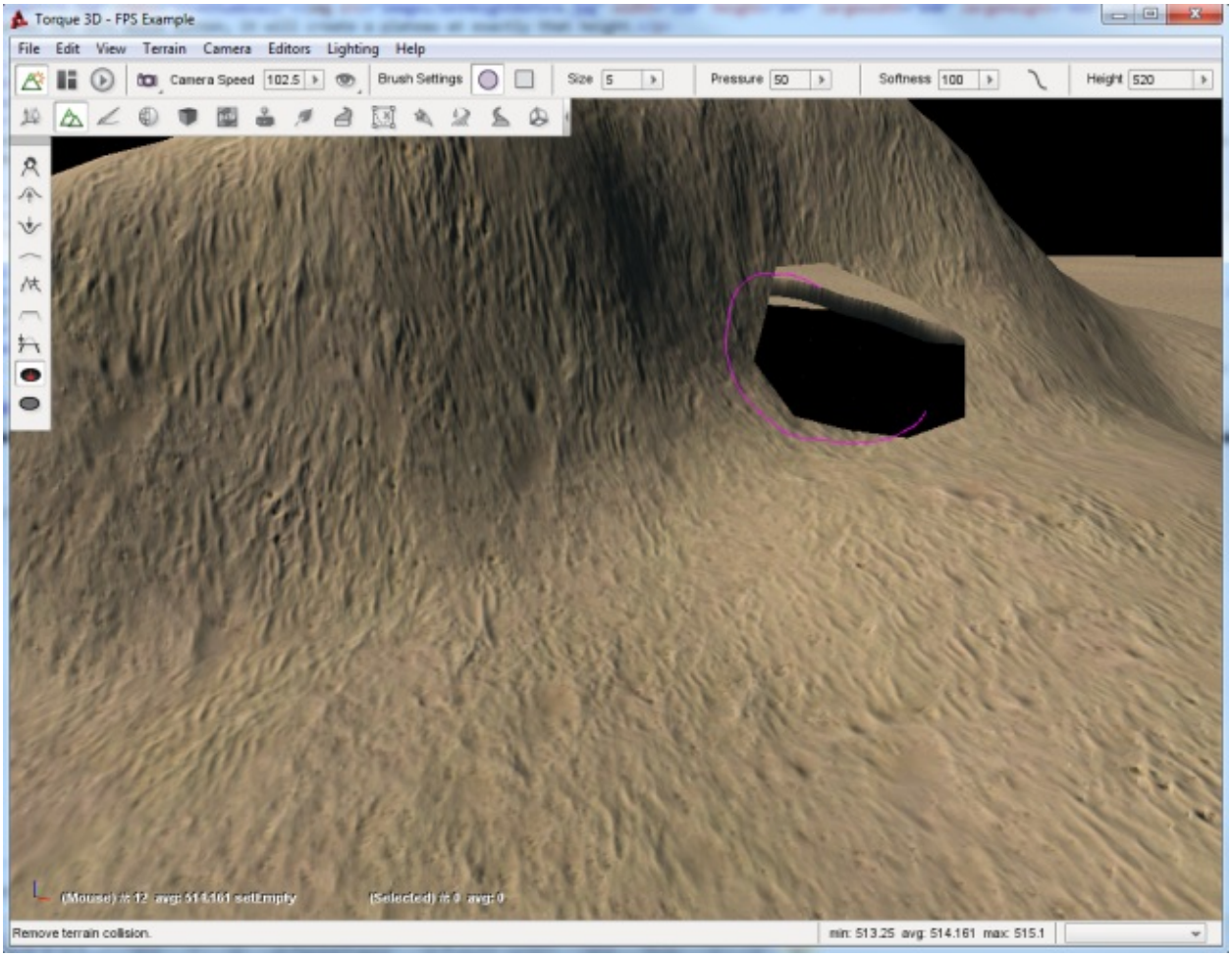
The Clear Terrain tool will allow you to remove pieces of the terrain. This is an effective way to carve out entrances to caves. That way your artist can create a detailed cave level and your level editor can "carve" out an entrance in the terrain. Using the previous tools, create a small hillside.



Now, set your brush size to 5 and zoom into an area that looks like a promising cave entrance.



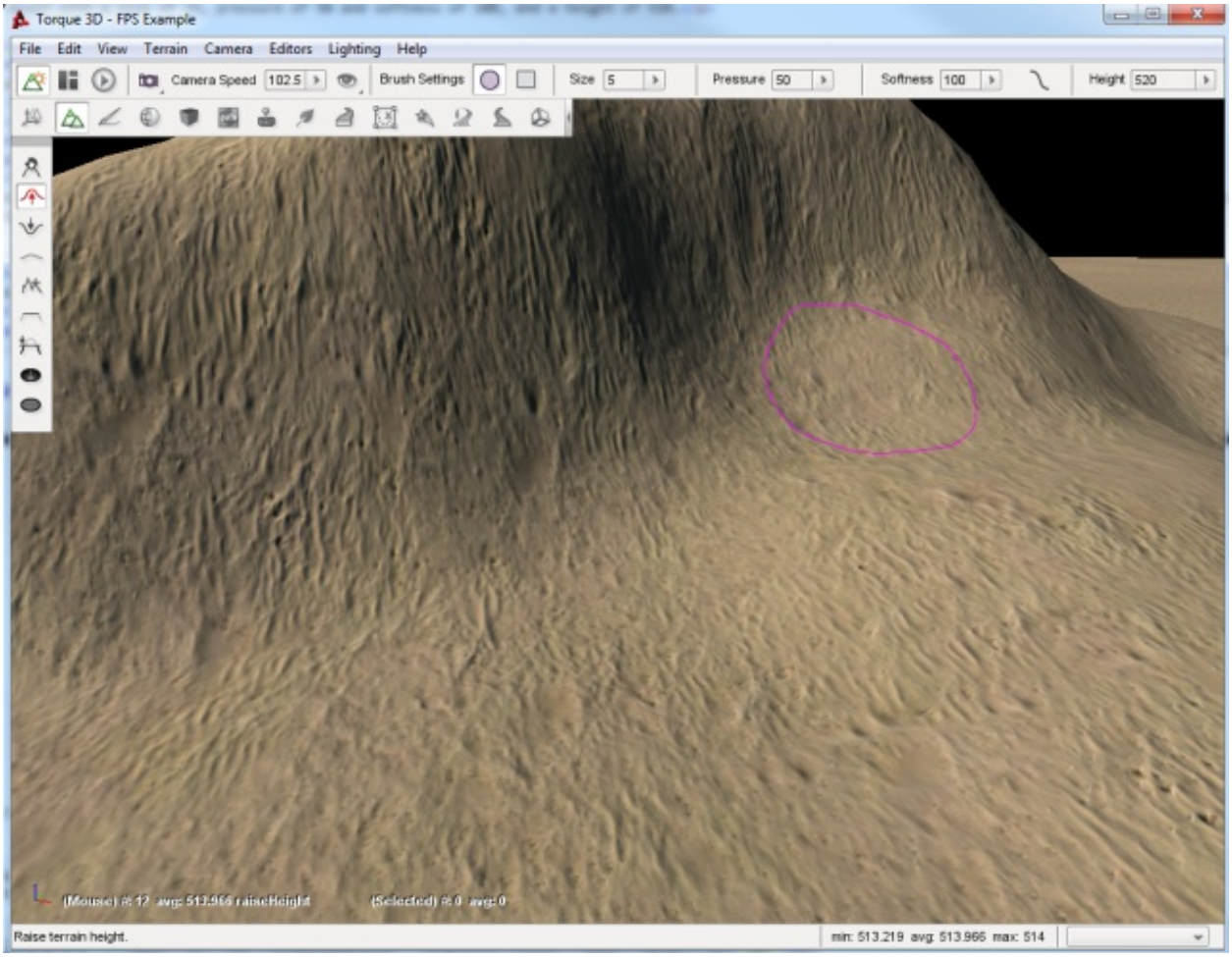
When you select the terrain area, it will remove the mesh data from the terrain, creating an opening.



Now you can place your cave model underneath the terrain so that the player can explore the world under your terrain.

Restore Terrain Tool

The Restore Terrain tool complements the Clear Terrain tool. It will restore the mesh data for the terrain. That way, you can have better control over the transition between your models and terrains. If you select the Restore Terrain button and then left-click on the previously cleared area, you will see it restore the terrain to its previous state.



2.4.2 Terrain Painter

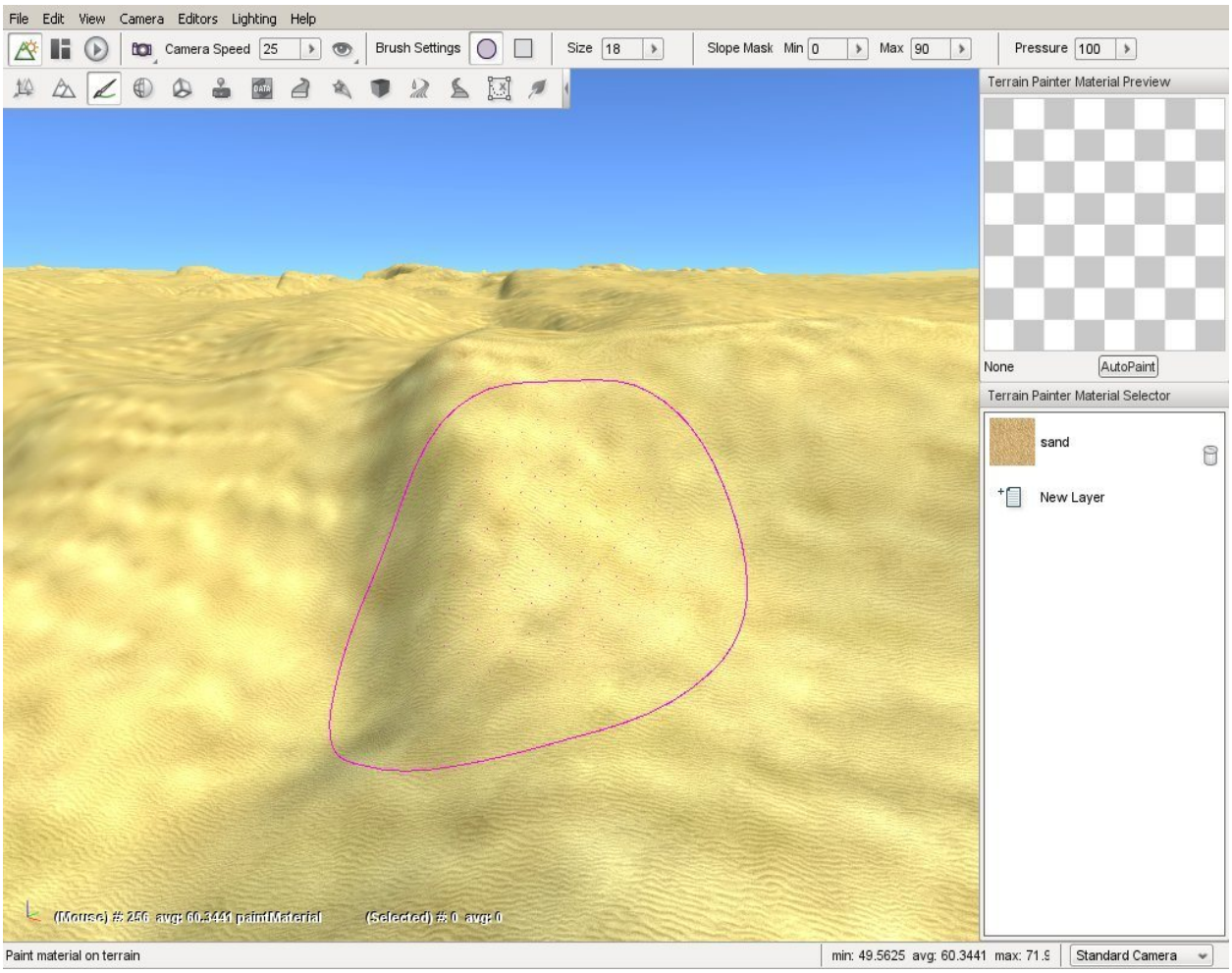
Just as the name implies, the Terrain Painter is a tool built into Torque 3D's World Editor which allows you to paint your terrain with various materials, such as grass, dirt, rocks, and so on.

Like the rest of the editors, the Terrain Painter is a WYSIWYG editor. As you change your terrain materials and paint the surface, you can see what the changes will look like in real time as if you were playing your game.

You can use the Terrain Painter to make wide-spread modification to a blank terrain, or use finer and more detailed brushes to touch up imported terrain layers/textures. Let's get started by setting up your environment.

Interface

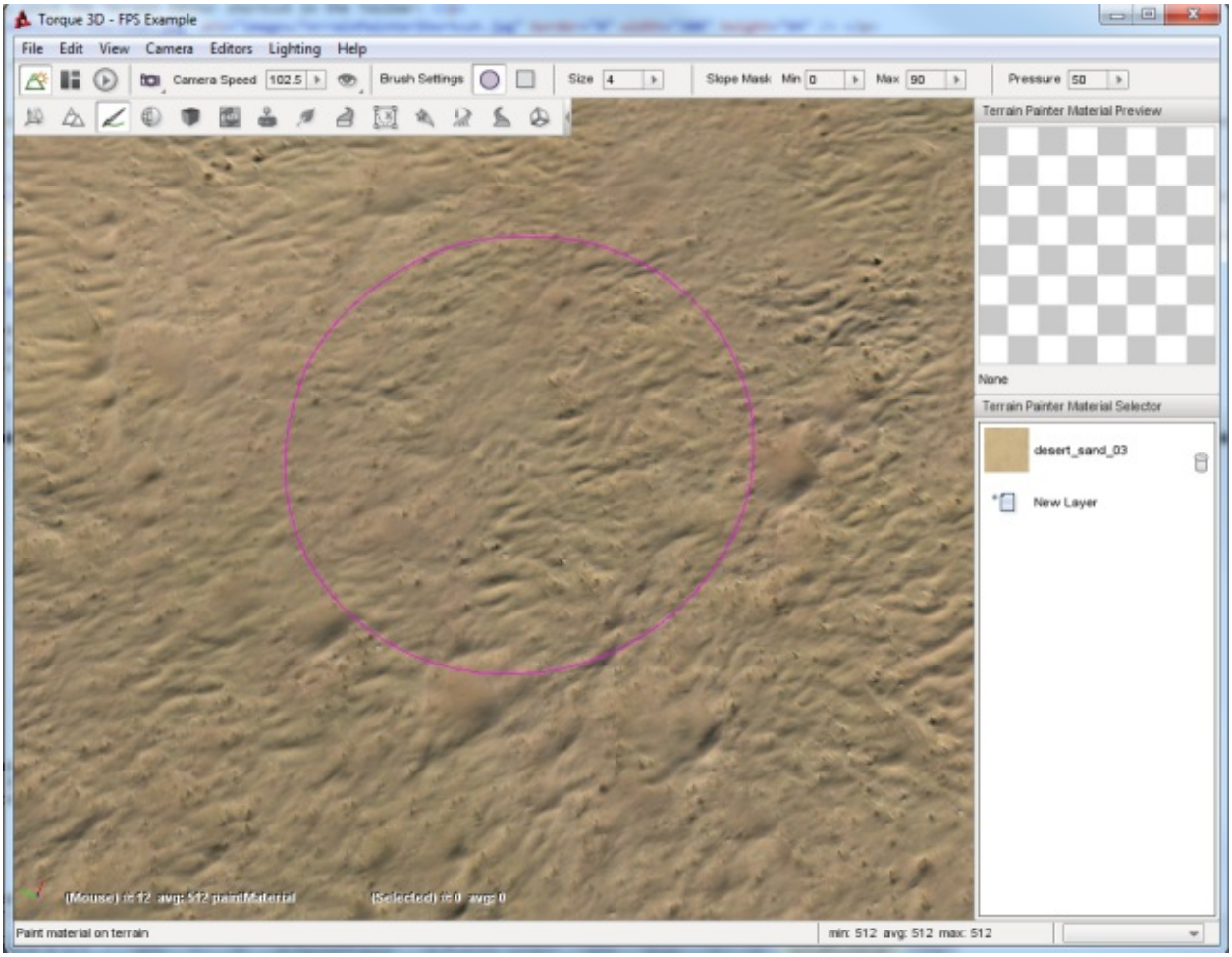
To switch to the Terrain Editor press the F3 key or from the main menu select Editors > Terrain Painter.



There are four main areas of the interface you will focus on while using this tool.

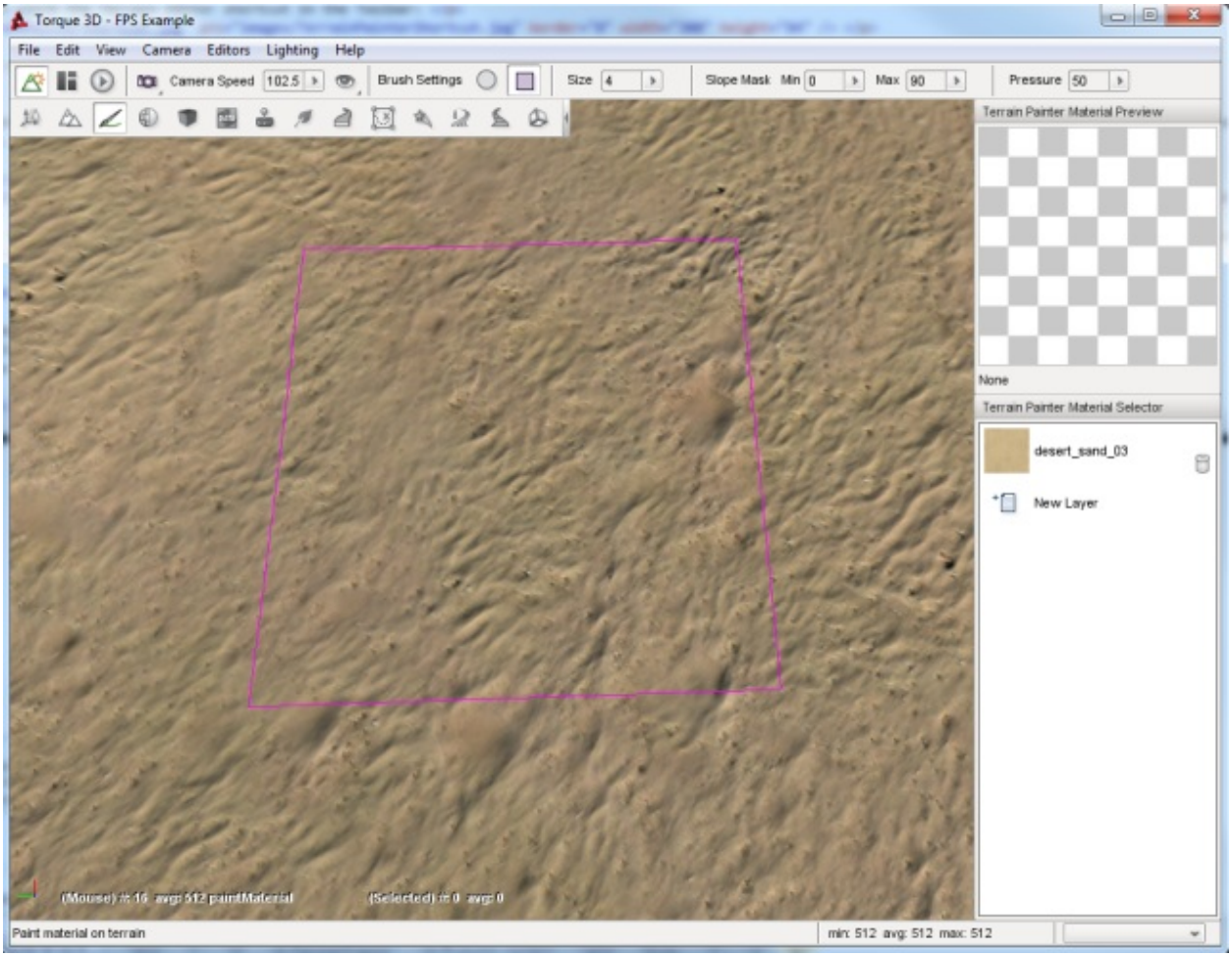
The Brush

Using the Terrain Painter is very similar to painting on a piece of paper with a brush except here you are painting on the terrain by dragging the mouse across the screen. Your brush is represented as a circle or a square in your scene's view. This visual outline allows you to know where your brush is located and what portion of the terrain it will affect when you move it.

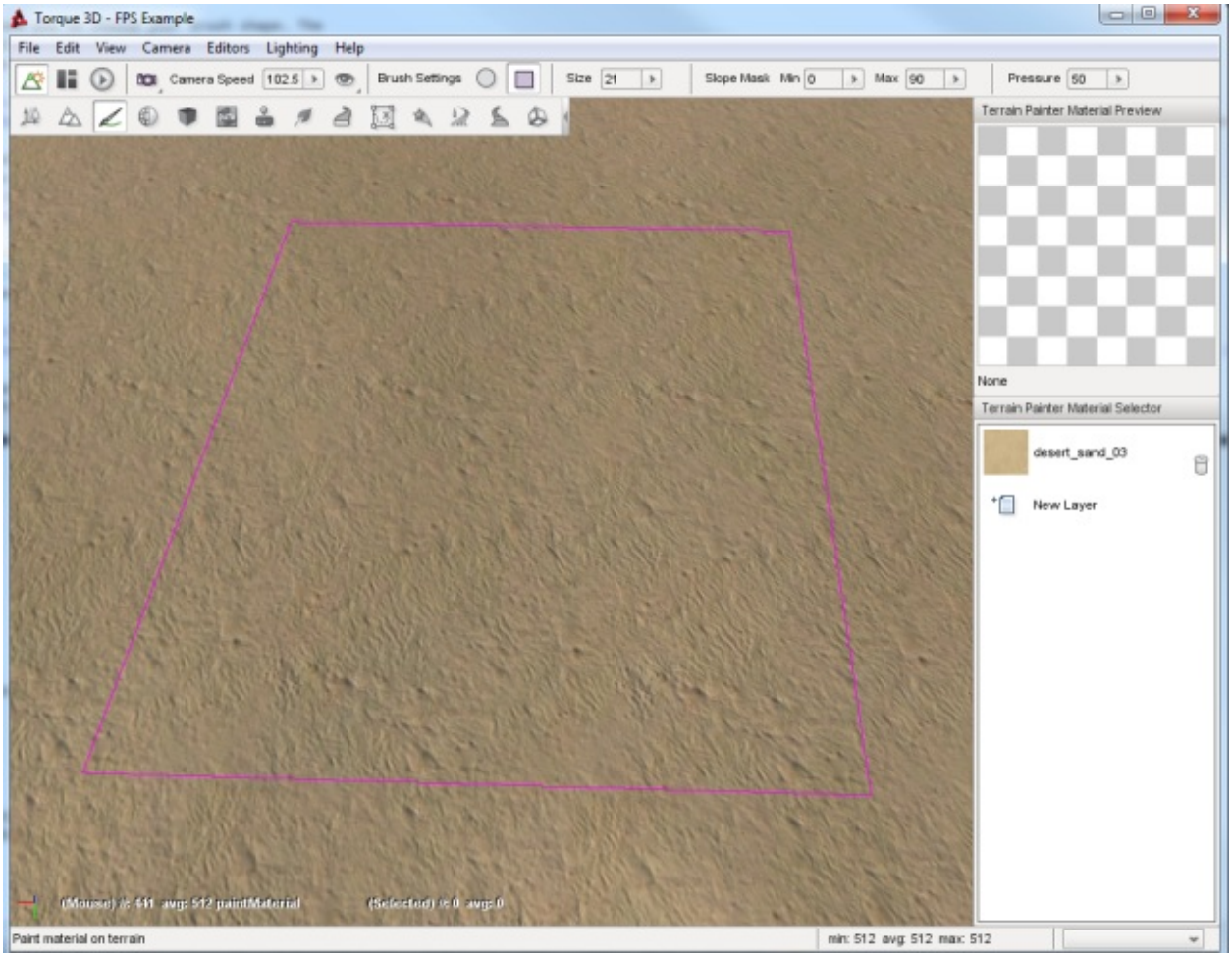


The image shown above is displaying the default brush style when you first open the Terrain Painter. If you wish to change your brush type, you can modify it via the Brush Settings found in the Tool Settings toolbar at the top of the screen. Brush Settings are only active while using the Terrain Painter.

The image shown above is displaying the default brush style when you first open the Terrain Painter. If you wish to change your brush type, you can modify it via the Brush Settings found in the Tool Settings toolbar at the top of the screen. Brush Settings are only active while using the Terrain Painter.



You will find the Brush Size slider next to the shape settings. You can move the slider from left (smaller) to right (larger) to change the size. The stock value is typically small, usually a 9x9 grid. The more you increase the slider value, the greater the grid will grow. The change will add an equal number of rows and columns, as shown below.

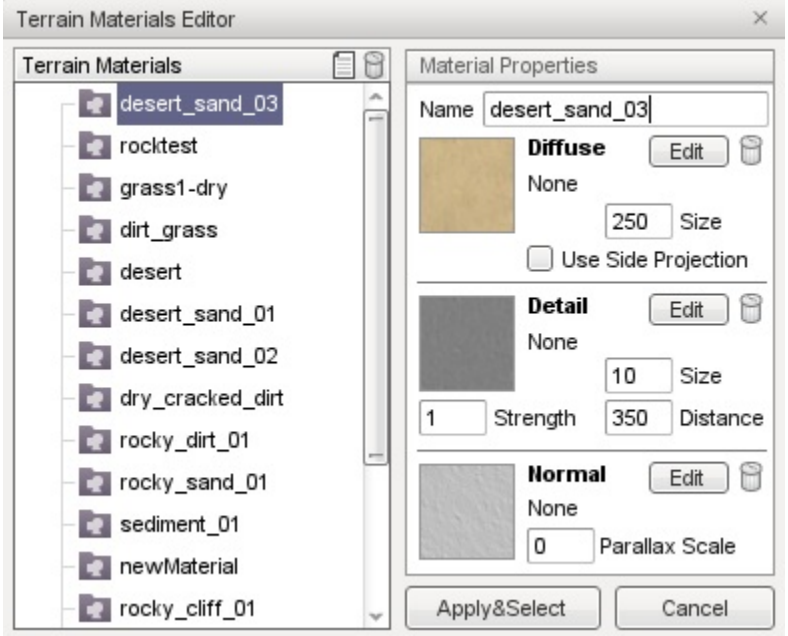


You can find the Terrain Painter palette docked on the right side of the editor. This panel is similar to a traditional painter's palette in the real world. Instead of swatches of color, the Terrain Painter's palette is populated by TerrainMaterials which you use to paint the terrain.

A TerrainMaterial is a collection of three textures combined into a single layer. The three textures are the base (also known as diffuse), detail, and normal map. A preview of which TerrainMaterial (or layer) is shown in the box at the top of the palette labeled Terrain Painter Material Preview.

Terrain Materials Editor

When you wish to add a new TerrainMaterial, click on the New Layer entry in the palette. Once you click on the entry, the Terrain Materials Editor window will appear. This tool is completely separate from the basic Material Editor, as TerrainMaterials are structured and used much differently than other Torque 3D materials which are used on shapes in the world placed with the World Editor.



Terrain Materials The TerrainMaterials list contains all the currently available textures for creating terrain materials.

New Button Clicking the Page icon in the Terrain materials header creates a new TerrainMaterial entry for editing.

Delete Button Clicking the Trash can icon in the Terrain materials headerdeletes the currently selected TerrainMaterial.

Apply & Select Button Clicking this button closes the Terrain Materials Editor and returns to what ever operation brought you to the dialog, for the purposes of this article it returns you back to the Terrain Painter Material Selector and adds the selected TerrainMaterial as a new material ready to be used for painting.

Cancel Button Close editor without making a choice.

Clicking on an entry in the Terrain Materials list updates the Material Properties pane on the right to display the current properties of that material.

The Material Properties pane contains a Name field, which is used as the label assigned to the material and three sub-sections which describe the textures that define the material.

The Diffuse sub-section shows a preview and the properties of the materials Diffuse texture, which provides the color and base appearance of the material. The Diffuse texture is also commonly referred to as the Base texture for this reason.

The Detail sub-section shows a preview and the properties of the materials Detail Map, which gives the material a more defined, crisp look. If you are familiar with advanced rendering concepts this is accomplished using additive blending and per-layer fade distance techniques.

The Normal sub-section shows a preview and the properties of the materials Detail Map, which gives the material a more defined, crisp look. If you are familiar with advanced rendering concepts this is accomplished using additive blending and per-layer fade distance techniques.

Name Assigns the name of the TerrainMaterial which will appear in the Terrain Materials list.

Edit Button Clicking this button allows you to select the texture to assign to this aspect of the material.

Trash Can Button Clicking this button clears the texture that has been selected for this section.

Use Side Projection Terrain diffuse textures are normally applied top-down, which can result in stretching. This toggle causes a material to smoothly merge and conform to steep terrain if needed.

Diffuse Size Controls the physical size, in meters, of the base texture.

Detail Size How close the camera must be before the detail map begins rendering in meters.

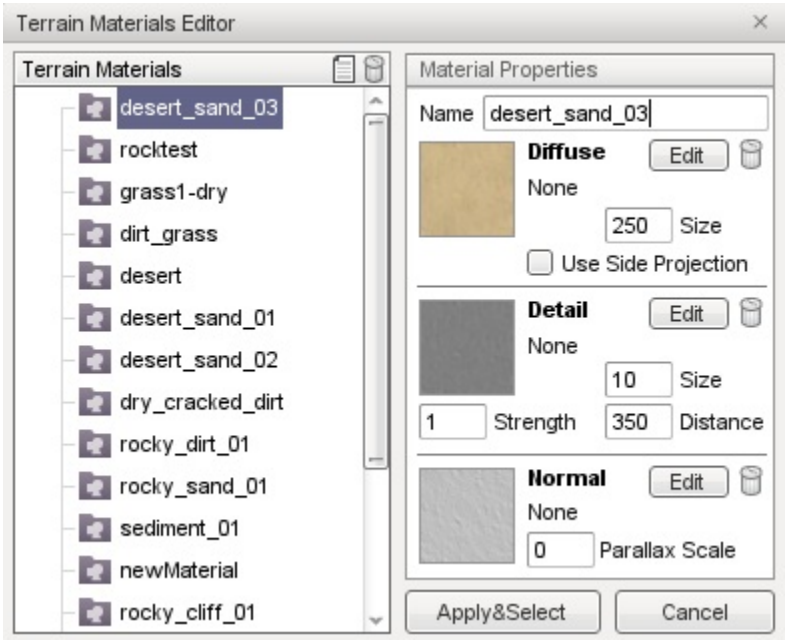
Detail Distance Determines how bold the detail appears on the base texture.

Parallax Scale Adjusts the intensity of the parallax depth in normal maps.

Painting

Before we begin painting, we will add a second TerrainMaterial to our palette (if the project you have open already has more than one feel free to skip this step).

To add a new material click the New Layer button in the Terrain Painter Material Selector. The Terrain Materials Editor will open. Click any TerrainMaterial in the list other than the one that is already in your palette, such as the "rocktest" material shown here.

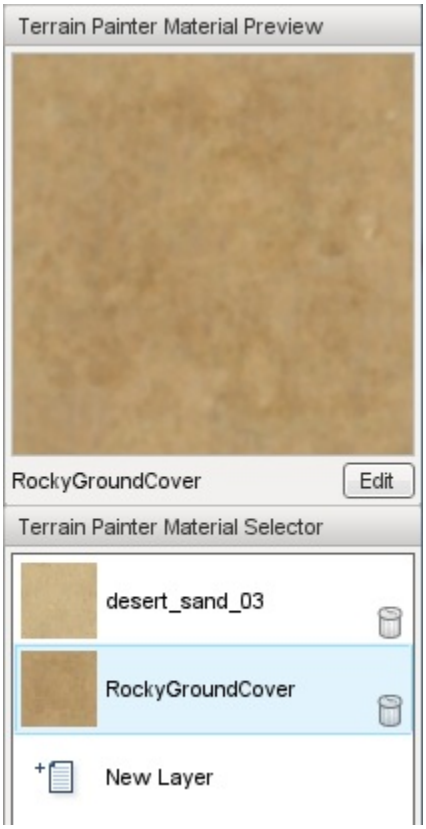


Once you have the material selected, click the Apply & Select button. Once you have done this, the new layer will have been added to your palette and available for painting.

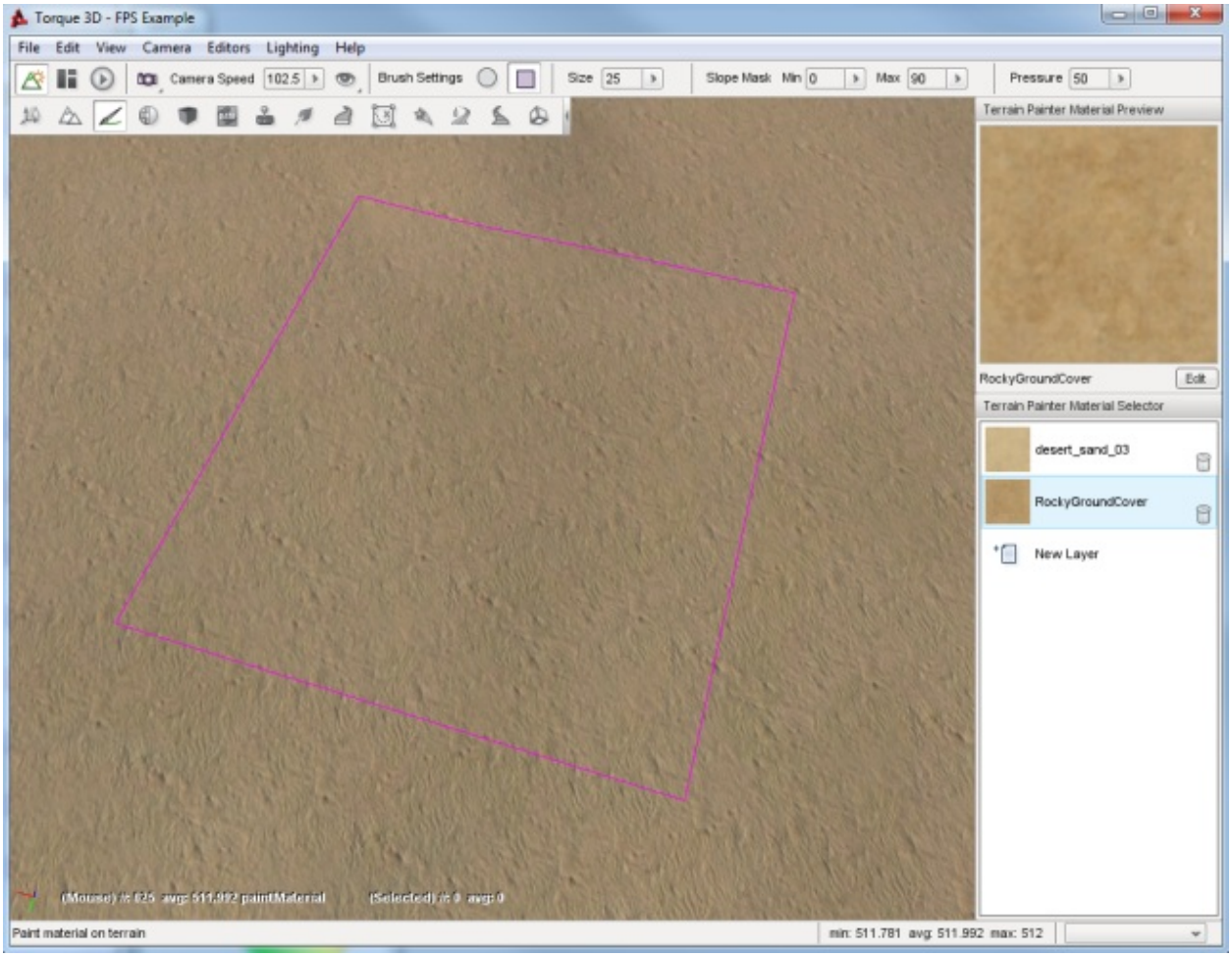
This is a good time to take a look behind the scenes to understand a little of how Torque 3D organizes materials and how it uses them for other operations to your advantage.What you can not see in the interface is that the system has associated each of the TerrainMaterials in your palette with a numbered layer. Throughout these documents you will see, or may have already seen, that material layers are used to control aspects of object placement such as which layer automatic object placement will occur on.

If you started with a project that was created with the Full template and added the rocktest material in the last step then the system now considersgrass1 to be layer0 and rocktest to be layer1. This allows you, whenever asked, to select layers using something meaningful to you rather than remembering some random numbering system. When asked to select a layer you can simply pick the grass or rocktest layer from a list and the system will use and apply the proper numbered layer to perform the related operation.

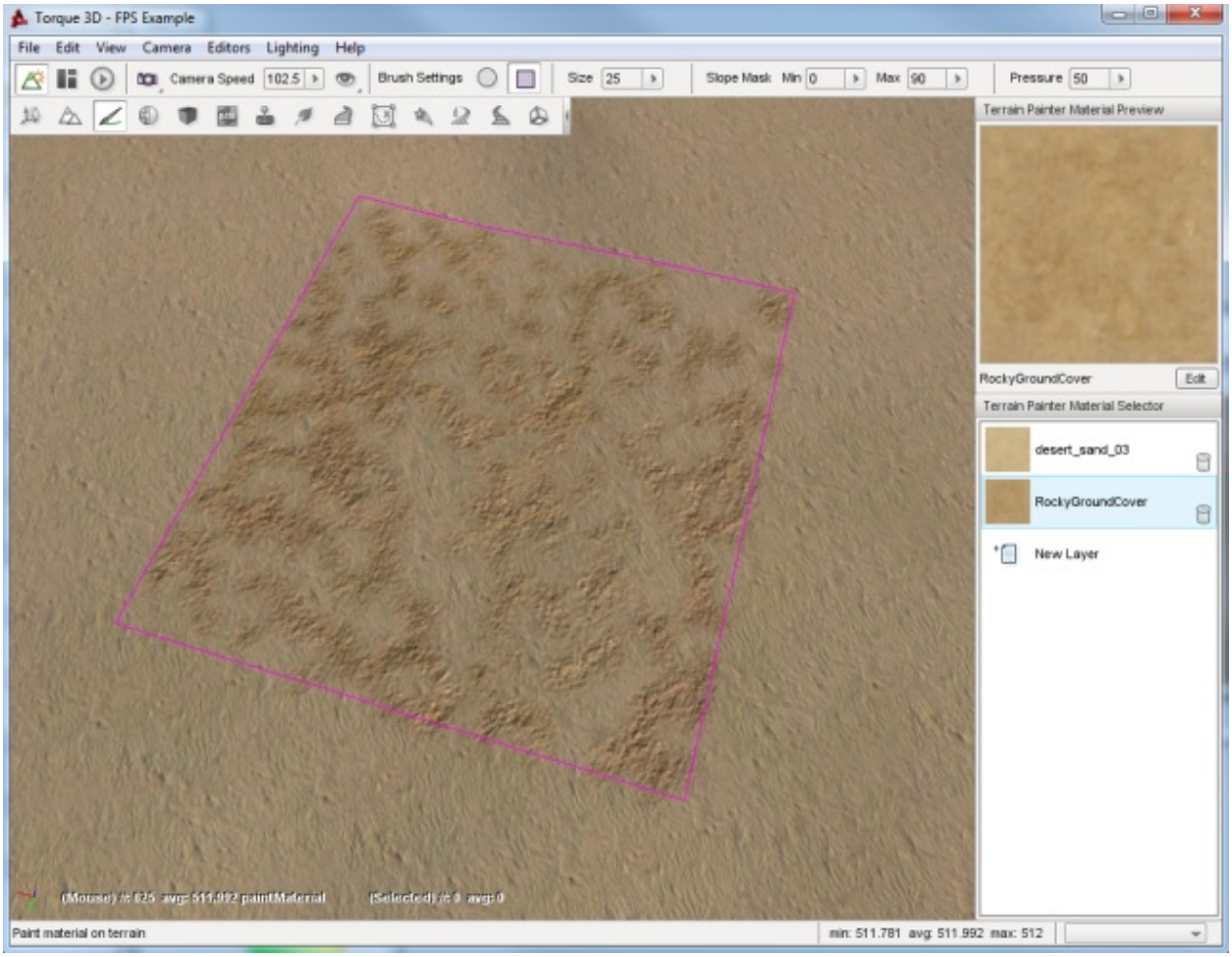
All this becomes very important in reducing the amount of work that is needed to create realistic terrain. The Terrain-Materials that you apply with the Terrain Painter tool not only give the terrain the appearance of natural materials but they can be used to automatically generate and restrict foliage and other shapes when used in conjunction with objects such as GroundCover.



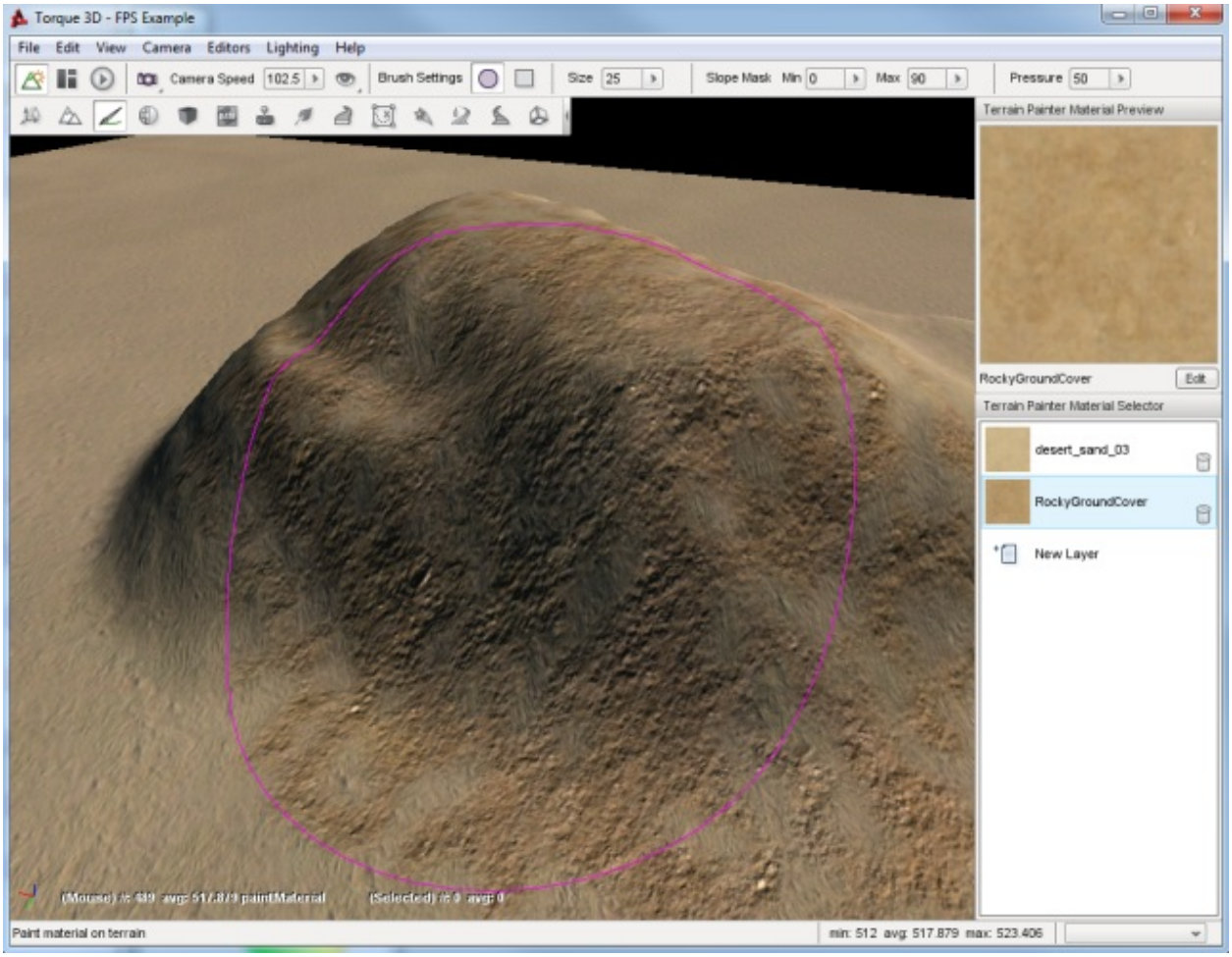
Now, on to learning to paint. Make sure you have the new material selected in the Terrain Painter Material Selector. So we can more easily see the modifications we are about to make, set your brush size to about 25. Now, find a section of the terrain you wish to paint. Here, we started in a corner of the TerrainBlock.



Click and hold down the left mouse button, then begin dragging the brush around the screen in a sweeping motion. The terrain will update in real time to reflect the painting of the new TerrainMaterial. When you let go of the mouse button, the Terrain Painter will stop laying down the material.

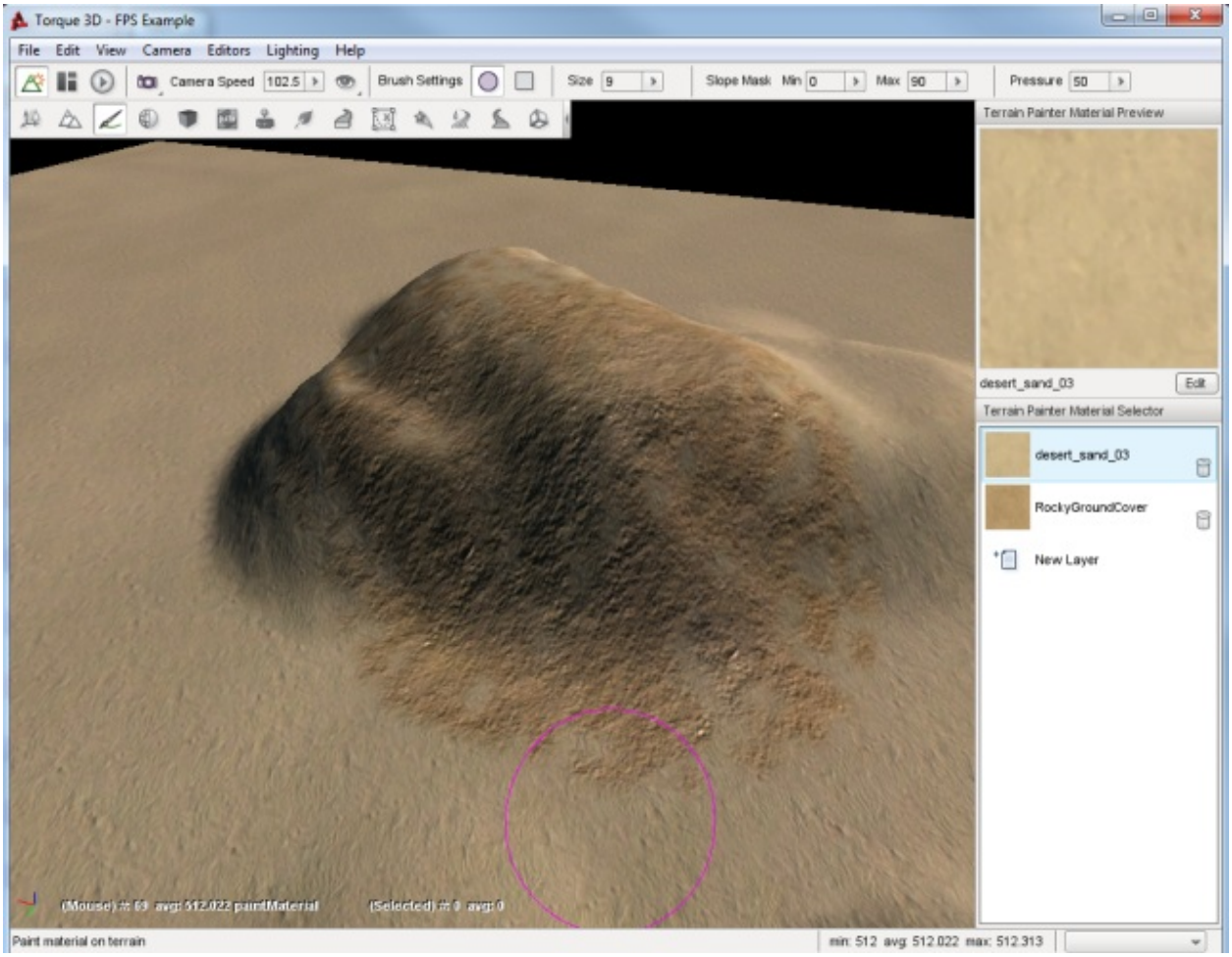


You should have noticed that your brush clamped to the terrain as long as the cursor was over the block. This happens regardless of any terrain modification or elevation occurring, as shown in the following example. Notice how the brush distorts to wrap around the elevated terrain.

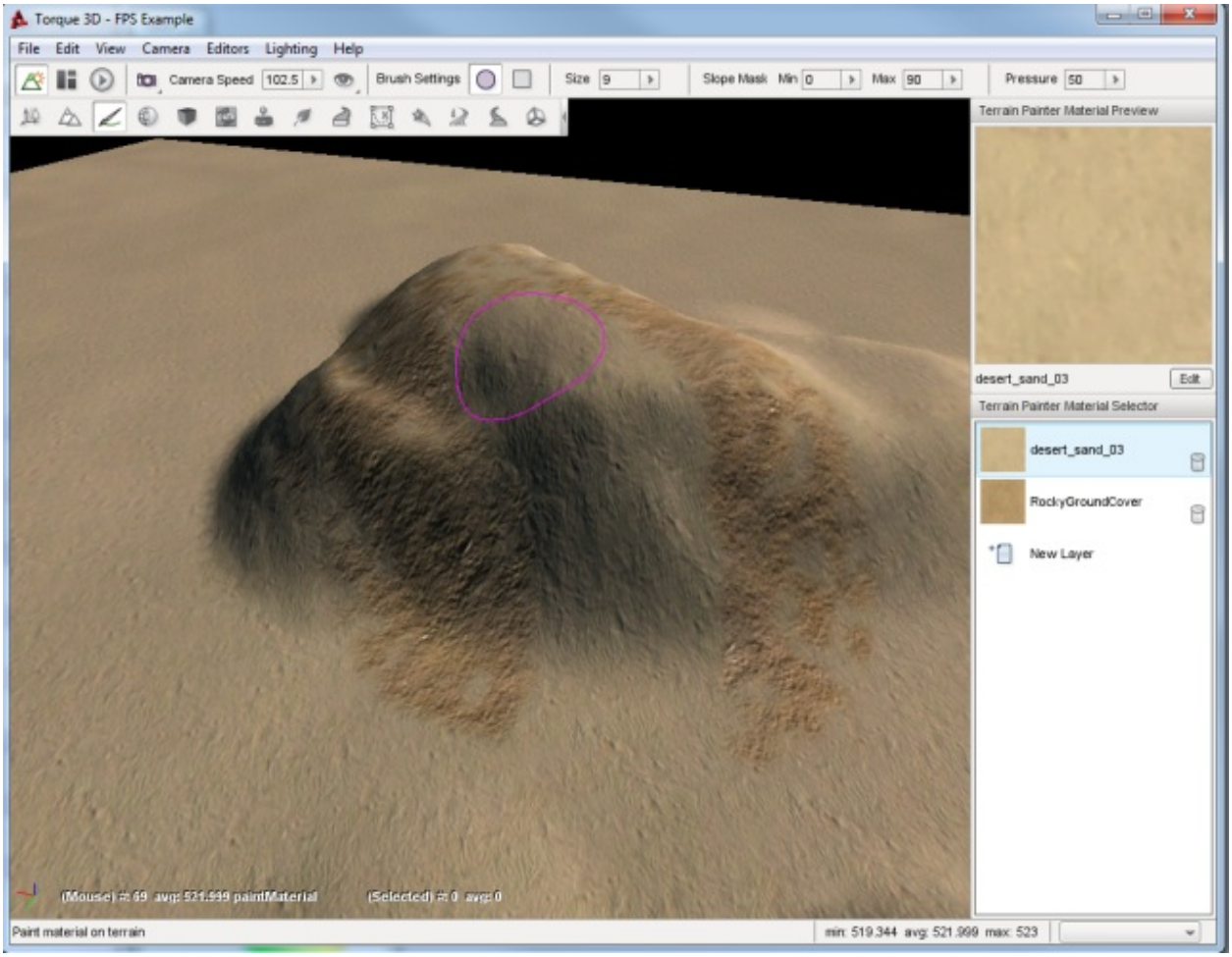


Even though you just paved a large section of rock material, you can still paint over it. Decrease the Brush Size to approximately 9, so we can paint a more exact line of terrain. We are going to paint a path over our rocky area. In the Terrain Painter palette, select the first material (desert_sand_03 in this image).

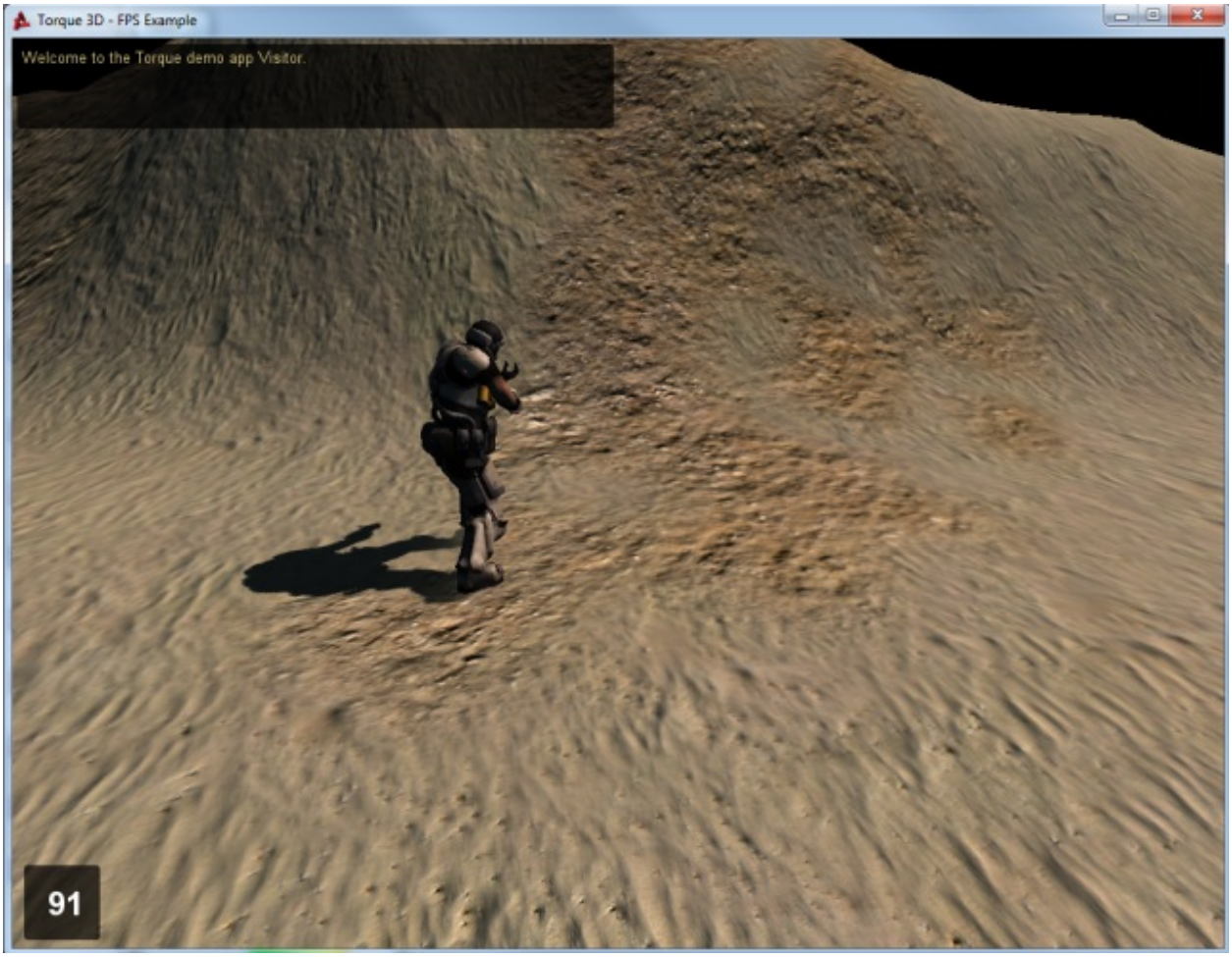
Now, using your mouse cursor move the brush to the edge of our rocky area. You can start it just before the rocky area, or even on top of it.



Click and hold down the left mouse button to begin painting then sweep your mouse in a curving motion across the rocky area. When you are finished, let go of the mouse and examine or your winding path made of grass.



If you were to drop down to the player's camera view, you can see where the two TerrainMaterials meet each other after editing.



Take the time to experiment with different brush sizes and shapes to see what kind of patterns you can come up with. When you are ready, read on to learn how to add a new TerrainMaterial with higher quality and detail.

2.4.3 Material Editor

Torque 3D's Material Editor allows an artist to quickly create and edit a game object's materials without ever touching a line of code. This tool can preview and edit materials mapped to an object in real time from the World Editor. Materials are categorized for ease of organization, and the editor is designed to be backwards-compatible with any existing script files. Materials are defined within script files named materials.cs

The Material Editor quickly comes into play when you are building your level by placing objects into the scene. As an example situation, let us assume you have a light fixture you or another artist has exported for use in Torque 3D. The creation of this object was a multi-step process.

The object's geometry had to be created in a 3D modeling app, such as 3DS Max, Maya or Blender. Once the geometry was finished, a 2D graphical application was used to create the various textures that make up the high quality appearance: base texture (diffuse map), normal map, detail map, etc.

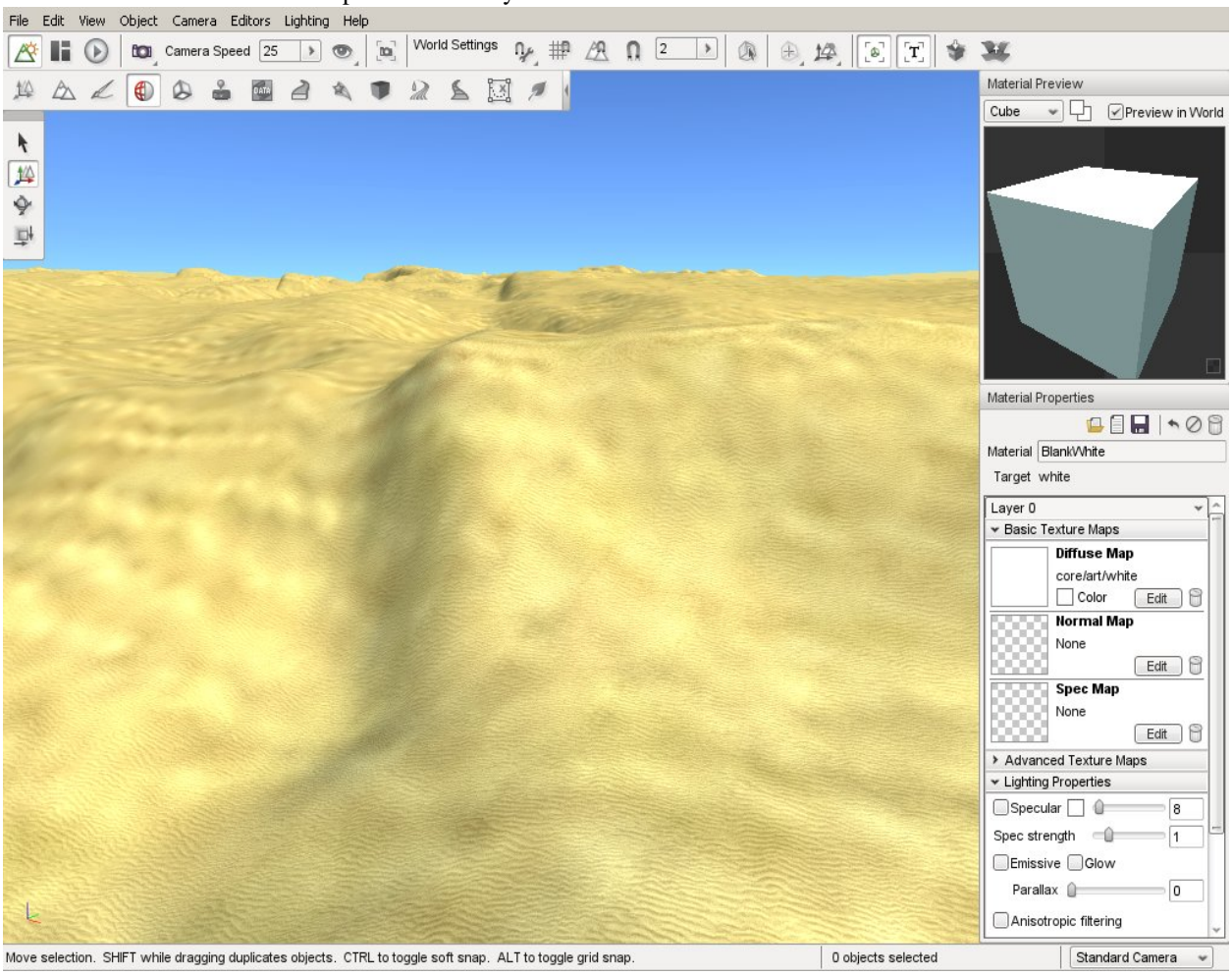
The rendered view of this object looks fine in the originating application, which leads you to believe it is ready to be imported into your game's level. When you drop the lighting fixture into the world, there is a good chance it will fit right in based on your existing design. The theme and color scheme are likely a match.

However, once your object is in the level you might notice something is off. While the stand alone object viewed in an external tool looked great, the object now seems out of sync with your level's lighting, tone, or specific room theme.

This is not the fault of the artist or design. What has happened is the materials for your object have either not been assigned or need to be tweaked to perfection in specific instances. Instead of going back into the art tool or adjusting properties in code, you can use the Material Editor to edit the appearance of your object by adjusting the texture maps and their properties.

Interface

To switch to the Material Editor press the `F4` key or from the main menu select Editors -> Material Editor.



Main Editor

Each major section of the Material Editor is separated via a header, such as Lighting Properties or Animation Properties. The fields found in these sections directly manipulate a materials properties in real time.

Material Preview

Cube Changes the Preview Mesh.

Square Symbols Change Normal Light Color.

Preview in World Show changes made in material editor on selected object in scene view.

Bottom Right Click square to change color of background preview.

Material Properties

Edit Material Select and Edit an Existing Material (E).

Floppy Disk Save material.

Trash Can Delete Material.

Name.dts Name of 3D asset using this material.

First Drop Down Texture associated with material.

Material Name of material.

Square with Ball Swap current material mapped to this mesh for another.

2nd Trash Icon Remove this material from mesh target.

Basic Texture Maps

Diffuse Map Base texture for material.

Normal Map Bump map that provides higher detail to mesh without extra polygons.

Overly Map Texture draw on top of other maps.

Detail Map Texture providing additional detail via lightening and darkening base map using high pass filter.

Light Map Texture using baked lighting info.

Tone Map Map which scales the RGB values of material. Used to calculate HDR.

Advanced Texture Maps

Diffuse Map Base texture for material.

Normal Map Bump map that provides higher detail to mesh without extra polygons.

Overly Map Texture draw on top of other maps.

Detail Map Texture providing additional detail via lightening and darkening base map using high pass filter.

Light Map Texture using baked lighting info.

Tone Map Map which scales the RGB values of material. Used to calculate HDR.

Lighting Properties

Specular Enables the use of Pixel Specular (shininess) for this layer. The slider adjust the strengt, and you can set the color of the specularity.

Glow Determines if this layer will Glow or not.

Exposure Intensifies glow and emission.

Emissive Causes an object to not be affected by lights. Good for materials from light source objects.

Animation Rotate Properties

Purpose Causes material to rotate along the surfaces of the mesh it is mapped to.

U and V Sliders Determines the direction of U/V coordinate rotation.

Speed Rate of coordinate rotation.

Animation Scroll Properties

Purpose Causes material to scroll along the surfaces of the mesh it is mapped to.

U and V Sliders Determines the direction of U/V coordinate scrolling.

Speed Rate of coordinate scrolling.

Animation Wave Properties

Purpose Causes the material to scroll in a wavy manner along the surfaces of the mesh it is mapped to.

Wave Type Switch between sine, triangle, and sqaure wave patterns.

Amplitude Changes the positive and negative crest of the wave (intensity).

Frequency Adjust wave length, which is the number of waves per time interval.

Animation Sequence Properties

Purpose Animates texture by frames.

Frames per Sec How many frames to display per second.

Frames Number of total frames in the sequence.

Advanced Properties

Purpose Adjusts advanced parameters that affects transparency calculations.

Transparency Blending Sets material to use transparent blending modes.

Transparent Z Write Can be used to help force a proper Z Ordering when Z Ordering issues occur. Only valid on materials with Transparency.

Alpha Threshold When enabled, causes pixels under a specific alpha threshold to get discarded rather than be computed.

Cast Shadows Material determines whether target mesh is allowed to cast shadows.

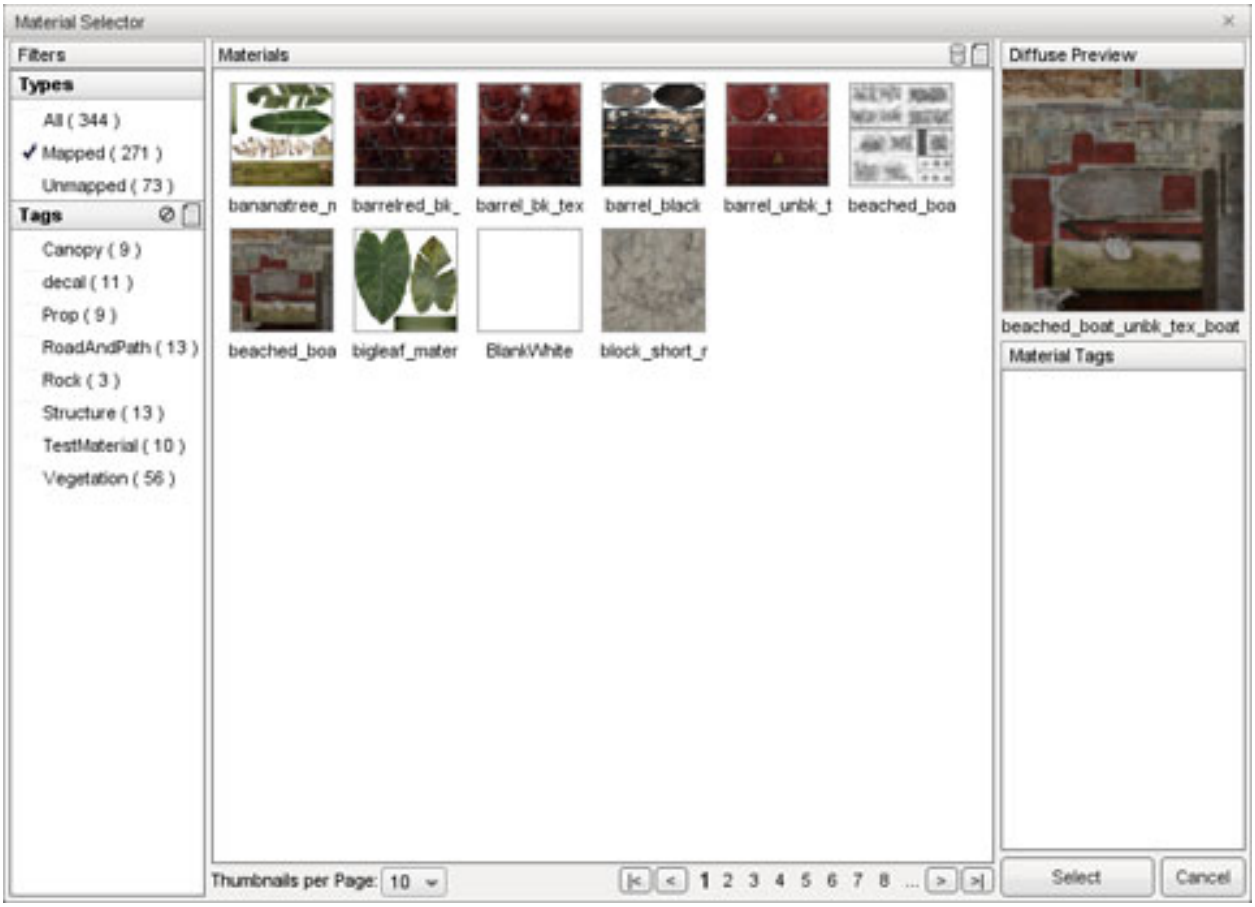
Double Sided Determines if this material will be rendered from both sides of a polygon.

Blending Box Determines type of blending and reflection applied on the transparent object.

Material Selector

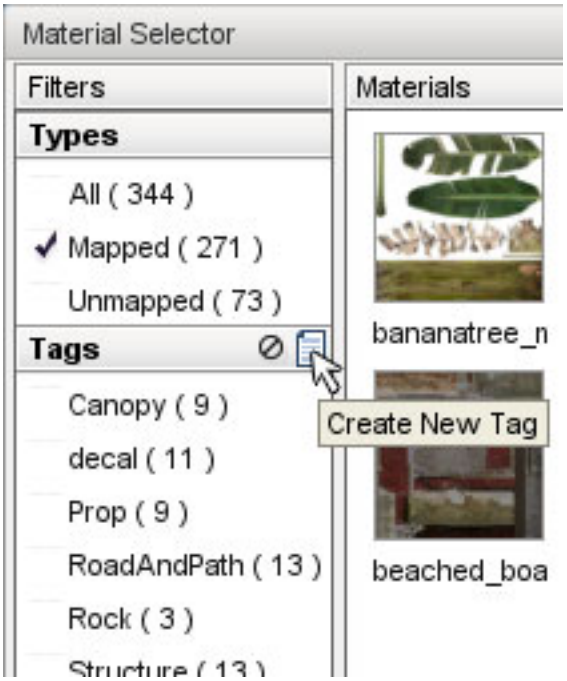
When you wish to swap the material mapped to an object or create a new material, you will use the Material Selector. To change the material on an object, it must first be selected. If you do not know how to select an object, refer to the Object Editor documentation, then switch back to the Material Editor (F4). The Material Properties pane on the right side of the screen displays the properties that describe the material of the selected object.

At the top-right of the pane there is a value named Material. Click on the globe to the right side of it. This will bring up the Material Selector window.



The center section of this dialog displays a list of all materials currently loaded in the game. OClicking on any material selects it which will cause the panes on the right to update and display information about the material. This information is limited to a preview of the material's Diffuse texture, the name of the diffuse texture, and a list of filter tags.

On the left is a list of filters. The filter system is used to organize your materials for ease of use, and contains types and tags. To create a new tag, click the new tag button:



The Create New Tag dialog will pop up. Enter a name for your new the tag then click the Create button. In this example, you will be grouping all the materials related to cliffs. Whenever a material is selected, the Material Tags section on the right will be updated to show all the tags that you have created, each with a checkbox. Clicking the box of a specific tag will associate that tag with the current material. Clicking a checked box will dissociate the tag from the material.

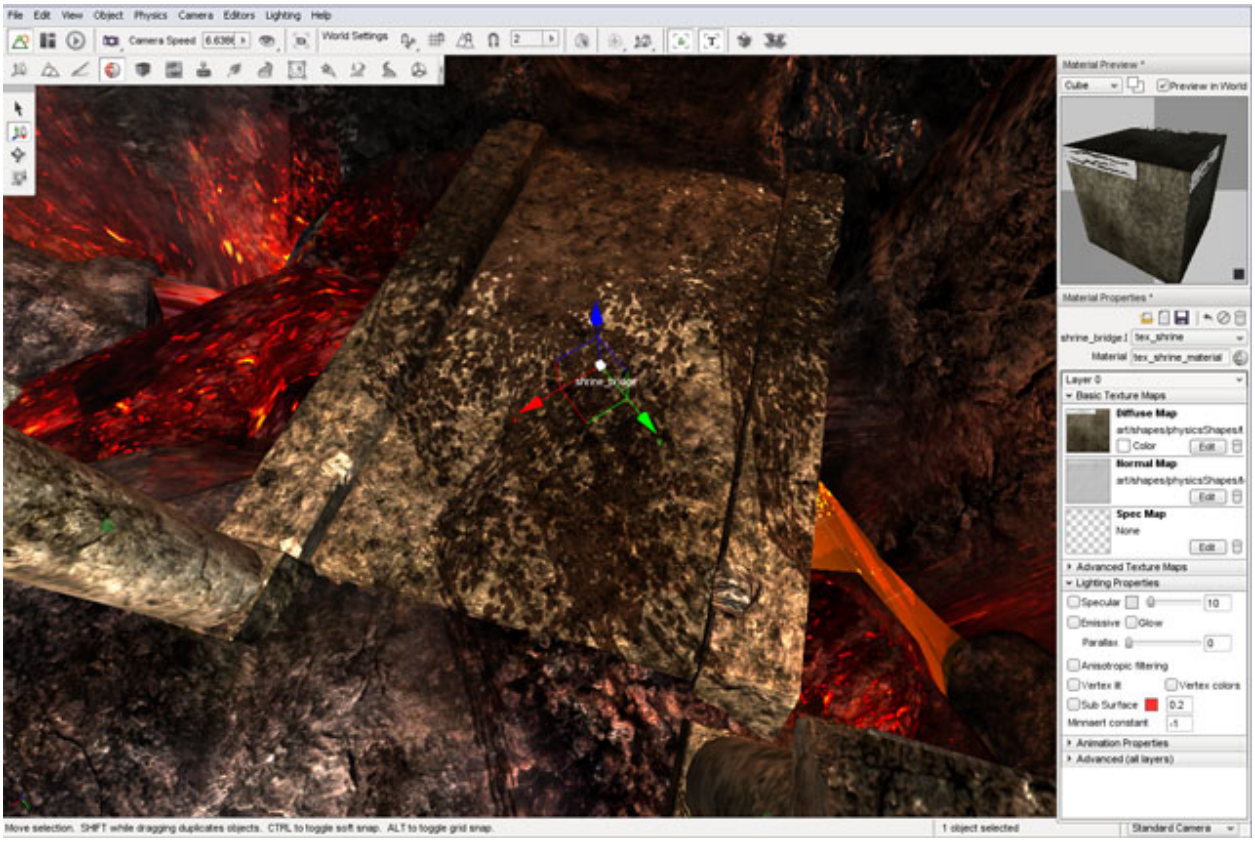
The list of materials can be filtered using the tags assigned to them. To filter the material list use the tags section on the far left. When you click on the check box for tag it tells the system to include materials that have that tag in the list. Any materials that do not have at least one of the checked tags will be filtered out of the list.

Editing an Existing Material

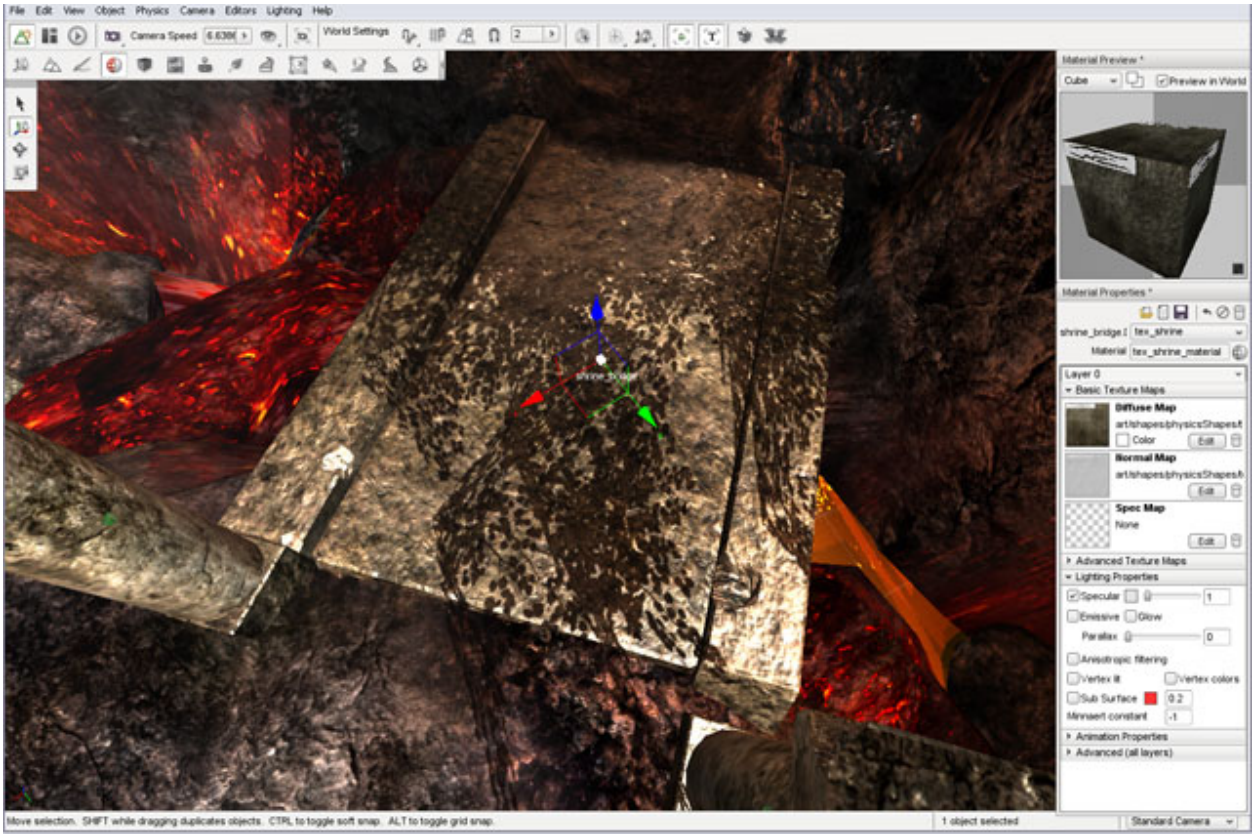
Your game's levels can potentially contain thousands of different objects with varying purposes: explosive barrels, ammo crates, static light fixtures, solid walls, etc. Each one will have a material that might need subtle tweaking to fit in, such as a glowing light bulb.

In this example, you will adjust the properties of this bridge materials.

Remember that you can preview the changes in the scene as well as the preview box in the Material Editor. You will start by toggling the Specular property of the material used for the metal pipe. Without Specular enabled, an object will not have a shine and will thus appear flat.

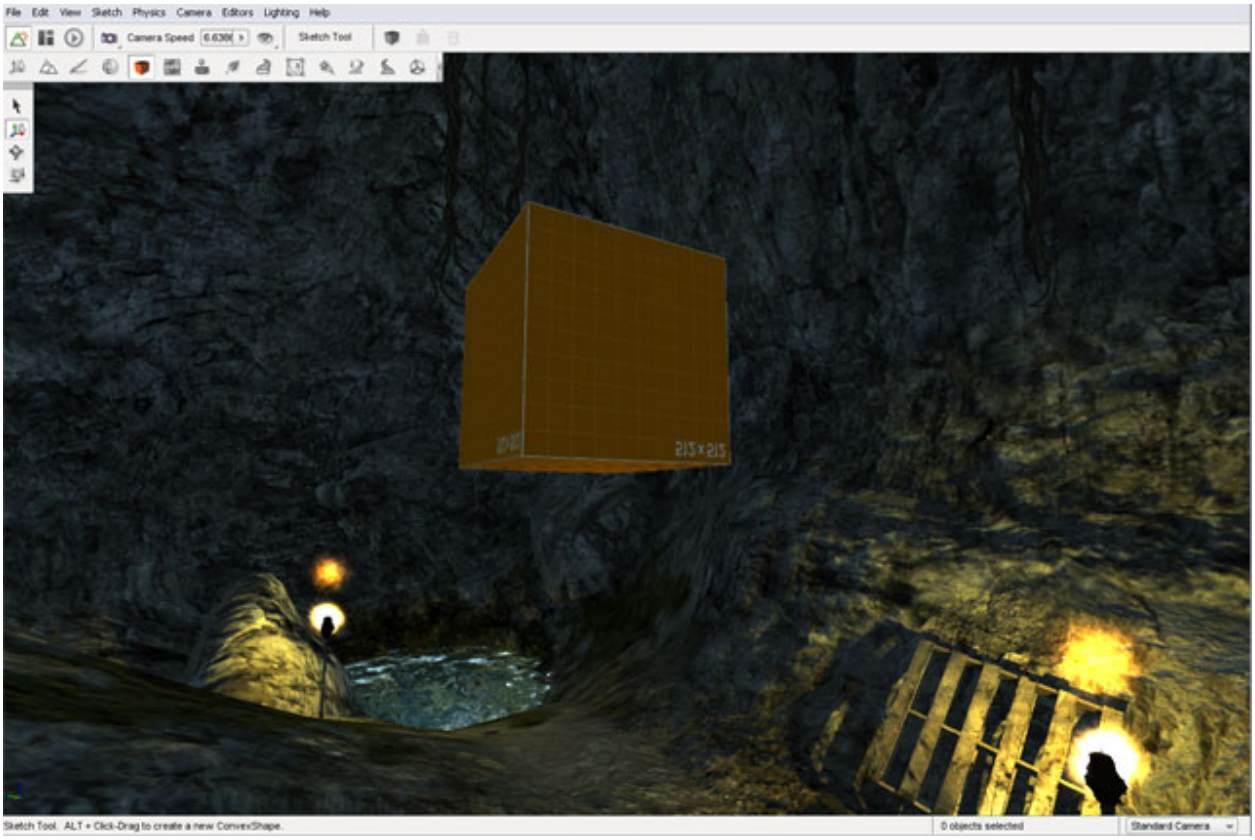


When the Specular property is enabled, the cube in the preview box will have a shiny appearance. In the scene, the metal will also be shinier due to the lighting reflection.



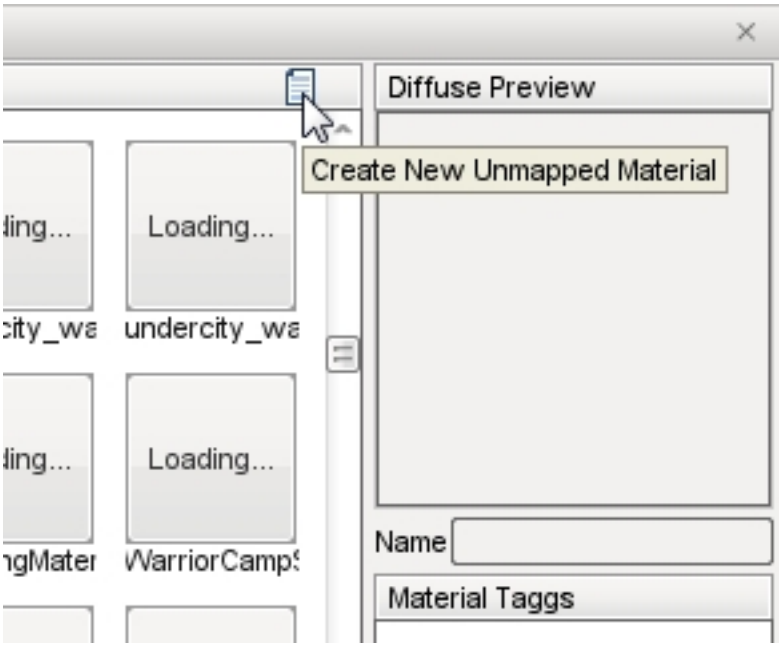
Creating a New Material

While developing your game, you will most likely be using your own assets. When you add a model to the scene, it will be assigned the default "No Material" texture which serves as a warning to the designer that no material has been assigned to an object. This material is automatically used for all assets before they have a mapped materials.



If you have already created the textures for your object, creating and assigning a material is a simple process. Start by clicking the globe symbol next to the Material name box.

The Material Selector dialog will appear. Click the Create New Unmapped Material button found at the top right of the Material section's header.



A new material will be added to the list with a name similar to newMaterial_0. Click on the material to view it in the Diffuse Preview section.

Click the Select button to use that selected material for the object you are editing. After the Material Selector closes, you will be prompted to save any material changes that you may have made before entering the Material Selector. Do so if you wish to retain any changes that you made prior to creating the material.

Your new material will have replaced the material selection in the Material Properties pane back in the Material Editor and should now be displayed in the Material field. Type in the real name you want for your new material to be known by then press the Enter key. In this example, the name of the material is "boxxy."

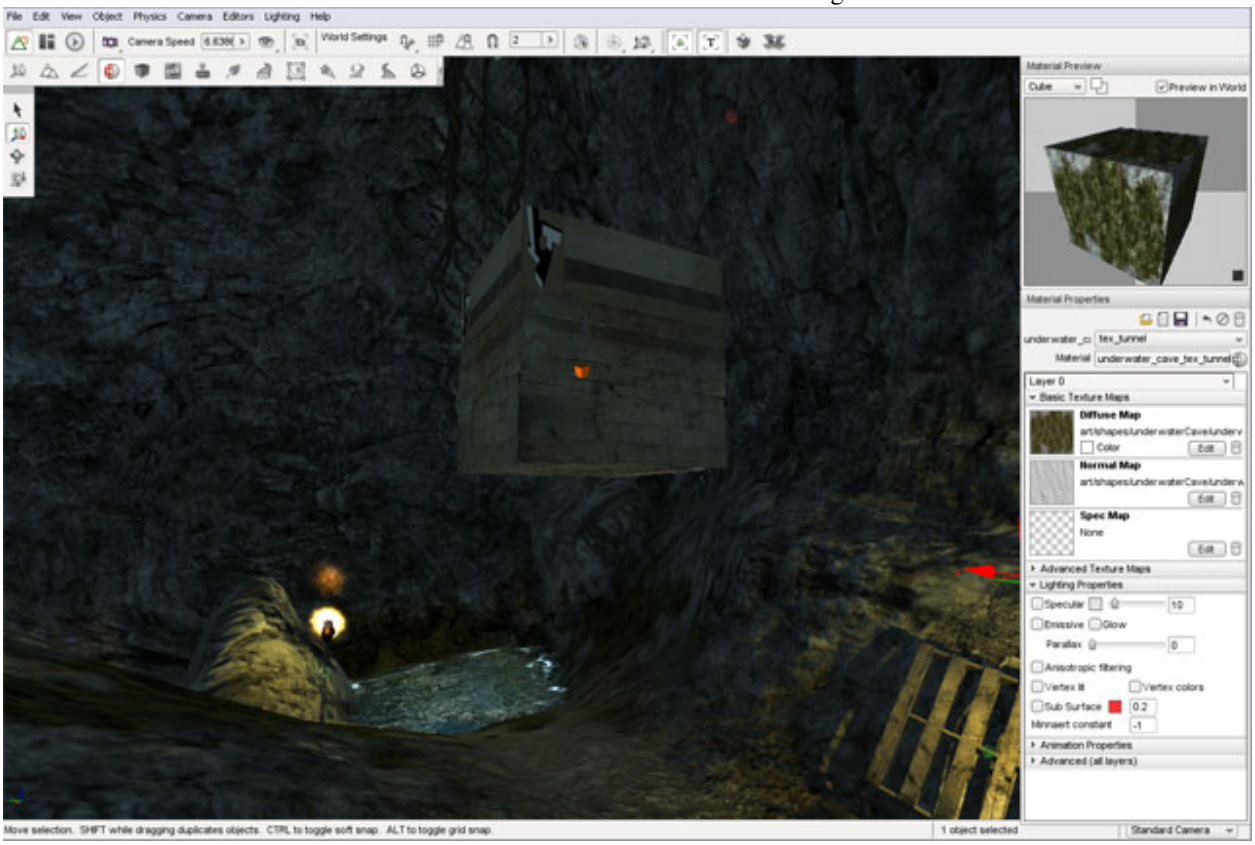
Before editing anything else, click the Save Material button, represented by the floppy disk symbol to save the new material.

Note: You MUST press the Enter key after typing the material name BEFORE clicking the Save Material button or the material will not be properly saved.

Now, scroll down to the Texture Maps section of the Material Editor. This is where you will be adding the actual texture files that define this new material. Click on the Diffuse Map preview or the Edit button in that section to open a file browser. Navigate to your diffuse texture, or sometimes referred to as the base texture. Select the file that you want to use as for this new material then click the Open button.

Your preview window and scene should immediately be updated to reflect the addition of your texture.

Repeat the process to add your Normal map. Click on the preview or edit button in the Normal Map section. When the file browser appears, select your normal map texture. Once again, your scene will be updated to reflect the changes that have been made to the material. Click the save button to retain these changes.



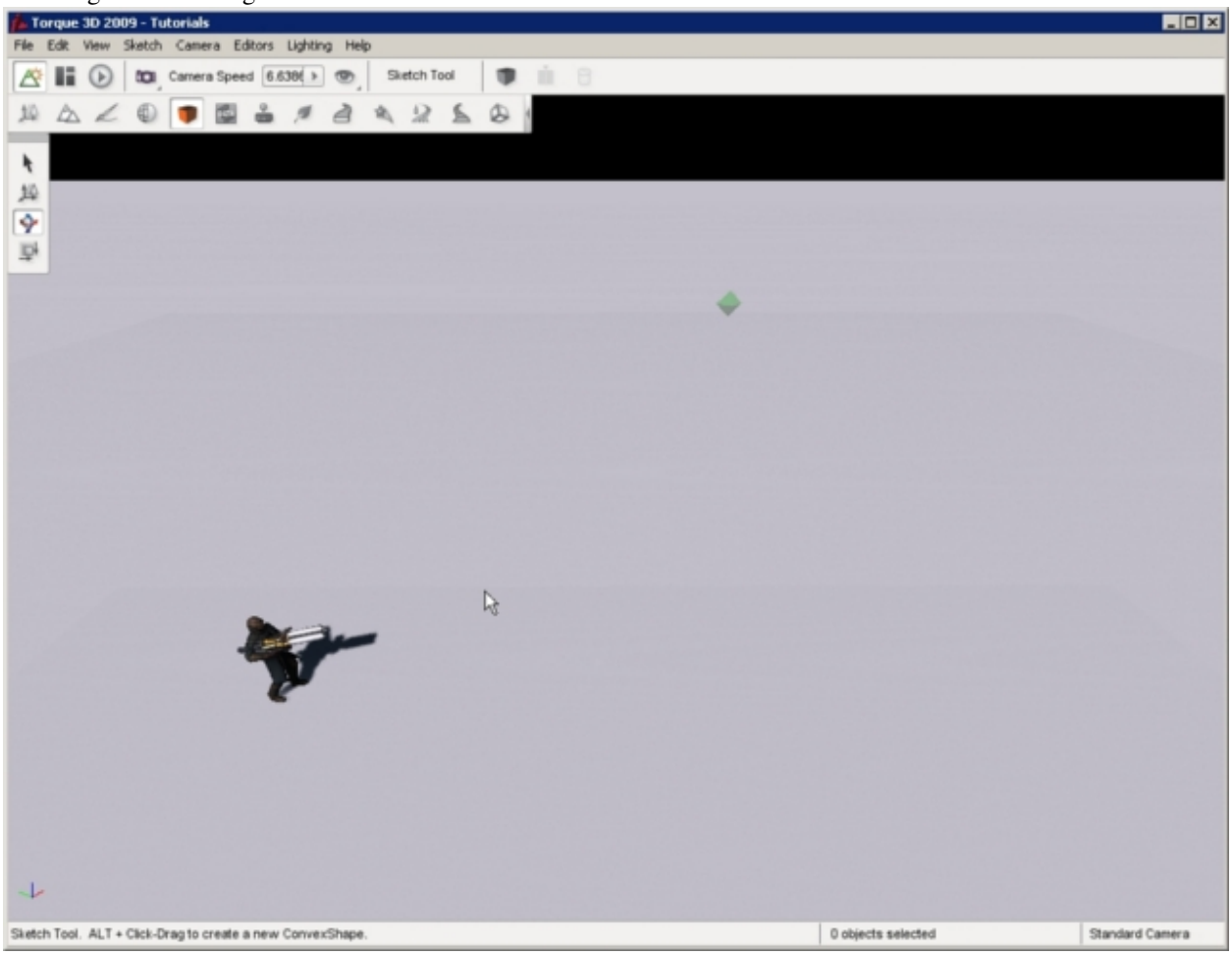
If you open the Material Selector again, you will notice your new material has been saved in the list. This material is now available to be assigned to any other meshes within the project without having to go through the whole process of redefining it again.

2.4.4 Sketch Tool

The Sketch Tool is a tool that allows you to quickly generate meshes without going to 3rd party modeling applications, such as Maya or 3DS Max. It is not meant to create final or game ready art, just rough shapes that are placeholders for your real art. For example, you can use this tool to sketch the shape of a building you want. The rough design can fit your needs for a simple design and estimated measurements.

Interface

To switch to the Material Editor press the `F4` key or from the main menu select Editors > Material Editor or click on the orange box icon to get started.



The Tools Palette will populate with basic manipulation icons:

Select Object Select a convex object or individual face

Translate Move an individual face

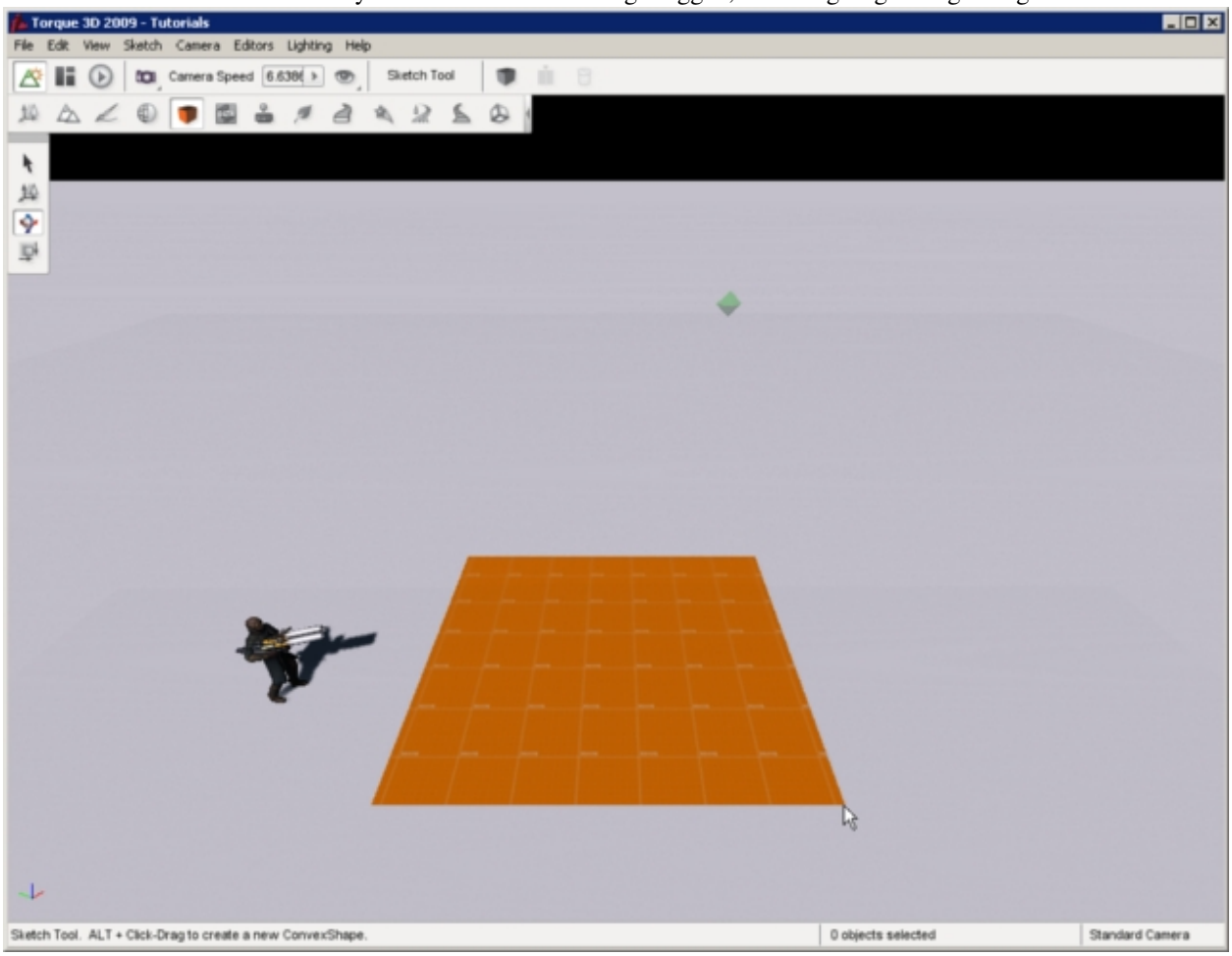
Rotate Object Rotate an individual face

Scale Object Grow or shrink an individual face

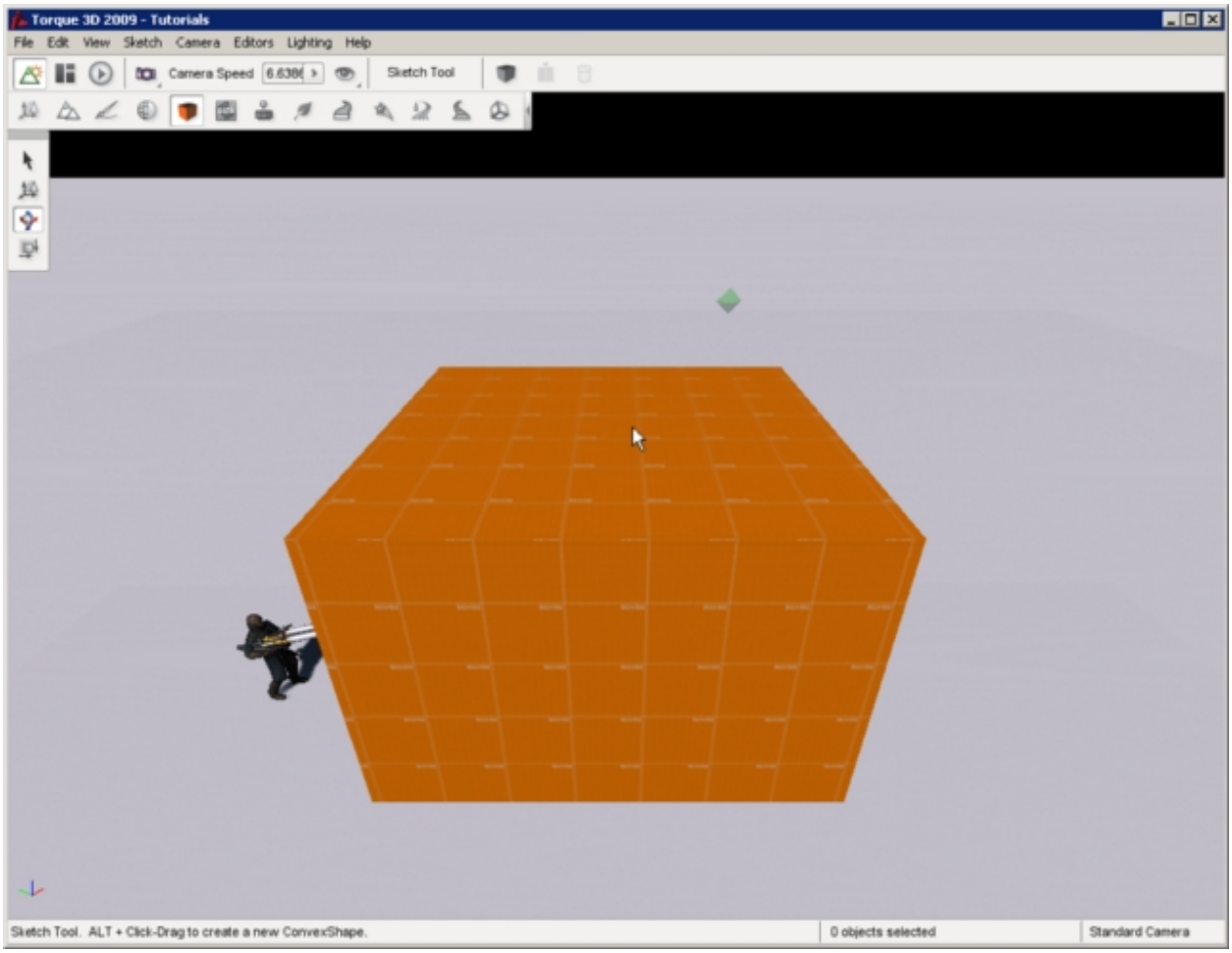
As with the other editors, extremely helpful usage hints will be displayed in the bottom left corner of the editor. Shortcuts and basic descriptions will appear based on which tool you are using.

Creating a Convex Shape

The very basic interface allows you to quickly sketch out convex shapes. All of your editing can be performed via mouse actions. To begin creating a convex shape, hold down the Alt key and left mouse button to begin drawing a base. The base will follow where your mouse cursor is being dragged, shrinking or growing as it goes.

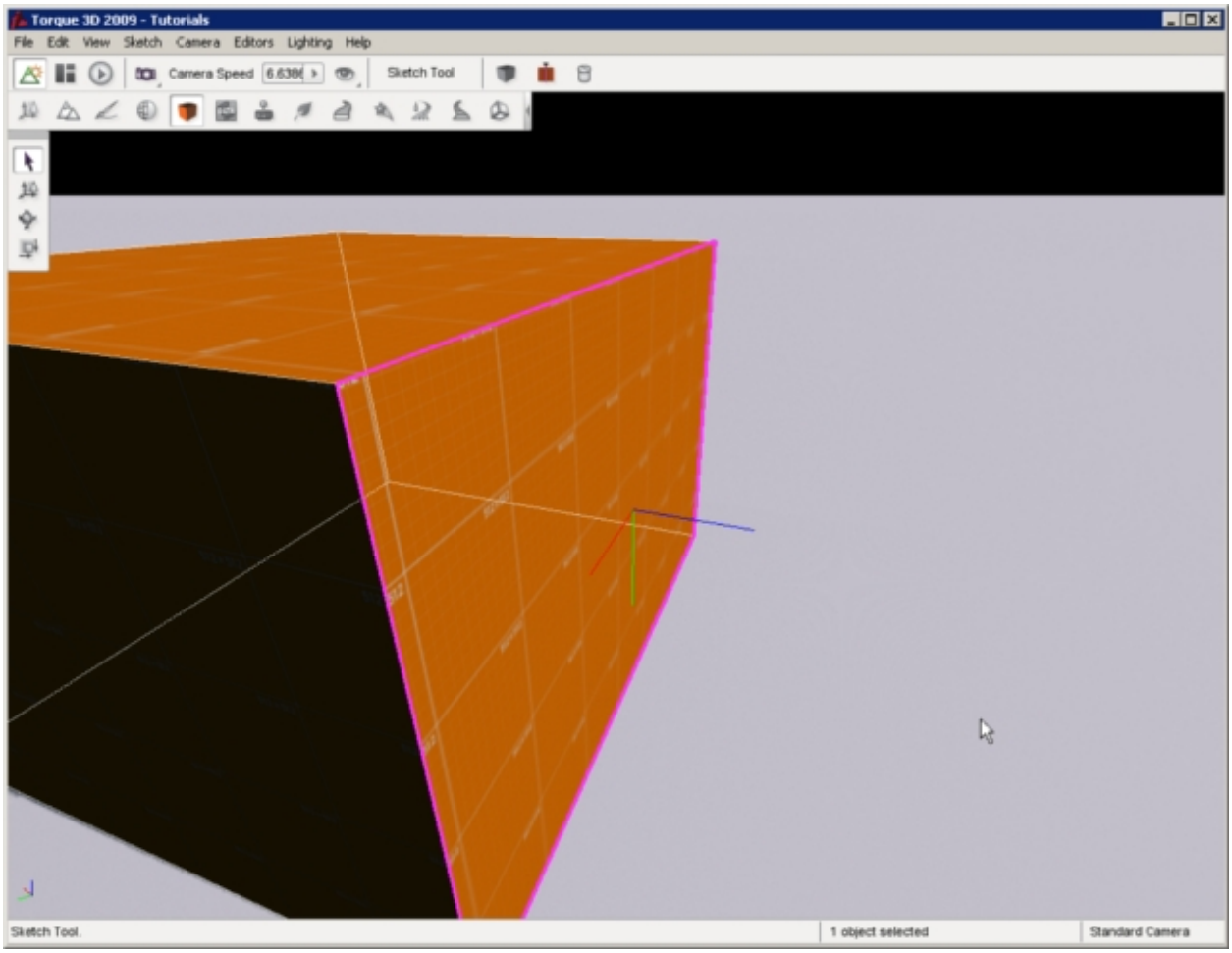


Once you let go of your mouse button, the base will stop growing. From here you can move your mouse cursor up and down to change the height of your new box. You do not have to hold down the mouse button during this time.



Once you are happy with your convex shape's height, left click one last time. The box will become a solid object and automatically be selected. If you make a mistake, hit `Ctrl-Z` to undo and erase the shape then repeat the process.

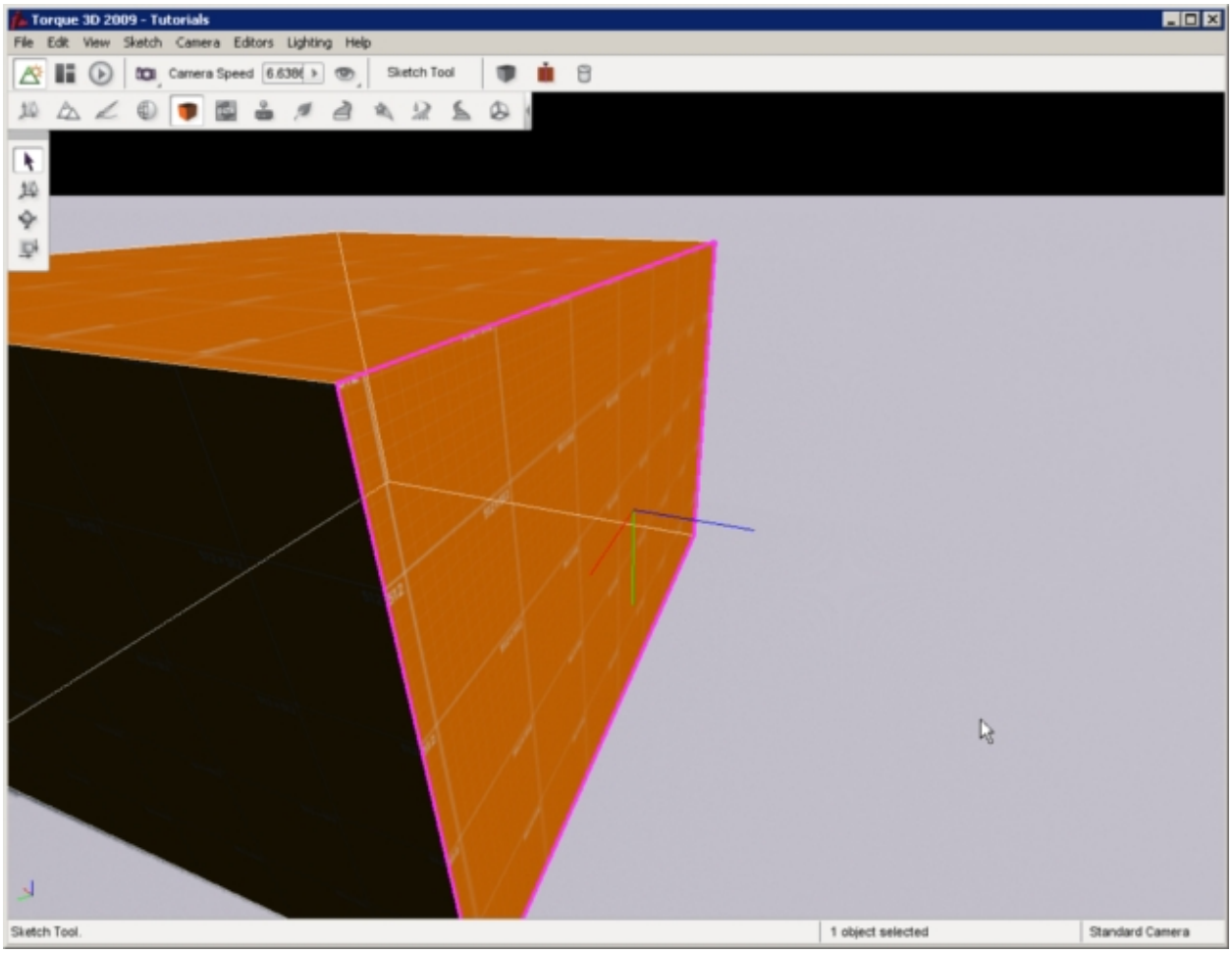
When you are ready to begin shaping the box, left click one of its faces. The currently selected face will be highlighted in bright pink:



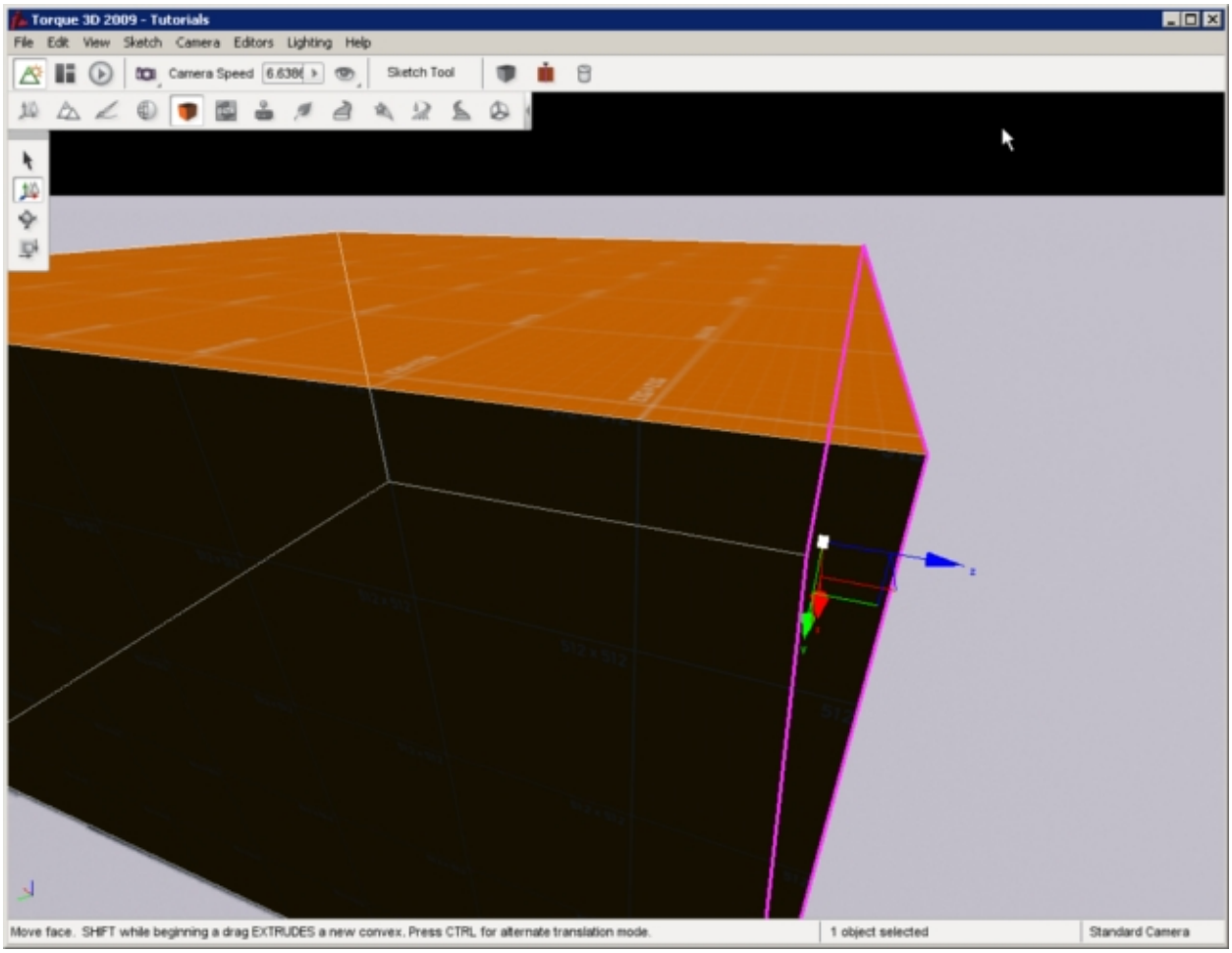
At this point, you can start using the Sketch tools to edit the specific face you have selected.

Editing a Convex Shape

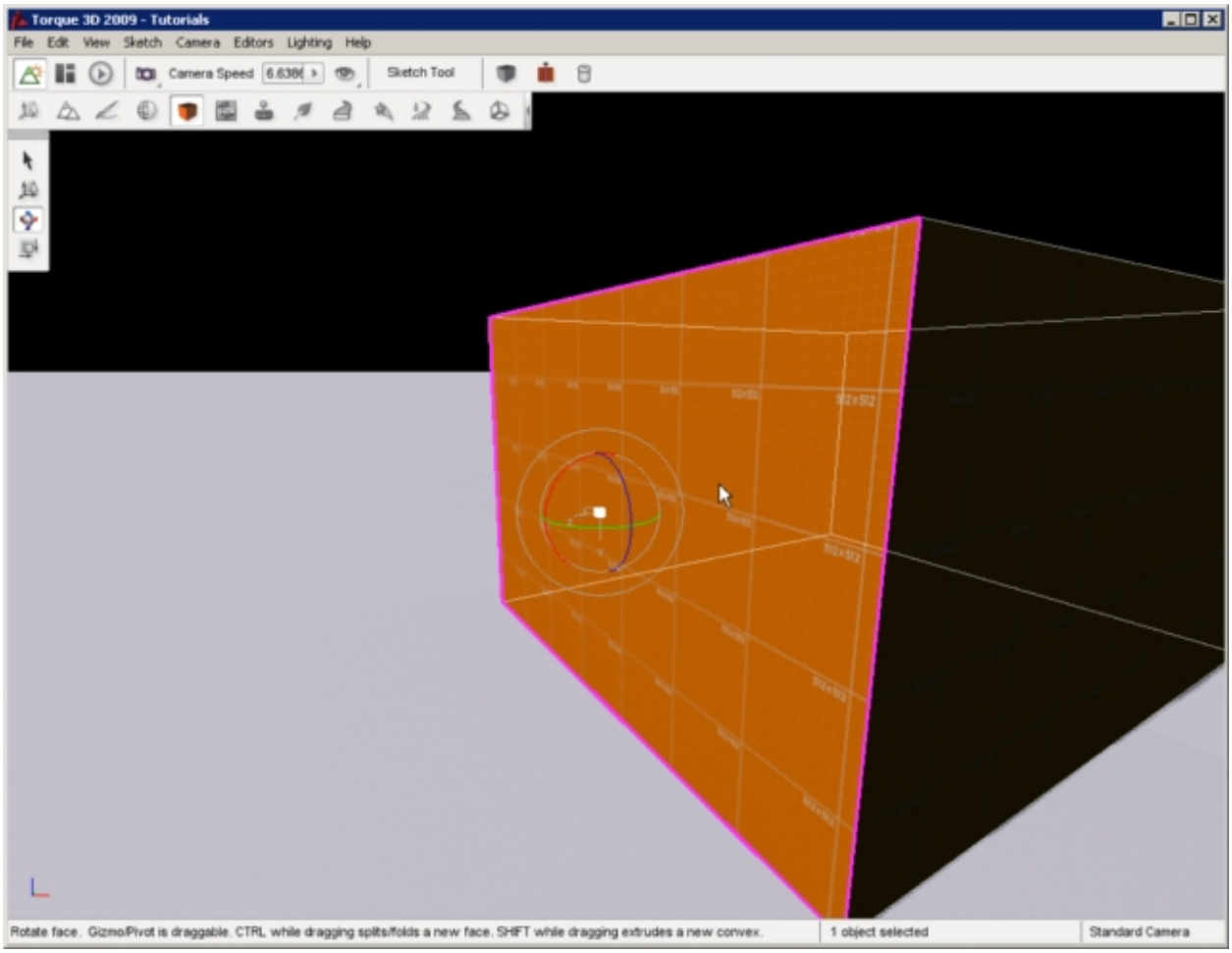
Let's move some surfaces around. Start by selecting a faceof the object by (left clicking on it). Three colored lines will now extend from the center of that face - these represent the axes for the three dimensions x, y and z. This is called the axis gizmo. Activate the Move Selection tool by clicking the icon on the Tools Palette on the left of the screen or press the shortcut key `2`:



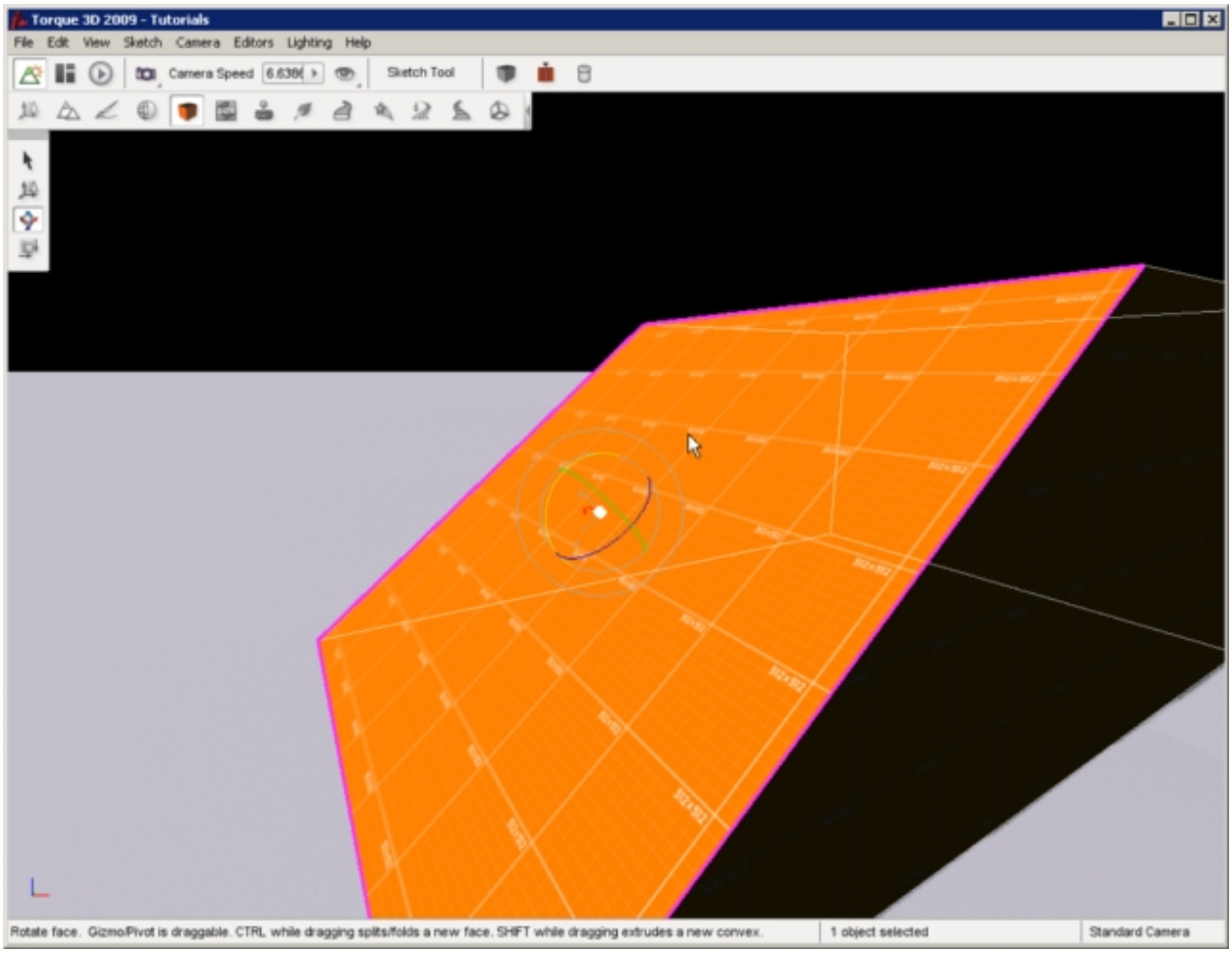
Once the Move Selection tool is activated, arrows will appear on the ends of the axis gizmo. Click on the X-axis and drag it outward. Your face will move in the direction you are dragging your mouse. The entire convex shape will adjust according to where the face as moved. You will be able to move the face in any direction in three-space:



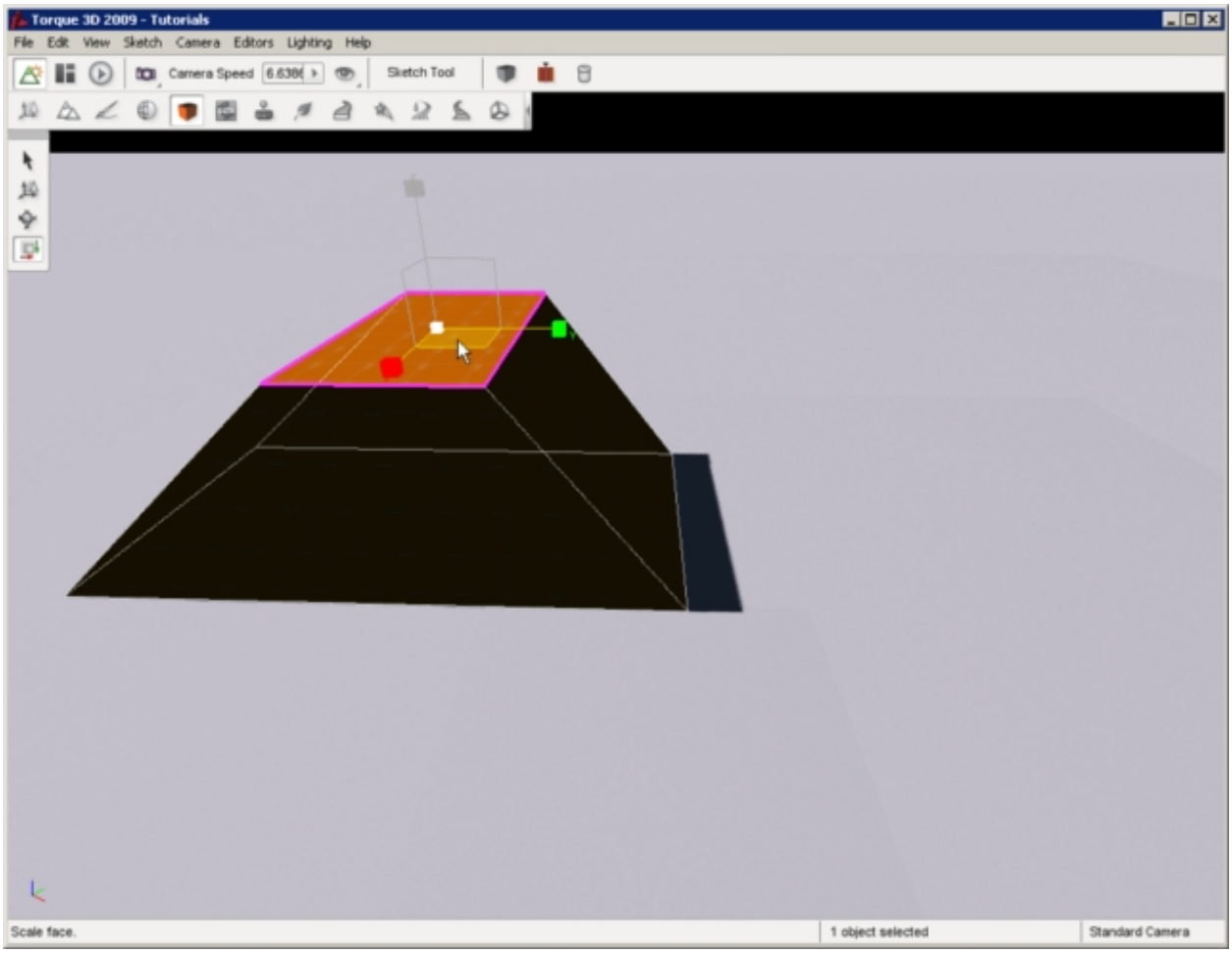
Next, activate the Rotate Selection tool by clicking the icon on the Tools Palette on the left of the screen or press the shortcut key `3`. A spherical gizmo will appear representing the orientation manipulators. The axis gizmo straight lines will now be displayed as three curved colored lines:



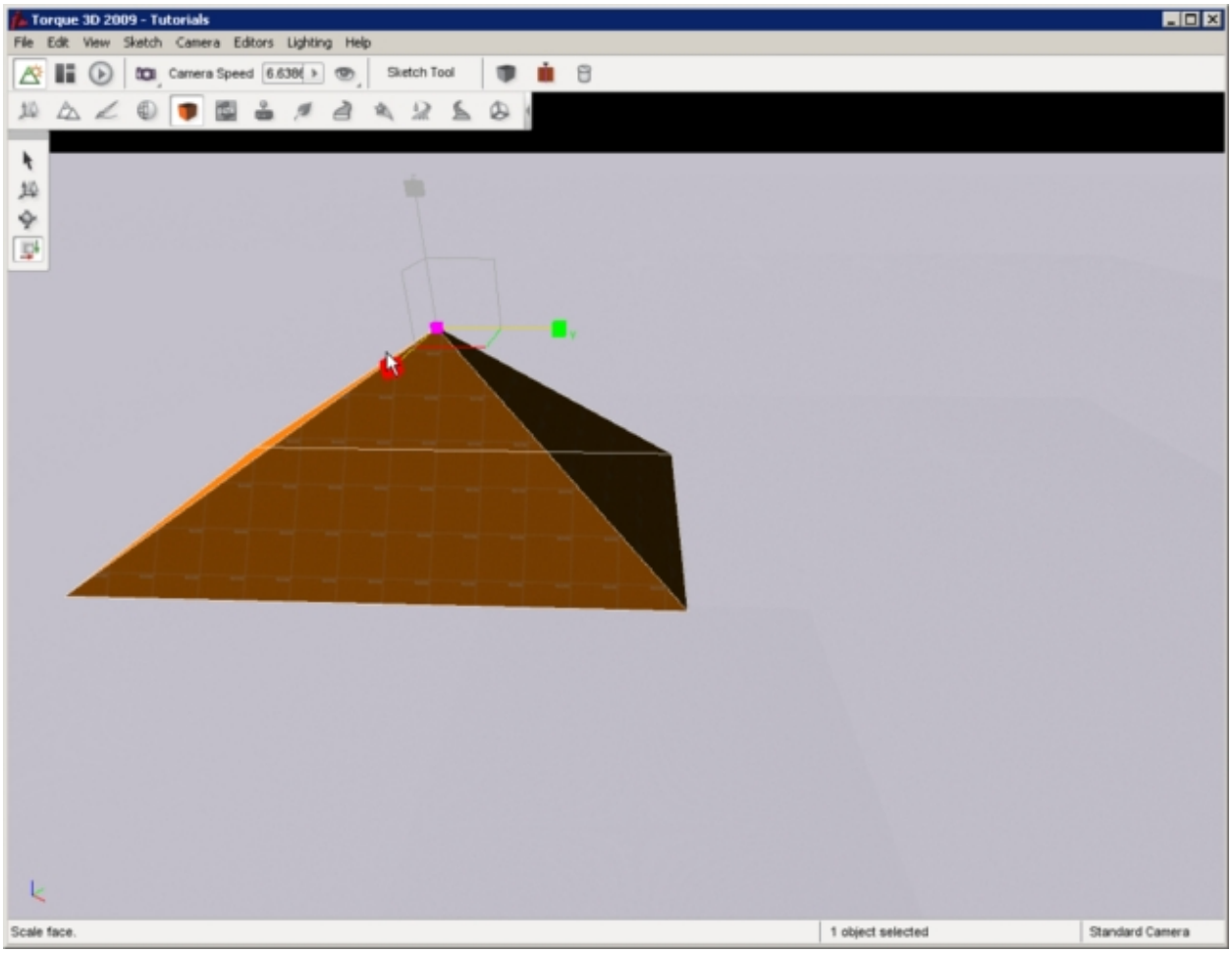
Click and hold one of the colored lines (an axis), and drag it in a direction. The selected face will begin to slope according to the new orientation:



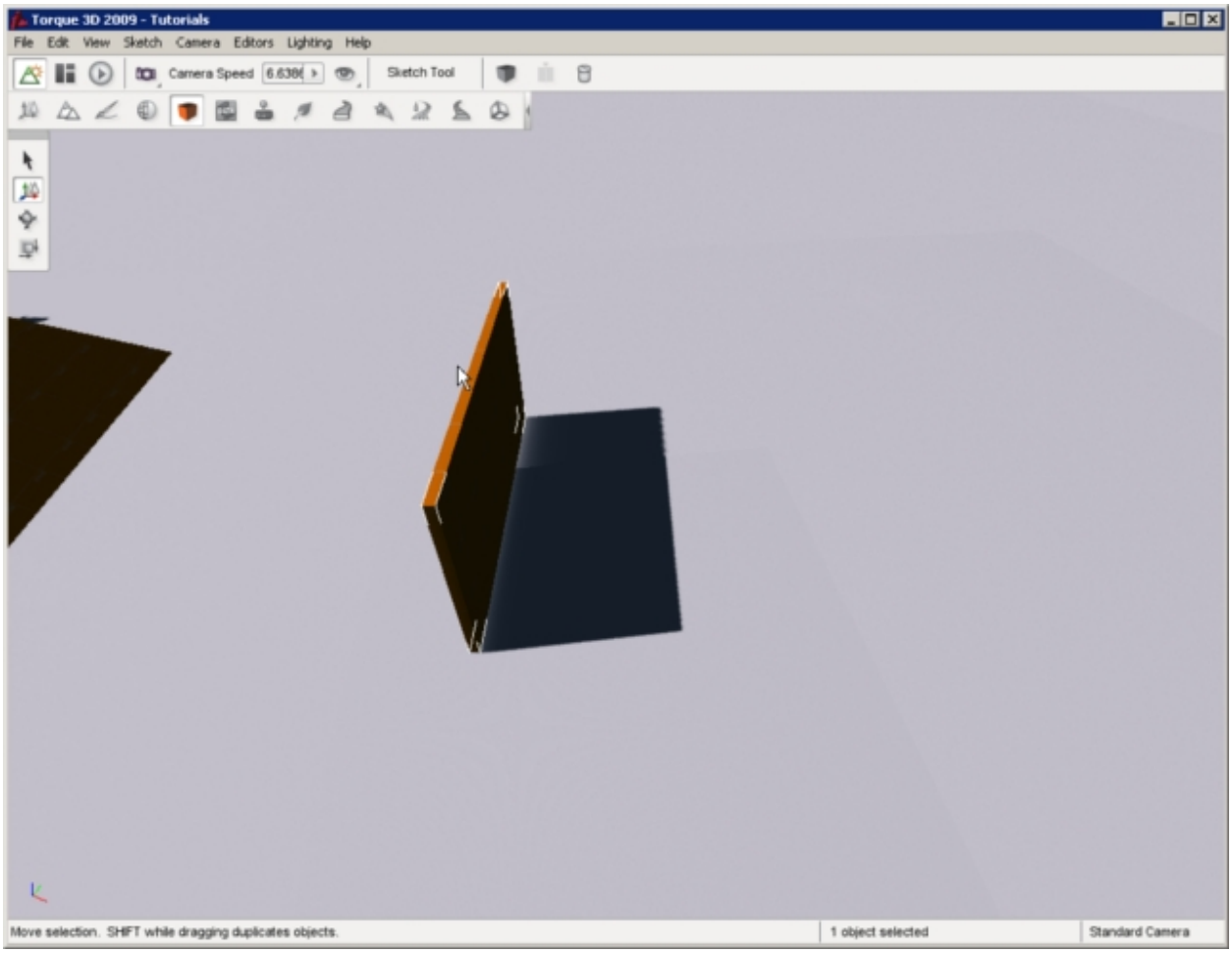
Finally, activate the Scale Selection tool by clicking the icon on the Tools Palette on the left of the screen or pressing the shortcut key `4`. Click on the top face of the box. A squared like gizmo will appear which will allow you to choose what parameters to adjust. You can adjust the (width, height, depth, or any combination of the three):



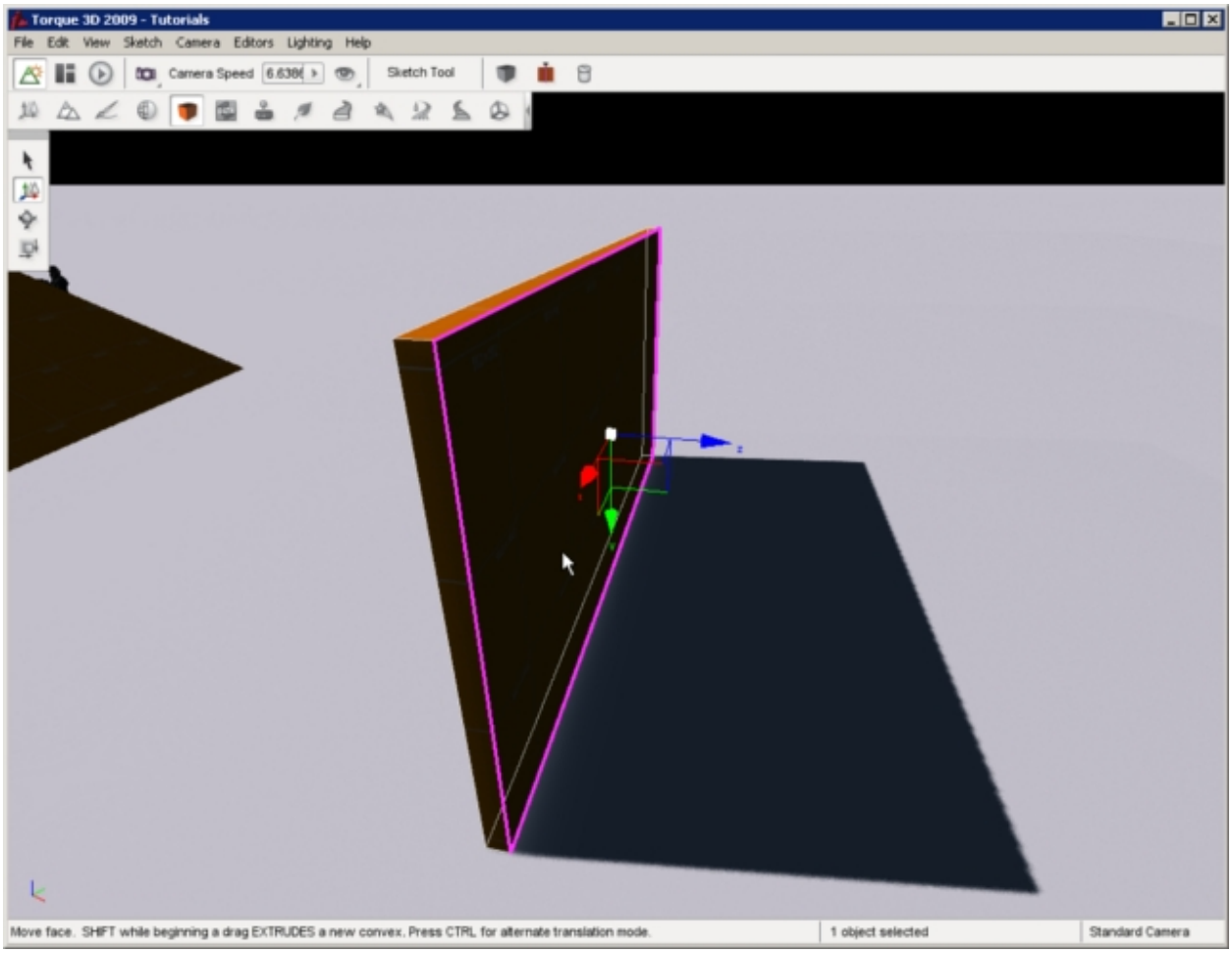
Instead of adjusting one parameter at a time, we are going to adjust width and height. Move your mouse over the different squares to see how they highlight. Click the bottom square of the gizmo, in between the red X and yellow Y axis and hold down the mouse button. Drag your mouse in either direction to shrink or grow the face. The more you shrink, the more like a pyramid it will be come:



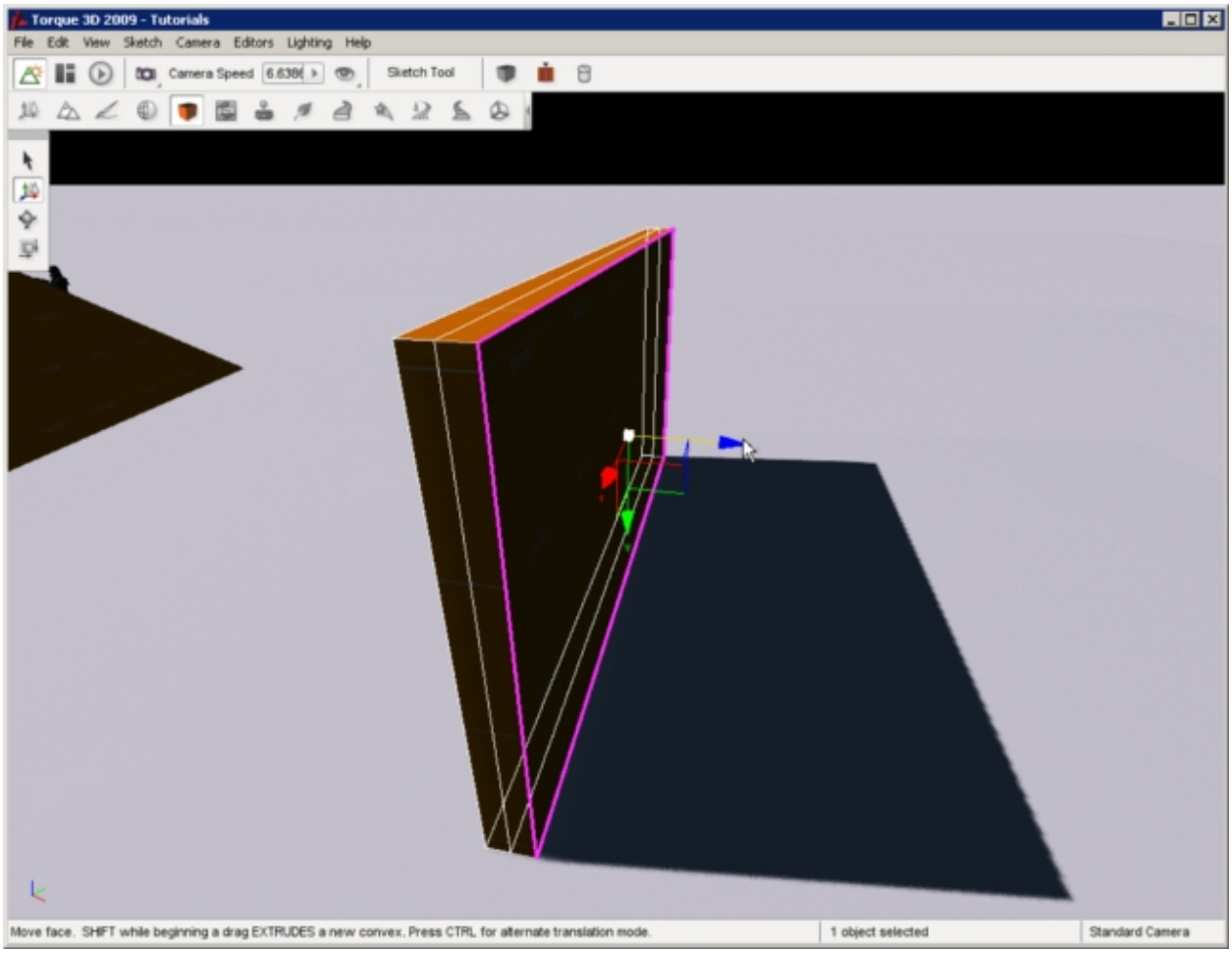
The last action this guide will address is extruding. The Sketch Tool extruding feature creates new geometry from a selected a face. Start by creating a new convex shape - review the section, Creating a Convex Shape, if you don't remember how.



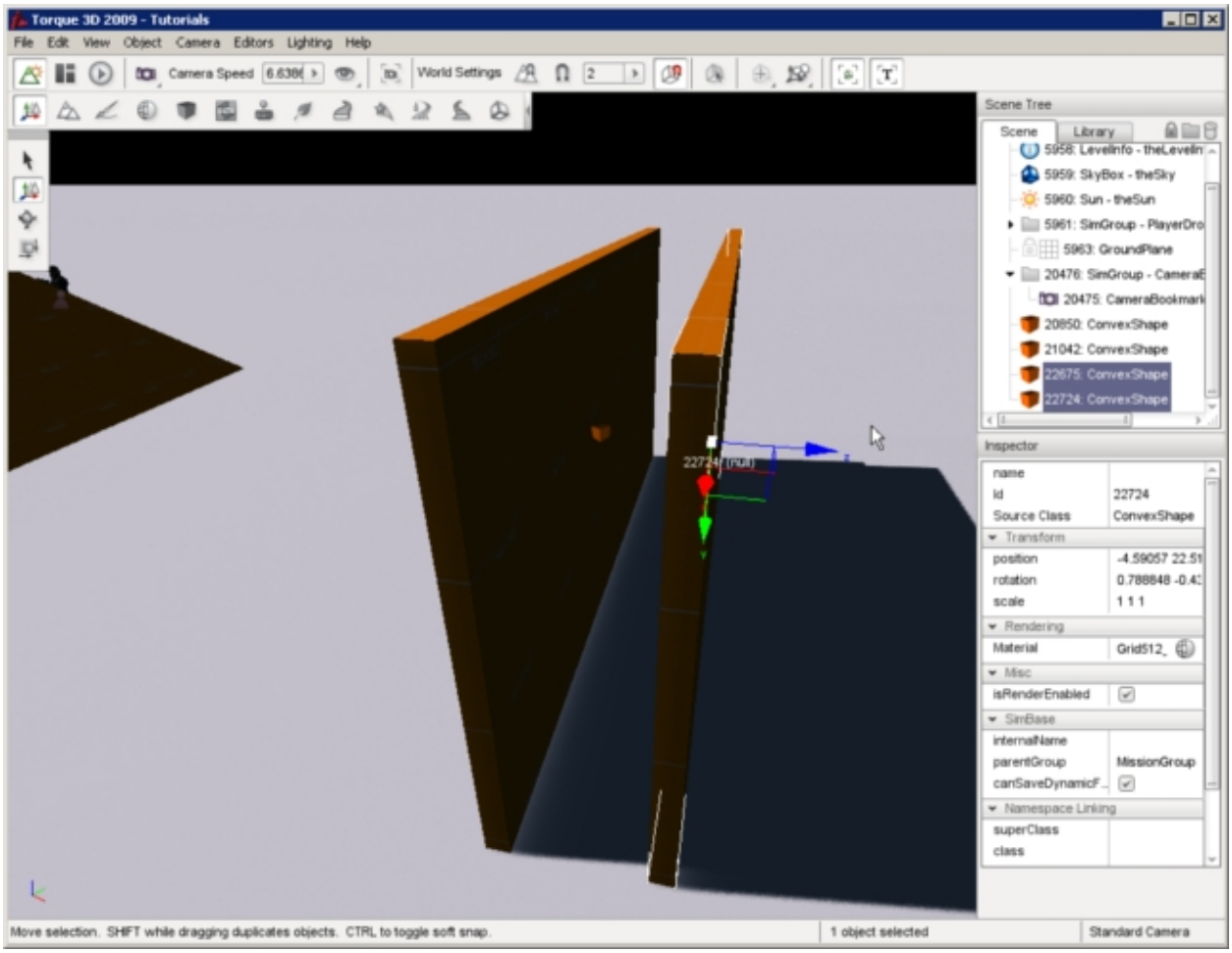
Next click on a single face of the shape. Make sure you have a face selected, and not the entire object. The selected face should be highlighted with a in bright pink. Activate the Move Selection tool. A hint will display at the bottom of the editor: "Move selection. `Shift` while beginning a drag *extrudes* a new convex."



Perform this action as described. With the face selected, hold down the `Shift` key, move the mouse over one of the colored arrows, and click and drag outwards from the object. The exact dimensions of the original face will be duplicated, constructing a new convex based on those parameters. This may not be apparent until you click on a face and see that the area of the new face is separate from the original:



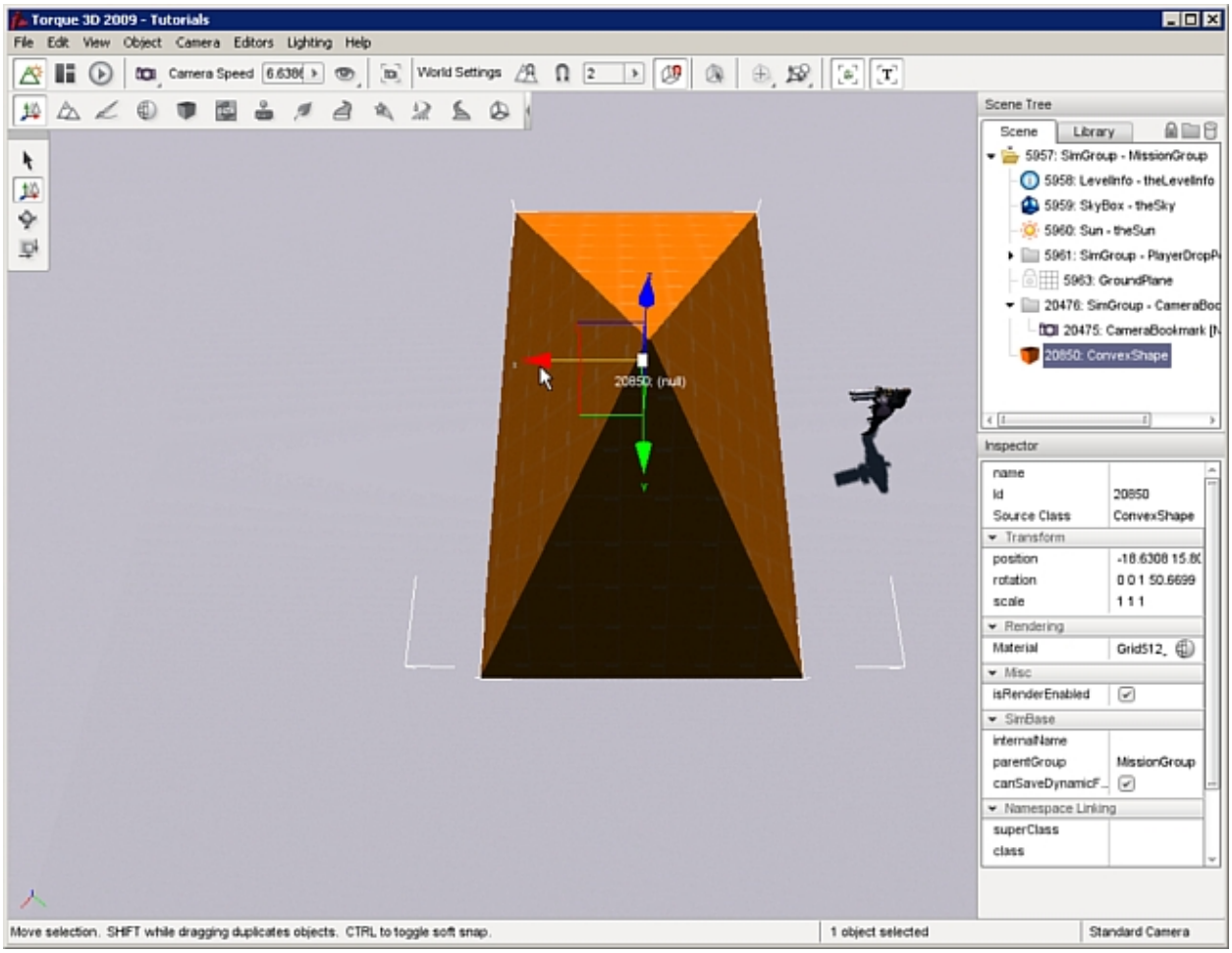
You can now select faces on the new convex object and continue editing it as a new object:



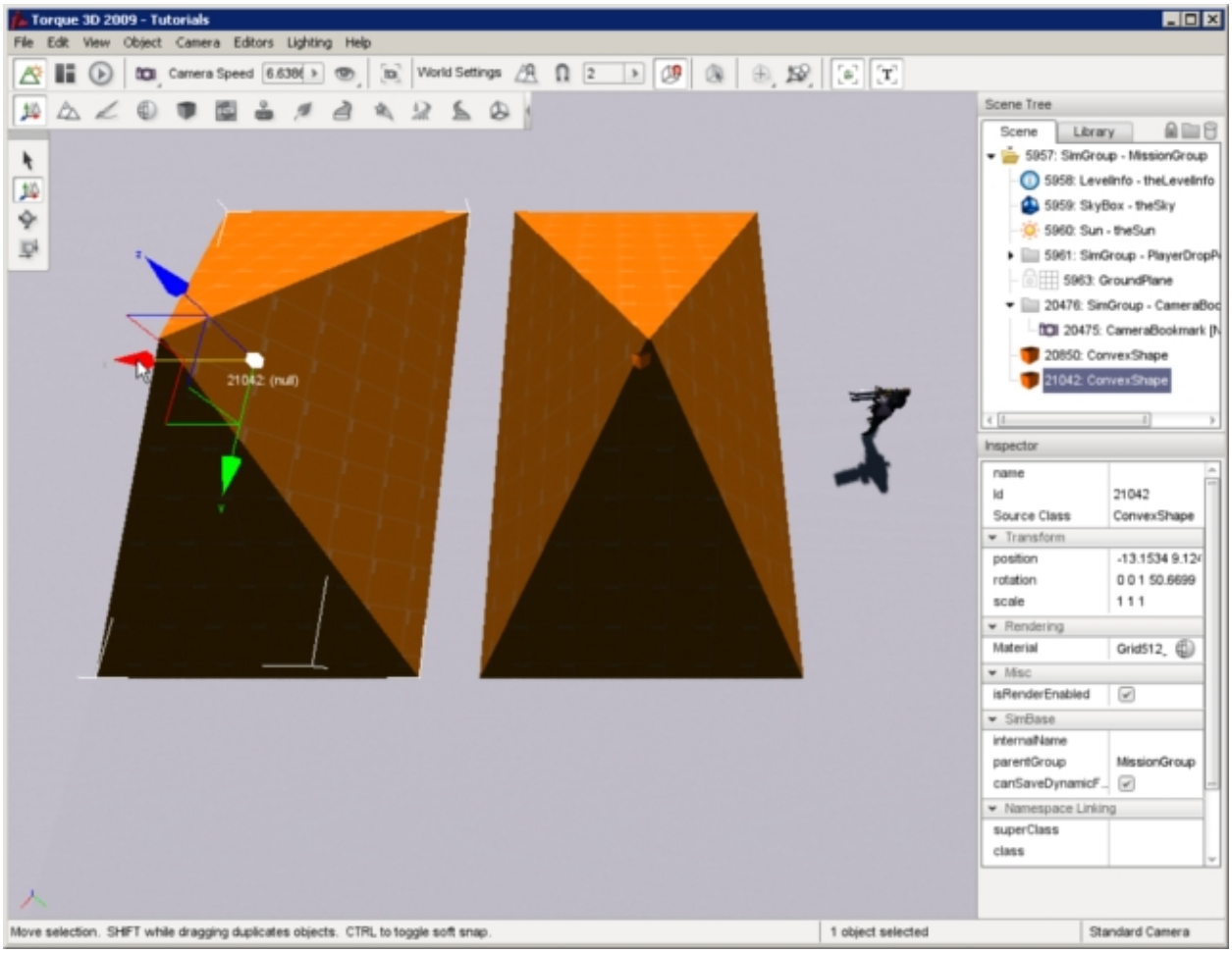
Object Manipulation

When you are finished sculpting a convex shape, you can manipulate it as you would with any other game object using the Object Editor. This includes selection, translation/rotation/scaling, and editing specific properties.

Unlike the Sketch Tool, selecting a convex shape using the Object Editor it treats the object as a whole. There is no individual face selection. Switch to the Object Editor by pressing the `F1` key then click on one of your objects:

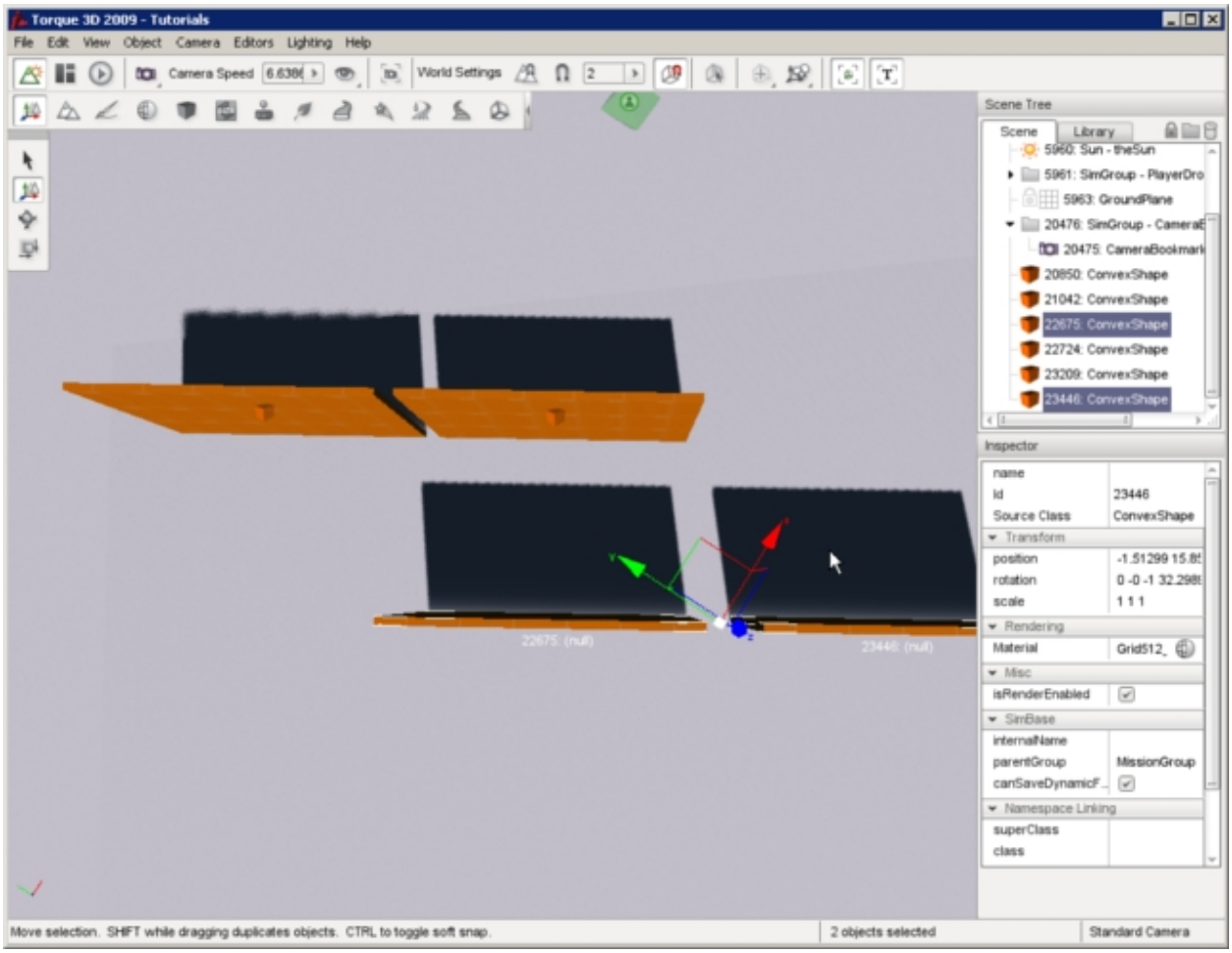


You may then manipulate the object using the normal Move Selection, Rotate Selection, and Scale Selection tools of the Object Editor. You can even use the other more complex Object Editor commands such as copying an object. To copy the object: hold the `Shift` key; activate the Move Selection tool by pressing the `2` hotkey; press and hold the `Shift` key; then drag the mouse to a new position in any direction. When you release the mouse button you will have a new duplicate copy of the original object:

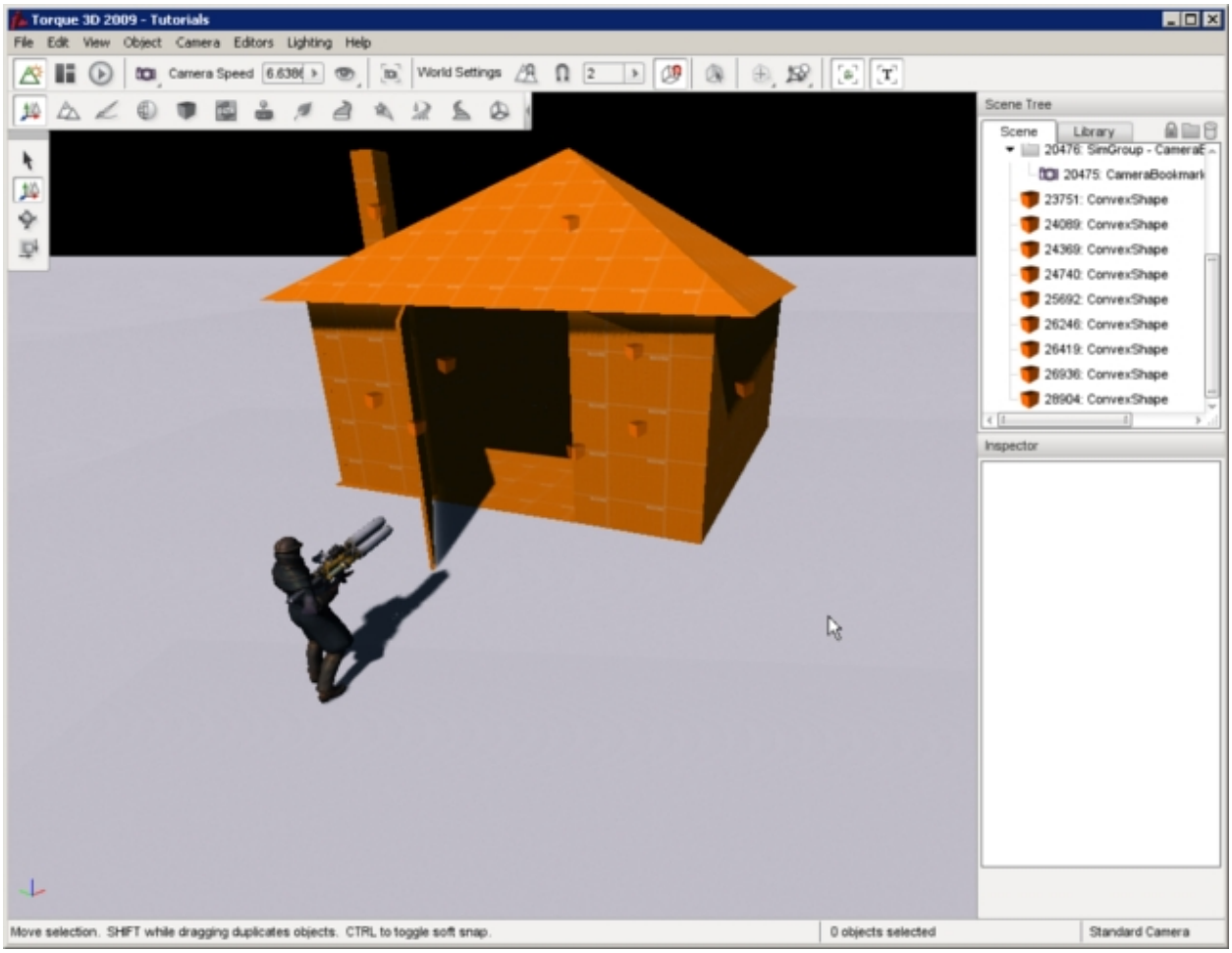


This can also be used for mass production of objects by copying multiple objects at once. Change back to the Select Arrow tool by pressing the `1` hotkey. Click one of your objects to select it. Take note of the position of the gizmo that appears and the little cube at the gizmos origin. Now select the other object. Again take note of the position of the gizmo and the cube for this object. Now press and hold down the `Shift` key then click your other object again. You will notice there are now two small cubes, one over each object, and one gizmo relatively near the center of the two objects. This indicates that both objects are currently selected. You now have a selection group.

Change to the Move Tool by pressing the `2` hotkey. The cubes will disappear, and large arrows will appear on the ends of the gizmo. If you mouse over either object, you will see a faint transparent cube pop up. This indicates that object is a part of the selection group. Clicking any arrow and dragging the mouse will not move all the objects at once. Likewise, pressing and holding down the `Shift` key, then clicking and drag will duplicate all the objects in the selection group:



The new copy of the objects will now be the current selection and they can be moved as a group or immediately copied again with another `Shift`-drag operation. The above method combined with rearranging the individual objects after copying them is a great way to piece together multiple convex shapes to create more complex arrangements. For example, you might have unique convex objects for a roof, wall, chimney, and so on. You could only create one wall, then duplicate it four times so that they are all the same size then arrange them into a building with the Move Selection tool:



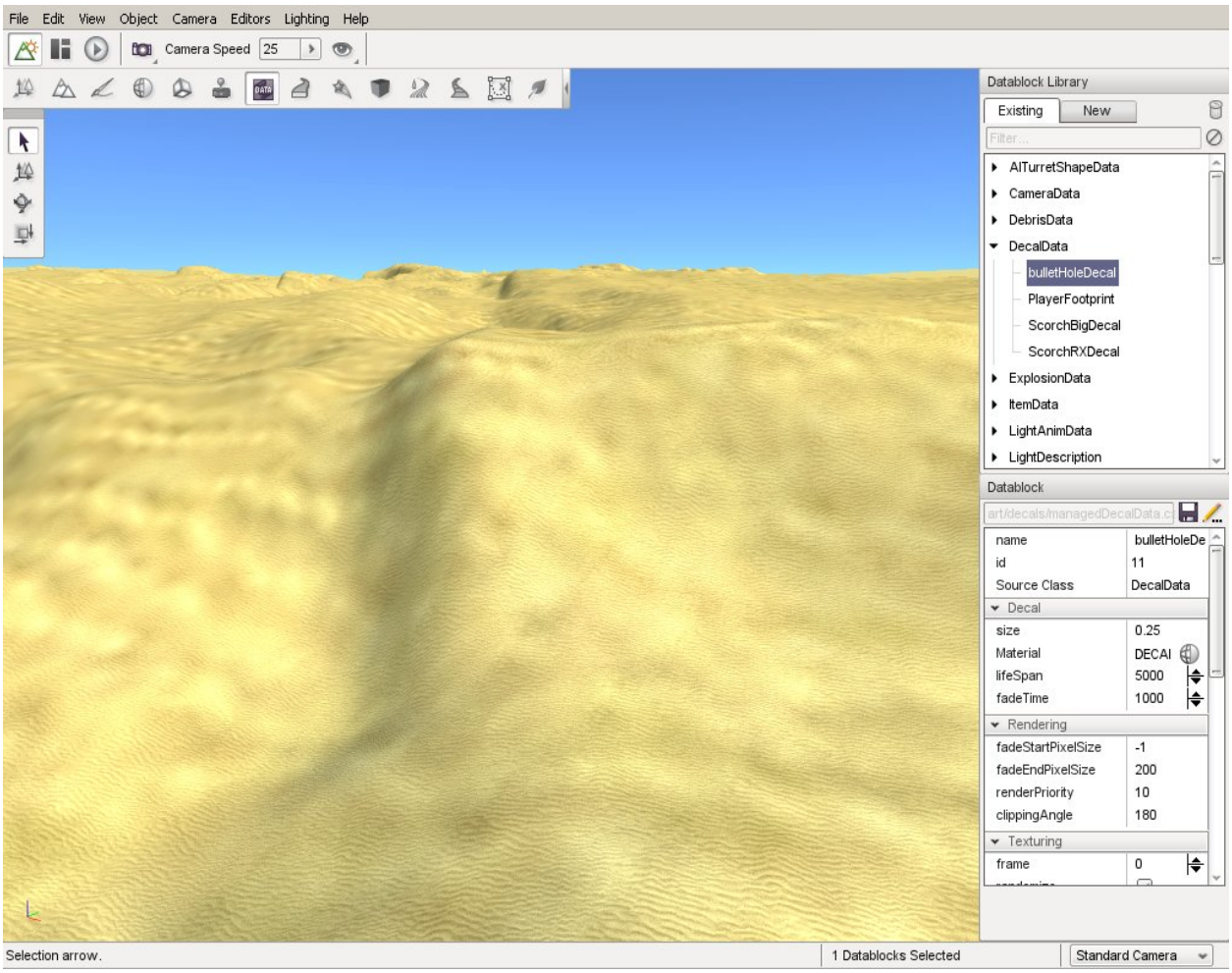
Once you get the hang of the Sketch Tool, you can sculpt unique and complex shapes. Entire levels can be prototyped to use placeholder art, created right inside Torque 3D, while you or your artists work on the final assets using the tools that they are familiar with.

2.4.5 Datablock Editor

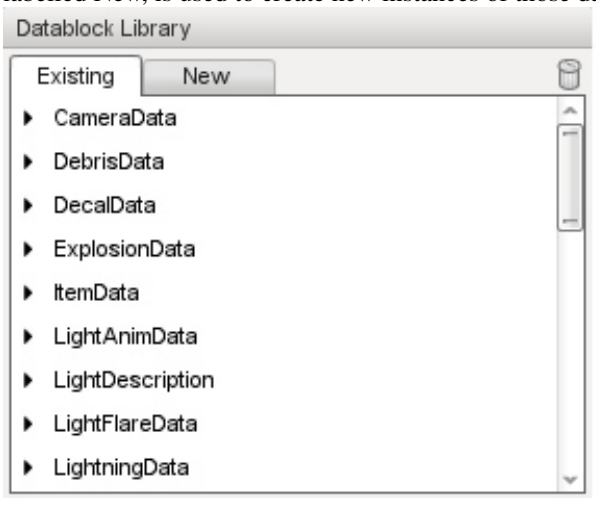
The configuration properties that describe dynamic objects in Torque 3D are stored in information structures called datablocks. The T3D Datablock Editor is used to quickly and easily change any parameter of any datablock from within the world Editor.

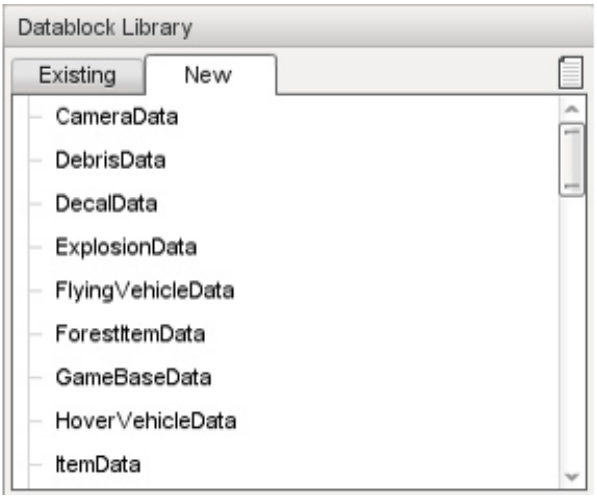
Interface

To switch to the Datablock Editor press the `F6` key or from the main menu select Editors > Datablock Editor. Or alternately click the Datablock icon from the World Editor toolbar.



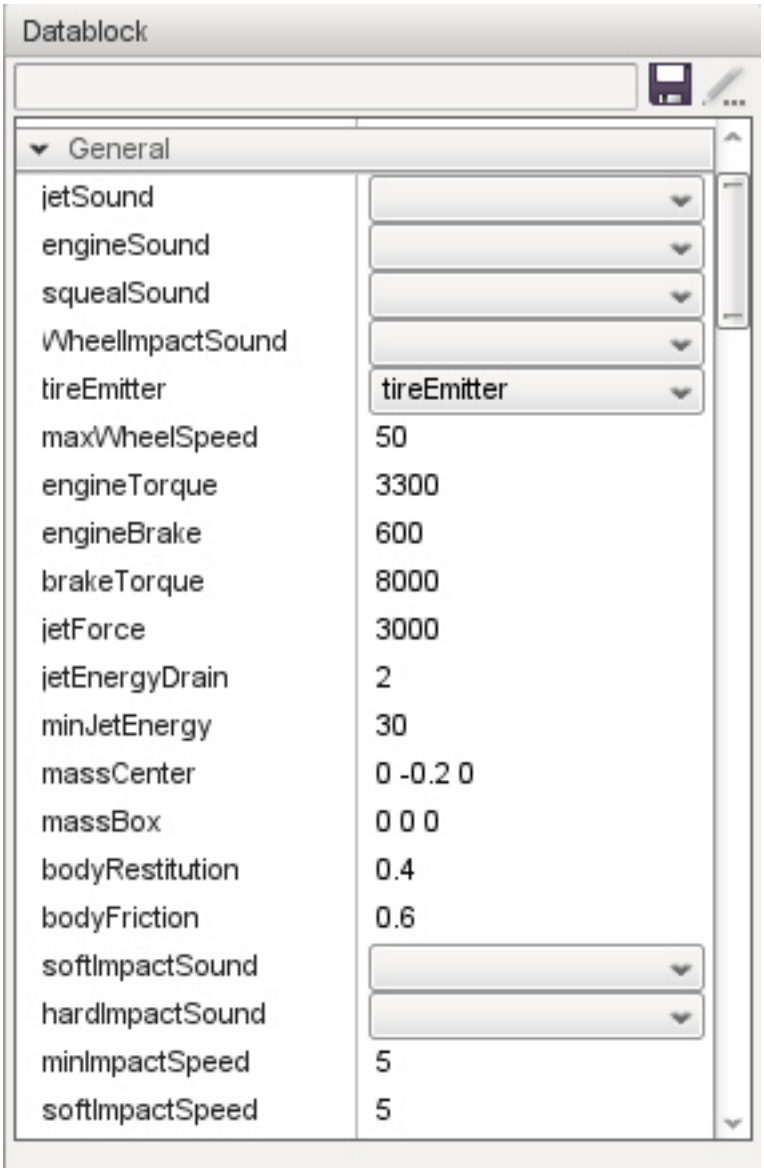
The Datablock editor has two components: the Datablock Library pane and the Datablock properties pane. These panes appear at the right of the screen whenever the Datablock Editor is active. The Datablock Library pane is further divided into two tabs. The first, labelled Existing, contains a categorized list of all the existing datablocks. The second, labelled New, is used to create new instances of those datablocks.





Clicking any existing datablock will cause the Datablock properties pane to update to display the current properties of that datablock.

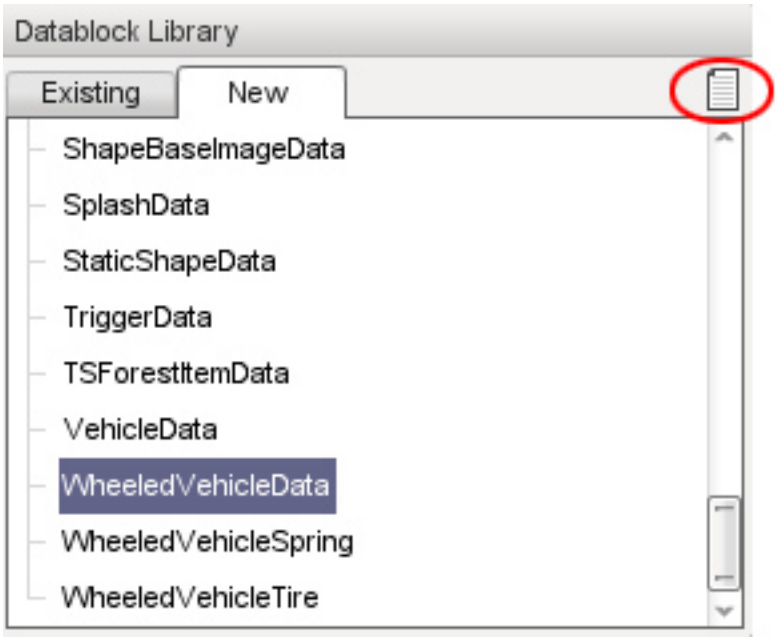
The image below shows the selection of the DefaultCar datablock, under the WheeledVehicleData category. This datablock contains variables related to vehicle performance.



Creating a new Datablock

Creating a new datablock can be done by creating a copy from an already existing datablock. To do so first select the New tab in the Datablock Library pane.

Then choose the type of datablock you wish to create from the list. Then press the New icon.



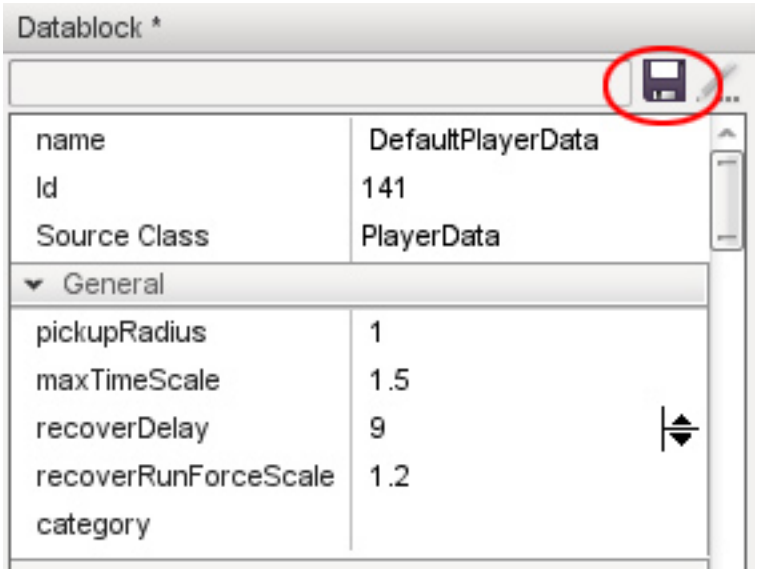
You will be presented with a new window giving you the option to name the new datablock and to copy values from one of the existing instances of the datablock type, if you want to. For example, in this scenario the DefaultCar datablock would be available in the dropdown box because it already exists at the time when creating a new datablock.

After clicking the Create button a new copy of the datablock will be added to the library, under the datablock type you first selected. In this example, you will create a new WheeledVehicleData datablock and name the new version "raceCar". This new version can now be found in the Library, under the Existing tab, in the WheeledVehicleData section.

Saving a Datablock

After editing the new datablock or any other datablock, you will need to save it. You will see a small "*" in the header of the properties right after the Datablock label if the datablock needs saving.

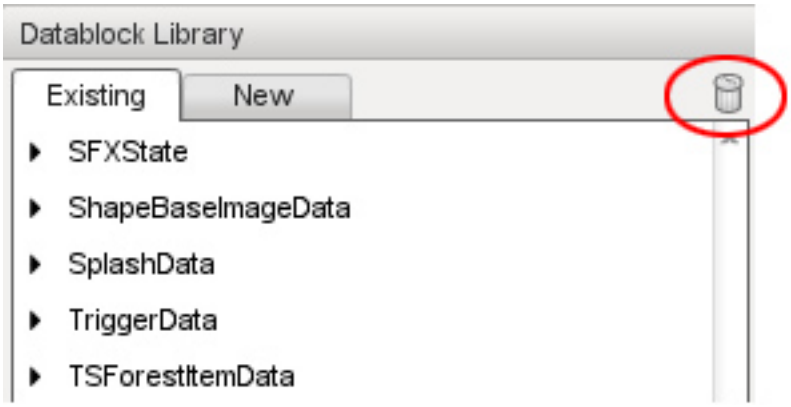
Click the small floppy disk icon to save your datablock changes.



Note: Any new datablock which has been saved will be added into the managedDatablocks.cs document which can be found at the location: projectgameartdatablocksfor your scripters to access later.

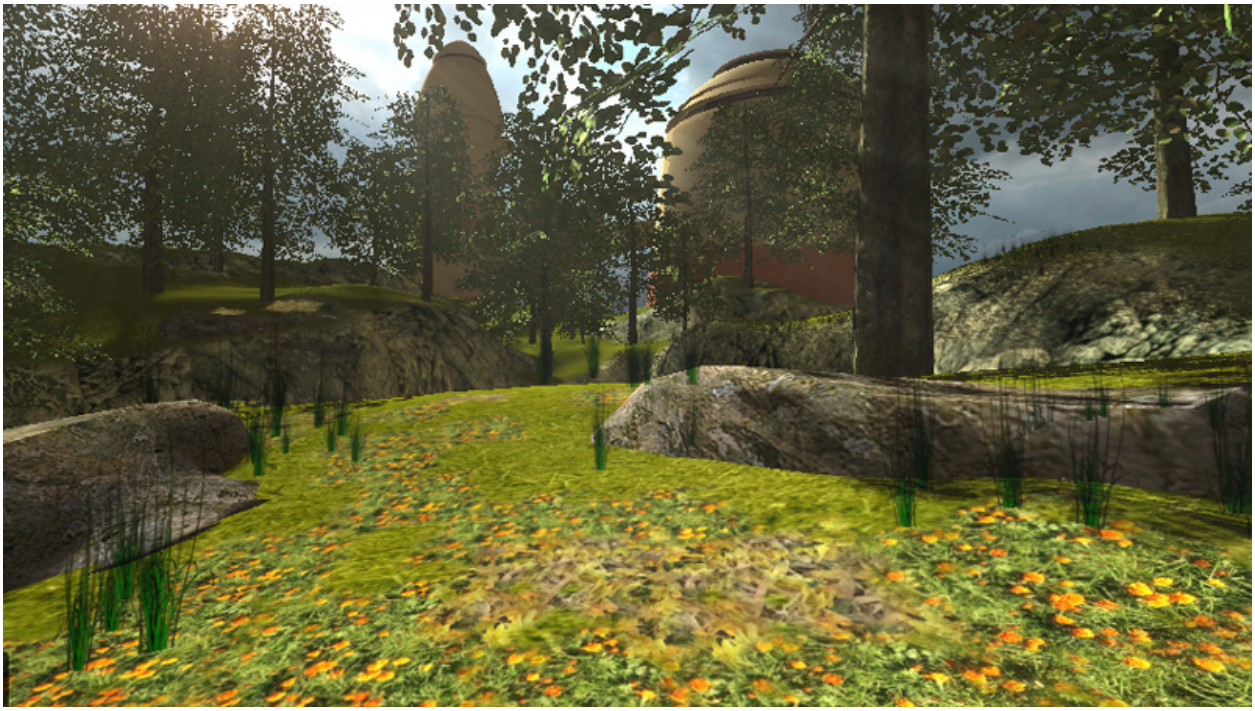
Deleting a Datablock

If you no longer need a datablock you can easily delete it by selecting the Delete icon.



After pressing this icon you will get a notification window stating that the datablock has been removed. The World Builder will need to be restarted to completely remove the file.

2.4.6 Decal Editor



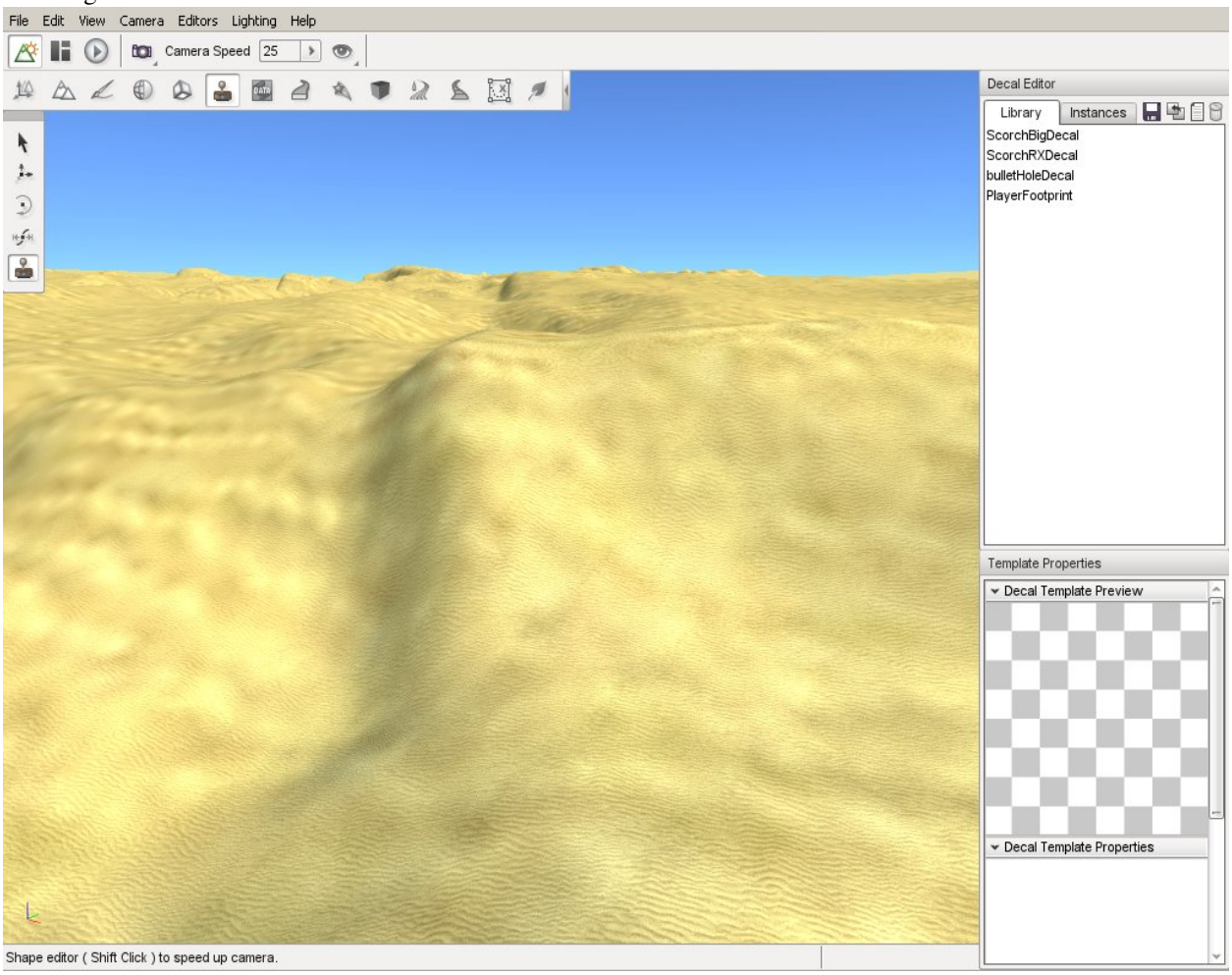
Decals in Torque 3D refer to image textures that are overlaid on objects such as the terrain or players to give the appearance of surface affects such as leaves, flowers, or litter on the ground, or for dynamic changes to the environment, such as foot prints, or burn marks from explosions without the need to create and place special terrain materials or

objects to represent the effects. Torque 3D's Decal Editor provides you with control over decals of any type, including placement and other attributes. Decals can be placed via the editor to any visible surface within the world.

As with the other Torque 3D editors this can be easily achieved by using the built-in WYSIWYG (What-You-See-Is-What-You-Get) editing tools.

Interface

To access the Decal Editor press the `F7` key or select it from the drop down menu at the top of the World Editor, by choosing Editors > Decal Editor. Or select the Decal icon from the World Editor tool bar.



The Decal editor has two main parts, one where you can add and manipulate the size, rotation and position of the decal, and the other for adjusting the decal properties. On the upper-left hand side of your screen, you'll see a toolbar which provides tools for decal placement and manipulation. On the right of the screen, there is the Decal Editor pane which displays a list of decals, and the Template Properties pane below it, which displays properties of the selected decal along with a texture preview.

In the Decal Editor pane on the top right, under the Library tab, is a list of the decal datablocks defined in the system, which are really decal descriptions. In the Instances tab, there is a list of all decals that have been created within the current project.

Adding a New Decal Datablock to the Library

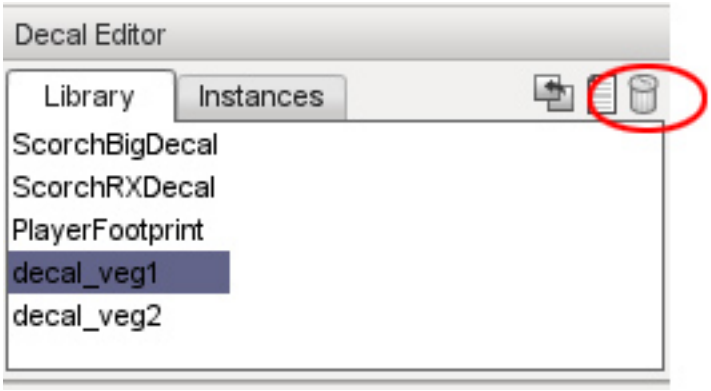
Before you can paint a decal to your world, the datablock needs to exist in the decal library. To add a new decal, simply press the New Decal icon at the top of the Decal Editor pane. This will add a new blank decal to the library, ready for you to select or add a material and set up its properties.

Naming a New Decal Datablock

After creating a new decal you will want to name it. This can be achieved by selecting the name property and entering a new name, then pressing the Enter key.

Removing Decal Datablock from the Library

To remove an existing decal from the library simply select the decal in the list then click the Delete icon.

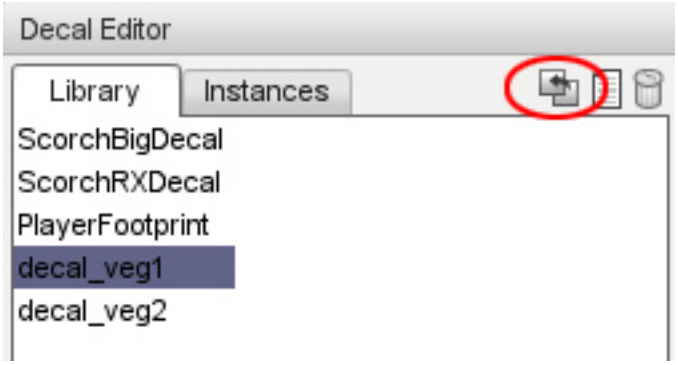


If you select Yes all instances of the selected decal will be removed. The datablock will continue to exist until the World Editor is restarted.

Missing Decals

If a decal exists in the level, but is not rendering, it means the datablock for it has either been deleted, renamed, or corrupted. The Retarget button allows you to assign an existing datablock to a missing decal.

To open the retarget dialogue, select the decal that you wish to fix then click the Retarget icon at the top of the Decal Library pane.



The Retarget Decal Instances dialog box will open. From here you can reassign the decal datablock. Select a Decal Datablock from the drop down list that you wish to use for the selected decal.

Editing Tools

The Decal Editor placement and manipulation tools appear on the toolbar down the left of your screen whenever the Decal Editor is active. Each tool can be activated by clicking the appropriate icons or by pressing its hotkey. Hotkeys are assigned 1 through 5 from top to bottom in the toolbar and are visible by hovering the mouse over the icon. These tools will enable you to quickly place, scale and rotate your decals.

Adding a Decal

Before adding a decal, a datablock for the decal must exist in the Decal Library. Please see the section "Adding a New Decal Datablock to the Library" above.

To add a decal to a level select a decal from the Library tab of the Decal editor pane, click the Add Decal tool, and then click on the terrain where you would like to see your decal instance appear. The same decal can be placed in a level multiple times. Each such copy is referred to as an instance of the decal.

Selecting a Decal

The selection tool enables you to directly select a decal instance that has already been placed in the level by simply clicking on it. Its datablock properties will then be shown in the Template Properties pane for viewing and/or editing.

Deleting a Decal Instance

There will be times when you need to delete an a decal instance. The decal can be selected by the Selection tool (see the "Selecting a Decal" section) and then pressing the Delete key. Or you can select the required decal from the Instances tab in the Decal Editor pane, then press the Delete key or press the Delete button, represented by the trash can icon.

Note: This will only delete the selected decal instance in the world, not the decalâ€™s datablock listed on the Library tab.

Moving a Decal

To move a decal instance simply select the decal using any method described above then click the Move tool icon. The normal Object Editor movement gizmo will appear. Click any axis arrow using the left mouse button, then hold the button down and drag the mouse to move the decal in that direction. Release the mouse button to drop the decal at the new location.

Scaling a Decal

If you find that the decal is either too small or too large you can use the Scale tool to resize the decal. This uses the standard world gizmo, but will not scale on the vertical axis due to decals being restricted to two-dimensions. Click any axis cube using the left mouse button, then hold the button down and drag the mouse to scale the decal in that direction. Release the mouse button to leave the decal at the new scale.

Rotating a Decal

If for any reason you find that you need to rotate a decal, you can use the Rotate Tool do so. To rotate a decal, select the decal by any method described above, and then click the Rotate Tool icon. The standard world rotational gizmo will appear. Click any rotation circle using the left mouse button, then hold the button down and drag the mouse to rotate the decal in that direction. Release the mouse button to leave the decal at the new location.

Note: Because decals are two-dimensional rotating a decal will never cause the decal to leave the surface upon which it has been placed. Rather, when a decal is rotated by any axis the decal will rotate in two-dimensions locked to that surface. This can cause some strange effects if the surface that contains a decal is curved with a radius less than the size of the decal. As with all the other T3D editors, the more that you experiment and use the tools the more familiar you will be with them. With practice and time you will find many uses for the Torque 3D decal system.

Properties

A Decal has only a small amount of properties which can be edited using the Template Properties pane:

Size The size of the decal rendered onto the surface.

Material Specifies the Material selected to display as the decal.

Lifespan Time in Ms (milliseconds) that the decal will exist in the world after being placed dynamically.

FadeTime Time for the decal to fade out in Ms (milliseconds).

Frame Index of texture rectangle to use for this decal, if the texture consists of multiple images.

Randomize Randomizes the texture rectangle (frame) used for each instance of the decal. So it essentially uses a random frame.

TexRows Defines the number of image rows in a multiple image material.

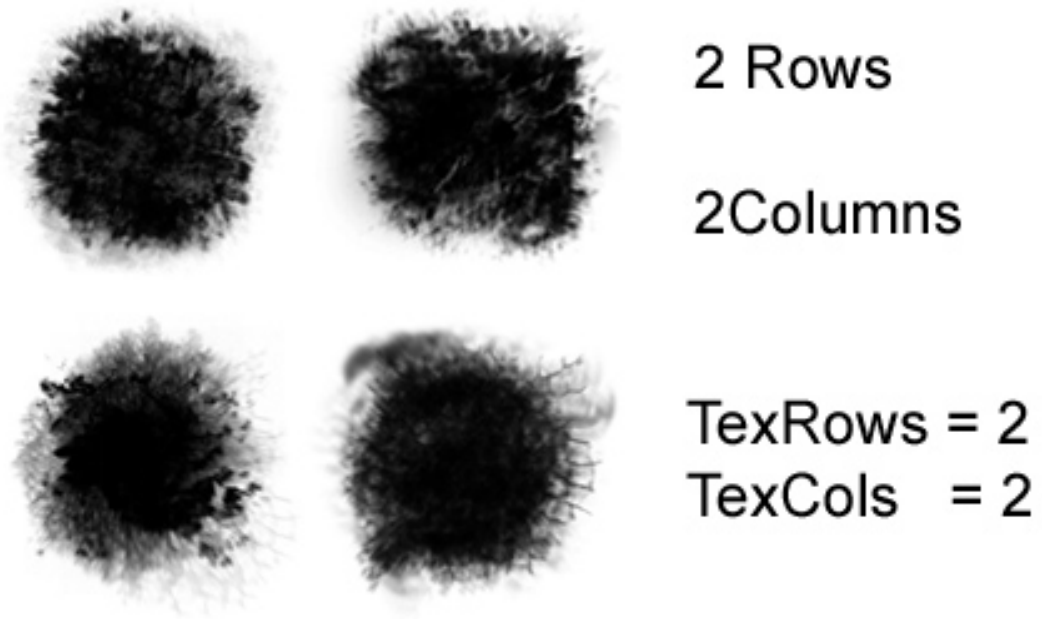
TexCols Defines the number of image columns in a multiple image material.

ScreenStartRadius Distance check for rendering the alpha channel of the decal.Visibility check based on the scale of the decal

ScreenEndRadius Distance check for rendering the alpha channel of the decal.Visibility check based on the scale of the decal

Render Priority If more than one decal are on top of each other the decal with the higher priority will rendered first

Clipping Angle The angle in degrees used to display geometry that faces away from the decal projection direction.



2.4.7 Forest Editor

The Forest Editor is a tool that allows you to quickly create massive amounts of vegetation for your level including patches of trees, forests, and fields of smaller elements such as shrubs and plants. Entire forests can be laid down using simple techniques similar to painting on a canvas, where instead of paint your brushes, lay down 3D models on the terrain.

Interface

To access the Forest Editor press the `F8` key, or select it from the drop down menu at the top of the World Editor, by choosing Editors > Forest Editor, or click on the leaf icon to get started.

The Tools Palette on the left of the screen will populate with Forest Editor specific tool buttons represented by icons.

Select Item Select an individual object in forest

Translate Item Move the currently selected object

Rotate Item Rotate the currently selected object

Scale Item Grow or shrink the currently selected object

Paint Used for painting objects on terrain

Erase Used for erasing objects from a terrain

Erase Selected Used to delete the currently selected objects

The Forest Editor has two main panels which will appear on the right of the screen whenever the Forest Editor is active.. On the top is the Forest Editor pane which is divided into two tabs: Brushes and Meshes. The Forest Editor works in a manner similar to painting on a canvas with a brush, except instead of paint the Forest Editor lays down shapes onto the terrain of your level. A Brush in the Torque 3D Forest Editor is composed of one or more mesh elements, which can be alternated between when painting.

The Meshes tab contains a list of all Forest Mesh elements which can be assigned to a brush. A forest mesh is really a datablock which is an information structure that defines a model and the properties which control it in the forest.

On the bottom of the right side of the screen is the Properties pane. The Properties pane displays information about the currently selected element in active tab of the Forest Editor pane.

Before we can use these tools and paint a forest, we need to import a forest mesh and set up a brush.

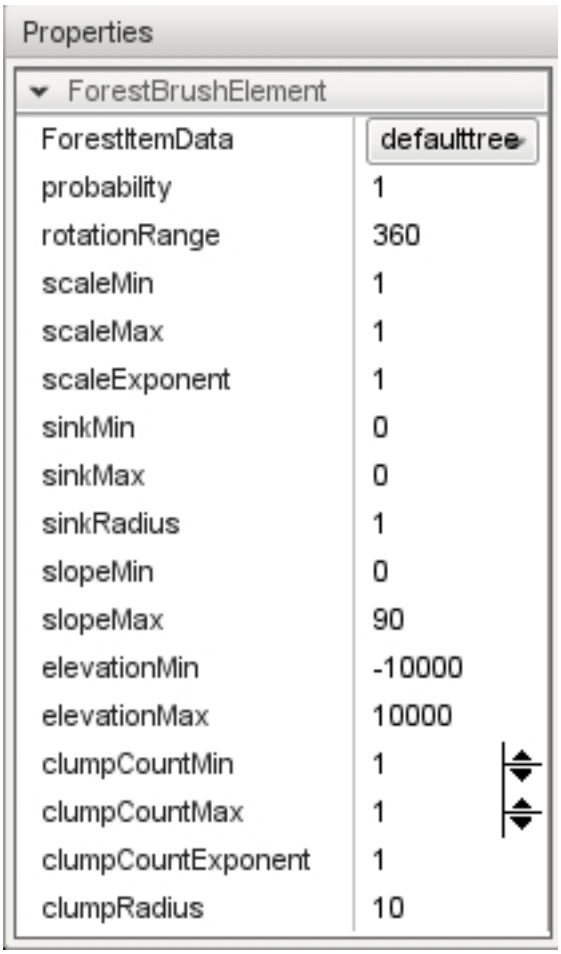
Creating a Forest Mesh

To create a forest mesh tart by clicking on the Meshes tab in the Forest Editor pane. There are two icons in the top right. The trash bin deletes the currently selected existing mesh, and the leaf icon will adds a new mesh. Click on the Add New Mesh icon.

A file browser should appear. Locate the sample tree mesh file, defaulttree.DAE, which can be located in the game/art/shapes/trees/defaulttree folder.

A new mesh will be added to the tab using the same name as the file you selected:

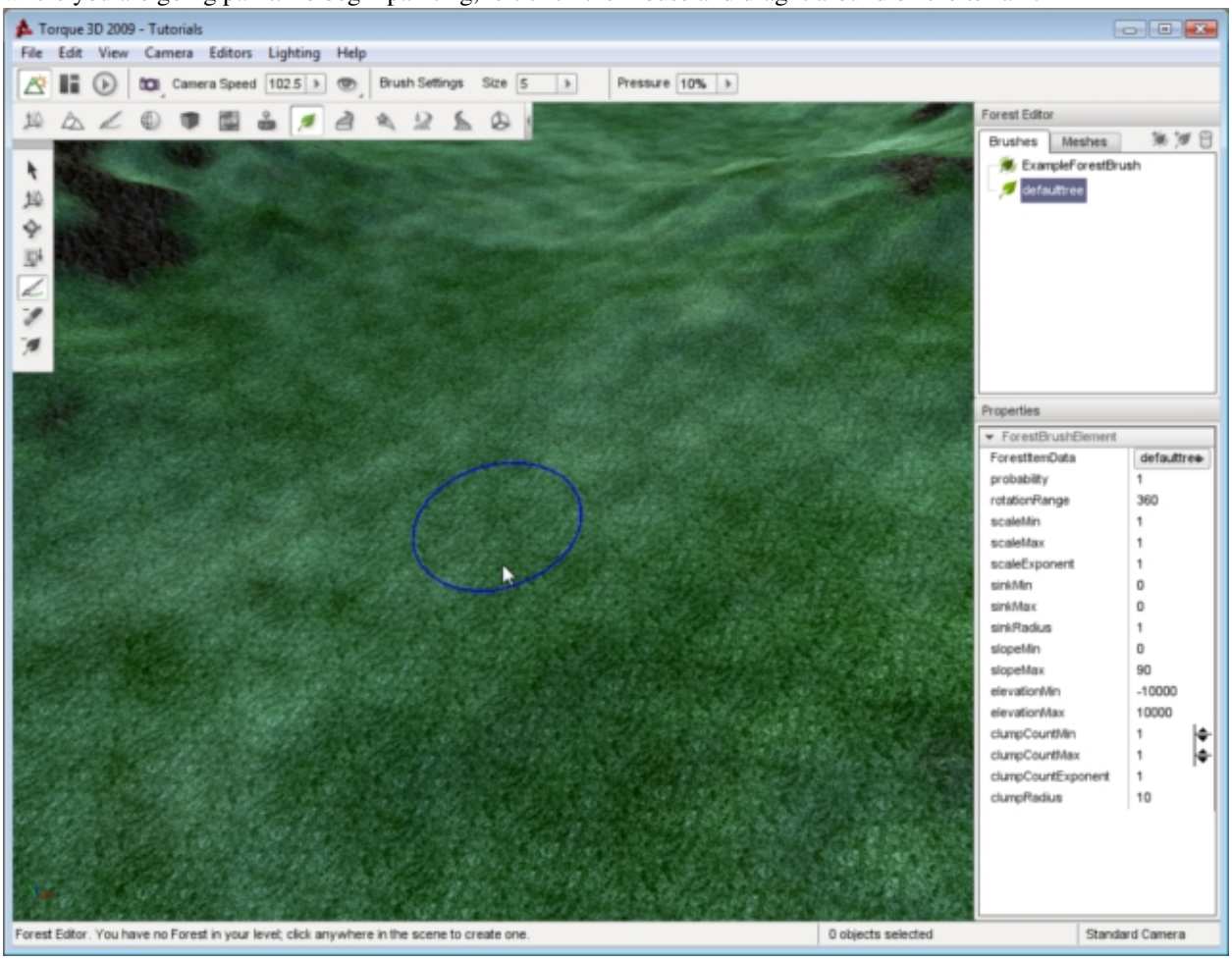
The Properties pane will also be populated with fields and values which describe the new mesh.



Switch to the Brushed Tab. You will see that the new mesh has also automatically been added to the list of Brushesallow you to select it as the element to paint with.Select the new brush by clicking on its name. The Properties pane will be updated to display the properties of the brush which can be used to randomize the placement and appearance of the selected mesh.

Using a Brush

Now that you have an available brush you can begin painting a forest. Select the defaulttree brush from the sample assets. Move the mouse until a blue circle appears on the terrain. This is the outline of your forest brush and shows where you are going paint. To begin painting, left click the mouse and drag it around on the terrain.



If this is the first time you have painted a forest in a level then no Forest object exists in the level yet. However, a forest object must exist , so you will be prompted to confirm that you want to add one.

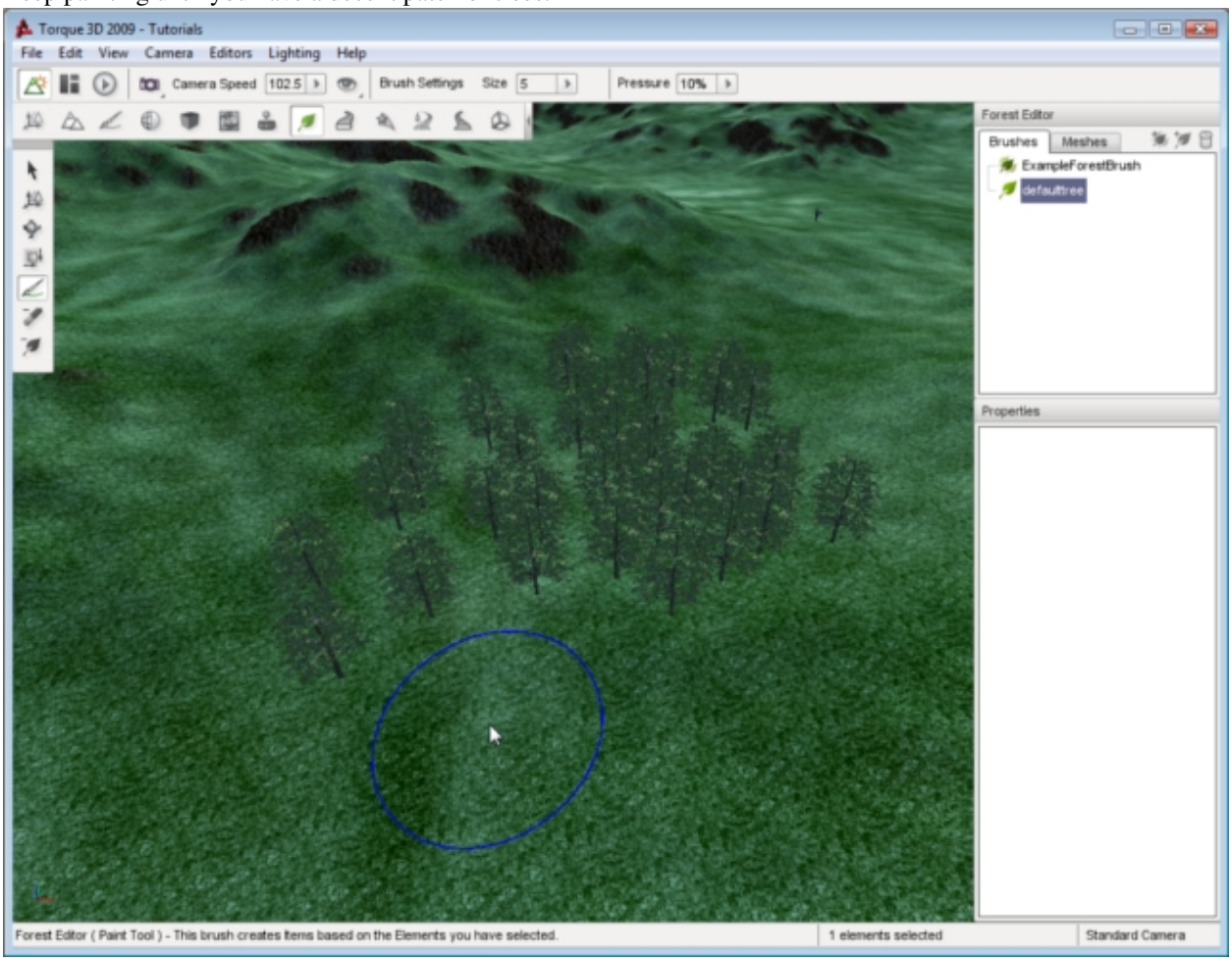
Answering No will abort the forest painting operation. Answering Yes will automatically create a new Forest object, add it to your level, and return you to the level with the brush still active ready to continue painting. You can examine the forest object the same way as any other object using the Object Editor but it has no useful properties to edit so lets skip it for now an continue on with our forest painting operation.

Once again, hold down the left mouse button and drag the mouse over terrain. As you move the brush trees will begin showing up in the wake of your brush. To change the size of your brush, pull the mouse wheel toward you to increase the size or push it away to decrease the size. The blue circle will grow or shrink to indicate your new size.

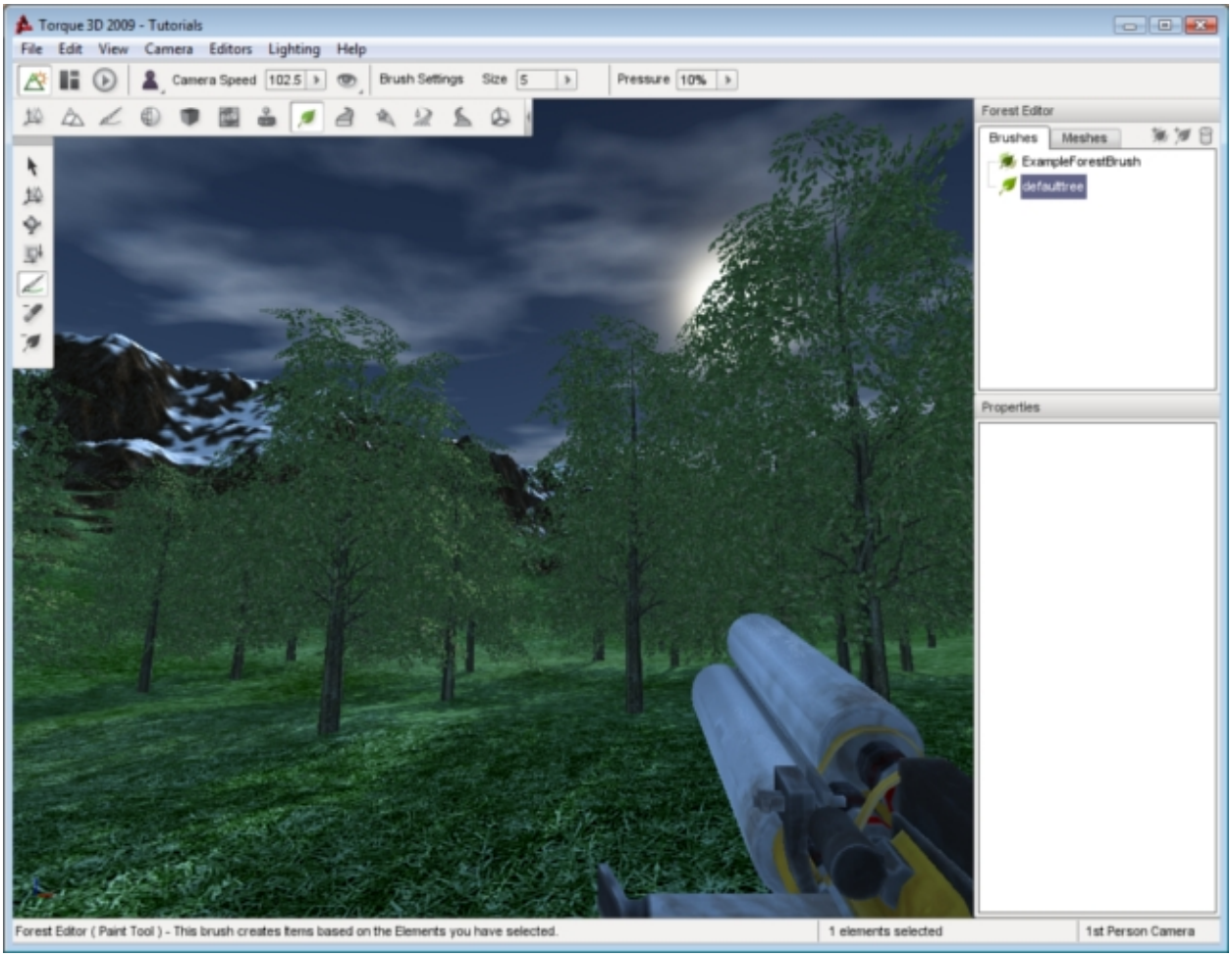
Note that you do not have the ability to move the camera forward and back in the Forest Editor because of the availability of the brush resizing feature. To move the camera forward and back while using the Forest Editor press

the Up Arrow to move forward and the Down arrow to move backward.

Keep painting until you have a decent patch of trees:



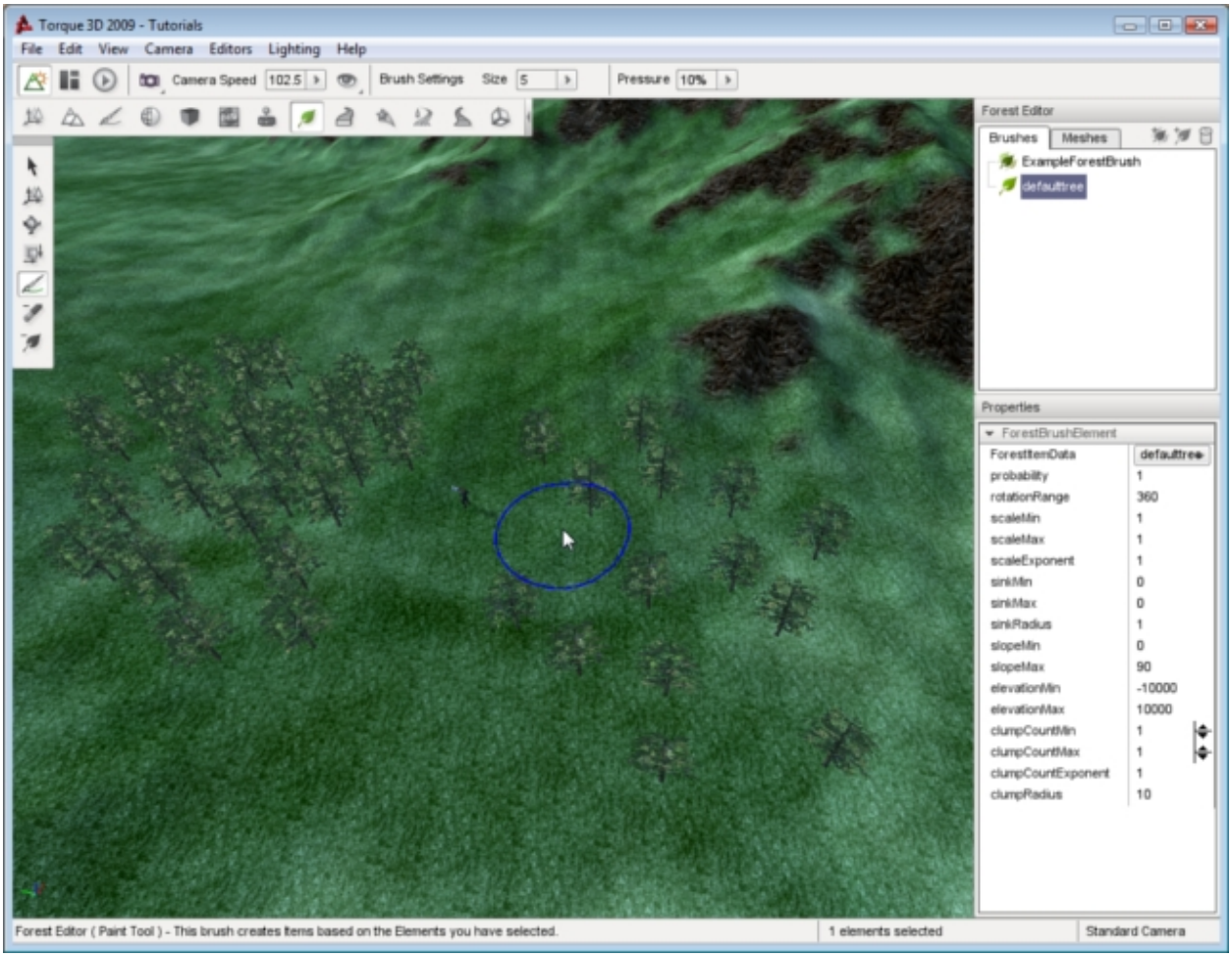
If you move your camera down to ground level, you can see how your forest will look from a player perspective. Youll notice that these are full 3D objects that react to collision, sunlight, and external forces.



Adjusting Properties

You can edit the properties of a Mesh to adjust how each tree is placed when painting. To adjust the density of mesh placement switch to the Meshes tab then select your defaulttree entry. The Properties pane will update to display the properties of your mesh. Change the radius property from 1 to 2 then press the Enter key.

This radius tells the tool a rough amount of space this item takes up. The value is a decimal value and has no limits, but remember that if the value is too low your trees may overlap, and if it is too high you may not get any trees to appear because the spacing might be larger than the brush itself. Now when you paint you should get more spacing between the placed meshes.

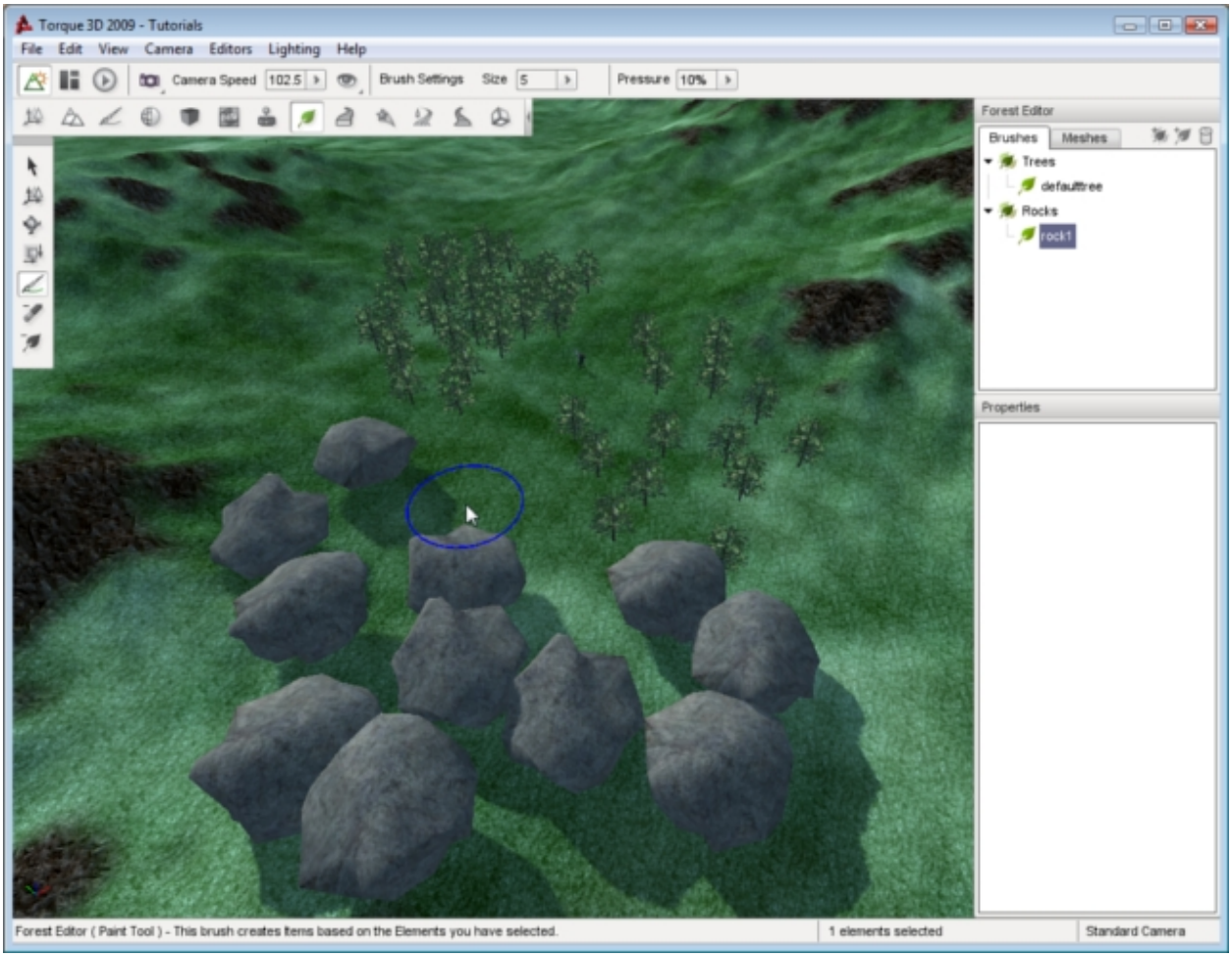


As mentioned previously, you can use the Forest Editor to paint additional environmental objects such as rocks, shrubs, or any other 3D model. Since you can paint different types of objects, you might want to organize your brushes and meshes.

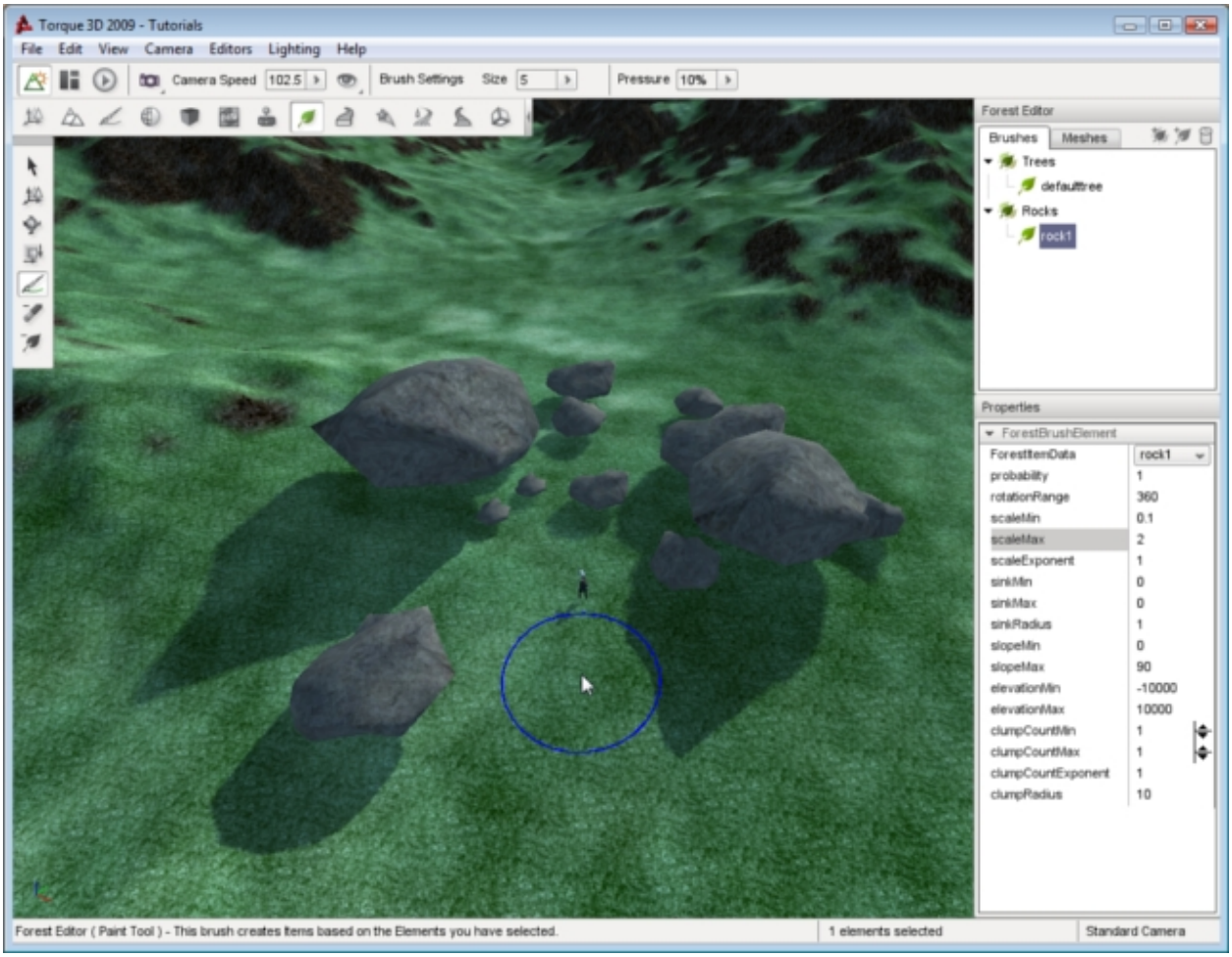
In the Brushes tab, click on the Add New Brush Group icon. This will add a new entry in the brush list, called “Brush”. Click on the text of the new brush group. This will allow you to edit the text of the brush. Name the brush group “Trees” then press the enter key. Now, you can click on the defaulttree element and drag it onto the Trees brush group. Switch to the Meshes tab, and click the Add New Mesh tree icon to add a new one. Select game/art/shapes/rocks/rock1.dts as your model.

The rock1 mesh will be added to your Meshes list. Unlike trees, the rock1 mesh is fairly large and somewhat spherical. Spreading out the placement of this mesh will help prevent dense blobs of rocks being placed. In the Mesh properties tab, increase the rock1 radius to 3.

Switch back to the Brushes tab. Create a new brush group and name it Rocks. Your rock1 mesh element should already be in the list, so drag it onto the Rocks brush group to keep things organized. Go ahead and paint down some rocks in your level. You should end up with a patch of huge boulders with fairly even spacing:

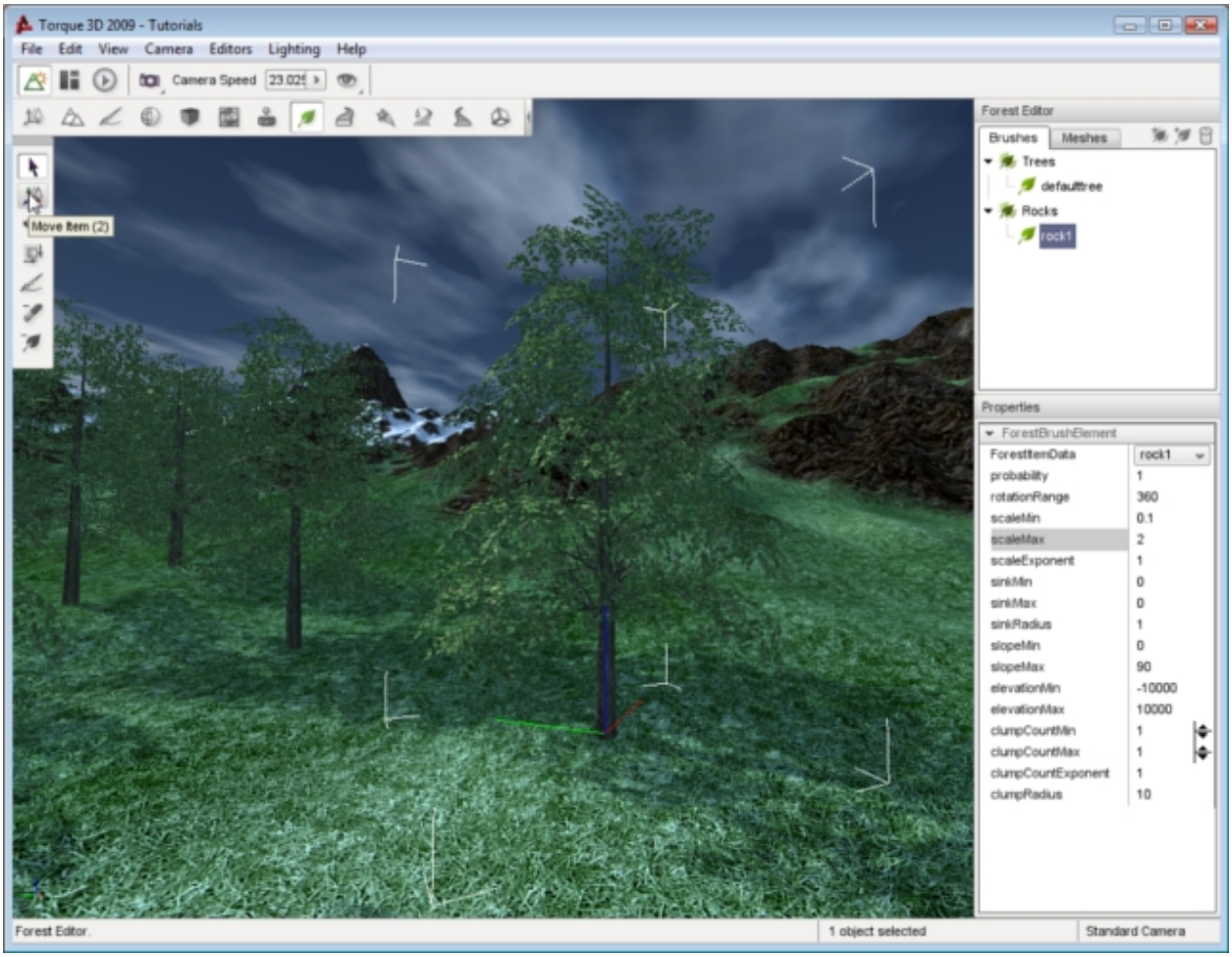


You might have noticed all the boulders are the same size. For added realism, you can adjust the brush properties to randomize its appearance. Select rock1, then decrease the scaleMin and increase the scaleMax. Begin painting a new set of rocks. Now, you will end up with rocks of varying sizes. Some will be as small as your player, while others could be twice the size of the original mesh.

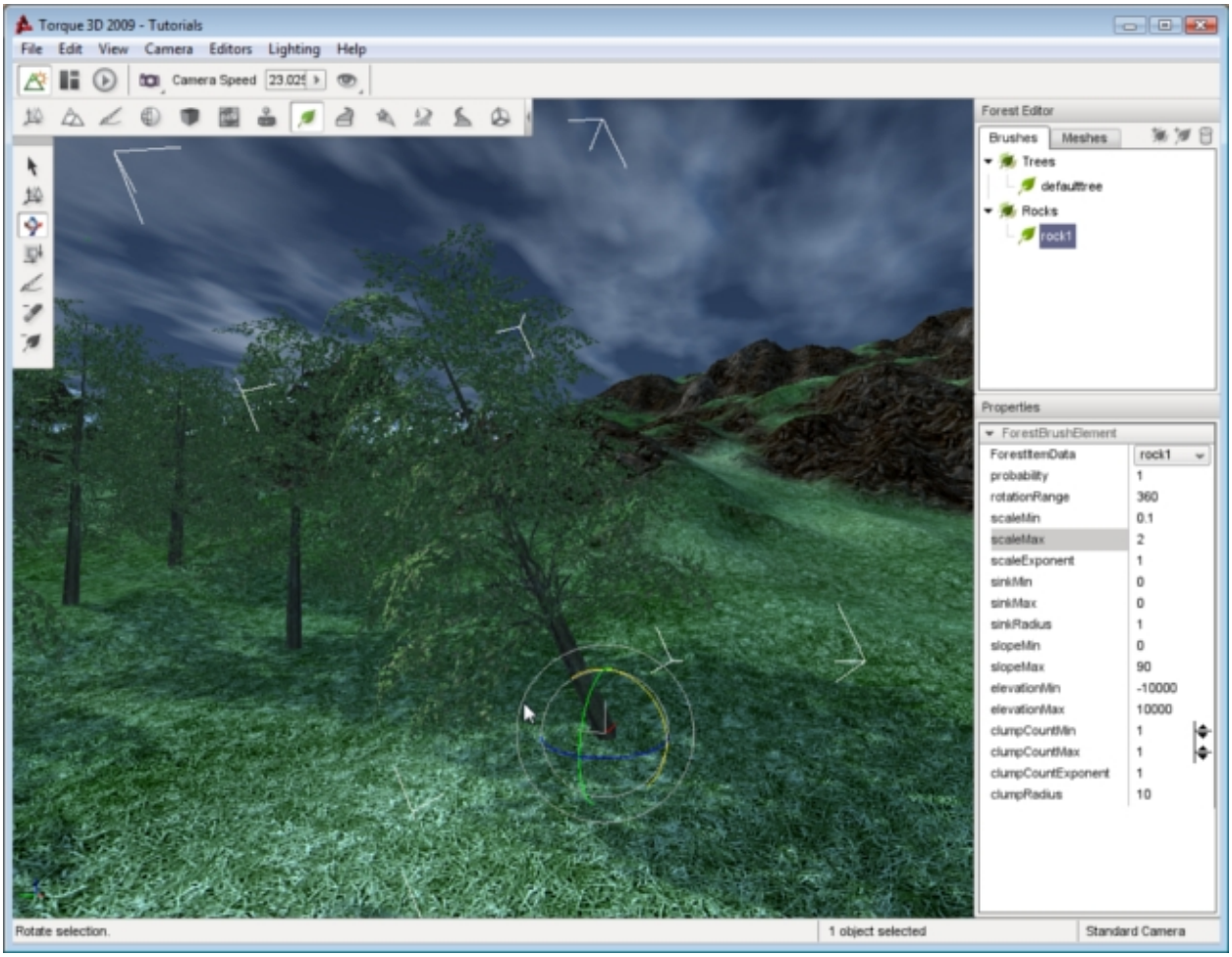


Editor Settings

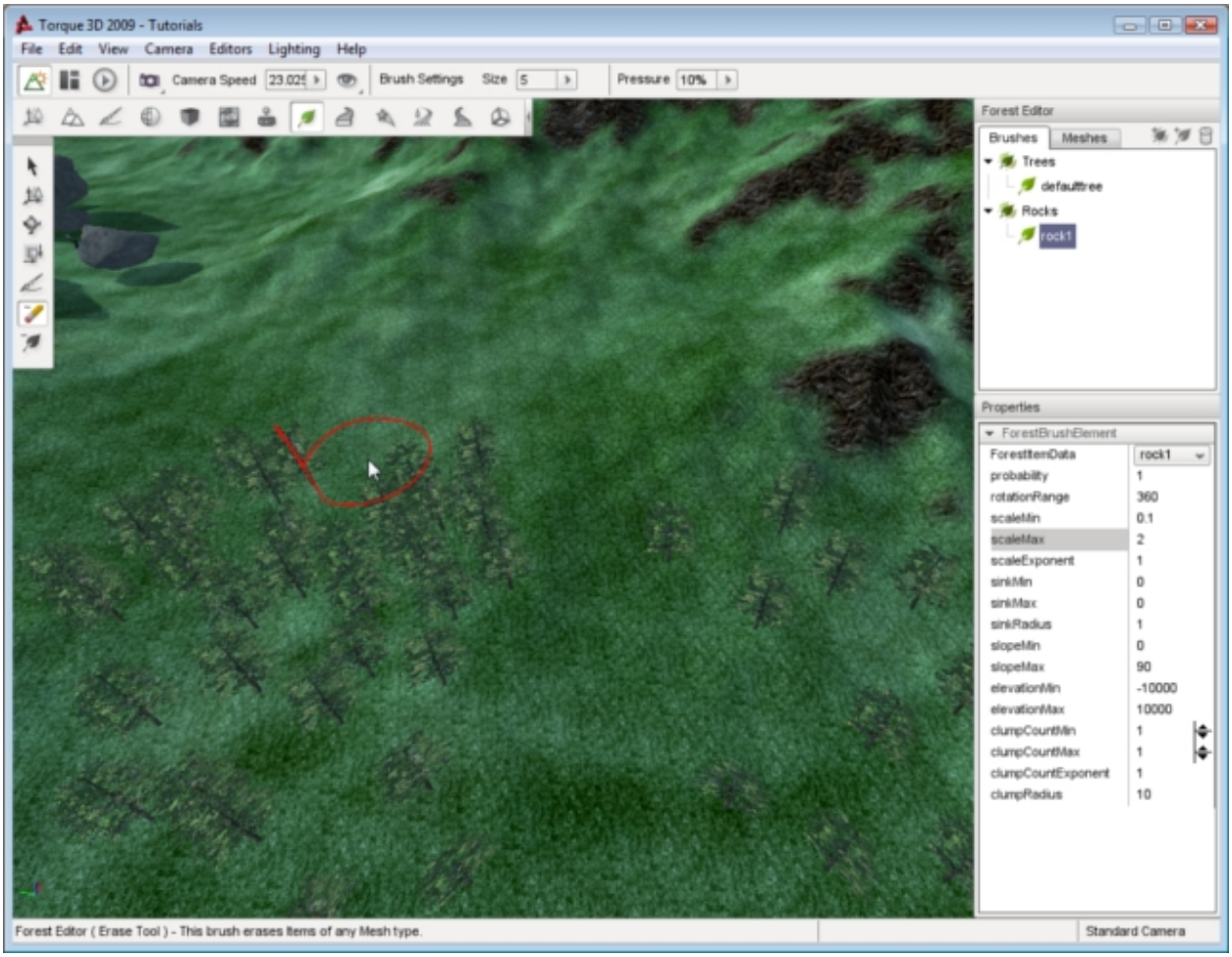
The actions available in the tools palette give you absolute control of your forest placement. The first four tools allow you to adjust individual elements of your forest, such as a single tree. The Select Item tool allows you to select an individual element, which is indicated by a colored axis gizmo appearing on top of the item:



Once you have a tree selected, you can change its location without moving the entire forest. With the tree selected, activate the Move Item tool. The arrows gizmo will appear, allowing you to drag the tree around in the world. The Rotate tool, represented by a spherical gizmo, allows you to adjust the orientation of the tree in 3D space. You can use this to make individual trees lean in a specific direction. The Scale tool can be used to shrink or grow an individual tree. When you need to tidy up a forest, such as removing rogue trees, pressing the delete key when you have a tree selected will remove it from the scene.

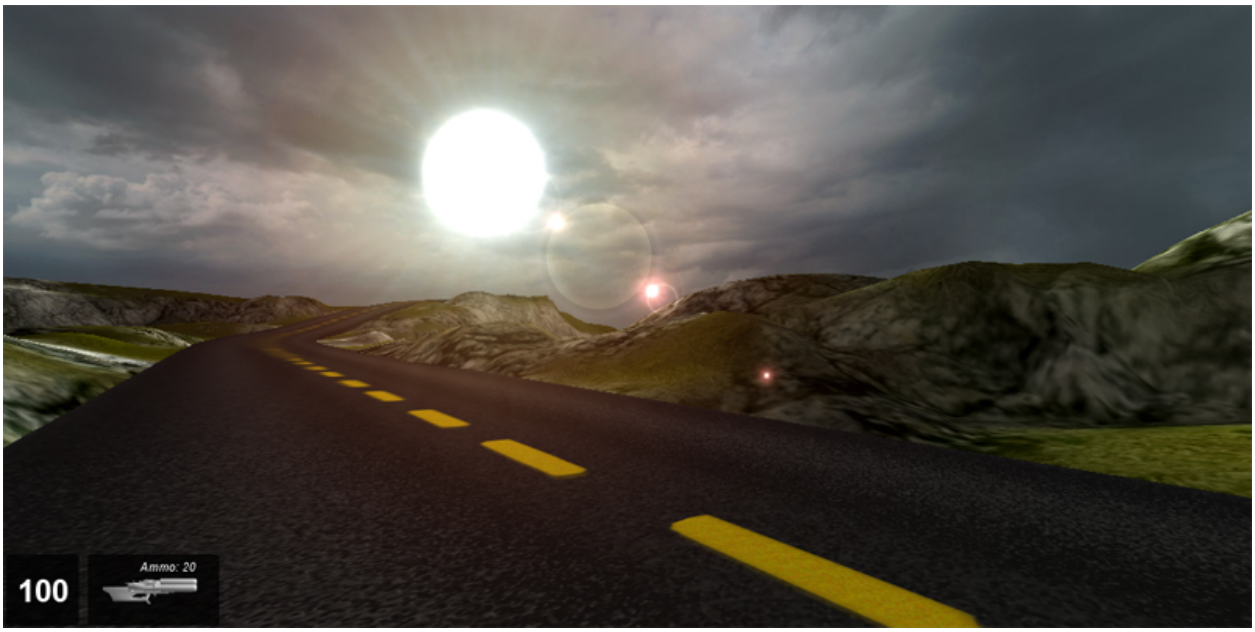


If you need to delete entire sections of a forest, you may not want to delete each tree individually. Instead, you should use the Erase tool. The Erase tool is located directly below the Paint tool. When activated, the circle representing your brush in the world will turn from blue to red when you move your brush over the terrain:



Left click your mouse and drag the brush over a section of trees. Any trees under your brush will be removed from the forest object. This is much faster than deleting individual trees. If you want to remove a larger amount of trees such as clearing an area for a road, you can set the width of the brush to a specific width. Locate the Size dropdown on the Tool Settings bar and click on it. A slider will appear so you can increase the circumference of your brush. Set it to something fairly large, like 20.

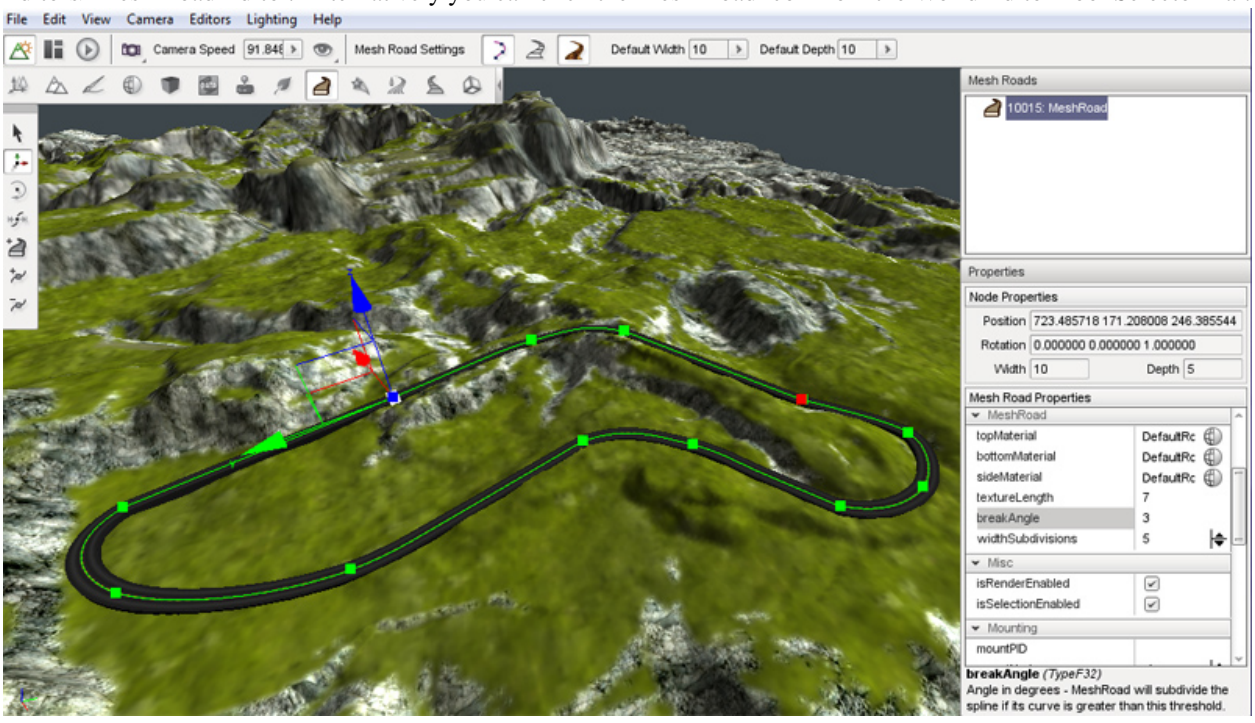
2.4.8 Mesh Road Editor



The Mesh Road editor will help you create a solid mesh road structure upon your terrain. A mesh road is actually a 3D model representation of your road and is not solely dependent upon the terrain height. A mesh road can be raised above the terrain, or suspended between hills unlike a decal road, which painted on the surface of your terrain as a texture, and must follow the terrain exactly. Decal roads are created with the Decal Road Editor.

Interface

To access the Mesh Road Editor press the `F9` key or activate it from the main menu of the World Editor by selecting Editors>Mesh Road Editor. Alternatively you can click the Mesh Road icon from the World Editor Tool Selector Bar.



Whenever the Mesh Road Editor is active three sections of the screen are updated to contain the editors tool. On the right side of the screen are the Mesh Roads pane and the Properties pane. At the top is the Mesh Roads pane which contains a list of all the road meshes currently in the level, if any are present. At the bottom is the Properties Pane which displays the properties of the currently selected road mesh.

At the left of the screen the Mesh Road placement tools will appear and are used to create and modify road meshes. At the top of the screen in the World Editor Tool Settings bar, a new set of icons will appear. These icons and their associated values will enable you to quickly set up the width and depth of the control points and modify the editor to show and hide some visual aids which can be used to guide your road placement.

Adding a Mesh Road

A road mesh is created by placing a number of control points across the terrain. Each point can be edited for Road height, Road width and Road depth. By adjusting these points we have full control over how our road will look. The default width and depth of control points can be set using the Default Width and Default Height properties on the Tool Settings Bar at the top of the editor window. Any new road meshes will be created using these settings until you change their values again.

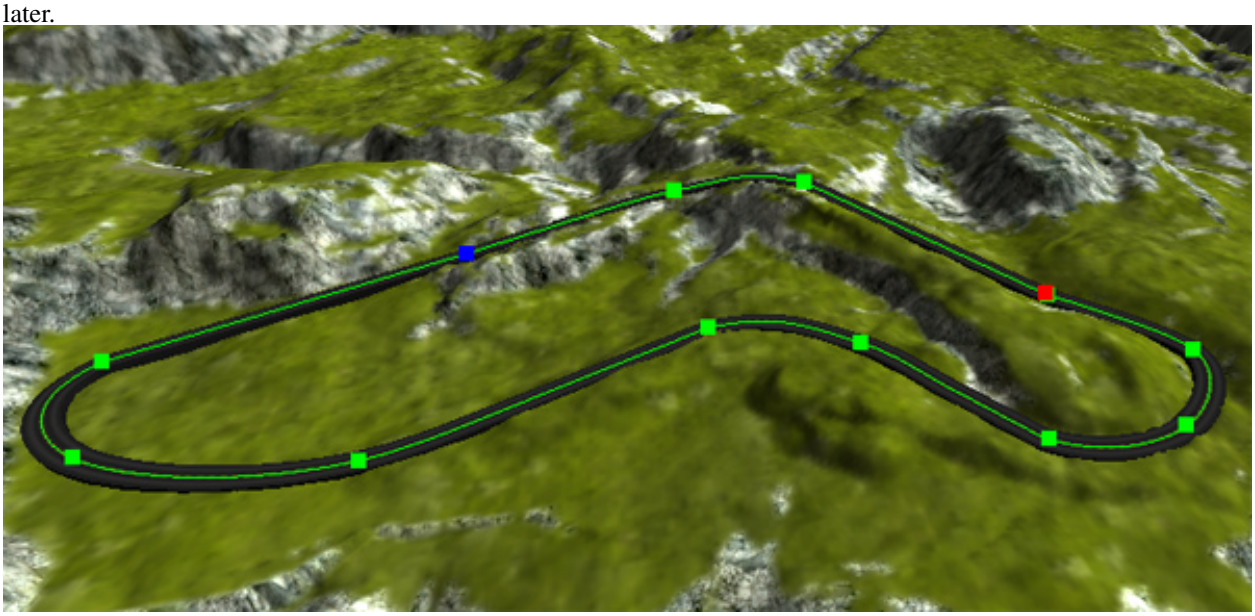
To create a new road mesh select the Create Road icon from the tool bar then click on the terrain with the left mouse button where you would like to start your road. Move the mouse away from the clicked location to see the results. Each time you click the terrain you will see three things:

1. a green square which represents the road location that you just placed
2. a blue square which represents the next location that will be placed the road if you press mouse button again
3. the surface of the road that will be placed the next time you click the button

Move the mouse to the next point on the terrain that you wish your road to travel to and then click again. Continue moving and clicking until you are finished with the initial placement of your road.

To complete the road placement process press the `Esc` key. This action will exit the Create Road tool leaving your new road selected and ready for adjustments.

To abort a road creation operation without placing a road at all press the `Esc` key before selecting a second road point. Once a second road point has been placed the only way to remove the road completely is to delete it, as explained later.



The new road will also show up in the Roads Mesh pane above the road Properties pane.

Editing a Mesh Road

The Road Mesh Editor provides several tools for modifying roads after they have been created. If at any time you make a mistake with any tool, you can press CTRL+Z to undo it.

Selection Tool

Once you have created your initial road you may need to edit some or all of the control points. This tool will allow you to directly select any created point for further editing. To activate the Selection Tool click its icon on the Tool Selector bar. Note that the Road Mesh Editor will automatically select this tool when you have finished creating a new road.

An entire road mesh can be selected by clicking anywhere on a road mesh other than one of its control points. This type of selection will result in the road being highlighted with a "spline", which is a curved line that runs along the center line of the road, and a series of green squares which represent the roads control points. There are no operations that can be performed on a road as a whole within the Mesh Road Editor. Selecting a road allows you to see its centerline and it control points for individual selection and manipulation. To perform operations on the entire road such as moving it to a new location use the Object Editors tools as with any other shape in your level.

Control points can be selected individually to adjust each point as necessary. To select a control point left click on one of the colored squares that represent a roads control points. The selected control point will turn blue.

Selecting a control point also causes the Properties pane on the right of the screen to be updated to display the current property values of the control point. The Node Properties section will display the position, rotation, width and depth of the selected control point. Values can be directly entered into these fields to modify the point or the Move Tool can be used to manipulate the point using the mouse.

Moving a Road

If at any time you are unhappy with the placement of a selected Road Mesh control point you can use the Move Tool to adjust its position. To activate the Move Tool click its icon in the Tool Selector bar. The move gizmo will appear. The move gizmo is used to move the road point to a new location. Left mouse click on any arrowhead then drag the mouse to move the point along that arrows axis. Release the mouse button to relocate the control point to that new location. Left mouse click on the colored square at the origin of the axes then drag the mouse to freely move the point to without regard to any axis.

Scaling a Road

The width and depth of a road can be directly adjusted at a selected control point by using the Scale tool. To activate the Scale Tool click on its icon on the Tool Selector. The scaling gizmo will appear. Left mouse click on the colored cube at the end of any axis then drag the mouse while holding the button down to increase or decrease the size of the road along that axis. Note that if you drag the blue cube to adjust the depth of the road you may not visibly see the adjustment take place because the road depth may be increasing down into the terrain. To adjust the width and depth at the same time left mouse click on the colored cube at the origin of the axes then drag the mouse while holding down the button. Release the mouse button to change the road to that new width and depth.

Rotate a Road

The Rotate Tool can be used to rotate a road at any selected control point. To activate the Rotate Tool click its icon on the Tool Selector. The rotate gizmo will appear. Left click on any colored circle then drag the mouse while holding the button down to rotate the roads surface around that axis at the control point.

Inserting Extra Points

The Insert Point tool can be used to add extra points in a road to create a smoother curve. In order to insert a new point into a road the road must first be selected. See the Selection Tool above for details on how to select a road. To activate the Insert Point tool once a road has been selected click its icon on the Tool Selector bar. To place a new point on the selected road click on the road where you would like the new point to be placed. A new point will be added to the road mesh and will immediately the currently selected point as indicated by the blue square.

Removing Points

The Remove Point tool can be used to delete a control from a road mesh. In order to remove a new point from a road the road must first be selected. See the Selection Tool above for details on how to select a road. To activate the Remove Point tool click its icon on the Tool Selector bar. To remove a control from the selected road point click on the control point. This will remove only the selected point leaving all the others in place. No adjustments will be performed on the other existing control points.

Properties

The Properties pane on the right side of the screen can be used to configure a Mesh Road.

Transform

The transform section contains properties which control the placement, rotation and scale of the Road Mesh as a whole.

Position The transform section contains properties which control the placement, rotation and scale of the Road Mesh as a whole.

Rotation Indicates the rotation of the entire Road Mesh in the level.

Scale Indicates the scale of the entire Road Mesh in the level.

Mesh Road

The Mesh Road section contains properties which determine and control the textures used to display the Road Mesh. To change any of the textures for the Road mesh click the globe icon to its right. Clicking one of these icons you will open up the Material Selection window.

Click on the material you want to use for the road mesh property then click the Select button. The material will be entered in the propertys field and will be used as the material for that portion of the Road Mesh.

Top Material Indicates the Material to use for the top surface of the road mesh.

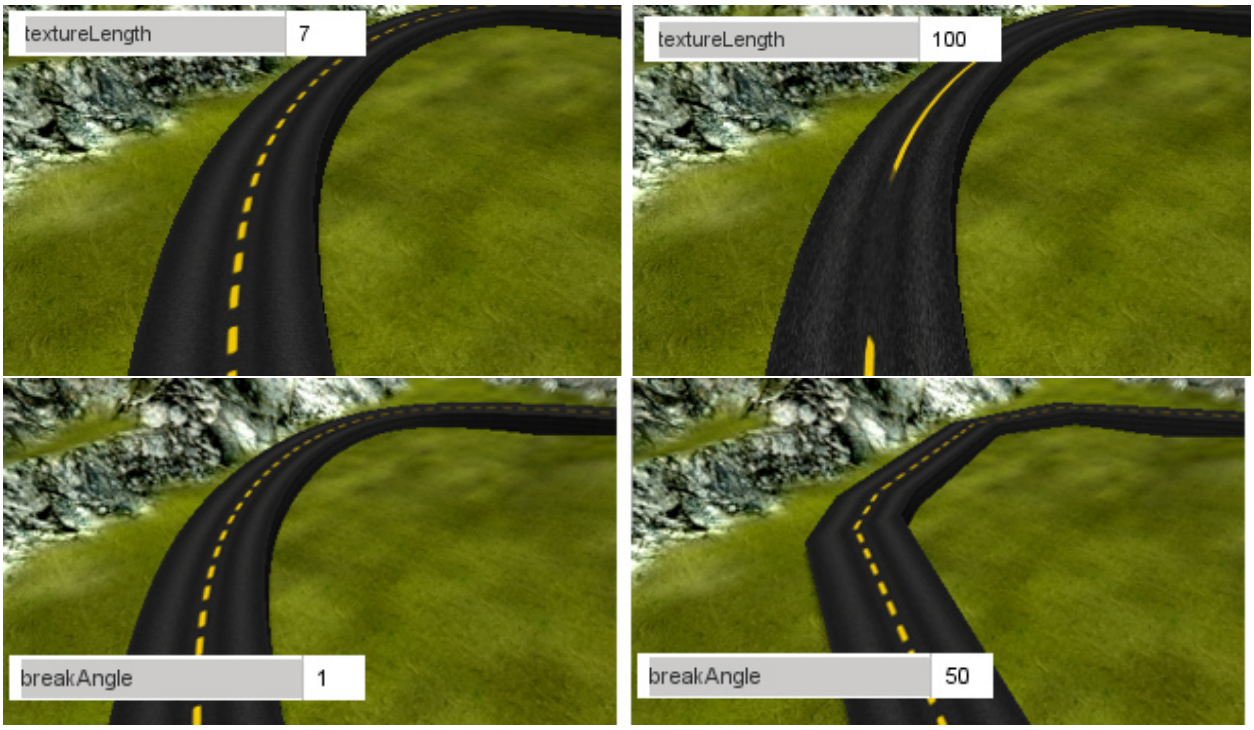
Bottom Material Indicates the Material to use for the underside surface of the road mesh.

Side Material Indicates the Material to use for the sides of the road mesh.

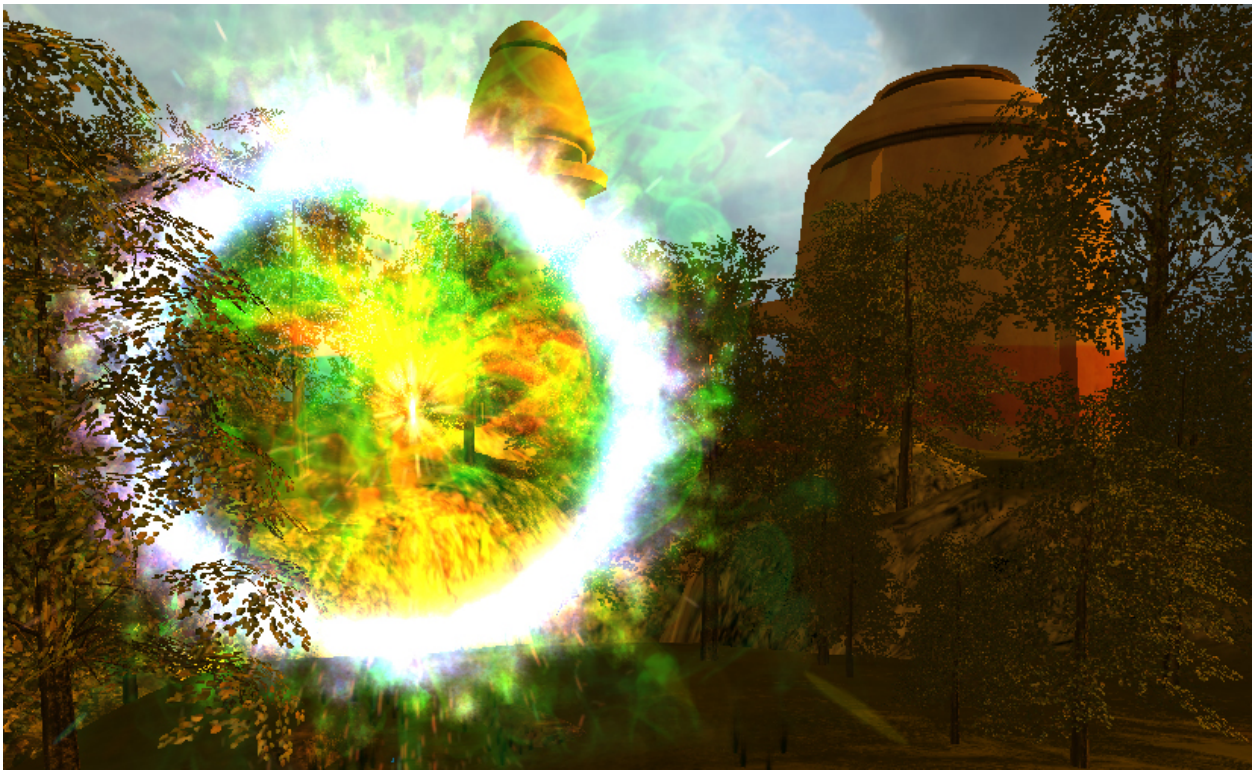
Texture Length Indicates the size in meters of the texture measured along the road center.

Break Angle Indicates the angle in degrees that the mesh roads spline will be subdivided into if its curve becomes greater than this threshold.

Width Subdivisions Subdivide segments width-wise this many times when generating vertices.



2.4.9 Particle Editor



Torque 3D provides a full featured particle system with many parameters which can be manipulated to fine tune your particle effects. Particle effects are things such as fire balls, smoke, and water splashes that you create and place into your levels. The Torque 3D Particle Editor is the tool of choice for full control over the look and feel of your effects.

At its most basic level a particle effect consists of: an emitter, a particle to be emitted from the emitter, and an image rendered to represent that particle.

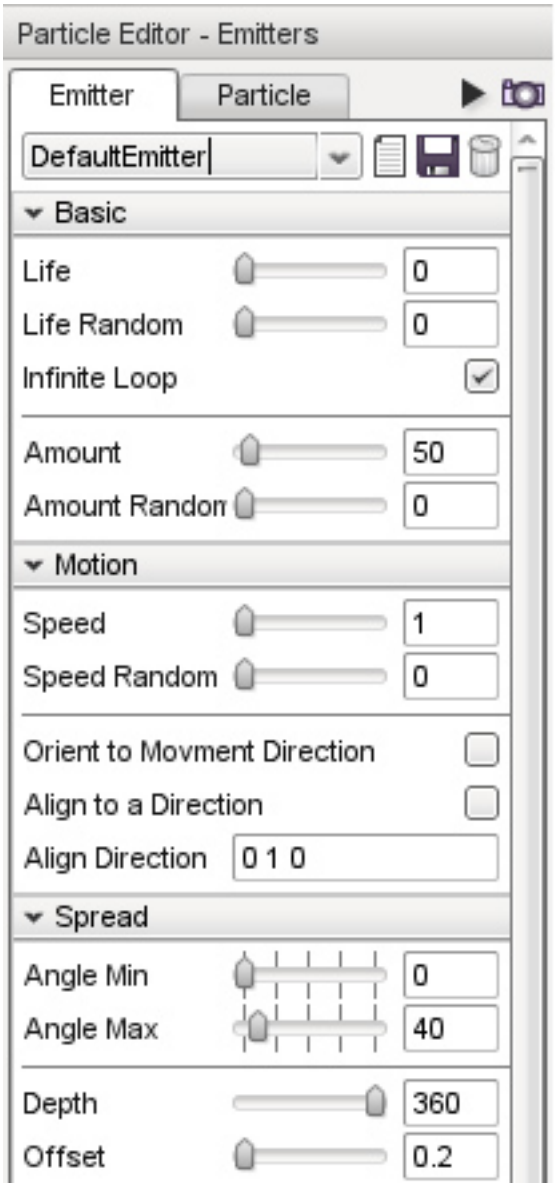
The emitter controls: the creation of the particles; their movement; which directions the particles will travel, also referred to as the spread pattern and: how each particle blends into the world.

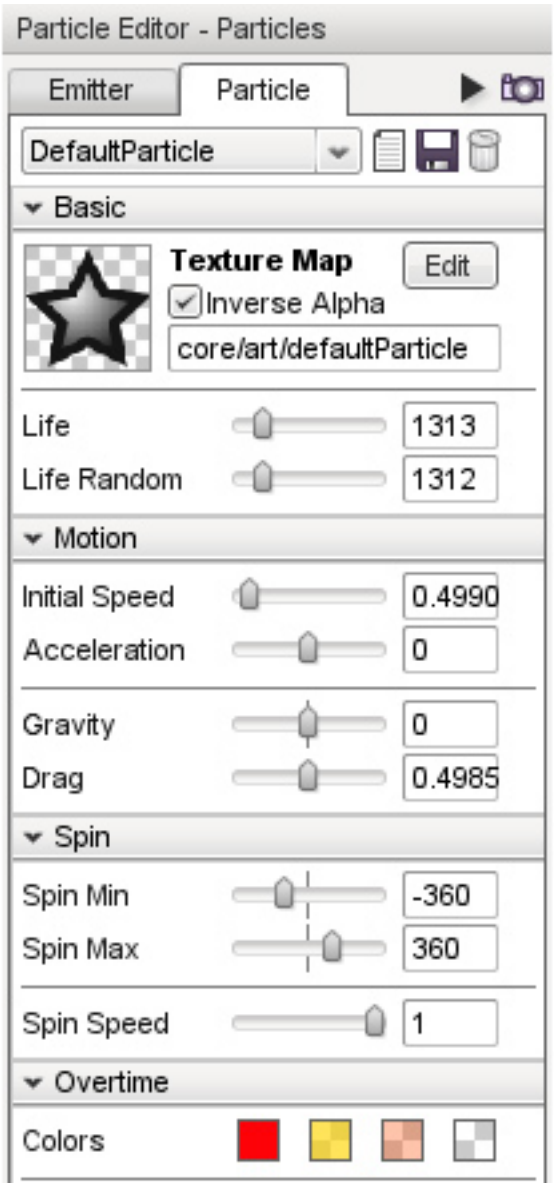
The particle controls its own life span, what image will be shown; how big the image is; what it's color over time is; and some basic force settings.

Interface

The Particle Editor can be activated from the dmain menu by selecting Editors -> Particle Editor. Or alternately, click the Particle Icon from the Tool Selector bar.Whenever the Particle Editor is active the *Particle Editor – Emitters* pane will be present on the right side of the screen. This pane is further divided into two tabs:

1. The Emitter tab contains properties about the currently selected emitter
2. The Particles tab contains properties about the currently selected particle.





Select either the Emitter tab or particle tab depending upon which object you wish to work with. In adition to the tabs there are also two buttons within the header of the Particle Editor.

One Shot Effect Types

There are two types of particle effect:

1. continuous effects, which constantly emit particles
2. one-shot effects, which only produce particles for a short time and then stop

Continuous effects run constantly so your changes can be seen in real-time as you adjust the properties of the emitter and its particles. In order to see your changes for one-shot emitters you need to replay the emission. To replay a one-shot emitter click the arrow icon to the right of the tabs.

The Temporary Emitter

When you open the Particle Editor you may have noticed it creates a temporary particle emitter in your current view. This temporary emitter is very useful for quickly trying out different particle editor settings.If your view is changed and you no longer see temporary emitter, press the little camera icon to the right of the tabs to place it back into view. It will always be placed in the center of your current view. The temporary particle emitter can be moved, rotated, and scaled like any other shape using the Object Editor.

New Emitter / Particle

To create a new blank emitter or particle that is ready to be configured, press the new icon on the Emitter or Particle tab as appropriate.

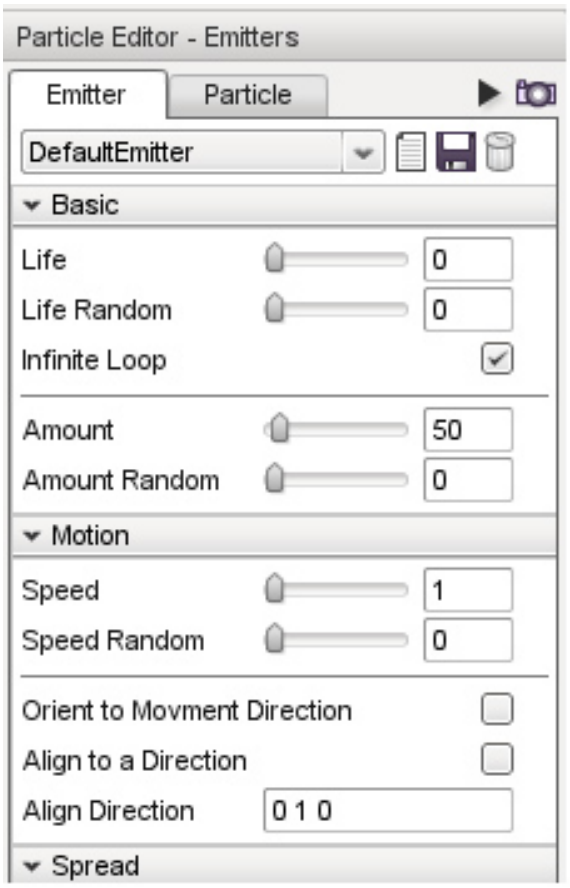
Save Emitter / Particle

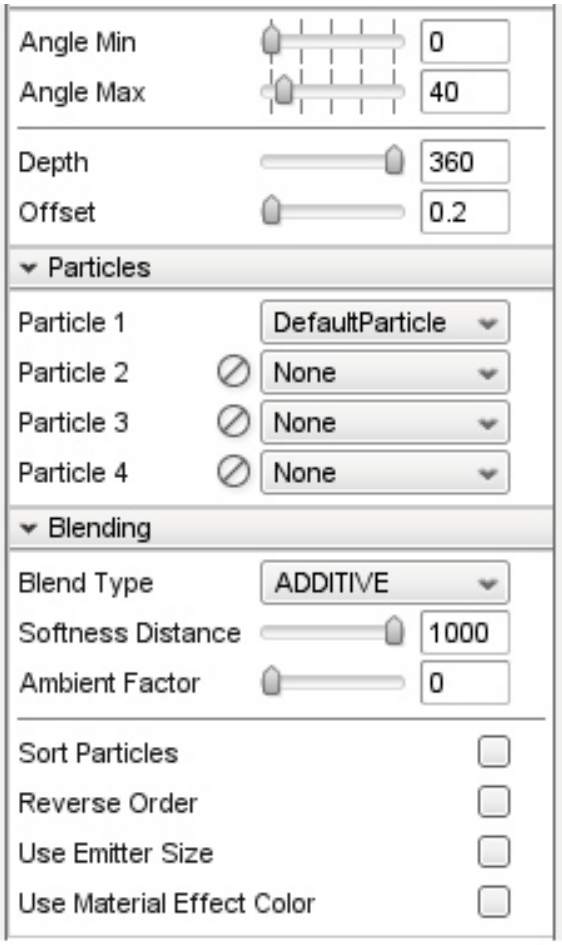
After editing an emitter or particle save the new settings by pressing the save icon on the Emitter or Particle tab as appropriate. Particle emitters are updated in real-time. Any changes to a particle or emitter will be reflected through out your level when changes are saved. Any instances of the emitter or particle that you are editing will also be changed. As with a lot of Torque 3D Editors the Particle Editor writes the resulting data to script files which the engine runs to create the particle emitter when you game is being played.

- Emitters can be found in a file named: projectName/game/art/levels/levelName.mis
- Particles can be found in a file named: projectName/game/art/shapes/particles/managedParticleData.cs

Emitter Properties

The Emitter tab contains the properties that define an Emitter. Properties are grouped into sections:





Basic

Basic properties affect the base emitter:

Life The time duration in ms that the effect will emit particles.

Life (Random) Substitutes a random value for the life property.

Infinite Loop When enabled this emitter will continuously produce particles. This setting effectively causes the Life and Random Life properties to have no affect on the emitter.

Amount The time in ms between each individual particle released from the emitter.

Amount Random Random Variation amount to be applied to the amount setting.

Motion

These settings will affect the emitter spread pattern, speed, and particle image orientation:

Speed The velocity the particle will leave the emitter in the defined spread pattern.

Speed Random A random setting for varying the speed.

Orient to Movement Direction Enabling this option fixes the particles image to the velocity direction of the particle. Note this will over ride any particle spin settings.

Align to a Direction Enabling this option aligns the particles to a predefined vector set up in Align Direction option.

Align Direction The vector used for particle alignment if the Align to a Direction option is checked.

Spread

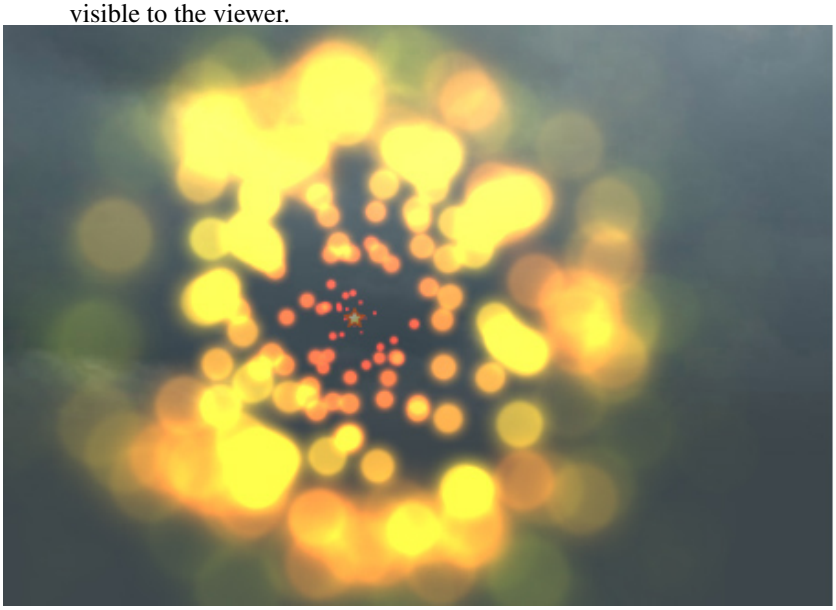
These setting affect how the spread pattern will be dispersed:

Angle Min The minimum angle for the emitter spread pattern.

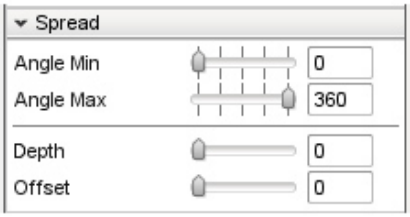
Angle Max The maximum angle for the emitter spread pattern.

Depth The depth of the released pattern. A setting of 360 will create a spherical spread pattern when Angle Max is set to 360.

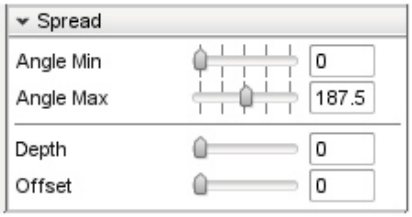
Offset The distance from the emitter that particles will be released. Effectively the distance that the particle will be visible to the viewer.



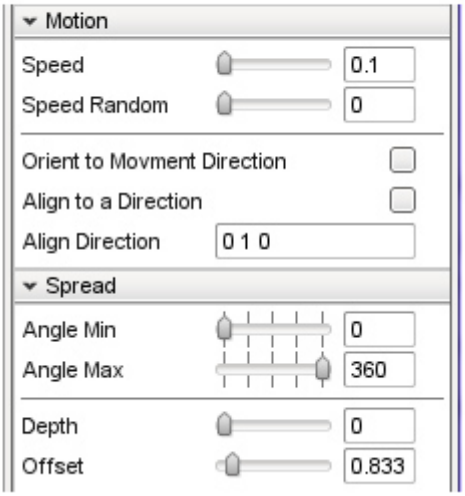
Example of Spread
Angle min /max



Example of Spread
Angle min /max



Example of Spread Offset



Particles

This affect assigns which particle(s) will be emitted from this emitter:

Particle 1 - 4 Select the particle from the drop down list to be used with this emitter. If at any time you need to remove a particle press the clear icon. Particle 1 can not be removed.

Blending

These setting affect how the particle(s) are rendered.

Blend Type The types of blending available to be applied to the particles.

Softness Distance The particle edge blending distance. Removes the hard edges where the particle meets an object.

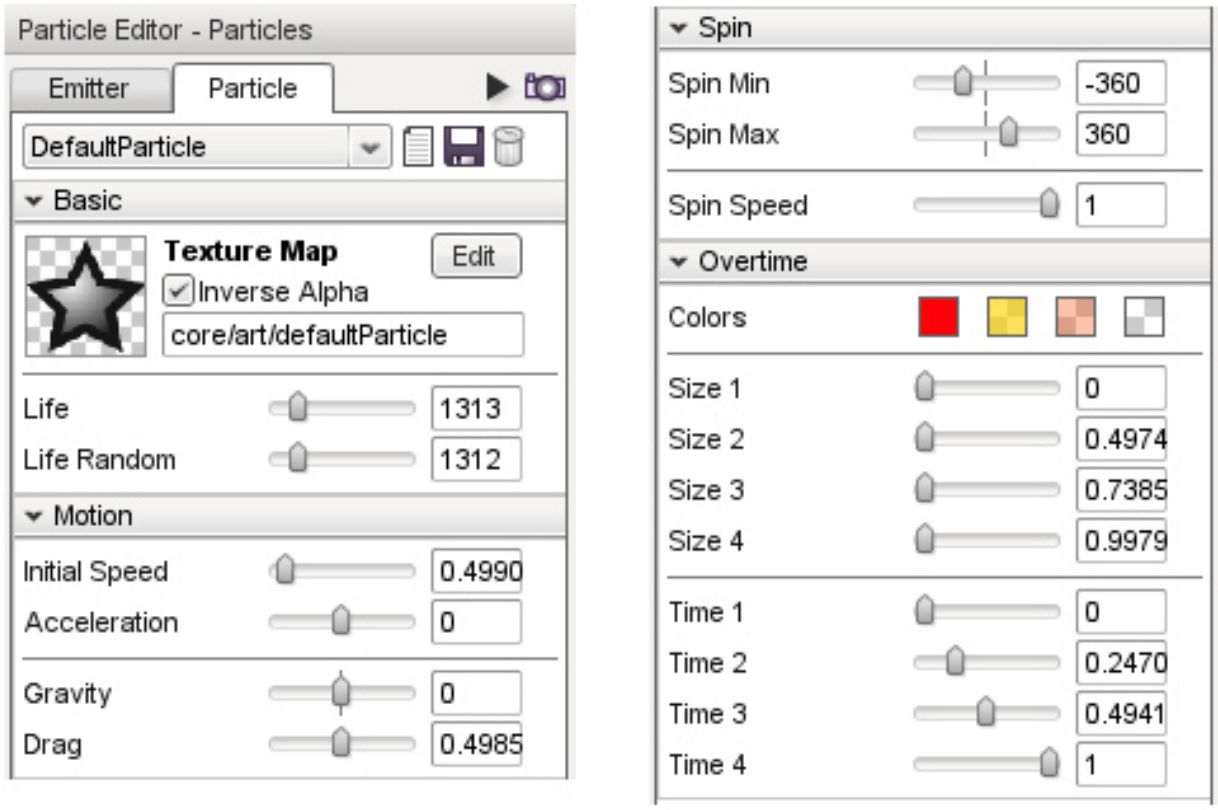
Ambient Factor Adjusts the alpha blend (level of the particles which affects how transparent they are).

Sort Particles The order in which particles are rendered.

Reverse Order When enabled, reverses the render order set in the Sort Particles setting

Particle Properties

The Particle tab contains the properties that define a Particle. Properties are grouped into sections:



Basic

Particle basic settings.

Texture Map The image that will be used on the emitted particle. The Edit button will open a file browser to locate and select a particle image.

Inverse Alpha Invert the alpha channel on the partice image (if one exists).

Life The time in ms (milliseconds) after its creation that the particle will exist for.

Life Random Random variation to the particle life span.

Motion

These settings affect the velocity of the particle.

Initial Speed The initial velocity, that the particle will travel at after being emitted. (Not to be confused with emitter spread speed.)

Acceleration The rate at which the particle's velocity with increase or decrease. Positive values cause a particle to speed up over time after being emitted. Negative values cause a particle to slow down over time after being emitted.

Gravity The gravitational force to be applied to particle. Positive values cause the particle to fall to the ground. Negative values cause the particle to rise from the ground.

Drag The amount of force working against the particle velocity. Drag will slow a particle's movement.

Spin

These settings affect if, and how, a particle rotates in degrees.

Spin Min The minimum rotation to be applied to the particle.

Spin Max The maximum rotation to be applied to the particle.

Spin Speed The speed of particle's rotation.

Overtime

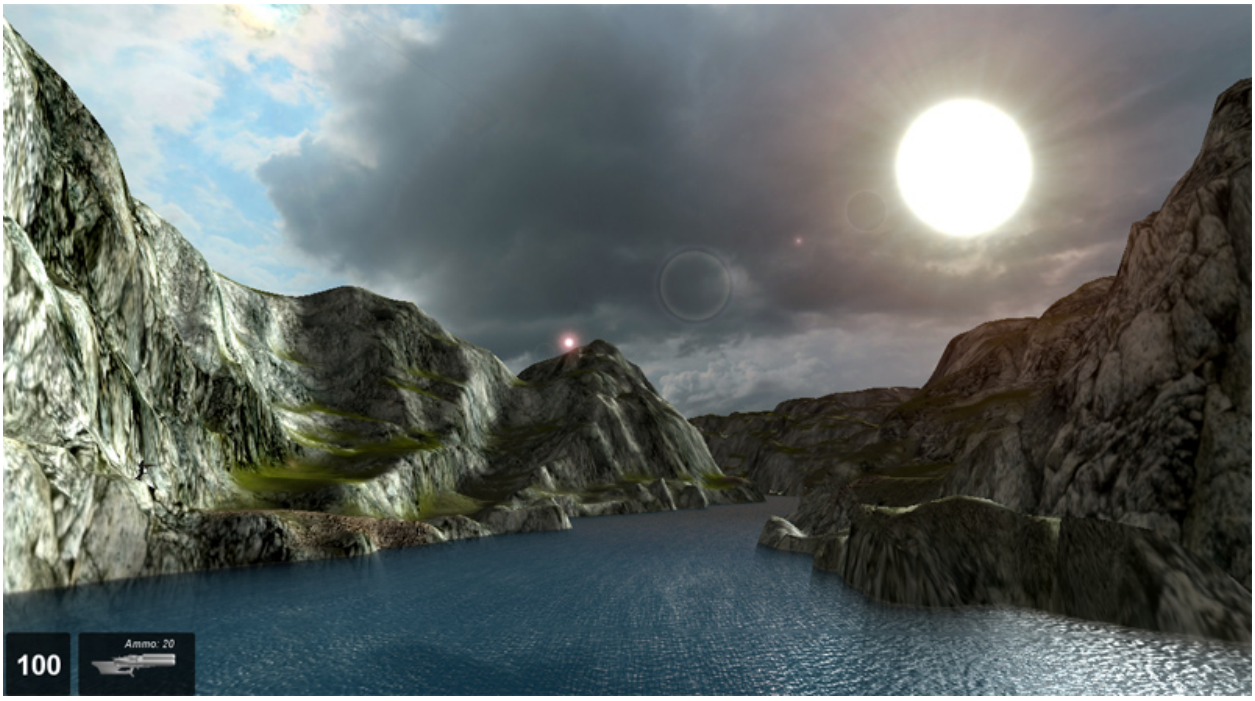
These settings affect the particle based upon how long it has been in existence for. Each particle can have up to four color and size settings, which can be set to change over time.

Colors Four color swatches indicate the color phases which a particle can pass through. To set any color click that swatch. To set a color value you may: enter R (red), G (green), and B (blue) color values; click anywhere within the gradient on the left or; click anywhere in the vertical "rainbow" strip. Red, green and blue color values range from 0 to 255 and indicate the amount of that color present in the overall particle color. The alpha value which represents the transparency of the particle color can be set by entering a decimal number between 0.0 and 1.0 in the Alpha field or by moving the slider with the mouse. The higher the number the less transparent the color will be.

Size 1-4 Each slider sets the size for the particle during each time stage.

Time 1-4 Each slider sets the time for that stage.

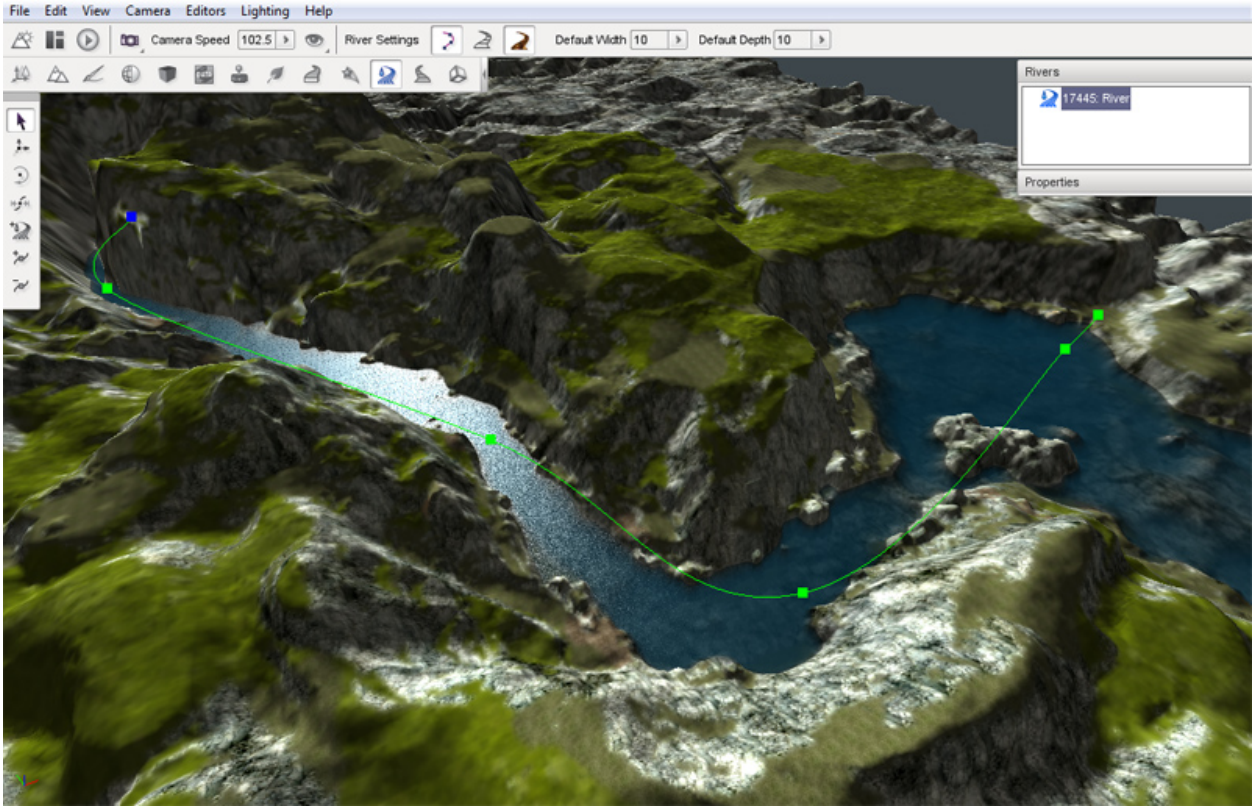
2.4.10 River Editor



The Torque 3D World Editor has a complete system for creating rivers and other small bodies of water. The River Editor is built-in WYSIWYG (What-You-See-Is-What-You-Get) editor with real-time feedback, giving you full control over how you would like to your river to appear.

Interface

To access the River Editor you can either activate it from the main menu by selecting Editors > River Editor. Alternatively you can click the River icon from the World Editor Tools Selector Bar.



The editor has two main parts, one where you can add and manipulate the river nodes (control points) for creation, the other for adjusting the river properties to create the river style (depth, width, flow, ripples etc). Whenever the River Editor is active three sections of the screen are updated to contain the editors tool. On the right side of the screen are three panes. At the top is the Rivers pane which contains a list of all the rivers currently in the level, if any are present. In the middle is the River Nodes pane which displays the properties of the currently selected river control point. At the bottom is the Properties Pane which displays the properties of the currently selected river. At the left of the screen the River placement tools will appear and are used to create and modify rivers and their control points. At the top of the screen in the World Editor Tool Settings Bar, a new set of icons will appear when the River Editor is active. These icons and their associated values will enable you to quickly set up the width and depth of the river and modify the editor to show and hide some visual aids which can be used to guide your river placement.

Adding a River

The river is created by placing a series of control points across the terrain which defines the path you would like your river to follow. Each control point, also called a “node”, will give you control of how the river will look at any given point. By adjusting each of these points we can have full control of where our river will go, its size, and its orientation.

The default width and depth of control points can be set using the Default Width and Default Height properties on the Tool Settings Bar at the top of the editor window. Any new rivers will be created using these settings until you change their values again.

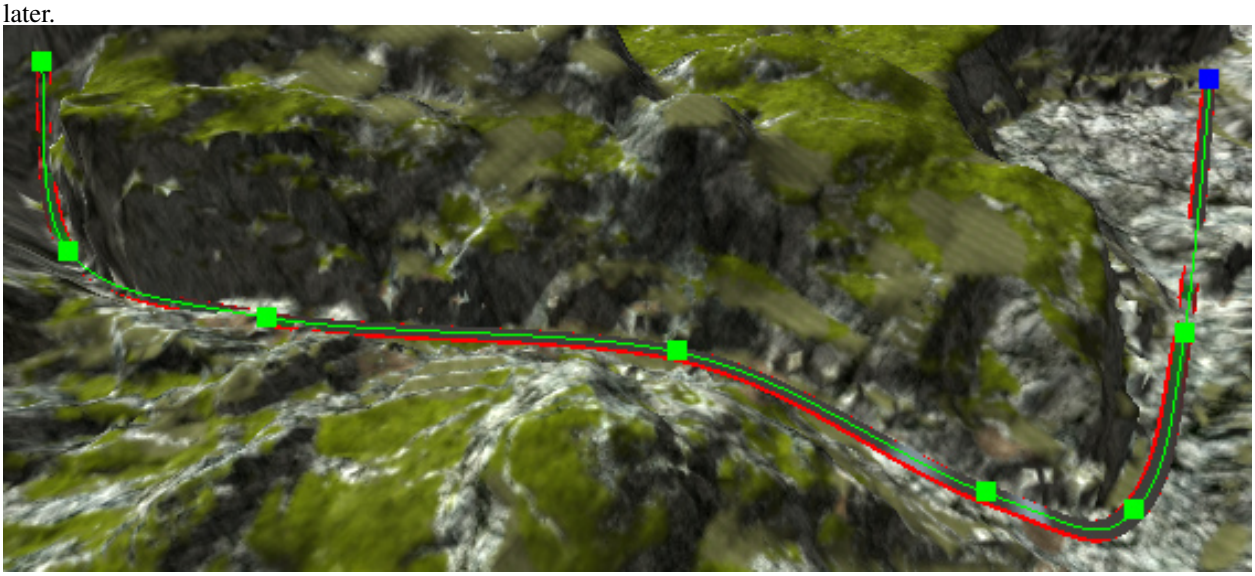
To create a new river select the Create River icon from the tool bar then click on the terrain with the left mouse button where you would like to start your river. Move the mouse away from the clicked location to see the results. Each time you click the terrain you will see three things:

1. a green square which represents the river location that you just placed
2. a blue square which represents the next location that will be placed the river if you press mouse button again
3. the surface of the river that will be placed the next time you click the button

Move the mouse to the next point on the terrain that you wish your river to travel to and then click again. Continue moving and clicking until you are finished with the initial placement of your river.

To complete the river placement process press the `Esc` key. This action will exit the Create River tool leaving your new river selected and ready for adjustments.

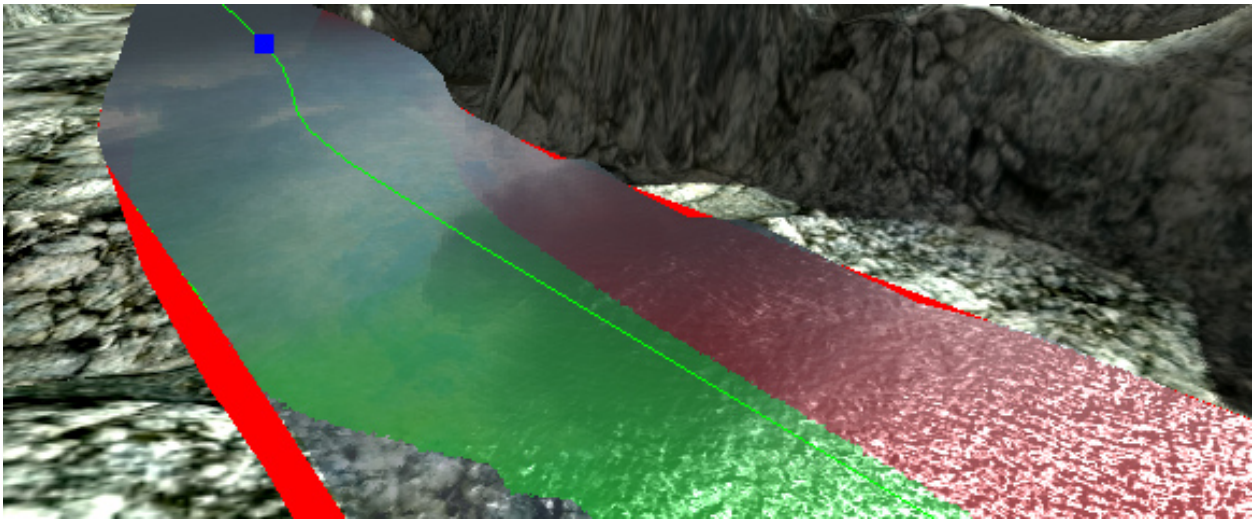
To abort a river creation operation without placing a river at all press the `Esc` key before selecting a second river point. Once a second river point has been placed the only way to remove the river completely is to delete it, as explained later.



The new River will also show up in the Rivers list at the right top of the screen. You will notice that the river itself is color-coded. These visual aids will be of help in adjusting your river.

RED The red lines at the edges of the river represent the river bounds. These lines need to be moved so that they are hidden by the terrain river bank to avoid gaps between the edge of the water and the surrounding terrain.

GREEN The green surface represents the depth of the river. This surface needs to be adjusted to be just below the terrain at every point in order for underwater views to be correct. Whenever this surface is above the terrain it will cause an "air bubble" between the bottom of the water and the terrain.



The new River may not look correct but with the following set of tools you will be able to adjust the width, depth and path.

Editing a River

The River Editor provides several tools for modifying rivers after they have been created. If at any time you make a mistake with any tool, you can press CTRL+Z to undo it.

Selection Tool

Once you have created your initial river you may need to edit some or all of the control points. This tool will allow you to directly select any created point for further editing. To activate the Selection Tool click its icon on the Tool Selector bar. Note that the River Editor will automatically select this tool when you have finished creating a new river.

An entire river can be selected by clicking anywhere on a river other than one of its control points. This type of selection will result in the river being highlighted with a "spline", which is a curved line that runs along the center line of the river, and a series of green squares which represent the rivers control points. There are no operations that can be performed on a river as a whole within the River Editor. Selecting a river allows you to see its centerline and it control points for individual selection and manipulation. To perform operations on the entire river such as moving it to a new location use the Object Editors tools as with any other shape in your level.

Control points can be selected individually to adjust each point as necessary. To select a control point left click on one of the colored squares that represent a rivers control points. A selected control point will turn blue.

Selecting a control point also causes the Properties pane on the right of the screen to be updated to display the current property values of the control point. The Node Properties section will display the position, rotation, width and depth of the selected control point. Values can be directly entered into these fields to modify the point or the Move Tool can be used to manipulate the point using the mouse. A selected control point will turn blue.

Moving a River

If at any time you are unhappy with the placement of a selected River control point you can use the Move Point tool to adjust its position. To activate the Move Point tool click its icon in the Tool Selector bar. The move gizmo will appear. The move gizmo is used to move the river point to a new location. Left mouse click on any arrowhead then drag the mouse to move the point along that arrows axis. Release the mouse button to relocate the control point to that new

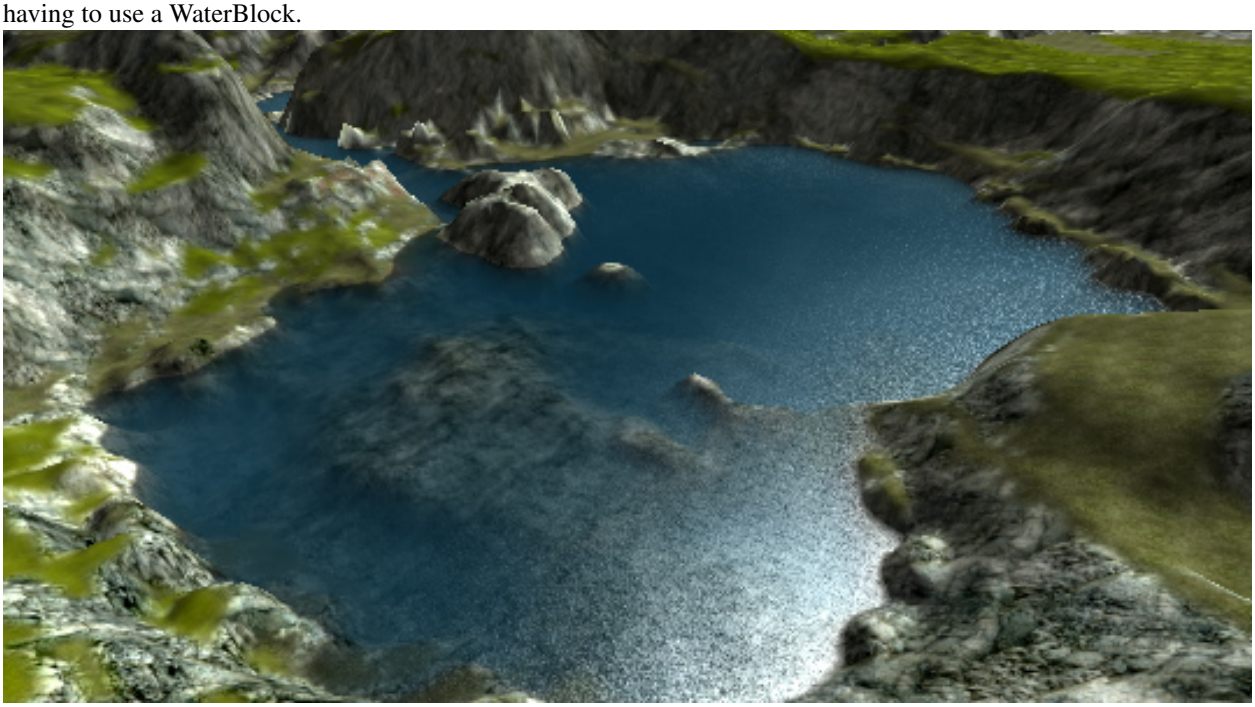
location. Left mouse click on the colored square at the origin of the axes then drag the mouse to freely move the point to without regard to any axis.

Scaling a River

The width and depth of a river can be directly adjusted at a selected control point by using the Scale Point tool. To activate the Scale Point tool click on its icon on the Tool Selector. The scaling gizmo will appear.

Left mouse click on the colored cube at the end of any axis then drag the mouse while holding the button down to increase or decrease the size of the road along that axis. To adjust the width and depth at the same time left mouse click on the colored cube at the origin of the axes then drag the mouse while holding down the button. Release the mouse button to change the river to that new width and depth. Changing the width and depth of the river in this manner is the main method to make sure that the red edges and the green surface, mentioned above, are concealed by the terrain.

The Scale point tool will allow you to quickly create very wide river sections, even as wide as a small lake, without having to use a WaterBlock.



Rotating a river

The Rotate Tool can be used to rotate a river at any selected control point. To activate the Rotate Tool click its icon on the Tool Selector. The rotate gizmo will appear. Rotating a river at each control in along the path of a river can make a river appear to be flowing downhill as opposed to a flat surface as is created by default.

Adding extra Points

The Insert Point tool can be used to add extra points in a river to create a smoother curve. In order to insert a new point into a river the river must first be selected. See the Selection Tool above for details on how to select a river. To activate the Insert Point tool once a river has been selected click its icon on the Tool Selector bar. To place a new point on the selected river click on the river where you would like the new point to be placed. A new point will be added to the river and will immediately the currently selected point as indicated by the blue square.

Removing Points

The Remove Point tool can be used to delete a control from a river. In order to remove a new point from a river the river must first be selected. See the Selection Tool above for details on how to select a river. To activate the Remove Point tool click its icon on the Tool Selector bar. To remove a control from the selected road point click on the control point. This will remove only the selected point leaving all the others in place. No adjustments will be performed on the other existing control points.

Properties

The Properties pane on the right side of the screen can be used to configure or modify various facets of the river object, such as its flow, colors, underwater effects, etc.

Transform

This section contains properties which control the placement, rotation and scale of the River as a whole.

Position Indicates the position of entire River in the level.

Rotation Indicates the rotation of the entire River in the level.

Scale Indicates the scale of the entire River in the level.

River

This section contains properties which control how the River is rendered which in turn will have an effect on the wave settings.

Segment Length The river will be divided into segments of this length, in meters.

Subdivide Length River segments will be subdivided in a way that each quad (four-sided polygon) is not any wider or longer than this distance in meters.

Flow Magnitude The magnitude of the force vector applied to dynamic objects that are within the River. This will affect how thing floating or suspended in the water are driven by the flow of the river.

Low LOD Distance Segments of the river at this distance in meters or more will be rendered as a single un-subdivided area without any undulation wave effects.

Water Object

This section contains properties that control the look and action of the water and contains several sub-sections.

Density Affects the buoyancy of an object entering the water.

Viscosity Affects a submerged object's drag force.

Liquid Type Type of datablock used to represent the type of liquid contained in the river (i.e. water, lava, etc.)

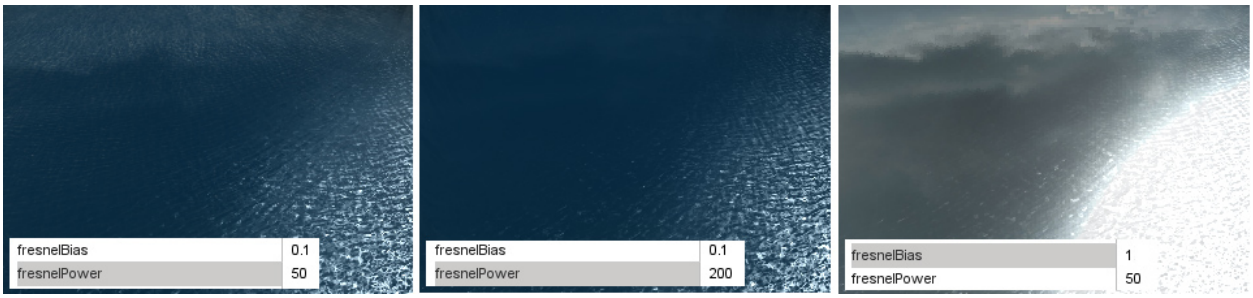
Base Color Changes the color of the underwater fog which has the effect of coloring the water surface.

Fresnel Bias Extent of Fresnel (reflection level based on viewing angle) affecting reflection fogging

Fresnel Power Measures the intensity of the effect on the reflection, based on fogging.

Specular Power Power used for secularity (lighting reflection) on the water surface (sun only)

Specular Color Specular color used for the water surface (sun only)



Waves

This sub-section contains properties that control the undulations on the water. Note: This effect actually moves the vertices of the mesh. This section has further sub-sections for controlling three wave sets, each sub-section is composed of the following properties that define the wave set.

Wave Dir A vector describing the direction the waves flow towards the river banks.

Wave Speed Speed of the wave undulation.

Wave Magnitude Height of the wave.

Overall Wave Magnitude This master parameter affects the depth of all the wave subsets, like a global wave height parameter.

Ripple Texture The Normal map used for simulating the surface ripples.

Ripples

This sub-section contains properties that control the animation that simulates the effect of ripples bouncing off the river bank. This animation is performed using normal map to represent the ripples. This section has further sub-sections for controlling three ripple sets, each sub-section is composed of the following properties that define the ripple set.

Ripple Dir A vector that modifies the surface ripple direction.

Ripple Speed Controls the ripple speed.

Ripple Tex Scale Intensifies the affect of the surface ripples by scaling the texture.

Ripple Magnitude Intensifies the ripple effect.

Overall Ripple Magnitude This parameter affects the depth of all the ripple subsets, like a global ripple intensity variable.

Foam Tex The texture used to render the ripple effect.

Reflect

This section contains properties that control the rendering of surface reflections:

CubeMap Cubemap to use instead of the default reflection texture, which is the current sky, if Full Reflect is turned off. Handy if you have not yet set up a sky for your project.

FullReflect Enables dynamic reflection rendering, which causes the water surface to reflect the current sky, if available.

Reflect Priority Affects the sort order of reflected objects.

Reflect Max Rate Ms Affects the sort time of reflected objects.

Reflect Detail Adjust Scale up or down the detail level for objects rendered in a reflection.

Reflect Normal Up The reflection normal.

Use Occlusion Query Turn off reflection rendering when occluded.

Reflect Text Size Texture size used for the reflections.

Underwater Fogging

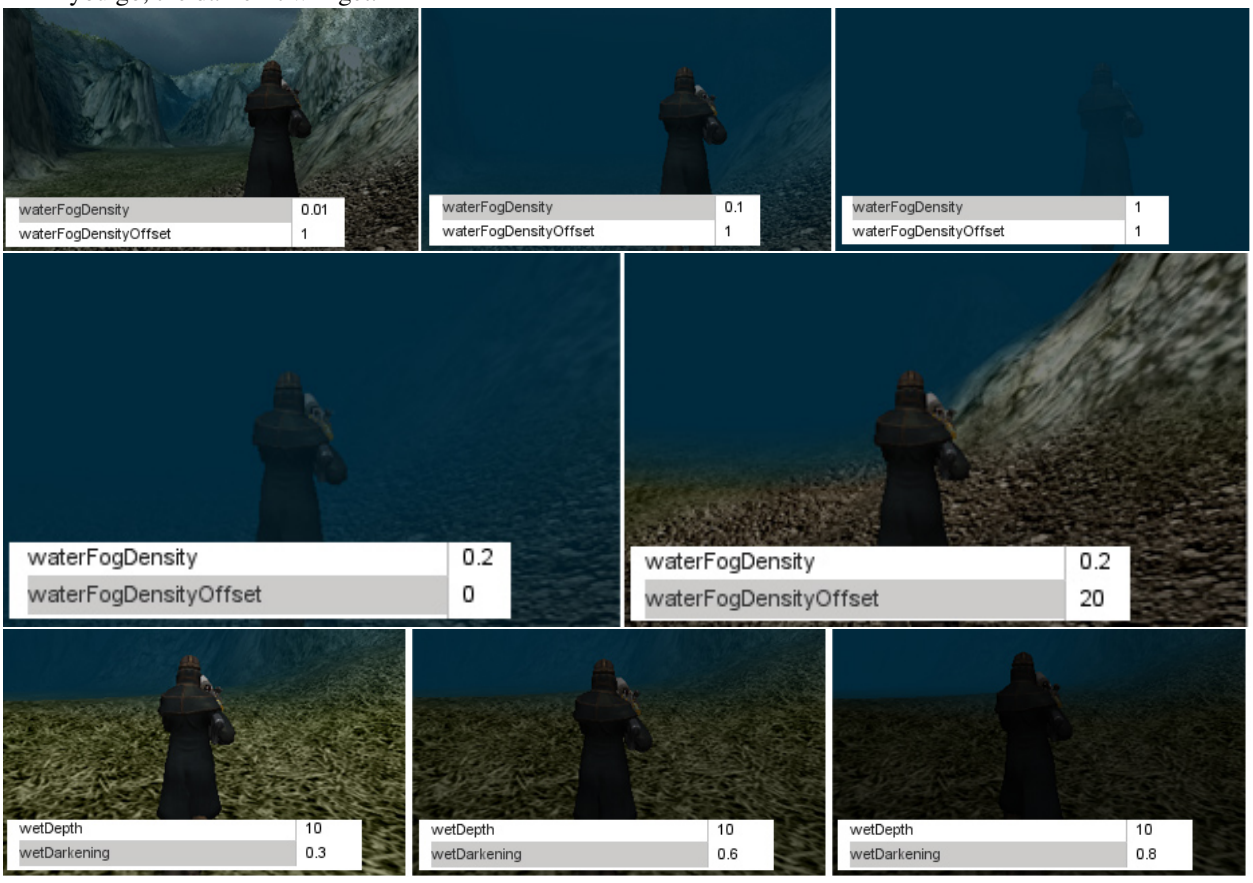
This section contains properties that control how the underwater view appears.

Water Fog Density The intensity of the underwater fogging.

Water Fog Density Offset The offset distance before the fogging occurs.

Wet Depth The depth in world units at which full darkening will occur, giving a wet look to objects underwater.

Wet Darkening The refract color intensity scaled to the depth of the player (wetDepth). The deeper under the water you go, the darker it will get.



Misc

Other uncategorized properties.

Depth Gradient Tex Texture for the gradient as the players moves deeper.

Depth Gradient Max Maximum depth for the gradient texture.

Foam

Foam Opacity Overall foam opacity.

Foam Max Depth The depth that the foam will be visible from underwater.

Foam Ambient Lerp An RGB color value that interpolates linearly between the base foam color and ambient color. This prevents bright white colors be viewable during situations such as Night.

Foam Ripple Influence Intensity of the foam effect on ripples.

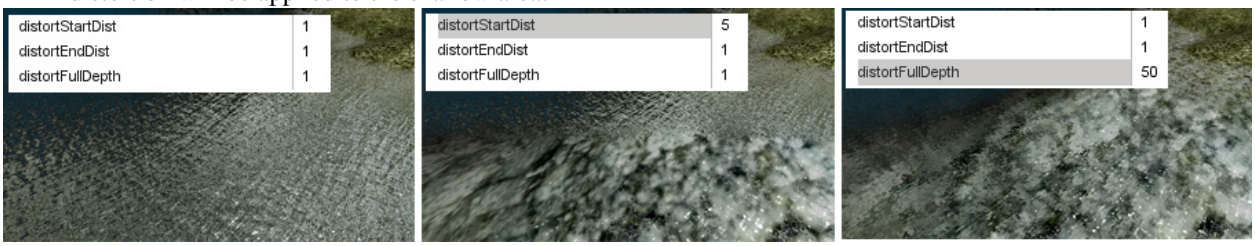
Distortion

This section contains properties that control how the water distorts the under water terrain when viewed from above.

Distort Start Dist Determines the start of the distortion effect from the camera.

Distort End Dist Max Distance that the distortion algorithm is performed. Lower values will show more of the distortion effect.

Distort Full Depth Sets the scaling down value for the distortion in shallow water. The lower the value the more the distortion will be applied to the shallow area.



Basic Lighting

This section contains properties that control the basic lighting effects on and in the water:

Clarity Opacity or transparency of the water surface.

Underwater Color Changes the color shading of objects beneath the water surface

Sound

This section contains properties that control sound under the water.

Sound Ambience Ambient sound environment for when the listener is submerged.

Editing

This section contains properties that control whether the river can be edited.

isRenderEnabled Toggles whether the object is rendered on the client.

isSelectionEnabled Toggles whether the object can be selected in the tools.

hidden Toggles whether the object is visible.

locked Toggles whether the object can be edited.

Mounting

This section contains properties that control whether the river can be mounted to another world object, for example a sewer pipe or a cave.

mountPID PersistentID of object we are mounted to.

mountNode Node we are mounted to.

mountPos Position where object is mounted.

mountRot Rotation where object is mounted.

Object

This section contains properties that control whether the river object is persistent in the world.

internalName Internal name of this object.

parentGroup Group to which this object belongs.

class Class to which this object belongs.

superClass SuperClass to which this object belongs.

Persistence

This section contains properties that control whether the river object is persistent in the world.

canSave Whether the object can be saved to the mission file.

canSaveDynamicField Whether dynamic properties are saved at runtime.

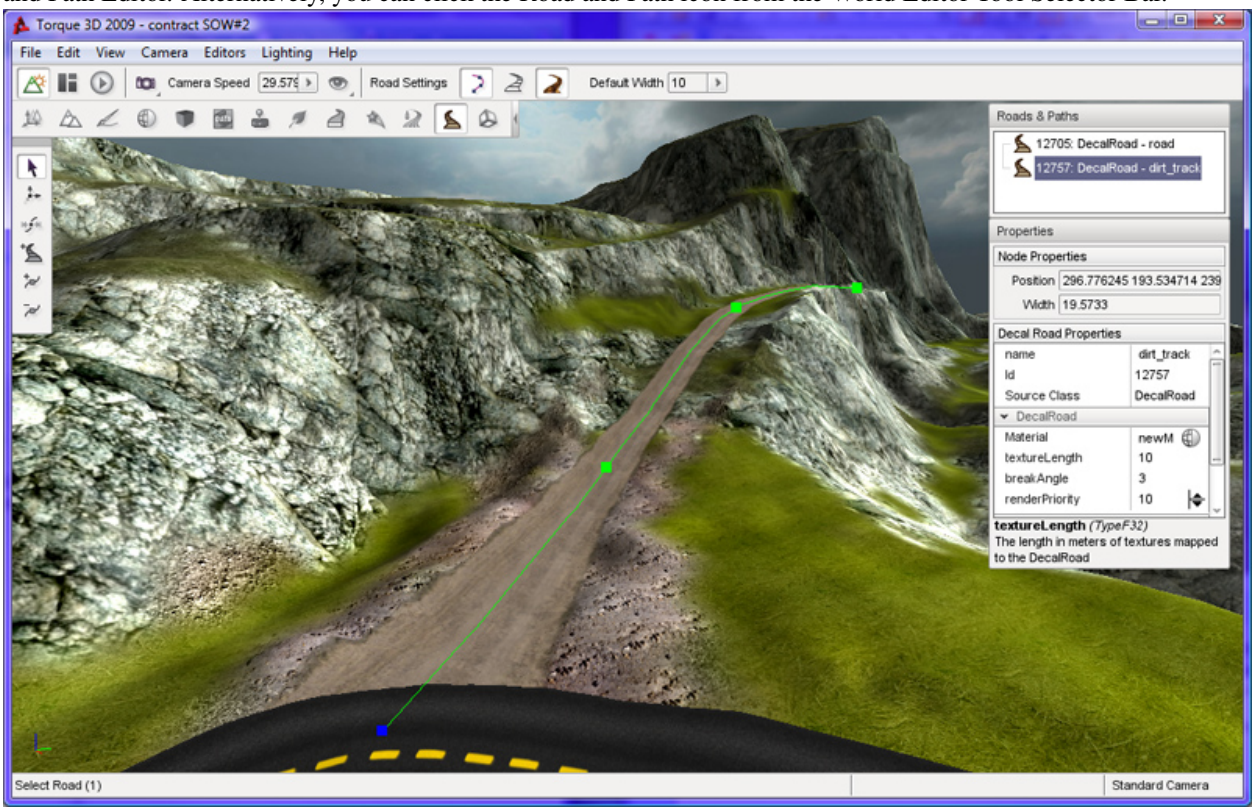
persistentID Unique ID of this object.

2.4.11 Decal Road Editor

The Road and Path Editor is used to create decal-based roads on your terrain. A decal-based road follows every contour of the terrain, unlike a Mesh Road which is a solid 3D object. By using the Road and Path Editor and a few choice materials, you can easily create dirt tracks, trails, paths, and simple roads. To create more complicate roads that rise above the terrain, span hills, or contain bridges use the Mesh Road Editor. The Road and Path Editor is a built-in WYSIWYG (What-You-See-Is-What-You-Get) editing tool which provides near real-time feedback so that you can see the changes and additions as you make them.

Interface

To access the Road and Path Editor activate it from the main menu of the World Editor by selecting Editors > Road and Path Editor. Alternatively, you can click the Road and Path icon from the World Editor Tool Selector Bar.



Whenever the Road and Path Editor is active three sections of the screen are updated to contain the editors tools. On the right side of the screen are the Roads and Paths pane and the Properties pane. At the top is the Roads and Paths pane which contains a list of all the decal-based roads currently in the level, if any are present. At the bottom is the Properties Pane which displays the properties of the currently selected road. At the left of the screen the Road and Path placement tools will appear which are used to create and modify your roads. At the top of the screen in the world editor tool bar, a new set of icons will appear after selecting the Road Editor. These icons and their associated values will enable you to quickly set up the width of the control points and modify the editor to show and hide some visual aids which can be used to guide your road placement.

There is no depth parameter for decal-roads as there is for Mesh based Roads. As mentioned earlier decal-based roads sit right on the terrain surface and follow the terrain exactly. They do not have their own geometry.

Adding a Decal Road

A decal-based road is created by placing a number of control points across the terrain. Each point can be edited for width at any time. By adjusting these points we have full control over how our road will look. The default width of

control points can be set using the Default Width property on the Tool Settings Bar at the top of the editor window. Any new roads will be created using these settings until you change their values again.

To create a new road select the Create Road icon from the tool bar then click on the terrain with the left mouse button where you would like to start your road. Move the mouse away from the clicked location to see the results. Each time you click the terrain you will see three things:

1. a green square which represents the road location that you just placed
2. a blue square which represents the next location that the road will be if you press mouse button again
3. the road decals that were just placed

Depending upon the power of your computer there may be a delay between when you click the terrain and when the decals appear. Move the mouse to the next point on the terrain that you wish your road to travel to and then click again. Continue moving and clicking until you are finished with the initial placement of your road.

To complete the road placement process press the `Esc` key. This action will exit the Create Road tool leaving your new road selected and ready for adjustments.

To abort a road creation operation without placing a road at all press the `Esc` key before selecting a second road point. Once a second road point has been placed the only way to remove the road completely is to delete it, as explained later.

Editing a Decal Road

The Road and Path Editor provides several tools for modifying roads after they have been created. If at any time you make a mistake with any tool, you can press CTRL+Z to undo it. As with road placement, depending upon the power of your computer there may be a delay between when you perform and editing action and when the change appears in the scene.

Selection Tool

Once you have created your initial road you may need to edit some or all of the control points. This tool will allow you to directly select any created point for further editing. To activate the Selection Tool click its icon on the Tool Selector bar. Note that the Road and Path Editor will automatically select this tool when you have finished creating a new road.

The selection tool allows two types of selection relating to roads:

- An entire road can be selected by clicking anywhere on a road other than one of its control points. This type of selection will result in the road being highlighted with a “spline”, which is a curved line that runs along the center line of the road, and a series of green squares which represent the roads control points. The only operations that can be performed on a road as a whole is deleting it. To delete an entire road press the Del key and confirm the operation using the dialog box that will pop up. Unlike a Mesh Road you can not move a decal road as a whole using either the Road and Path Editor or the Object Editor. Selecting a road allows you to see its centerline and it control points for individual selection and manipulation.
- Selecting a control point also causes the Properties pane on the right of the screen to be updated to display the current property values of the control point. The Node Properties section will display the position and width of the selected control point. Values can be directly entered into these fields to modify the point or the Move Point Tool and Scale Point Tool can be used to manipulate the point using the mouse.

Moving a Road

If at any time you are unhappy with the placement of a selected Road Mesh control point you can use the Move Tool to adjust its position. To activate the Move Tool click its icon in the Tool Selector bar.

This editor's move mode works a little bit different than the other editors as there is no gizmo over the selected control point when tool is active. To move the selected control point: click the left mouse button on the control point; hold down the button; and drag it to a new position. The road decal will always follow the contour of the terrain. There may be a small delay as the editor updates the decal road.

Scaling a Road

If you feel that the road is not the correct width, or you just want to make some variations in a dirt track, you can use the Scale Point tool to change the width. This tool works in a similar fashion to the Move Point tool as there is no gizmo over the selected control point when the tool is active. To activate the Scale Point tool click its icon on the Tool Selector.

To change the width of the road select the control point you would like to scale; click the control point using the left mouse button; hold the button down; and drag the point to the left to reduce the width, or drag it to the right to increase the width. As with the Move tool there may be a small delay as the editor updates the decal road. Release the button to leave the road at that width at any time.

Adding Extra Points

The Insert Point tool can be used to add extra points in a road to create a smoother curve. In order to insert a new point into a road the road must first be selected. See the Selection Tool above for details on how to select a road. To activate the Insert Point tool once a road has been selected click its icon on the Tool Selector bar. To place a new point on the selected road click on the road where you would like the new point to be placed. A new point will be added to the road and will immediately be the currently selected point as indicated by the blue square.

Removing Points

The Remove Point tool can be used to delete a control from a road. In order to remove a new point from a road the road must first be selected. See the Selection Tool above for details on how to select a road. To activate the Remove Point tool click its icon on the Tool Selector bar. To remove a control from the selected road point click on the control point. This will remove only the selected point leaving all the others in place. No adjustments will be performed on the other existing control points.

Properties

The Properties pane on the right side of the screen can be used to configure a decal-based Road.

Decal Road

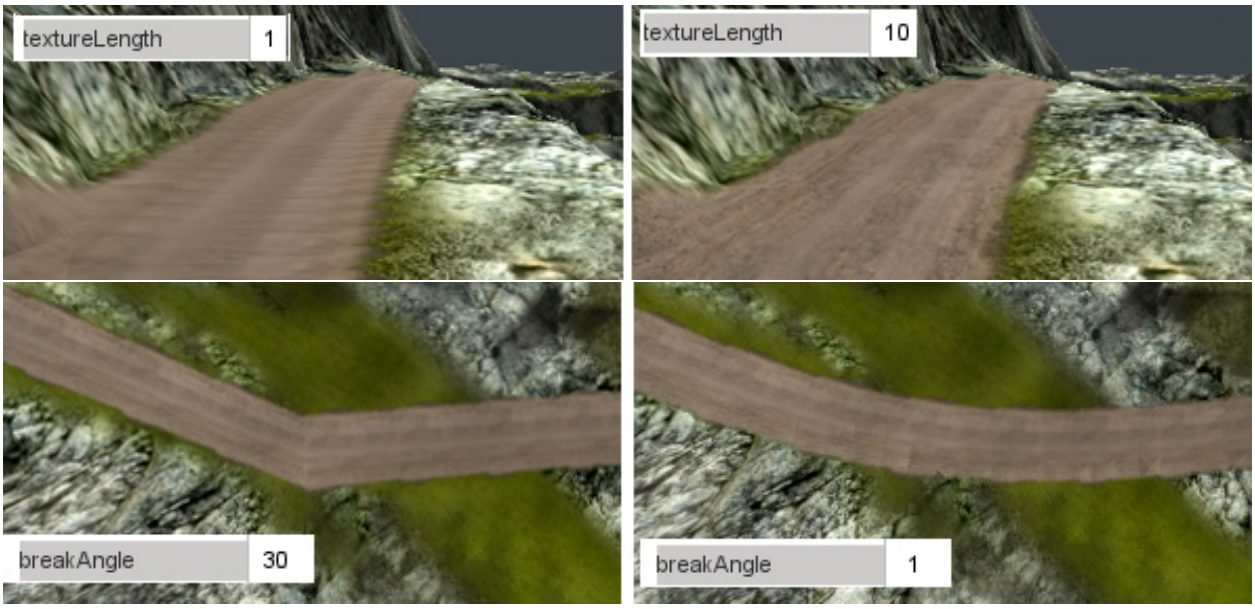
This section contains properties that control the roads appearance.

Material The texture assigned to this property will be used as the decal that displays on the terrain to represent the roads surface. Clicking the small round icon to it right will open the Torque 3D Material Selector window. From this window you can select a new material to assign to the Material property. For full details on how to use the Material Selector and how to create new materials see the Material Editor article.

Texture Length The length the texture will be rendered at in meters, measured along the centerline of the road.

Break Angle Indicates the angle in degrees that the mesh roads spline will be subdivided into if its curve becomes greater than this threshold.

Render Priority Decal roads are rendered in descending order.

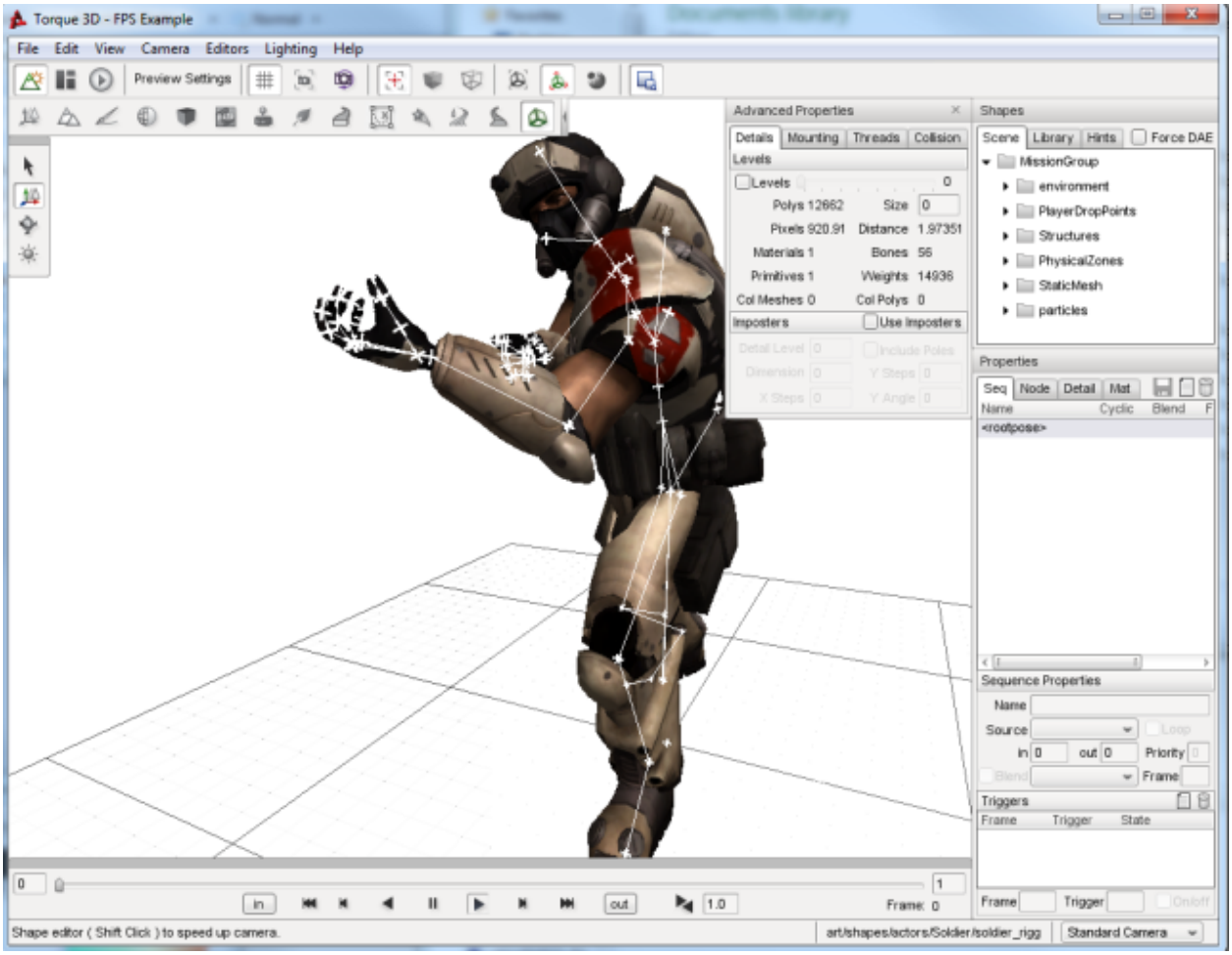


2.4.12 Shape Editor

The Shape Editor is used to view and edit the shapes that can be placed into your levels using the Object Editor. The Shape Editor can view and manipulate files in both the DTS (.dts) and COLLADA (.dae) formats. It can be used to quickly preview shapes before they are added to a level, and provides an easy way to add, edit and delete animation sequences, skeleton nodes, and rendering detail levels.

Interface

The Shape Editor can be activated from the main menu by selecting Editors > Shape Editor.



The Shape Editor interface consists of five primary sections whenever it is active.

Shape Selector The Shape Selector panel is used to choose a shape file for viewing and editing. It is composed of 3 tabs: Scene, Library and Hints. The Scene tab allows you to select a shape that has been placed in the current level. The Library tab allows you to browse and select any DTS or COLLADA shape from your project's art folder. Finally, the Hints tab displays information about which nodes and sequences are expected by Torque for a given type of shape object.

Shape View Window The main window shows a 3D view of the selected shape, and includes animation playback controls to play and single-step the selected sequence.

Properties Window The Properties panel is used to view and edit the sequences, nodes, details and materials in the shape.

Advanced Properties Window The Advanced Properties window displays Level-of-Detail information, and provides the ability to mount objects to other objects and animation thread control.

Toolbar and Palette The Shape Editor adds several buttons to the standard World Editor toolbar to control the 3D shape view, and uses the familiar select/move/rotate palette for node transform manipulation.

Shape Selection

To start using the Shape Editor, first you need to select a shape. There are three ways to do this:

1. Select an object in the World Editor, then activate the Shape Editor from the menu bar or the toolbar. If the object uses a DTS or COLLADA file, it will automatically be selected in the Shape Editor.
2. Select an object using the Scene tree in the Shape Editor. This view is the same as that used in the World Editor, and provides a convenient way to select objects that have already been placed in the level. Note that the Shape Editor only allows selection of objects that use DTS or COLLADA files; selection of Interior or ConvexShape objects will be ignored.
3. Select a shape file using the Library tab. This view is the same as that used in the World Editor Meshes tab, and allows you to browse the DTS and COLLADA assets in your project's art folder. This method allows you to view and edit shape files that have not yet been placed in the level.

The Force DAE option checkbox at the top of the Shapes panel forces Torque to load the COLLADA file, even if an up-to-date cached.dts file is present. Note that if the model is already present in the scene (and thus already loaded into the Torque Resource Manager), the Force DAE option will have no effect, as the shape will be opened from memory instead of from disk. This option is also available in the Editor Settings panel when working in the World Editor.

You will be prompted to save if there are unsaved changes in the current shape when a new shape is selected. The selected shape will appear in the View window, and listings of its sequences, nodes and materials will be displayed in the Properties window.

Shape Hints

The Hints tab in the Shape Selection window shows you which nodes and sequences are required for a particular type of shape to work with Torque.

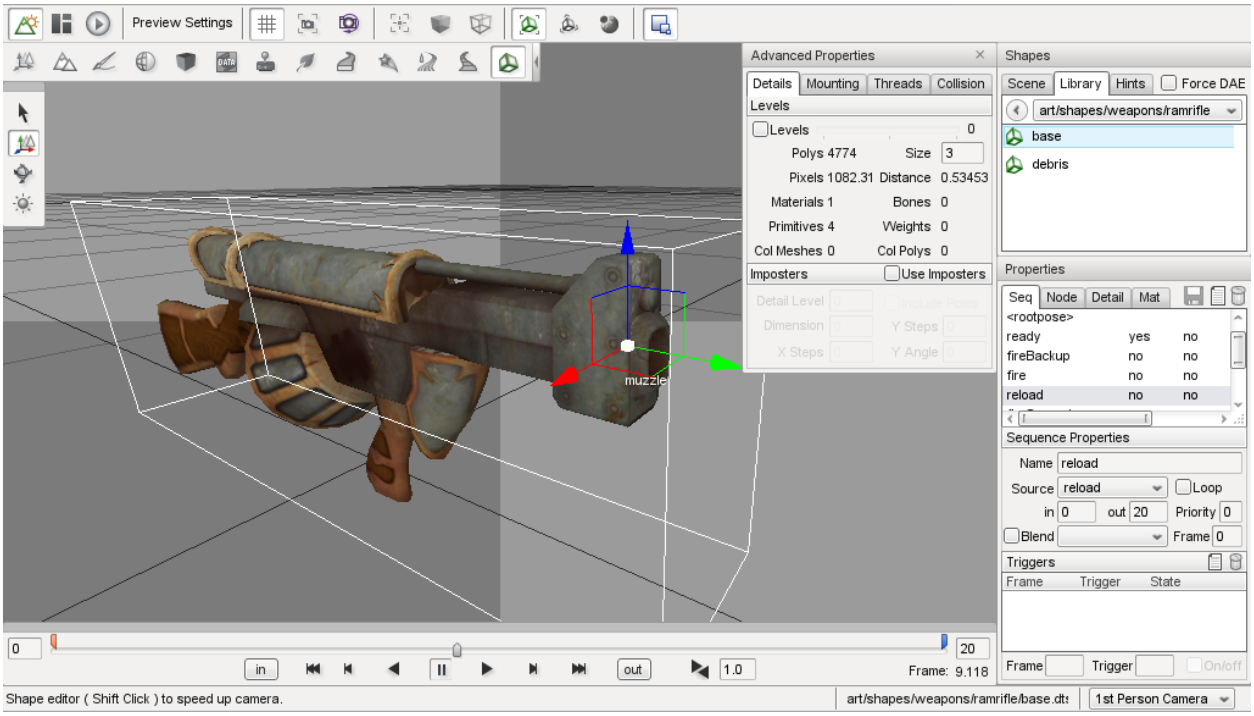
Simply select the desired object type from the dropdown menu and the list of required nodes and sequences will be displayed underneath. Items that are present in the selected shape will be marked with a tick mark. Hovering the mouse cursor over an item will display a short description of the item. Double-click the item to add it to the current shape.

Most items are optional - the shape will still load and run without a particular node or sequence, but the object may not perform correctly in-game. A Player object for example uses a node called cam as the 3rd person camera position. If this node does not exist, the shape origin is used instead, which will probably not be correct for most character shapes.

It is easy to extend the Shape Editor hints for custom object types by adding to the list in: tools/shapeEditor/scripts/shapeEditorHints.ed.cs.

Shape View

The Shape View window displays the shape as it would be seen in-game, and also provides helpful rendering modes such as transparent, wireframe and visible nodes.



The camera can be rotated by dragging the right mouse button, translated by dragging the middle mouse button (drag left+right buttons if your mouse does not have a middle button), and zoomed using the mouse wheel. Use the Camera->View menu (or the dropdown list in the bottom right corner) to switch between the Standard/Perspective view and the orthographic views (Top, Bottom, Left, Right, Front, Back).

Hovering the mouse over a shape node will display the node name, and left clicking a node in the view will select it in the Node Properties panel (and vice versa). Once a node is selected, its transform can be modified by dragging the 3D gizmo similar to how objects are positioned in the World Editor.

Animation Controls

At the bottom of the Shape View window are the animation playback controls:

As well as allowing the selected sequence to be scrubbed with the slider, stepped one frame at a time, or played normally, the start and end frames of the sequence can be easily modified to facilitate sequence splitting or to correct off-by-one-frame looping errors. Sequence triggers appear as thin, vertical bars at the appropriate frame (as shown in the image above).

Pressing the in or out button, or modifying the text box directly (remember to hit Return to apply the change), will set the start or end frame of the sequence to the current slider position.

Properties Window

The Properties window is where you can view and edit the sequences, nodes, detail levels and materials in the shape. The top right corner has three buttons, which do the following:

- Save the shape
- Add a new sequence, node or detail; and

- Delete the selected sequence, node or detail

Sequences Tab

The Sequences tab (displayed as "Seq" onscreen) lists the sequences available in the shape, as well as a number of different properties about the selected sequence. In addition, the 'root' (non-animated) pose can be selected for display in the Shape View window. The sequence properties available to view and edit are:

Name The name of the sequence. To rename a sequence, simply edit the value and press Enter.

Source The source animation data for the sequence, for example, the path to an external DSQ file, or the name of another sequence in the shape.

Priority The priority of the sequence. This determines which sequence will take precedence when more than one sequence is attempting to control the same node.

in The first frame in the source sequence used for this sequence. Change this value to clip the start of the source sequence. This sequence will then start on the specified frame of the source regardless of what other frames may be before it in the source sequence.

out The last frame in the source sequence used for this sequence. Change this value to clip the end of the source sequence. This sequence will then end on the specified frame of the source regardless of what other frames may be after it in the source sequence.

Loop Flag indicating whether this sequence loops around when it reaches the last frame.

Blend sequence Name of the sequence to use as a reference for generating blend transforms.

Blend flag Flag indicating whether this sequence is a blend, that is, whether it can be played over top of another sequence.

Blend frame Frame in the Blend sequence to use as a reference.

Triggers The list of triggers in the sequence. Select a trigger and edit the values to modify the trigger.

Adding a sequence An important feature of the Shape Editor is the ability to add new sequences to a shape from external animation files (DSQ or DAE). This allows animations to be shared by shapes that have a common skeleton (such as character models).

To add a new sequence click the New Sequence button. If a sequence is currently selected when the button is clicked, the new sequence will use that selected sequence as its initial source for animation keyframes. You can change the Source using the dropdown menu to select a different sequence, or to Browse for an external DSQ or DAE file. If the <rootpose is selected, pressing the New Sequence button will open the Browse window automatically.

Once the sequence has been created, you can edit its properties - including the start and end frames - using the Sequence Properties panel.

COLLADA <animation_clips Currently, very few 3D modeling packages support the COLLADA <animation_clip element, which means a model with several animations will appear to have only a single sequence (or 'clip') when loaded into Torque. The Shape Editor allows you to split this single animation into multiple sequences by specifying different start and end frames for each sequence. The procedure for splitting animations is as follows:

1. Select the combined animation sequence (usually called ambient).
2. Press the New Sequence button to make a copy of this sequence, then rename the new sequence as desired.
3. Use the animation slider in the 3D view to find the desired keyframe that you want the new split sequence to start at. Press the In button to set the start frame.

4. Use the animation slider in the 3D view to find the desired keyframe that you want the new split sequence to stop at. Press the Out button to set the start frame.

Blend Animations A blend animation is special in that it stores node transforms relative to a reference keyframe, instead of absolute transforms like other animations. This allows the sequence to be played on top of another sequence without forcing the animated nodes to a particular position.

The Shape Editor allows you to set and clear the blend flag for a sequence, as well as change the reference keyframe if desired. Each of these operations requires that a valid reference sequence and reference frame number is specified.

For example, most Player characters will have a blended look animation. The animation is a blend so that the character's head can be made to look around while also doing something else (like running or swimming). To make the look animation a blend, first we set the reference sequence (e.g. root) and frame (e.g. 0), then we can set the blend flag.

Nodes Tab

The nodes tab shows the node hierarchy and various properties of the selected node. The node properties available to view and edit are:

Name The name of the node. To rename, simply edit the value and press Enter.

Parent The parent of the node in the hierarchy. A new parent can be selected from the dropdown menu if desired.

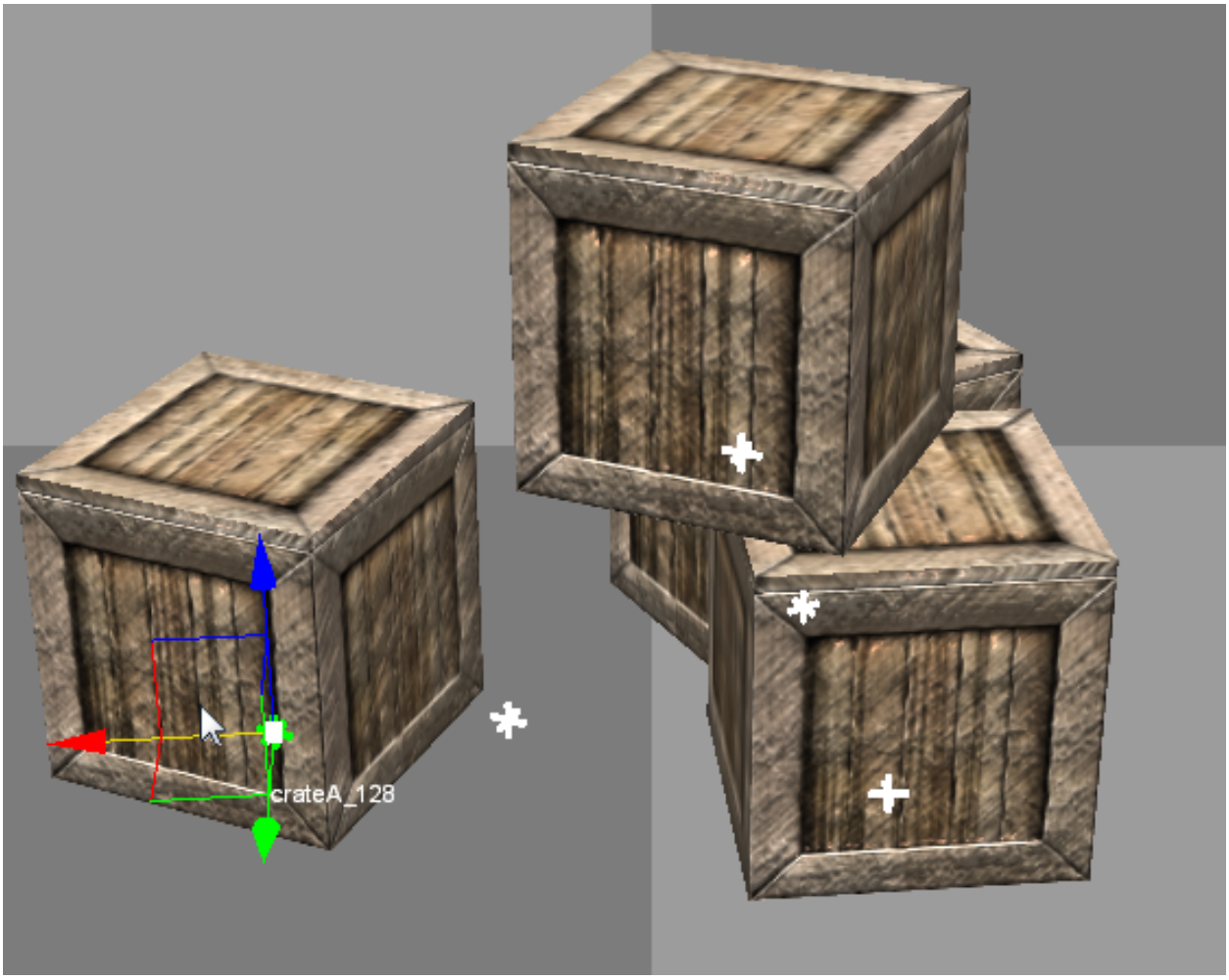
Transform The position and orientation of the node. Node transforms can be edited in either World mode (where the transform is relative to the shape origin), or Object mode (where the transform is relative to the node's parent). Node transforms can also be edited visually in the Shape View window by selecting the node and dragging the axis gizmo, similar to how object transforms are edited in the World Editor. In World mode, the gizmo uses the global X,Y,Z axes, while in Object mode, the gizmo uses the node relative X,Y,Z axes (useful for seeing which way the node points for eye or cam nodes)

Editing Nodes The Shape Editor allows shape nodes to be added, moved, renamed and deleted.

To add a node, simply press the New Node button in the top right corner of the Properties panel. If a node is currently selected, it will automatically be used as the initial parent for the new node. A new parent node can be selected using the dropdown menu. Renaming the node is as simple as typing a new name in the edit box and pressing Enter to apply the change.

There are two ways to edit node transforms: The first way is to manually edit the position and rotation values in the Node Property panel. This method is most useful when trying to set an explicit value. For example, you may require that a node be offset by exactly 2 units in the X direction from its parent node. Node transforms can be specified as either relative-to-parent (Object mode) or relative-to-origin (World mode).

The second way to edit node transforms is in the 3D Shape View. Simply select the desired node in the 3D view or in the node tree then drag the axis gizmo to the correct position and orientation.



It should be noted that the Shape Editor tool is not intended as a replacement for a fully-functional 3D modeling application, and as such, it only allows the non-animated transforms of the shape nodes to be edited. That is, the node transforms when the shape is in the root pose. You cannot use the Shape Editor tool to define new animation keyframes. For this reason, it is recommended to edit node transforms only when the <rootpose is selected in the Sequence Properties list. Node transforms can be edited when any sequence is selected, but the results may not be as expected, since animated parent nodes will affect the node transform as seen in the Shape View.

To delete a node, simply select it in the 3D Shape View or the node tree and press the Delete button in the top right corner of the Properties panel. Note that deleting a node will also delete all of its children.

Detail Tab

The Detail tab of the Properties pane lists the detail levels and associated geometry (meshes) in the shape, as well as allowing certain properties to be edited.

If the Levels checkbox is checked on the Details tab in the Advanced Properties window, then selecting a mesh or detail level in the tree will switch to that detail level in the 3D view. The bounding box for the selected object can be displayed using the Toggle Bounding Box button on the toolbar. To view all collision geometry (as wireframe) no matter which detail level is selected, use the Toggle Collision Mesh button on the toolbar.

Name (top-left field in Detail/Object Properties section) The name of the mesh or detail level. Note that an object may contain multiple level-of-detail meshes. Changing the object name will change the name of all meshes for that object.

Size (top-right field in Detail/Object Properties section) The pixel size of the mesh or detail level. Changing the size for a mesh will move it from one detail level to another (creating a new detail level if required). Changing the size of a detail level will change the size for all meshes in that detail.

Billboarding Allows a mesh to be set as a billboard.

Object Node The name of the node this object is attached to. Changing this value will change the node for all level-of-detail meshes of the object.

Import Shape into... Import geometry from another shape file. See Importing Geometry for more details.

Re-compute bounds Recalculate the shape bounding box using the current pose and detail level.

The Shape Editor also allows meshes to be hidden inside the 3D view (equivalent to the ShapeBase::setMeshHidden script method). Simply right-click a mesh in the detail tree to toggle the hidden state. Note that all detail-level meshes for that object share the same hidden state, so hiding the Head 2 mesh will also hide any other meshes for the Head object.

Importing Geometry The Shape Editor allows you to import geometry from another DAE or DTS file into the current shape via the "Import Shape into..." button. Geometry in the external file may be added to the currently selected detail level or to a new, automatically created detail level. The size of the new detail level can be edited after the geometry has been added.

The dropdown to the right of the "Import Shape into..." button has two options:

1. The current detail option is useful when combining separate files that you want to be rendered at the same detail level. For example, if a player character was split into body part models as follows:

```
player_torso.dts
player_head.dts
player_left_arm.dts
player_right_arm.dts
player_left_leg.dts
player_right_leg.dts
```

To combine the models, open player_torso.dts in the Shape Editor, switch to the Details tab then Import each of the other files into the current detail. When the shape is rendered, all body parts will be rendered together.

```
+-base01
        +-start01
            +-Torso                 Object Torso with details: 2
            +-Head                  Object Head with details: 2
            +-LeftArm               Object LeftArm with details: 2
            +-RightArm              Object RightArm with details: 2
            +-LeftLeg               Object LeftLeg with details: 2
            +-RightLeg              Object RightLeg with details: 2
```

2. The new detail option is useful when combining separate files that represent different detail levels of the same shape. For example, a vehicle model may have the following detail level files:

```
truck_lod400.dts
truck_lod200.dts
truck_lod60.dts
truck_col_lod-1.dts
```

To combine the models, open truck_lod400.dts in the Shape Editor, switch to the Details tab then Import each of the other files into new detail levels. The single truck object now has 3 visible detail levels (at pixel sizes 400, 200 and 60), and a single, invisible collision detail level (size -1).

```
+-base01
  +-start01
    +-Truck                  Object Truck with details: 400 200 60
    +-Collision              Object Collision with details: -1
```

Note that when the new detail option is selected, the Shape Editor examines the filename of the imported model to determine the detail size. If the filename ends in "_LODX" (where X is a number), the new detail level will be created with size X. The detail level size can be changed after import if needed.

Materials Tab

The Materials tab (labelled as "Mat" in the window) shows the materials specified in the shape, as well as the Material each one is mapped to.

Selecting a material while the Highlight selected Material option is set will highlight all of the primitives that use the material in the shape view. Pressing Edit the selected Material will open the Material Editor dialog, allowing you to modify the Material properties and view the results in real-time in the Shape Editor view window. Hit the Back to Previous Editor button in the upper-left corner of the Material Properties pane to return to the Shape Editor. Do not forget to save any changes you make before returning to the Shape Editor.

Advanced Properties Window

The Advanced Properties Window allows you to further change the settings of the model loaded in the shape editor.

Details Tab

The detail size and mesh characteristics for each LOD need to be carefully determined in order to reduce the visual artifacts associated with switching and rendering detail levels. The Details Tab of the Advanced Properties window provides a convenient way to view and edit detail levels without having to re-export the model. It also allows non-rendered collision and LOS-collision detail levels to be visualised. The detail level properties available to view and edit are:

Levels When set, the current detail level is selected by moving the slider. When unset, the current detail level is selected based on the camera distance, in the same way as LOD is handled in-game.

Current DL The index of the currently selected detail level is shown to the right of the slider track.

Polys The number of polygons (triangles) in the current detail level.

Size The size (in pixels) above which the current detail level will be selected. This value can be edited to change the size of the current detail level (remember to press Return after editing the value to apply the change).

Pixels The current size (in pixels) of the shape. This value is an approximation based on the shape bounding box, viewport height, and camera distance.

Distance The distance from the shape origin to the camera.

Materials The number of different materials used by all meshes at the current detail level.

Bones The number of bones used by all skinned meshes at the current detail level. Non-skinned meshes will display 0 for this value.

Primitives The total number of primitives (triangle lists, strips or fans) in all meshes at the current detail level. This is the minimum number of draw calls that will be executed for this detail level.

Weights The number of vertex weights used by all skinned meshes at the current detail level. Non-skinned meshes will display 0 for this value.

Col Meshes The total number of collision meshes in this shape.

Col Polys The total number of polygons (triangles) in all collision meshes in this shape.

The Details Tab of the Advanced Properties window allows imposter detail levels to be added and edited. Imposters are a series of snapshots of the object from various camera angles which are rendered instead of the object when this detail level is selected. An imposter detail level is usually the last visible detail level (smallest positive size value).

Mounting Tab

The Mounting Tab of the Advanced Properties window allows you to attach other models to the main shape to visualise how they would look in-game, or to fine tune the position and rotation of mount nodes. When a model is mounted, it inherits the position and rotation of the node it is mounted to and will animate along with it. Press the Mount New Shape or Delete Mounted Shape buttons to add or remove mounted models respectively. The following properties of the selected mount can be modified:

Shape DTS or DAE model file to mount.

Node Node (on the main shape) to mount to. Only nodes that follow the mountX and hubX naming conventions will appear here.

Type

- Object: Mount the model as a SceneObject. The model's origin is attached to the selected mount node. This is equivalent to mounting the object using the following script call:

```
%obj.mountObject(%obj2, 0);
```

- Image: Mount the model as a ShapeBaseImage. The model's mountPoint node (or origin if not present) is attached to the selected mount node. This is equivalent to mounting the object using the following script call:>

  %obj.mountImage(%image, 0);

- Wheel: Mount the model as a WheeledVehicle tire. The mounted shape's origin is attached to the selected mount node, and it is rotated to face the right way (whether on the left or right side of the vehicle). This is equivalent to mounting the object using the following script call:

```
%car.setWheelTire(0, %tire);
```

Sequence Select a sequence for the mounted shape to play. Playback can be controlled using the slider and play/pause button to the right of the sequence dropdown box.

Threads Tab

The Threads Tab of the Advanced Properties window allows you to set up threads to play multiple sequences simultaneously, and to view transitions between sequences. A set of animation sequence playback controls that mirror the main animation controls are provided as a convenience so you don't have to mouse too far to test out a new thread. The mini-timeline slider is also used to indicate sequence transition information.

Thread The index of the thread. Press the Add New Thread or Delete Selected Thread buttons to add or remove threads respectively. If the shape contains any sequences, there will always be at least one thread (index 0) defined.

Sequence Select the sequence for this thread to play. Changing the selected sequence while the thread is playing (and transitions are enabled) will cause a transition to the new sequence.

Transition flag If enabled, changing the selected sequence for the thread will cause a transition from the current pose to the target pose. During the transition period, node transforms are smoothly interpolated towards the target pose. If transitions are disabled, changing the selected sequence for the thread will switch node transforms to the new sequence immediately.

Transition lasts Transition duration in seconds. The default for Torque 3D is 0.5.

Transition to Selects the start frame in the target sequence; the target sequence begins playing from this point. When slider position is selected the target sequence will play from wherever the mini-timeline slider has been set. Torque 3D defaults to having the new sequence start at position 0.0 so it is likely that you'll want to keep the mini-timeline slider all the way to the left when in this mode. When synched position is selected, the new sequence will start playing at the same position along the timeline as the currently playing sequence. While in this mode, the mini-timeline slider will change from yellow to red during the transition period.

Target anim Controls whether the target sequence plays during the transition period. When plays during transition is selected the target sequence will play during the transition; node transforms will be interpolated towards the changing target pose. When pauses during transition is selected the target sequence will not play during the transition, but will start once the transition has ended. Node transforms will be interpolated towards the initial target sequence frame.

Collision Tab

The Shape Editor can auto-fit geometry to a part or the whole of the shape for use in collision checking. Each time the settings are changed, the geometry in detail size -1 is replaced with the new auto-fit geometry. The node Col-1 (and any child nodes) may also be modified.

Fit Type The type of mesh to auto-fit for this collision detail (see table below for details).

Fit Target The geometry used to generate the auto-fit mesh. The target is either 'Bounds' (fit to the whole shape) or one of the shape sub-objects.

Max Depth For convex hull auto-fit meshes, this specifies the maximum decomposition recursion depth. Increase this value to increase the number of potential hulls generated.

Merge % For convex hull auto-fit meshes, this specifies the volume percentage used to merge hulls together. Increase this value to make merging less likely, and thus increase the number of final hulls.

Concavity % For convex hull auto-fit meshes, this specifies the volume percentage used to detect concavity. Decrease this value to be more sensitive to concavity (and thus more likely to split a mesh).

Max Verts For convex hull auto-fit meshes, this specifies the maximum number of vertices per hull. Increase this value to produce more complex (and CPU expensive) hulls.

Box % For convex hull auto-fit meshes, this specifies the maximum volume error below which a hull may be converted to a box. Increase this value to allow more hulls to be converted to boxes.

Sphere % For convex hull auto-fit meshes, this specifies the maximum volume error below which a hull may be converted to a sphere. Increase this value to allow more hulls to be converted to spheres.

Capsule % For convex hull auto-fit meshes, this specifies the maximum volume error below which a hull may be converted to a capsule. Increase this value to allow more hulls to be converted to capsules.

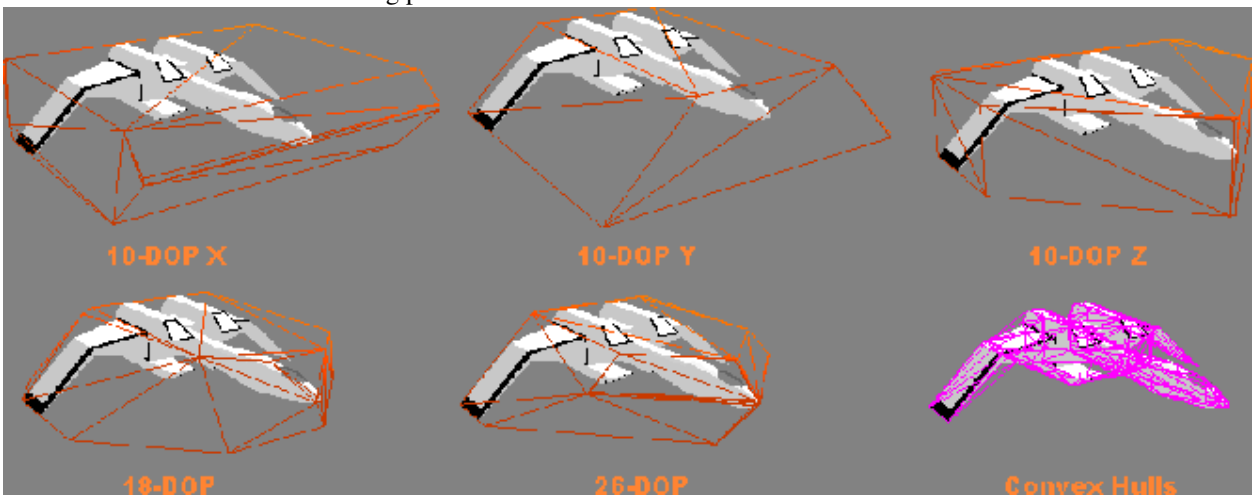
Update Hulls Re-compute convex hulls using the current parameters.

Revert Changes Revert convex hull parameters to the values used for the most recent hull update.

The following types of geometry can be generated. The Box, Sphere and Capsule types are generally the most CPU efficient, and are converted to true collision primitives when the shape is loaded. The other types are treated as convex triangular meshes - the more triangles, the more expensive it is to test for collision against the mesh.

Type	Desciption
Box	Minimum extent object aligned box
Sphere	Minimum radius sphere that encloses the target
Capsule	Minimum radius/height capsule that encloses the target
10-DOP	Axis-aligned box with four edges bevelled; you can choose X, Y or Z aligned edges
18-DOP	Axis-aligned box with all edges bevelled
26-DOP	Axis-aligned box with all edges and corners bevelled
Convex Hull	Set of convex hulls

The k-DOP (K Discrete Oriented Polytope) types push 'k' axis-aligned planes as close to the mesh as possible, then form a convex hull from the resulting points as shown below.



The Convex Hull fit type performs a convex decomposition of the target geometry to generate a set of convex hulls. The basic algorithm is described here. For each hull that is produced, the hull volume is compared to the volume of a box, sphere and capsule that would enclose the hull. The hull is replaced with the primitive type that is closest in volume to the hull with volume % difference less than Box, Sphere or Capsule % respectively. If none of the primitive volumes are less than their respective error setting, the hull will be retained as a triangular mesh.

Shape Editor Settings

The Shape Editor settings dialog can be accessed from the main menu by selecting Edit Editor Settings, and allows the appearance of the editor to be customized. These settings are persistent and will be automatically saved and restored between sessions.

Saving Changes

The Shape Editor does not modify the DTS or COLLADA asset file directly. Instead, changes made in the editor are saved to a TSShapeConstructor object in a separate TorqueScript file. This file is automatically read by Torque before the asset is loaded, meaning you can safely re-export the DTS or COLLADA model without overwriting changes made in the Shape Editor tool. The change set will be re-applied to the shape when it is next loaded by Torque.

If needed, you can also re-edit the generated TSShapeConstructor object, either manually with a text editor, or by using the Shape Editor tool again.

To save changes to the current shape, simply press the save button in the top right corner of the Properties window. The script filename is the same as the DTS or COLLADA asset filename, only with a .cs extension. For example, saving changes to ForgeSoldier.dts would save to the file ForgeSoldier.cs in the same folder.

The Shape Editor TSShapeConstructor object may also be accessed directly from the console. For example:

- Dump the shape hierarchy to the console (handy for debugging shape issues):

```
ShapeEditor.shape.dumpShape();
```

- Save the modified shape to DTS (instead of saving the change-set to a TSShapeConstructor script):

```
ShapeEditor.shape.saveShape("myShape.dts");
```

- Set ground transform information (not yet available in Shape Editor UI):

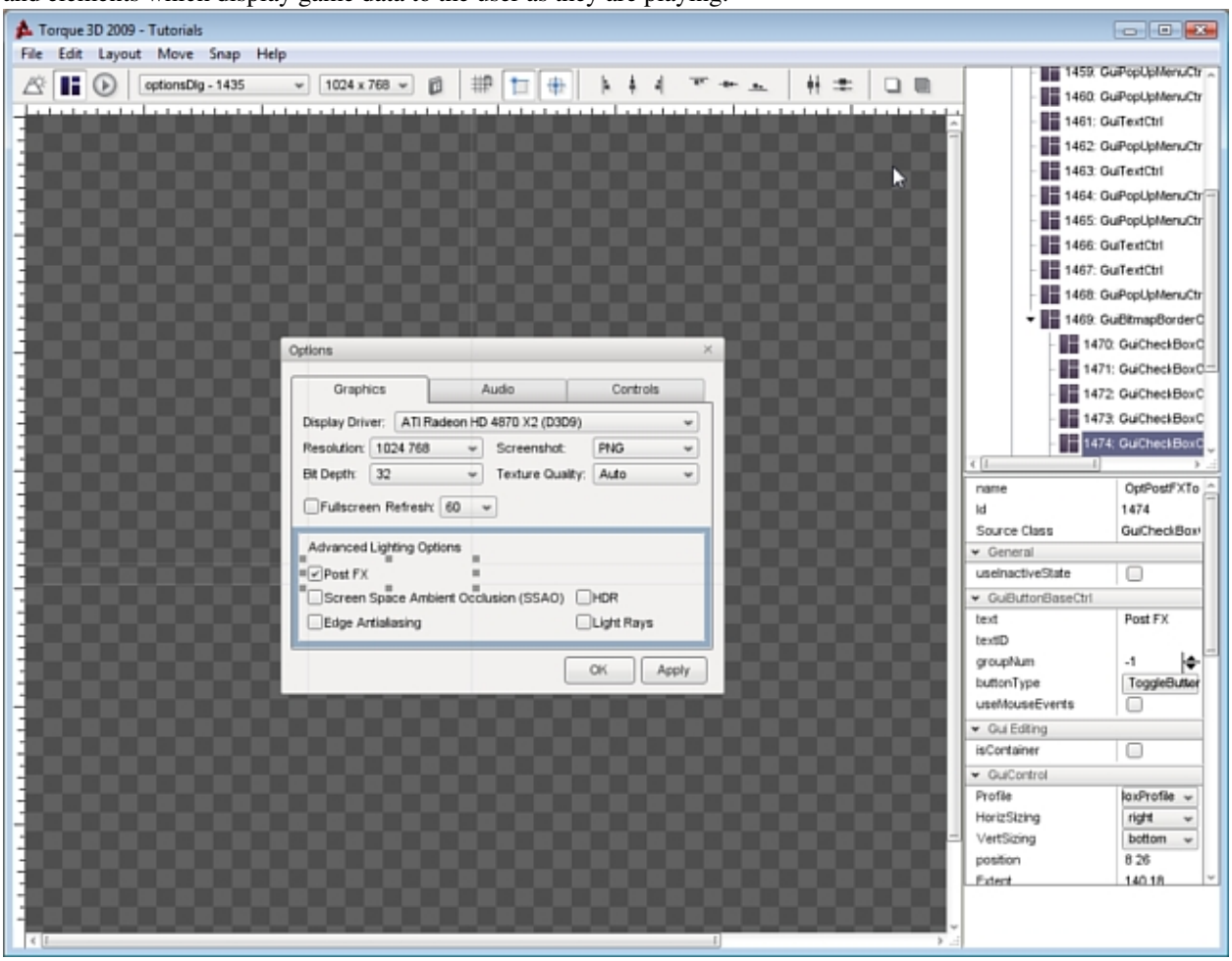
```
ShapeEditor.shape.setSequenceGroundSpeed("run", "0 4 0", "0 0 0");
```

CHAPTER 3

GUI Editor

3.1 Overview of GUI Editor

"GUI" stands for Graphical User Interface. It is the summation of all the controls (windows, buttons, text fields, etc.) that are used to interact with a game and its settings. Most interfaces in games consist of buttons to launch or join a game session, editing devices to change user preferences, options to change screen resolutions and rendering options, and elements which display game data to the user as they are playing.



GUI creation and design is extremely important to game development. Many decent games have been crippled by

inaccessible GUIs, which is why having a built in GUI editor can be a blessing. The Torque 3D editor provides drag and drop functionality, with minimal fill in the blank requirements.

Torque 3D features a WYSIWYG GUI Editor, which allows you to create, edit, and test your GUI in game with maximum fidelity. 90% of your GUI creation can be done in the editor, leaving 10% for scripting advanced functionality.

GUIs are saved as a script (.gui), which allows you to further tweak values using your favorite text editor. Additionally, you can declare variables and define functions at the end of a GUI script, which will not be written over when modifying the GUI using Torques editor.

Multiple controls which can be combined to make up a single interface. Each control is contained in a single structure, which can be embedded into other GUI elements to form a tree. The following is an example of a GUI control which displays a picture:

```
// Bitmap GUI control
new GuiBitmapCtrl() {
   profile = "GuiDefaultProfile";
   horizSizing = "width";
   vertSizing = "height";
   position = "8 8";
   extent = "384 24";
   minExtent = "8 8";
   visible = "1";
   helpTag = "0";
   bitmap = "art/gui/images/swarmer.png";
   wrap = "0";
};
```

Once the above GUI is active in your interface, it will display the following:

3.2 GUI Editor Interface

The main GUI Editor view consists of 5 primary sections:

File Menu Found at the very top, this is where you will find various menus that controls global functionality of the editor, such as creating/saving GUI files, manually locking, selecting, and aligning controls, toggle snapping, and so on.

The Toolbar Located just below the File Menu, this bar contains shortcuts to the GUI Selector, resolution adjuster, and common positioning actions (nudge, align, etc).

Control Palette The Control Palette contains all of the controls you can add to your current GUI. You can click a control in the list or manually drag it to a position in the view to add it to the scene.

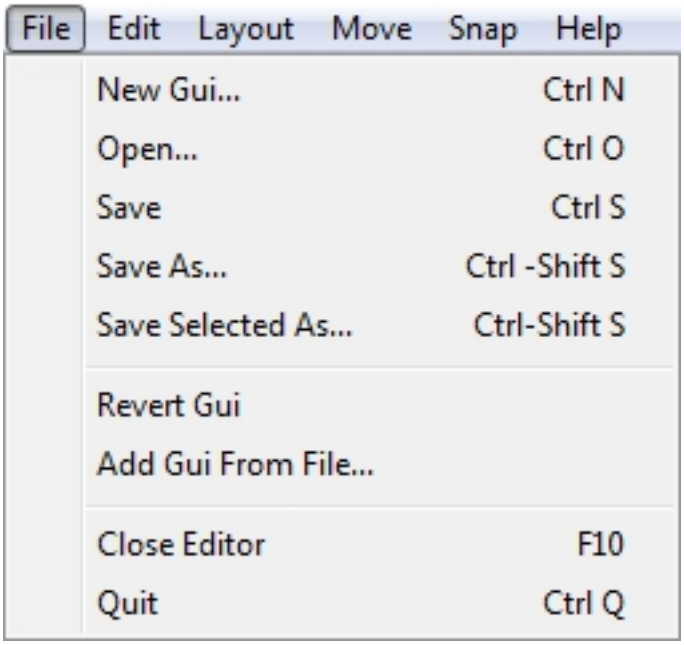
GUI Tree Panel This panel, located on the far right, contains all of the controls that make up your current GUI. They are listed in hierarchical tree, which is sorted by oldest to most recent (top to bottom) and parenting (described

later).

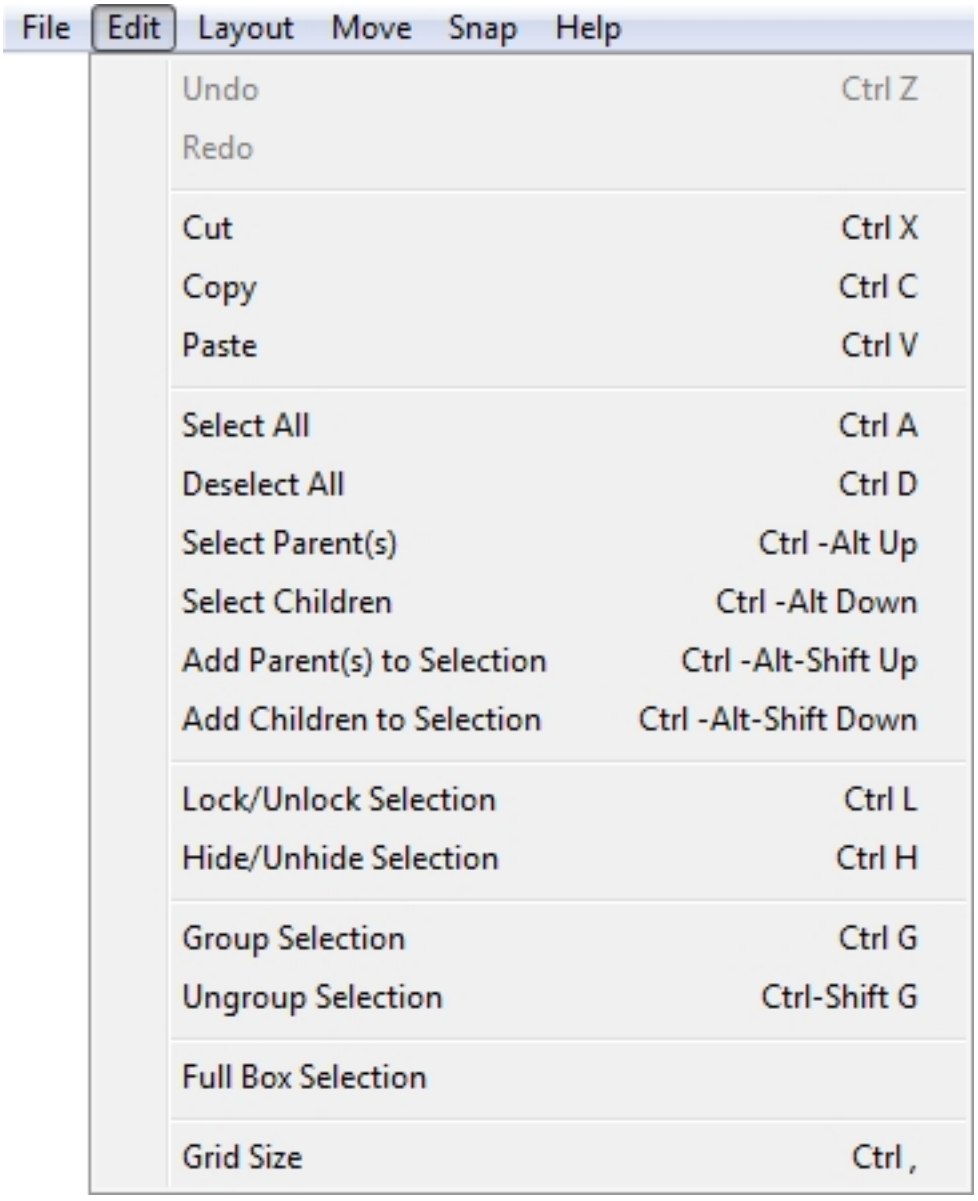
GUI Inspector Panel This panel, found directly below the GUI Tree Panel, is populated with all the properties that make up the currently selected GUI control. Most of your field editing will be performed here.

3.2.1 File Menu

File Menu allows you to create, save, and open GUI files. You can also revert GUIs or load from file.

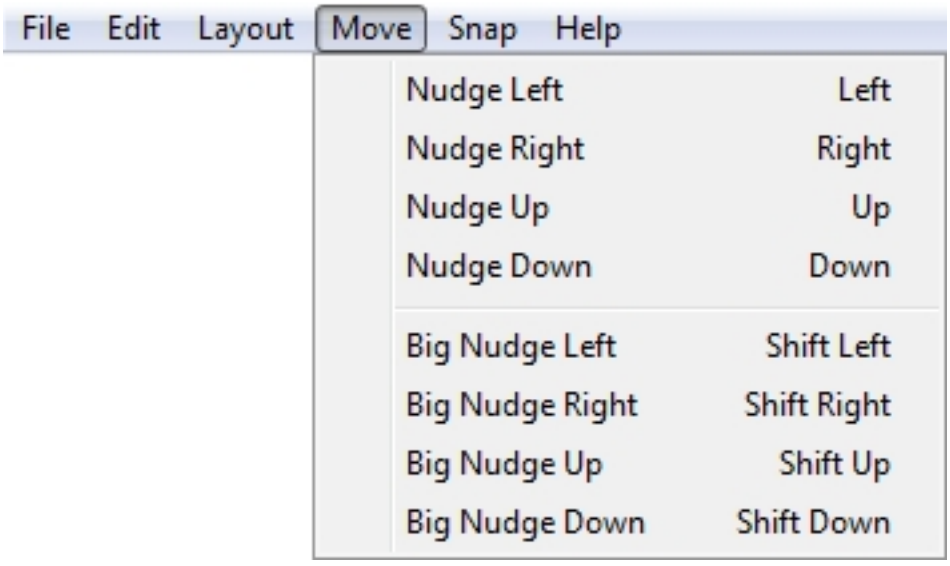


The Edit Menu controls various editor actions, such as undo and redo. The second function allows you to cut/copy/paste/delete objects you have selected. Finally, this is the menu that allows you to perform selection and group actions, such as toggling visibility and locking.

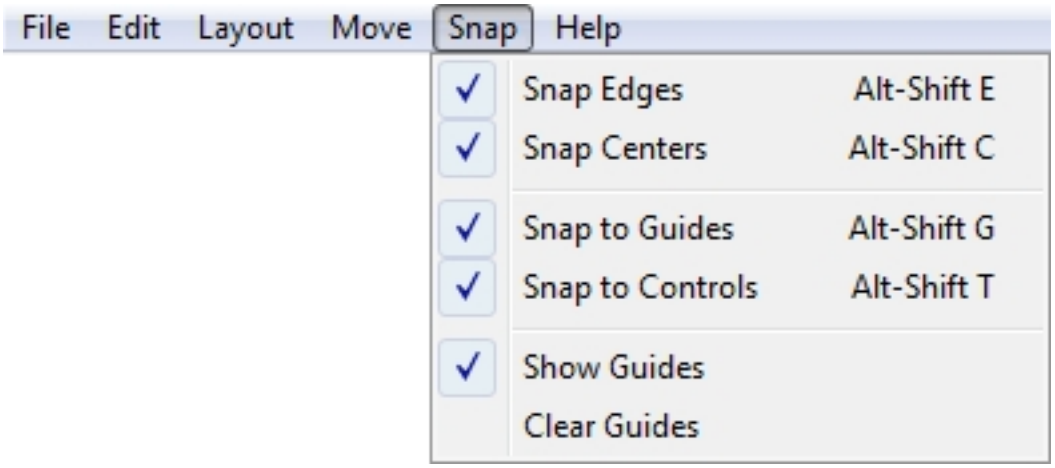


The Layout Menu contains actions that makes it easy for you align your GUI controls for a clean and neat appearance. This is very useful for a complex interface with multiple controls that stack vertically or horizontally.

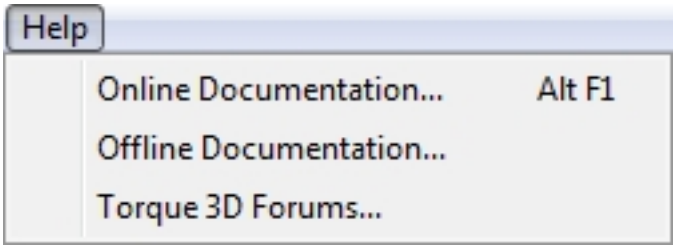
When you need to perform very subtle and precise movements on a GUI control, you can use the actions listed in the Move Menu. Each nudge is assigned a shortcut, so using this menu is optional.



When dragging GUI controls with your mouse, using the Snap Menu toggles will cause your mouse to immediately jump to specific points (depending on the toggle).



Contains shortcuts to documentation and forums for Torque 3D.



3.2.2 Tool Bar

The most useful and preferred shortcuts for quick edits can be found in the Tool Bar. While most of your control properties will be edited in the Inspector, you can use the Tool Bar to perform quick positioning actions and testing.

The first three icons toggle the editors, and are always available. The left most icon (looks like a mountain) toggles the World Editor. The next one (boxes) toggles the GUI Editor. The Play icon will exit the editors and let you play through the game.

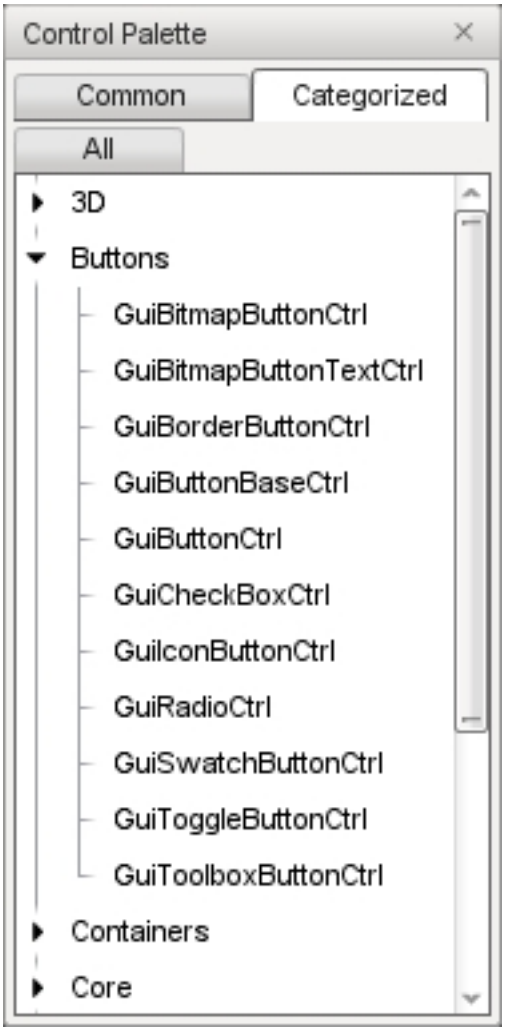
Next to the editor toggles, you will find three extremely important settings. The first two are drop down lists and the last one is a button toggle. These determine what you are editing and at what resolution. The first drop down is a list of every single GUI available to edit, including new ones you just created. You can jump to an individual GUI at anytime, which can be useful if you are editing multiple GUIs that which work together. The second drop down list contains three different resolutions you can build your GUI in.

The button toggles the Control Palette, which is explained in the next section. The next set of icons allow you to toggle the most commonly used and important settings for snapping. After the snapping icons, several shortcuts are available toggle the alignment of controls. The next two icons, which look like multiple boxes attached to lines, are used when you have multiple GUI controls selected. These distribution toggles will equally space the GUIs you currently have selected. The final two icons in the the Tool Bar can move the currently selected GUI between layers. The first button will move the selected GUI ahead in a layer, bringing it closer to view. The second button will start shoving your GUI behind others, obscuring it from view.

3.2.3 Control Palette

The Control Palette contains all of the controls you can add to your current GUI. You can click a control in the list or manually drag it to a position in the view to add it to the scene. There are ways to view the list of available controls,

depending on which tab you are using.

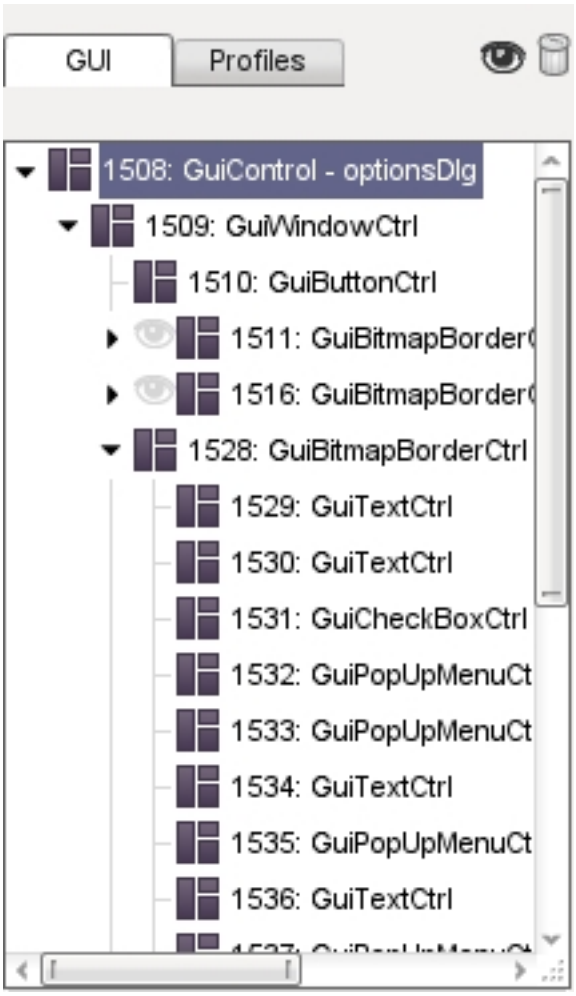


When you first toggle the Control Palette, you will see a list of the most commonly used controls. There are quite a few controls hidden, which are mainly used to create the Torque 3D editors. These controls are not typically used in games, so they have been hidden. You can click the All button to see every GUI control the engine contains. When you click on the Categorized tab, you can get list of all the GUI controls based on their functionality. The categories are straight forward and should be an excellent way to get to the exact controls you need to build your interface. To see what a category contains, just click on one of the arrows or text to expand it:

To add a control, locate it in the Palette's list. Next, click on the control and drag it to your main view using a mouse. When you let go of your mouse button, you new control will anchor to the view and become your current selection.

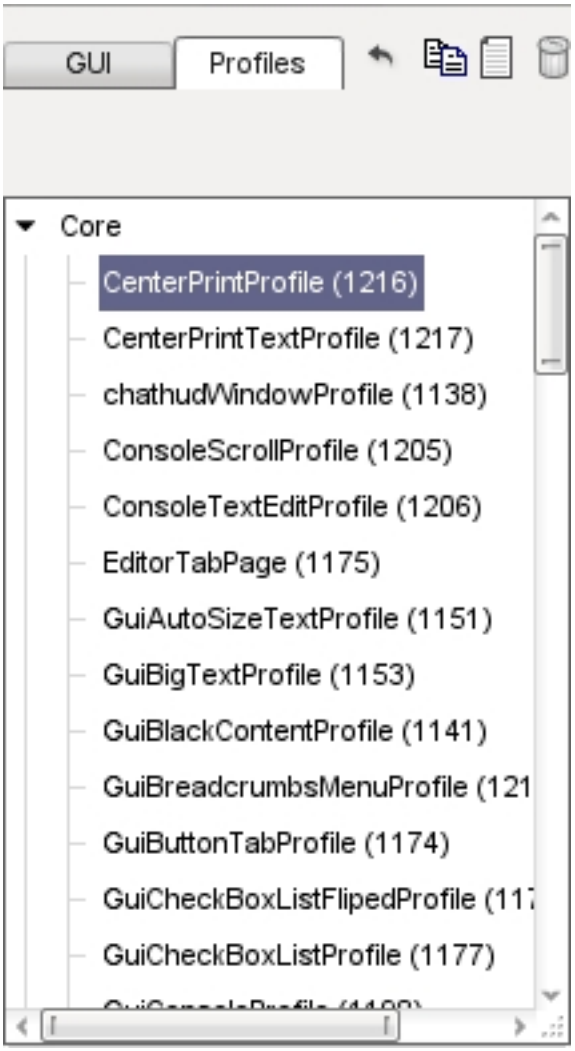
3.2.4 GUI Tree View

Every control added to your current GUI is kept in a sorted list. To view this list, go to the panel on the far right and click the "GUI" tab. This will list all of your controls in the order they were added, the most recent at the bottom of the list. Each control has a unique ID, and can be given a name.



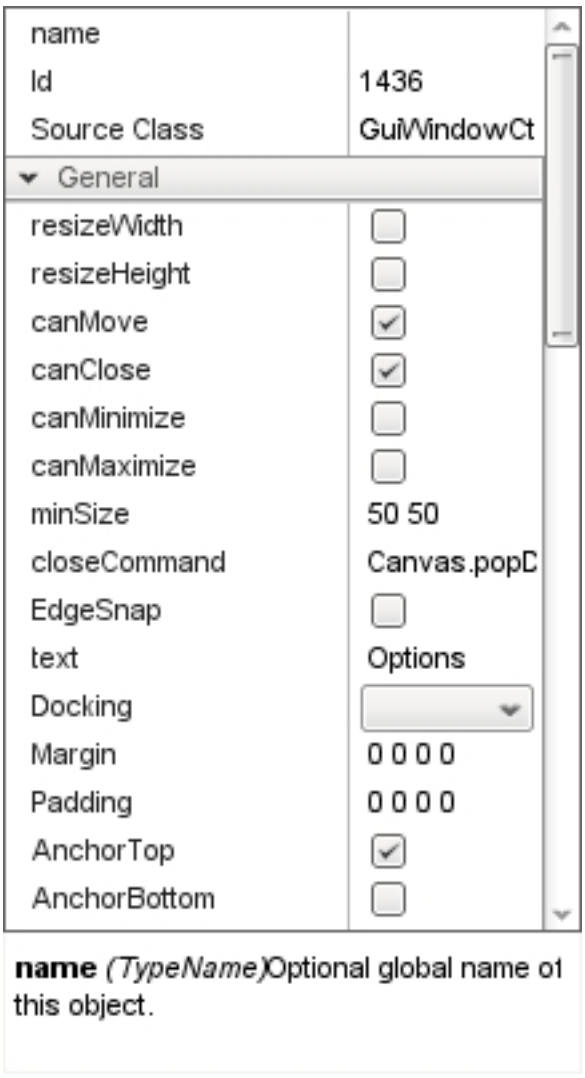
3.2.5 Profile Editor

In the same panel as the GUI Tree View, there is a tab called "Profiles." Clicking this tab will present you with a list of all the GUI profiles currently loaded by your game. GUI profiles contain data that personalizes your controls. This will allow you tailor an interface unique to your game.



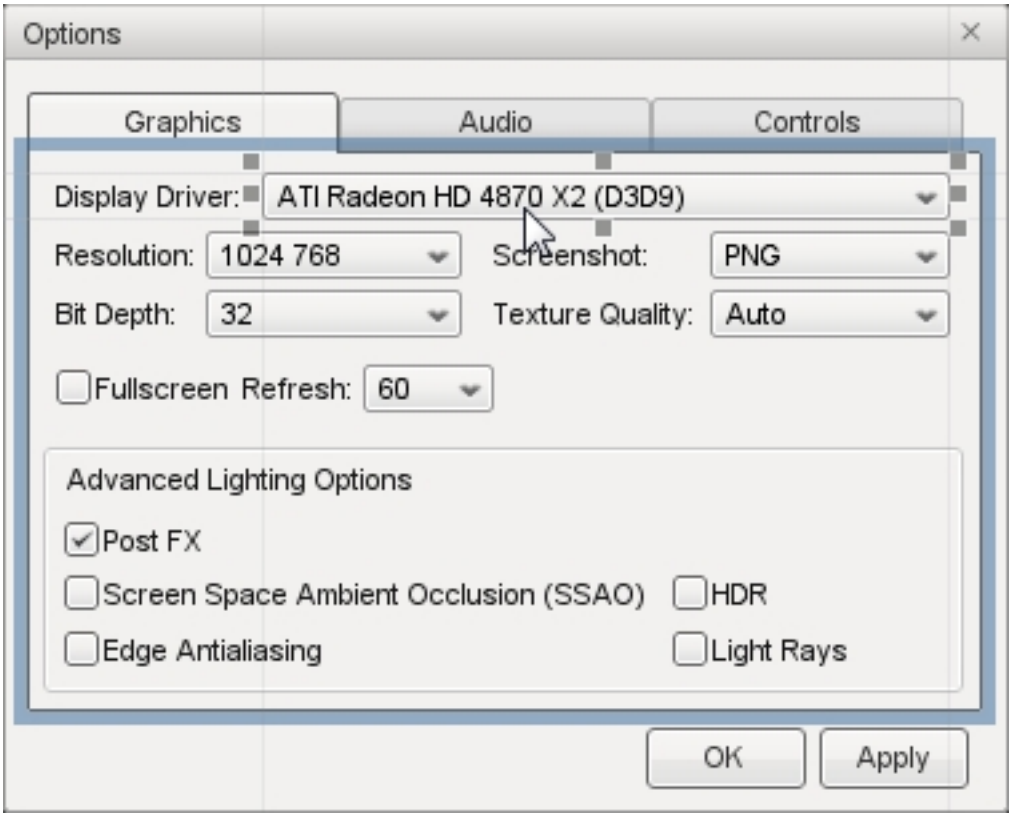
3.2.6 GUI Inspector

When you have a control or profile selected, the GUI inspector will be populated with the properties that make up the selection. These are the values that play an important role in assigning functionality to your GUI. Most of your editing will occur here.



3.2.7 Selection and Parenting

The last portion of the interface is how controls are selected. The following image shows the stock options GUI that ships with Torque 3D. This consists of dozens of controls working together to make up the audio and video options:



In the above image, I have selected the list box control that shows the display driver for a video card. The current selection is marked by six boxes surrounding the corners of the control, and several subtle lines. However, you should notice there is a large blue box surrounding multiple controls.

The large blue box shows the Parent control. When a control is the "parent", it can contain multiple sub-controls. The "children" controls will now adhere to the same behaviors as the parent control. For example, if the parent control is set to invisible, the children controls will become invisible as well. If the parent is moved, all the children controls will move with it.

CHAPTER 4

Artists Guide

4.1 Overview of Content Pipeline

The main tool for importing and manage content in Torque 3D is the World Editor. However the World Editor is not a tool for creating game objects. Objects must be created using applications appropriate for the object type.

4.1.1 Importing 3D Models

Torque 3D uses two different formats for 3D geometry and animation data: *Collada* and *DTS*.

Collada is intended to be the primary model format during development of a Torque 3D game, as there are Collada importers and exporters for almost every 3D modeling application around. Collada is a format for interchanging models between digital content creation applications. Torque 3D can load geometry, material and animation data directly from Collada DAE files and is based on version 1.4.1 of the Collada specification. Collada is an XML based file format, meaning DAE files may be opened, viewed and edited in any text editor.

The Torque 3D supports all of the Collada polygonal geometry elements: <triangles>, <tristrips>, <trifans>, <polygons> and <polylist>, with non-triangular polygons automatically converted to triangles during loading. Note that if polygons are non-planar, this may introduce seams on the model, so the best option is to triangulate in the modeling application prior to exporting.

Many 3D modeling applications come with built-in Collada import and export functionality, but third party plugins are also available that may be more stable, updated more frequently and give better results. In particular, OpenCOLLADA is recommended for both Autodesk 3ds Max and Maya.

While DTS is a format internal engine format intended for the models that ship with the released version of a game. It is an optimized, binary format that loads much faster than Collada and also provides a small measure of protection for art assets.

4.1.2 Shipping 3D Models

The normal workflow is to use Collada during development, then make sure all models are converted to DTS for a shipping release build. There are several different approaches available to achieve that.

It can be accomplished by either use the cached DTS files that Torque 3D saves in the same folder as the Collada file during import. Every time the DAE file would be loaded, Torque 3D first checks if there is a newer cached.dts file in the same folder and if so, loads that instead. For simple models, this means you can simply strip the DAEs from the released version of the game, leaving only the cached.dts files. All datablocks and mission files still refer to the DAE model, but Torque will automatically load the cached.dts in its place.

Note: The cached.dts file represents the converted DAE model only - changes applied using a TSShapeConstructor script are only made to the in-memory shape and are not included in this file.

Anther way is to use the dae2dts tool as a part of the build process to convert Collada files into DTS. If using this approach, datablocks and mission files should refer to the coverted DTS output file, not the original DAE file. The dae2dts tool bakes any model transformations made using TSShapeConstructor into the final DTS model, so make sure that the TSShapeConstructor script used with dae2dts is not run when loading the DTS output file into Torque 3D (by making the filenames different, or keeping the output DTS file in a different folder) or the changes will be applied twice!

Of course, there is no reason you could not use one TSShapeConstructor script with dae2dts to bake changes, then use another TSShapeConstructor script when loading the baked DTS file into Torque 3D to apply dynamic changes (like auto-loading all sequence files within a folder for example).

There also exist some modeling applications with DTS file export support. However, those are no longer recommended as they place limitations on the exported files while the Torque 3D Collada importer is much more flexible.

4.1.3 Coordinate System

Torque uses the same coordinate system as 3ds Max where characters and vehicles should are facing the +Y axis.

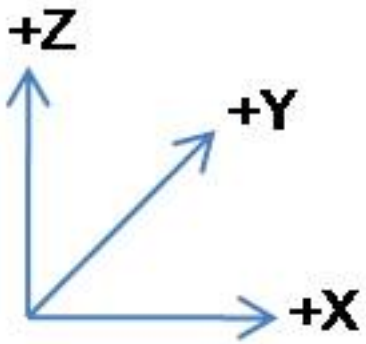


Fig. 4.1: The Torque 3D coordinate system.

Thus the coordinate system is equivalent to Z_UP in Collada and the importer will automatically convert models using X_UP or Y_UP to this coordinate system. However, some Collada exporters may generate models with a wrong or missing <up_axis> element (Z_UP is assumed if <up_axis> is not specified). In this case, the Collada import dialog can be used to override the value.

4.1.4 Units

When exporting to Collada, you are free to work in whatever units you like. They will be scaled appropriately when importing the model into Torque 3D.

When creating a model, it makes sense to work in units appropriate to the type of object being modeled. For example, it may be convenient to model a building in meters (or feet), but a small object like a pen would be better modeled in cm (or inches). Normally the modeler would be forced to choose a single unit for both objects, and one object would end up having awkward measurements; like a building that is 2000 units high, or a pen that is only 0.14 units long.

Collada provides a mechanism to specify the units used when modeling the object. This is very important, since it allows a set of objects modeled in completely different units to be imported into Torque 3D with the correct scale relative to each other. This is done using the Collada <unit> element, which appears near the top of the .dae file and specifies the units-per-meter used for all positional values in the file such as vertex and node positions, translation animations etc. Torque 3D uses this value as a global scale factor which is applied to the entire model on import.

Many modeling applications allow the user to specify the units to be used, or alternatively, the option may be available when exporting the model to Collada.

It is up to the modeler to ensure that models are created with appropriate units in mind. For example, if a chair was modeled at 5 units high in the modeling application (simply because that was a convenient value for the modeler), then exported to DAE with units set to feet, then the chair would appear to be the equivalent of 5 feet high in Torque 3D! The Torque 3D Collada import dialog allows you to override the <unit> scale specified in the DAE file, for cases like this when the modeler has not taken units into account.

4.1.5 Texture Files

Torque 3D supports several texture file formats: BMP, GIF, JEPG, JPG, JNG, MNG, PNG, TGA, DDS. Note that texture dimensions should be powers of 2 wherever possible such as 16, 32, 64, 128, 256, so forth, although they need not be square. Some older hardware is unable to process non-power-of-2 textures at all, and even modern hardware will pad such textures up to the next largest power of 2 when loaded which wastes VRAM.

4.1.6 Normal/Bump Maps

Torque 3D uses DirectX style normal maps, where the green (Y) channel contains the DOWN vector. This is opposite to the system OpenGL uses which has the UP vector in the green channel. If your normal maps appear backwards in Torque 3D, you may need to invert the green channel manually.

4.2 Rigging a Player Character

The purpose of this guide is to show how to rig and export a character for use in Torque 3D. This guide assumes that you are creating a new rig and animations from scratch and are not reusing any already existing rig or library of animations.

Already created animations will only work when applied to the rig they where created for. If you for example want to reuse the animation sequences that ships with the Torque 3D demo character, then you need to set up the rig for your new character exactly the same. However, this guide will not cover that specific case.

Deciding what rigs are necessary and setting them up should typically be done during technical pre-production of your game. Once it is done they should not be changed as that would mean re-targeting or in worst case re-creating all your animations.

How many and what type of rigs you need depend on your project. Are all your characters bipedal? Or do you also need to do quadrupedal animation as well? Will you use the same bipedal rig for all characters, player as well as all non-player characters? In some games you want a rig of higher fidelity for the hero player character, simpler rigs for the non-player characters and possibly a couple of special biped rigs for some monsters.

However, each rig you decide to create need its own library of animation sequences. Thus you can not share animations between say the hero rig and the non-player rig unless you set them up very carefully. For example you can set up the hero rig to be the exact same rig as the non-player but with some additional joints for any separate and additional animation.

With sharing animations in mind you should carefully name the rig and its components so that they apear logical in all skinned mesh files as well as sequence animation files.

4.2.1 Assets

Your game shapes should live in a directory somewhere under the *art/shapes* directory. You should create a new directory for each character. It will normaly hold the following files:

materials.cs All the material definitions for the character.

character.cs The TSShapeConstructor definition for the character.

character.dae Collada file with the characters rigged and skinned mesh.

character_d.dds The diffuse texture map for the character.

character_n.dds The normal map for the character.

character_s.dds The specular map for the character.

The exact naming convention of the mesh and texture files are entierly up to you as you export them from your 3D and 2D applications. While the script files are automatically generated by the Shape Editor upon importing the character mesh.

It is also recommended that you create a directory for all the animation sequences somewhere under the *art* directory. Each animation sequence should be stored in a separate DAE file. It's important to note that the characters skinned mesh need to use the same rig as the animation sequence files to be able to play them back.

4.2.2 Hierarchy

Torque expects for each type of shape a number of specific nodes to be present depending on the shape class. The Player class require thus the following nodes acting as mount points:

cam Used as 3rd person camera position.

eye Used as 1st person camera position.

ear Where the SFX listener is mounted.

mount0-31 You can have up to 32 mount nodes that is used to mount objects to the character.

Further the Player class expects a number of nodes acting as bones in an armature. The following nodes are used by the Player class for look, aim and recoil animations if no special sequences has specified by you for those actions (see Sequences). The naming convention for those nodes were taken from 3ds Max Biped and are therefore already present if the character is rigged in 3ds Max:

Bip01_Pelvis Hierarchial node (usually a bone) that act as the root.

Bip01_Spine Hierarchial node (usually a bone) whose parent is Bip01_Pelvis.

Bip01_Spine1 Hierarchial node (usually a bone) whose parent is Bip01_Spine.

Bip01_Spine2 Hierarchial node (usually a bone) whose parent is Bip01_Spine1.

Bip01_Neck Hierarchial node (usually a bone) whose parent is Bip01_Spine2.

Bip01_Head Hierarchial node (usually a bone) whose parent is Bip01_Neck.

The hierarchy should also contain axis-aligned bounding box mesh named *bounds* that fits around the shape at the root level that is used to define the shapes origin, determine which shape level to render and define the speed that the shape is intended to be moving at.

Lastly the hierarchy should contain one or more skinned meshes of the actual character shape. The number of meshes depend on the number of detail levels used. How to name the meshes is up to you, as long as you put the detail size as a number at the end of the mesh name.

While most nodes are optional, the shape will still load and run without a particular node or sequence, the object may not perform correctly in-game. However, a minimal rig for the Player class could look like this:

```
+ Bip01
| + Bip01_Pelvis
| | + Bip01_Spine
| |   + Bip01_Spine1
| |     + Bip01_Spine2
| |       + Bip01_Neck
| |         + Bip01_Head
| + cam
| + eye
| + ear
| + mesh0
+ bounds
```

Nodes for arms should normally be connected to the Bip01_Neck node and legs to the Bip01_Spine node. It is also recommended to add meshes of several detail levels, depending on the complexity of your mesh.

4.2.3 Level of detail

Level-of-Detail (LOD) is an extremely important concept to master in order to produce a great looking game that plays smoothly on low/mid-range hardware. Essentially, it involves rendering successively less complex versions of a shape in order to improve performance.

The metric used to control LOD is the estimated size in pixels of the shapes bounding box on screen. As the shape gets further from the camera it will become smaller on screen, and a simpler version of the mesh may be rendered without loss of fidelity. Before rendering the shape, Torque estimates how large it would appear on-screen and selects the mesh, or meshes, of the appropriate detail level to be rendered.

Only a mesh with a detail level equal or higher to the estimated size will be considered. Note that detail levels with negative sizes will never be chosen for rendering. Once the estimated size is less than the smallest positive detail size, no geometry will be rendered for the shape. You can force a shape to always render something by making the smallest positive detail level have a size of zero.

The level of detail size is expressed as a number put at the end of the mesh name. The LOD type should be set to "TrailingNumber" in the Shape Editor when you import the shape.

4.2.4 Bounding box

Every shape includes an axis-aligned bounding box. This box appears around the shape when it is selected in the World Editor, and can be used for simple collision detection or mouse-hit picking. The bounding box is also used to determine which shape detail level to render and optionally to define the speed that the shape is intended to be moving at. The size of the bounding box is not fixed to the shape geometry. The modeler is free to define a custom bounding box extent for an object. This is done prior to export from the 3D application by creating a cube mesh called *bounds* with the appropriate dimensions.

If the exported DAE file does not contain a root level node called bounds with geometry attached to it, the Collada importer will automatically calculate a bounding box that encloses all of the geometry in the scene. For animated models, only the root (non-animated) pose is considered, but a walking character animation may move the feet or arms of the model outside the box containing the shape in its root pose, so use a custom bounding box to explicitly specify the bounding box extents.

4.2.5 Ground transform

Animation sequences that move the character should also include a ground transform. This tells the engine how fast the character would move along the ground when the animation is played back at normal speed. In the case of a Player object, this allows Torque to scale the animation playback speed to match the in-game speed that the Player is moving. For example, if the model was animated such that it would normally move at 3 units per second, but in-game was moving at 6 units per second, then the animation can be played back at double speed so the feet do not look like they are skating along the ground. Another use for ground transforms is to automatically switch between walking and running animations based on the in-game velocity of the Player.

The exact details of how to export ground transforms will depend on the 3D application. In general, the animation should be created so the character moves through space, rather than running or walking in-place. Animate the *bounds* node to move with the character so there is no translation relative to the bounds node. On export, the ground transform is determined by subtracting the movement of the bounds node from the walking or running animation so that it will play in-place in Torque 3D.

The ground transform information can also be set in the shapes script file (not yet available in Shape Editor UI) by using the setSequenceGroundSpeed member method on the shape object.

4.2.6 Mounts

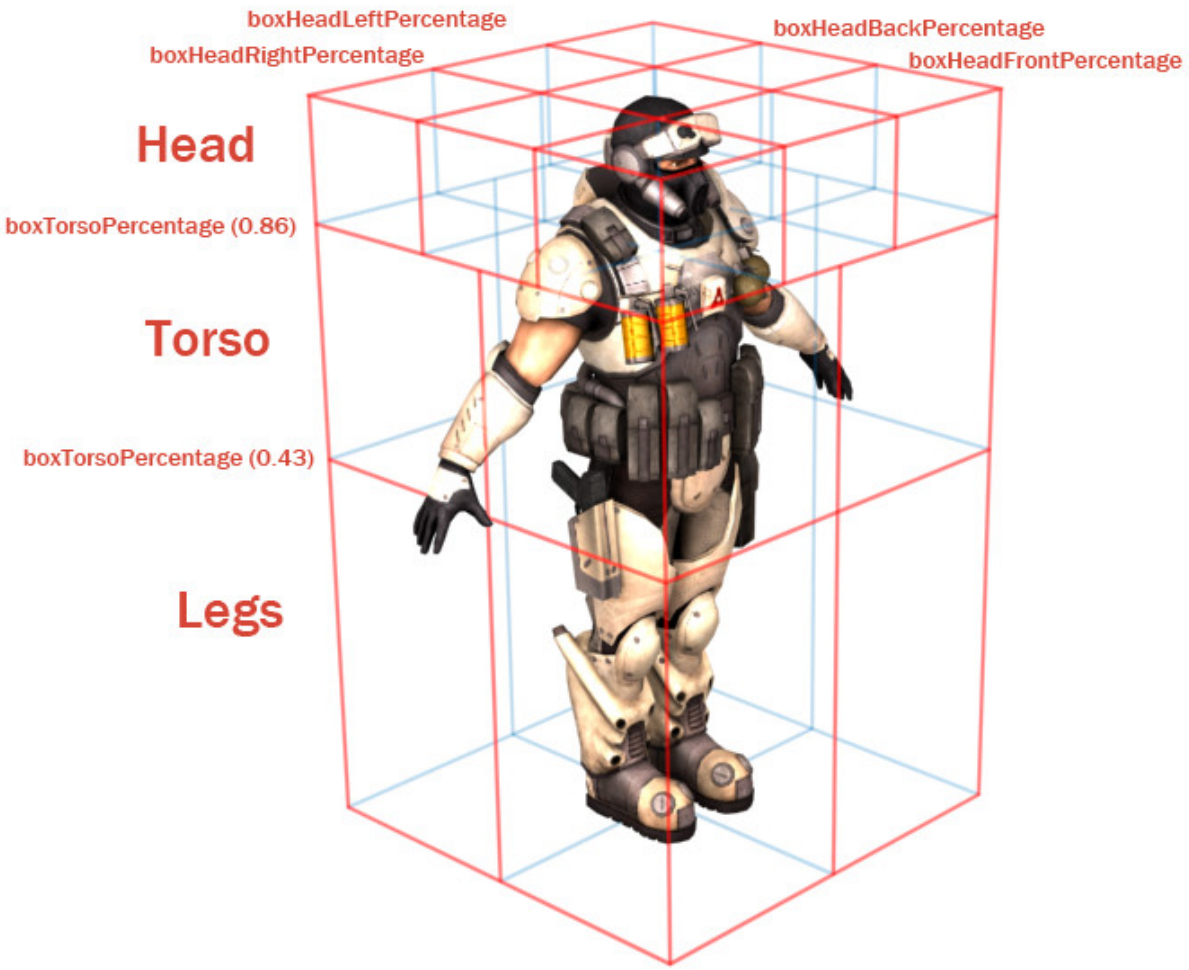
Objects in Torque may be mounted to other objects, such as a Player riding a WheeledVehicle or a weapon placed in the player's hands. Usually the object to be mounted has a node named *mountPoint*. A weapon will be mounted in the player model's hand at node *mount0*. The *mountPoint* node is not essential however, if not present the mounted object's origin is used as the mount point.

4.2.7 Hitboxes

Currently, the player's hitbox defined by their bounding box. In order to get damage locations we have cut the player's world box up into pieces as defined by the following sections in the Player's datablock:

- boundingBox
- boxHeadPercentage
- boxTorsoPercentage
- boxHeadLeftPercentage
- boxHeadRightPercentage
- boxHeadBackPercentage
- boxHeadFrontPercentage

The player's boundingBox determines the length in each dimension the bounding box should encompass. From the standard player datablock, its sections would look like the following:



It may be easiest to come up with these numbers by taking a render of the player, and using an imaging program to determine what percentage of the player makes up their legs/head/torso.

4.2.8 Sequences

Player characters can be setup to use different weapon animations and share those animations between different skinned meshes with the same skeleton hierarchy. A character can have Collada files for the character's skinned mesh and skinned skeleton as well as for the character's animations with just the skeleton for each weapon pose. The animations can either be exported individually or combined in one DAE file that is split up through the Shape Editor.

Just as like with nodes the Player class also make use of a number of animation sequences to work properly in the game:

head Vertical head movement blend animation, start frame is fully up and end frame is fully down. Usually implementd as a 9 frame animation that only affects the neck and head nodes.

headside Horizontal head movement blend animation, start frame is full left and end frame is full right. Usually implemented as a 9 frame animation that only affects the neck and head nodes.

look Vertical arm movement blend animation, start frame is fully up and end frame is fully down. Usually implemented as a 9 frame animation that only affects the spine.

light_recoil Player has been hit lightly.

medium_recoil Player has been hit moderately hard.

heavy_recoil Player has been hit hard.

root Looped idle animation, just the character standing and breathing.

run Looped running forward animation.

back Looped runing backward animation.

side Character side stepping to the right. This looped animation will be played in reverse when moving to the left.

crouch_root Looped crouched idle animation.

crouch_forward Looped crouched forward walk animation.

crouch_backward Looped crouched backward walk animation.

crouch_side Looped crouched right movement animation. Will be played in reverse when moving to the left.

prone_root Looped idle animation of player lying down and not moving.

prone_forward Looped animation of player lying down and moving forward.

prone_backward Looped animation of player lying down and moving backward.

prone_side Looped animation of player lying down and moving to the right. Will be played in reverse when moving to the left.

swim_root Looped treading water animation.

swim_forward Looped animation of swimming forward.

swim_backward Looped animation of swimming backward.

swim_right Looped swiming right animation.

swim_left Looped swiming left animation.

fall Looped falling animation.

jump Character jump up from a moving start animation.

standjump Character jump up from a standing start animation.

land Character landing animation after falling or jumping.

jet Looped jetting animation.

reload Reloading the weapon blend animation.

sitting Looped animation of character sitting in a vehicle.

death# Where # is can be a number for multiple death sequences that will be picked randomly.

A number of these animation sequences are optional.

You add the animations through the Shape Editor by going to sequence tab labeled *Seq* and click on the new sequence icon and a file browsing dialog will open. Select the sequence Collada file you want. Now define the time range that you want by changing the numbers at the beginning and end of the timeline. Complete this process for each sequence that you wish to add.

4.2.9 Blends

Blend animations allow additive animation on the node structure of the shape. These will not conflict with other threads, and can be played on top of the node animation contained in other threads; such animations are relative. Blends only store the changes that occur over the course of the animation and not the absolute position of the nodes.

This means that if a node is transformed by a blend animation, it includes only the transform information for that node, and it will add that transformation on top of the existing position in the base shape. Common uses for blend animations are facial expressions, head turning or nodding, and arm aiming.

Bear in mind that a blend can be played as a normal sequence, or it can be played on top of other sequences. When another sequence is playing, it will alter the root position, and the blend will be applied on top of that.

If you try to do a blend sequence where the root position is different than the 'normal' root (in the default root animation), you might expect that the blend will blend it to the new root (the position the character is positioned in during the blend animation). However, it does not work this way. Since nothing would actually be animating, it doesn't move the bones to the new position. What is contained in the blend sequence is only transform offsets from the blend sequence root position. Thus it is not a good idea to have a different root position in your 'normal' animations and your blends, as they can easily get out of sync!

The values added from the blend animation are based on the root position in the Collada file. This root position does not have to be the beginning of the animation. You can pick any position for the blend animation to reference. This is useful, because you can have a blend animation that can have a reference position that is the 'root' position. For animation like hip twists and arm movements (as in the 'look' animation), the character can be in a natural default state. In this way, you can have one animation control the character through the base pose to an extreme in either direction while referencing the default 'base' state, which will exist somewhere in the middle of the blend animation.

4.2.10 Threads

Animation threads allow multiple sequences to play at the same time on a single shape. For example, a "headside" animation could rotate the player's head to look at something at the same time as a running animation is playing. Each animation sequence is played using a thread. Threads for non-blend sequences are applied first (in order of increasing priority), then blend sequence threads are applied on top (in order of increasing priority). The following rules determine what happens when more than one thread controls the same node in the shape:

1. If two non-blend sequences control the same node, the sequence with higher priority will animate it.
2. If two non-blend sequences with the same priority control the same node, the thread that was created last will animate it.
3. Blend sequences are applied on top of any previous thread, so if two blend sequences control the same node, both will animate it (applied in order of increasing priority, or thread creation order if priority is the same).

4.2.11 Triggers

Triggers are arbitrary markers that can be used to call events on specific frames in a sequence. For example, a trigger can be responsible for generating footstep sounds and footprints when the feet hit the ground during walk and run animations. There can be up to 30 independent trigger states each with their respective on (1 to 30) and off (-1 to -30) states. You decide what each of those trigger states means. You should work with your programmer to define what the trigger states mean and how you should use them.

For example, you could have one trigger for each foot of a character that creates a footprint when the foot is down on the ground. Let's say that a triggerState of 1 is the left foot down and a triggerState of 2 is the right foot down. When the sequence plays the frame during which the left foot touches the ground, you could have a trigger on that frame that has a triggerState of 1 to create a footprint. You would then create another trigger with a triggerState of 2 for the right foot. You don't necessarily need to turn off the footprints (let's assume that the programmer will turn them off when it is necessary), but you could by creating two more triggers with triggerStates -1 and -2.

CHAPTER 5

Scripting

5.1 What is TorqueScript?

TorqueScript (TS) is a proprietary scripting language developed specifically for Torque technology. The language itself is derived from the scripting used for Tribes 2, which was the base tech Torque evolved from. Scripts are written and stored in .cs files, which are compiled and executed by a binary compiled via the C++ engine (.exe for Windows or .app OS X).

The CS extension stands for "C Script," meaning the language resembles C programming. Though there is a connection, TorqueScript is a much higher level language and is easier to learn than standard C or C++.

5.1.1 Basic Usage

Like most other scripting languages, such as Python or Java Script, TorqueScript is a high-level programming language interpreted by Torque 3D at run time. Unlike C++, you can write your code in script and run it without recompiling your game.

All of your interfaces can be built using the GUI Editor, which saves the data out to TorqueScript. The same goes for data saved by the World Editor or Material Editor. Most of the editors themselves are C++ components exposed and constructed via TorqueScript.

More importantly, nearly all of your game play programming will be written in TorqueScript: inventory systems, win/lose scenarios, AI, weapon functionality, collision response, and game flow. All of these can be written in TorqueScript. The language will allow you to rapidly prototype your game without having to be a programming expert or perform lengthy engine recompilation.

5.1.2 Scripting vs Engine Programming

As mentioned above, TorqueScript is comprised of the core C++ objects needed to make your game. For example, you will use the PlayerData structure to create player objects for your game. This structure was written in C++:

```
struct PlayerData: public ShapeBaseData {
        typedef ShapeBaseData Parent;
        bool renderFirstPerson;    ///< Render the player shape in first person


        mass = 9.0f;          // from ShapeBase
        drag = 0.3f;          // from ShapeBase
        density = 1.1f;       // from ShapeBase
```

Instead of having to go into C++ and create new PlayerData objects or edit certain fields (such as mass), PlayerData was exposed to TorqueScript:

```
datablock PlayerData(DefaultPlayerData)
{
        renderFirstPerson = true;

        className = Armor;
        shapeFile = "art/shapes/actors/gideon/base.dts";

        mass = 100;
        drag = 1.3;
        maxdrag = 0.4;

        // Allowable Inventory Items
        maxInv[Pistol] = 1;
        maxInv[PistolAmmo] = 50;
};
```

If you want to change the name of the object, the mass, the inventory, or anything else, just open the script, make the change, and save the file. When you run your game, the changes will immediately take effect. Of course, for this example you could have used the in-game Datablock Editor, but you should get the point. TorqueScript is the first place you should go to write your game play code.

5.2 Language Reference

5.2.1 Basic Syntax

Like other languages, TorqueScript has certain syntactical rules you need to follow. The language is very forgiving, easy to debug, and is not as strict as a low level language like C++. Observe the following line in a script:

```
// Create test variable with a temporary variable
%testVariable = 3;
```

The three most simple rules obeyed in the above code are:

1. Ending a line with a semi-colon `;`
2. Proper use of white space.
3. Commenting.

The engine will parse code line by line, stopping whenever it reaches a semi-colon. This is referred to as a statement terminator, common to other programming languages such as C++, JavaScript, etc. The following code will produce an error that may cause your entire script to fail:

```
%testVariable = 3
%anotherVariable = 4;
```

To the human eye, you are able to discern two separate lines of code with different actions. Here is how the script compiler will read it:

```
%testVariable = 3%anotherVariable = 4;
```

This is obviously not what the original code was meant to do. There are exemptions to this rule, but they come into play when multiple lines of code are supposed to work together for a single action:

```
if(%testVariable == 4)
        echo("Variable equals 4");
```

We have not covered conditional operators or echo commands yet, but you should notice that the first line does not have a semi-colon. The easiest explanation is that the code is telling the compiler: "Read the first line, do the second line if we meet the requirements." In other words, perform operations between semi-colons. Complex operations require multiple lines of code working together.

The second rule, proper use of whitespace, is just as easy to remember. Whitespace refers to how your script code is separated between operations. Let's look at the first example again:

```
%testVariable = 3;
```

The code is storing a value `3` in a local variable `%testVariable`. It is doing so by using a common mathematical operator, the equal sign. TorqueScript recognizes the equal sign and performs the action just as expected. It does not care if there are spaces in the operation:

```
%testVariable=3;
```

The above code works just as well, even without the spaces between the variable, the equal sign, and the `3`. The whitespace rule makes a lot more sense when combined with the semi-colon rule and multiple lines of code working together. The following will compile and run without error:

```
if(%testVariable == 4) echo("Variable equals 4");
```

Comments

The last rule is optional, but should be used as often as possible if you want to create clean code. Whenever you write code, you should try to use comments. Comments are a way for you to leave notes in code which are not compiled into the game. The compiler will essentially skip over these lines.

There are two different comment syntax styles. The first one uses the two slashes, `//`. This is used for single line comments:

```
// This comment line will be ignored
// This second line will also be ignored
%testVariable = 3;
// This third line will also be ignored
```

In the last example, the only line of code that will be executed has to do with `%testVariable`. If you need to comment large chunks of code, or leave a very detailed message, you can use the `/*comment*/` syntax. The `/*` starts the commenting, the `*/` ends the commenting, and anything in between will be considered a comment:

```
/*
While attending school, an instructor taught a mantra I still use:

"Read. Read Code. Code."

Applying this to Torque 3D development is easy:

READ the documentation first.

READ CODE written by other Torque developers.

CODE your own prototypes based on what you have learned.
*/
```

As you can see, the comment makes full use of whitespace and multiple lines. While it is important to comment what the code does, you can also use this to temporarily remove unwanted code until a better solution is found:

```
// Why are you using multiple if statements. Why not use a switch$?
/*
if(%testVariable == "Mich")
  echo("User name: ", %testVariable);

if(%testVariable == "Heather")
  echo("User Name: ", %testVariable);

if(%testVariable == "Nikki")
  echo("User Name: ", %testVariable);
*/
```

5.2.2 Variables

A variable is a letter, word, or phrase linked to a value stored in your game's memory and used during operations. Creating a variable is a one line process. The following code creates a variable by naming it and assigning a value:

```
%localVariable = 3;
```

You can assign any type value to the variable you want. This is referred to as a language being type-insensitive. TorqueScript does not care (insensitive) what you put in a variable, even after you have created it. The following code is completely valid:

```
%localVariable = 27;
%localVariable = "Heather";
%localVariable = "7 7 7";
```

The main purpose of the code is to show that TorqueScript treats all data types the same way. It will interpret and convert the values internally, so you do not have to worry about typecasting. That may seem a little confusing. After all, when would you want a variable that can store a number, a string, or a vector?

You will rarely need to, which is why you want to start practicing good programming habits. An important practice is proper variable naming. The following code will make a lot more sense, considering how the variables are named:

```
%userName = "Heather";
%userAge = 27;
%userScores = "7 7 7";
```

TorqueScript is more forgiving than low level programming languages. While it expects you to obey the basic syntax rules, it will allow you to get away with small mistakes or inconsistency. The best example is variable case sensitivity. With variables, TorqueScript is not case sensitive. You can create a variable and refer to it during operations without adhering to case rules:

```
%userName = "Heather";
echo(%Username);
```

In the above code, `%userName` and `%Username` are the same variable, even though they are using different capitalization. You should still try to remain consistent in your variable naming and usage, but you will not be punished if you slip up occasionally.

There are two types of variables you can declare and use in TorqueScript: *local* and *global*. Both are created and referenced similarly:

```
%localVariable = 1;
$globalVariable = 2;
```

As you can see, local variable names are preceded by the percent sign `%`. Global variables are preceded by the dollar sign `$`. Both types can be used in the same manner: operations, functions, equations, etc. The main difference has to do with how they are scoped.

In programming, scoping refers to where in memory a variable exists during its life. A local variable is meant to only exist in specific blocks of code, and its value is discarded when you leave that block. Global variables are meant to exist and hold their value during your entire programs execution. Look at the following code to see an example of a local variable:

```
function test()
{
   %userName = "Heather";
   echo(%userName);
}
```

We will cover functions a little later, but you should know that functions are blocks of code that only execute when you call them by name. This means the variable, `%userName`, does not exist until the `test()` function is called. When the function has finished all of its logic, the `%userName` variable will no longer exist. If you were to try to access the `%userName` variable outside of the function, you will get nothing.

Most variables you will work with are local, but you will eventually want a variables that last for your entire game. These are extremely important values used throughout the project. This is when global variables become useful. For the most part, you can declare global variables whenever you want:

```
$PlayerName = "Heather";

function printPlayerName()
{
   echo($PlayerName);
}

function setPlayerName()
{
   $PlayerName = "Nikki";
}
```

The above code makes full use of a global variable that holds a player's name. The first declaration of the variable happens outside of the functions, written anywhere in your script. Because it is global, you can reference it in other locations, including separate script files. Once declared, your game will hold on to the variable until shutdown.

5.2.3 Types

TorqueScript implicitly supports several variable data-types: numbers, strings, booleans, arrays and vectors. If you wish to test the various data types, you can use the `echo(...)` command. For example:

```
%meaningOfLife = 42;
echo(%meaningOfLife);

$name = "Heather";
echo($name);
```

The echo will post the results in the console, which can be accessed by pressing the tilde key ~ while in game.

Numbers

TorqueScript handles standard numeric types:

```
123     (Integer)
1.234   (floating point)
1234e-3 (scientific notation)
0xc001  (hexadecimal)
```

Strings

Text, such as names or phrases, are supported as strings. Numbers can also be stored in string format. Standard strings are stored in double-quotes:

```
"abcd"    (string)
```

Example:

```
$UserName = "Heather";
```

Strings with single quotes are called "tagged strings":

```
'abcd'  (tagged string)
```

Tagged strings are special in that they contain string data, but also have a special numeric tag associated with them. Tagged strings are used for sending string data across a network. The value of a tagged string is only sent once, regardless of how many times you actually do the sending.

On subsequent sends, only the tag value is sent. Tagged values must be de-tagged when printing. You will not need to use a tagged string often unless you are in need of sending strings across a network often, like a chat system:

```
$a = 'This is a tagged string';
echo("  Tagged string: ", $a);
echo("Detagged string: ", detag($a));
```

The output will be similar to this:

```
Tagged string: 24
Detagged string:
```

The second echo will be blank unless the string has been passed to you over a network.

Booleans

Like most programming languages, TorqueScript also supports booleans. Boolean numbers have only two values- true or false:

```
true    (1)
false   (0)
```

Again, as in many programming languages the constant "true" evaluates to the number 1 in TorqueScript, and the constant "false" evaluates to the number zero. However, non-zero values are also considered true. Think of booleans as "on/off" switches, often used in conditional statements:

```
$lightsOn = true;

if($lightsOn)
  echo("Lights are turned on");
```

Arrays

Arrays are data structures used to store consecutive values of the same data type:

```
$TestArray[n]   (Single-dimension)
$TestArray[m,n] (Multidimensional)
$TestArray[m_n] (Multidimensional)
```

If you have a list of similar variables you wish to store together, try using an array to save time and create cleaner code. The syntax displayed above uses the letters `n` and `m` to represent where you will input the number of elements in an array. The following example shows code that could benefit from an array:

```
$firstUser = "Heather";
$secondUser = "Nikki";
$thirdUser = "Mich";

echo($firstUser);
echo($secondUser);
echo($thirdUser);
```

Instead of using a global variable for each user name, we can put those values into a single array:

```
$userNames[0] = "Heather";
$userNames[1] = "Nikki";
$userNames[2] = "Mich";

echo($userNames[0]);
echo($userNames[1]);
echo($userNames[2]);
```

Now, let's break the code down. Like any other variable declaration, you can create an array by giving it a name and value:

```
$userNames[0] = "Heather";
```

What separates an array declaration from a standard variable is the use of brackets []. The number you put between the brackets is called the index. The index will access a specific element in an array, allowing you to view or manipulate the data. All the array values are stored in consecutive order.

If you were able to see an array on paper, it would look something like this:

```
[0] [1] [2]
```

In our example, the data looks like this:

```
["Heather"] ["Nikki"] ["Mich"]
```

Like other programming languages, the index is always a numerical value and the starting index is always 0. Just remember, index 0 is always the first element in an array. As you can see in the above example, we create the array by assigning the first index (0) a string value ("Heather").

The next two lines continue filling out the array, progressing through the index consecutively:

```
$userNames[1] = "Nikki";
$userNames[2] = "Mich";
```

The second array element (index 1) is assigned a different string value ("Nikki"), as is the third (index 2). At this point, we still have a single array structure, but it is holding three separate values we can access. Excellent for organization.

The last section of code shows how you can access the data that has been stored in the array. Again, you use a numerical index to point to an element in the array. If you want to access the first element, use 0:

```
echo($userNames[0]);
```

In a later section, you will learn about looping structures that make using arrays a lot simpler. Before moving on, you should know that an array does not have to be a single, ordered list. TorqueScript also support multidimensional arrays.

An single-dimensional array contains a single row of values. A multidimensional array is essentially an array of arrays, which introduces columns as well. The following is a visual of what a multidimensional looks like with three rows and three columns:

```
[x] [x] [x]
[x] [x] [x]
[x] [x] [x]
```

Defining this kind of array in TorqueScript is simple. The following creates an array with 3 rows and 3 columns:

```
$testArray[0,0] = "a";
$testArray[0,1] = "b";
$testArray[0,2] = "c";

$testArray[1,0] = "d";
$testArray[1,1] = "e";
$testArray[1,2] = "f";

$testArray[2,0] = "g";
$testArray[2,1] = "h";
$testArray[2,2] = "i";
```

Notice that we are are now using two indices, both starting at 0 and stopping at 2. We can use these as coordinates to determine which array element we are accessing:

```
[0,0] [0,1] [0,2]
[1,0] [1,1] [1,2]
[2,0] [2,1] [2,2]
```

In our example, which progresses through the alphabet, you can visualize the data in the same way:

```
[a] [b] [c]
[d] [e] [f]
[g] [h] [i]
```

The first element `[0,0]` points to the letter 'a'. The last element `[2,2]` points to the letter 'i'.

Vectors

Vectors are a helpful data-type which are used throughout Torque 3D. For example, many fields in the World Editor take numeric values in sets of 3 or 4. These are stored as strings and interpreted as "vectors":

```
"1.0 1.0 1.0"   (3 element vector)
```

The most common example of a vector would be a world position. Like most 3D coordinate systems, an object's position is stored as (X Y Z). You can use a three element vector to hold this data:

```
%position = "25.0 32 42.5";
```

You can separate the values using spaces or tabs (both are acceptable whitespace). Another example is storing color data in a four element vector. The values that make up a color are "Red Blue Green Alpha," which are all numbers. You can create a vector for color using hard numbers, or variables:

```
%firstColor = "100 100 100 255";
echo(%firstColor);

%red = 128;
%blue = 255;
%green = 64;
%alpha = 255;

%secondColor = %red SPC %blue SPC %green SPC %alpha;
echo(%secondColor);
```

5.2.4 Operators

Operators in TorqueScript behave very similarly to operators in real world math and other programming languages. You should recognize quite a few of these from math classes you took in school, but with small syntactical changes. The rest of this section will explain the syntax and show a brief example, but we will cover these in depth in later guides.

Arithmetic Operators

These are your basic math ops.

Operator	Name	Example	Explanation
*	multiplication	$a * $b	Multiply $a and $b.
/	division	$a / $b	Divide $a by $b.
%	modulo	$a % $b	Remainder of $a divided by $b.
+	addition	$a + $b	Add $a and $b.
-	subtraction	$a - $b	Subtract $b from $a.
++	auto-increment (post-fix only)	$a++	Increment $a.
--	auto-decrement (post-fix only)	$b--	Decrement $b.

Note: ++$a is illegal. The value of $a++ is that of the incremented variable: auto-increment is post-fix in syntax, but pre-increment in sematics (the variable is incremented, before the return value is calculated). This behavior is unlike that of C and C++.

Note: --$b is illegal. The value of $a-- is that of the decremented variable: auto-decrement is post-fix in syntax, but pre-decrement in sematics (the variable is decremented, before the return value is calculated). This behavior is unlike that of C and C++.

Relational Operators

Used in comparing values and variables against each other.

Operator	Name	Example	Explanation
<	Less than	$a < $b	1 if $a is less than $b
>	More than	$a > $b	1 if $a is greater than $b
<=	Less than or Equal to	$a <= $b	1 if $a is less than or equal to $b
>=	More than or Equal to	$a >= $b	1 if $a is greater than or equal to $b
==	Equal to	$a == $b	1 if $a is equal to $b
!=	Not equal to	$a != $b	1 if $a is not equal to $b
!	Logical NOT	!$a	1 if $a is 0
&&	Logical AND	$a && $b	1 if $a and $b are both non-zero
||	Logical OR	$a || $b	1 if either $a or $b is non-zero
$=	String equal to	$c $= $d	1 if $c equal to $d.
!$=	String not equal to	$c !$= $d	1 if $c not equal to $d.

Bitwise Operators

Used for comparing and shifting bits.

Operator	Name	Example	Explanation
~	Bitwise complement	~$a	flip bits 1 to 0 and 0 to 1
&	Bitwise AND	$a & $b	composite of elements where bits in same position are 1
|	Bitwise OR	$a | $b	composite of elements where bits 1 in either of the two elements
^	Bitwise XOR	$a ^ $b	composite of elements where bits in same position are opposite
<<	Left Shift	$a << 3	element shifted left by 3 and padded with zeros
>>	Right Shift	$a >> 3	element shifted right by 3 and padded with zeros

Assignment Operators

Used for setting the value of variables.

Operator	Name	Example	Explanation
=	Assignment	$a = $b;	Assign value of $b to $a
op=	Assignment Operators	$a op= $b;	Equivalent to $a = $a op $b, where op can be any of: * / % + - & | ^ << >>

Note: The value of an assignment is the value being assigned, so $a = $b = $c is legal.

String Operators

There are special values you can use to concatenate strings and variables. Concatenation refers to the joining of multiple values into a single variable. The following is the basic syntax:

```
"string 1" operation "string 2"
```

You can use string operators similarly to how you use mathematical operators (=, +, -, *). You have four operators at your disposal:

Operator	Name	Example	Explanation
`@`	String concatenation	`$c @ $d`	Concatenates strings $c and $d into a single string. Numeric literals/variables convert to strings.
`NL`	New Line	`$c NL $d`	Concatenates strings $c and $d into a single string separated by new-line. Such a string can be decomposed with getRecord()
`TAB`	Tab	`$c TAB $d`	Concatenates strings $c and $d into a single string separated by tab. Such a string can be decomposed with getField()
`SPC`	Space	`$c SPC $d`	Concatenates strings $c and $d into a single string separated by space. Such a string can be decomposed with getWord()

Miscellaneous Operators

General programming operators.

Operator	Name	Example	Explanation
`? :`	Conditional	`x ? y : z`	Evaluates to `y` if `x` equal to 1, else evaluates to `z`
`[]`	Array element	`$a[5]`	Synonymous with `$a5`

`( )`	Delimiting, Grouping	`t2dGetMin(%a, %b)`
		`if ( $a == $b )`
		`($a+$b)*($c-$d)`
			Argument list for function call
			Used with if, for, while, switch keywords
			Control associativity in expressions

`{}`	Compound statement	`if (1) {$a = 1; $b = 2;}` `function foo() {$a = 1;}`	Delimit multiple statements, optional for if, else, for, while Required for switch, datablock, new, function
`,`	Listing	`t2dGetMin(%a, %b)` `%M[1,2]`	Delimiter for arguments
`::`	Namespace	`Item::onCollision()`	This definition of the `onCollision()` function is in the `Item` namespace
`.`	Field/Method selection	`%obj.field` `%obj.method()`	Select a console method or field
`//`	Single-line comment	`// This is a comment`	Used to comment out a single line of code
`/* */`	Multi-line comment	`/*This is a a multi-line comment*/`	Used to comment out multiple consecutive lines `/*` opens the comment, and `*/` closes it

Note: There is no "comma operator", as defined in C/C++; `$a = 1, $b = 2;` is a parse error.

5.2.5 Control Structures

TorqueScript provides basic branching structures that will be familiar to programmers that have used other languages. If you are completely new to programming, you use branching structures to control your game's flow and logic. This section builds on everything you have learned about TorqueScript so far.

if, else

This type of structure is used to test a condition, then perform certain actions if the condition passes or fails. You do not always have to use the full structure, but the following syntax shows the extent of the conditional:

```
if(<boolean expression>)
{
   pass logic
}
else
{
   alternative logic
}
```

Remember how boolean values work? Essentially, a bool can either be true (1) or false (0). The condition (boolean) is always typed into the parenthesis after the "if" syntax. Your logic will be typed within the brackets `{}`. The following example uses specific variable names and conditions to show how this can be used:

```
// Global variable that controls lighting
$lightsShouldBeOn = true;

// Check to see if lights should be on or off
if($lightsShouldBeOn)
{
   // True. Call turn on lights function
   turnOnLights();

   echo("Lights have been turned on");
}
else
{
   // False. Turn off the lights
   turnOffLights();

   echo("Lights have been turned off");
}
```

Brackets for single line statements are optional. If you are thinking about adding additional logic to the code, then you should use the brackets anyway. If you know you will only use one logic statement, you can use the following syntax:

```
// Global variable that controls lighting
$lightsShouldBeOn = true;

// Check to see if lights should be on or off
if($lightsShouldBeOn)
  turnOnLights();   // True. Call turn on lights function
else
  turnOffLights(); // False. Turn off the lights
```

switch, switch$

If your code is using several cascading if-then-else statements based on a single value, you might want to use a switch statement instead. Switch statements are easier to manage and read. There are two types of switch statements, based on data type: numeric (`switch`) and string (`switch$`):

```
switch(<numeric expression>)
{
   case value0:
       statements;
   case value1:
       statements;
   case value3:
       statements;
   default:
       statements;
}
```

As the above code demonstrates, start by declaring the switch statement by passing in a value to the `switch(...)` line. Inside of the brackets `{}`, you will list out all the possible cases that will execute based on what value being tested. It is wise to always use the default case, anticipating rogue values being passed in:

```
switch($ammoCount)
{
   case 0:
      echo("Out of ammo, time to reload");
      reloadWeapon();
   case 1:
      echo("Almost out of ammo, warn user");
      lowAmmoWarning();
   case 100:
      echo("Full ammo count");
      playFullAmmoSound();
   default:
      doNothing();
}
```

`switch` only properly evaluates numerical values. If you need a switch statement to handle a string value, you will want to use `switch$`. The `switch$` syntax is similar to what you just learned:

```
switch$ (<string expression>)
{
   case "string value 0":
                statements;
   case "string value 1":
                statements;
   case "string value N":
                statements;
   default:
                statements;
}
```

Appending the $ sign to switch will immediately cause the parameter passed in to be parsed as a string. The following code applies this logic:

```
// Print out specialties
switch($userName)
{
   case "Heather":
```

```
      echo("Sniper");
   case "Nikki":
      echo("Demolition");
   case Mich:
      echo("Meat shield");
   default:
      echo("Unknown user");
}
```

for

As the name implies, this structure type is used to repeat logic in a loop based on an expression. The expression is usually a set of variables that increase by count, or a constant variable changed once a loop has hit a specific point:

```
for(expression0; expression1; expression2)
{
    statement(s);
}
```

One way to label the expressions in this syntax are (startExpression; testExpression; countExpression). Each expression is separated by a semi-colon:

```
for(%count = 0; %count < 3; %count++)
{
    echo(%count);
}

OUTPUT:
0
1
2
```

The first expression creates the local variable `%count` and initializing it to 0. In the second expression determines when to stop looping, which is when the `%count` is no longer less than 3. Finally, the third expression increases the count the loop relies on.

foreach

Simplify the iteration over sets of objects and string vectors. To loop over each object in a SimSet, use the foreach statement:

```
foreach( %obj in %set )
   /* do something with %obj */;
```

To loop over each element in a string vector, use the foreach$ statement:

```
foreach$( %str in "a b c" )
   /* do something with %str */;
```

while

A while loop is a much simpler looping structure compared to a for loop.

```
while(expression)
{
    statements;
}
```

As soon as the expression is met, the while loop will terminate:

```
%countLimit = 0;

while(%countLimit <= 5)
{
   echo("Still in loop");
   %count++;
}

echo("Loop was terminated");
```

5.2.6 Functions

Much of your TorqueScript experience will come down to calling existing Console Functions and writing your own. Functions are a blocks of code that only execute when you call them by name. Basic functions in TorqueScript are defined as follows:

```
// function - Is a keyword telling TorqueScript we are defining a new function.
// function_name - Is the name of the function we are creating.
// ... - Is any number of additional arguments.
// statements - Your custom logic executed when function is called
// return val - The value the function will give back after it has completed. Optional.

function function_name([arg0],...,[argn])
{
    statements;
    [return val;]
}
```

The function keyword, like other TorqueScript keywords, is case sensitive. You must type it exactly as shown above. The following is an example of a custom function that takes in two parameters, then executes code based on those arguments.

TorqueScript can take any number of arguments, as long as they are comma separated. If you call a function and pass fewer parameters than the function's definition specifies, the un-passed parameters will be given an empty string as their default value:

```
function echoRepeat (%echoString, %repeatCount)
{
   for (%count = 0; %count < %repeatCount; %count++)
   {
      echo(%echoString);
   }
}
```

You can cause this function to execute by calling it in the console, or in another function:

```
echoRepeat("hello!", 5);

OUTPUT:
"hello!"
"hello!"
```

```
"hello!"
"hello!"
"hello!"
```

If you define a function and give it the same name as a previously defined function, TorqueScript will completely override the old function. This still applies even if you change the number of parameters used; the older function will still be overridden.

Torque 3D also contain Console Functions written in C++, then exposed to TorqueScript. These are global functions you can call at any time, and are usually very helpful or important. E.g. throughout this document, we have been using the Console Function `echo(...)`.

5.2.7 Objects

The most complex aspect of TorqueScript involves dealing with game objects. Much of your object creation will be performed in the World Editor, but you should still know how to manipulate objects at a script level. One thing to remember is that everything in TorqueScript is an object: players, vehicles, items, etc.

Every object added in the level is saved to a mission file, which is written entirely in TorqueScript. This also means every game object is accessible from script.

Syntax

Even though objects are originally created in C++, they are exposed to script in a way that allows them to be declared using the following syntax:

```
%objectID = new ObjectType(Name : CopySource, arg0, ..., argn)
{
   <datablock = DatablockIdentifier;>

   [existing_field0 = InitialValue0;]
   ...
   [existing_fieldN = InitialValueN;]

   [dynamic_field0 = InitialValue0;]
   ...
   [dynamic_fieldN = InitialValueN;]
};
```

%objectID Is the variable where the object's handle will be stored.

new Is a key word telling the engine to create an instance of the following ObjectType.

ObjectType Is any class declared in the engine or in script that has been derived from SimObject or a subclass of SimObject. SimObject-derived objects are what we were calling "game world objects" above.

Name (optional) Is any expression evaluating to a string, which will be used as the object's name.

CopySource (optional) The name of an object which is previously defined somewhere in script. Existing field values will be copied from CopySource to the new object being created. Any dynamic fields defined in CopySource will also be defined in the new object, and their values will be copied. Note: If CopySource is of a different ObjectType than the object being created, only CopySource's dynamic fields will be copied.

arg0, ..., argn (optional) Is a comma separated list of arguments to the class constructor (if it takes any).

datablock Many objects (those derived from GameBase, or children of GameBase) require datablocks to initialize specific attributes of the new object. Datablocks are discussed below.

existing_fieldN In addition to initializing values with a datablock, you may also initialize existing class members (fields) here. In order to modify a member of a C++-defined class, the member must be exposed to the Console.

dynamic_fieldN Lastly, you may create new fields (which will exist only in Script) for your new object. These will show up as dynamic fields in the World Editor Inspector.

The main object variants you can create are SimObjects without a datablock, and game objects which require a datablock. The most basic SimObject can be created in a single line of code:

```
// Create a SimObject without any name, argument, or fields.
$exampleSimObject = new SimObject();
```

The `$exampleSimObject` variable now has access to all the properties and functions of a basic SimObject. Usually, when you are creating a SimObject you will want custom fields to define features:

```
// Create a SimObject with a custom field
$exampleSimObject = new SimObject()
{
   catchPhrase = "Hello world!";
};
```

As with the previous example, the above code creates a SimObject without a name which can be referenced by the global variable `$exampleSimObject`. This time, we have added a user defined field called "catchPhrase." There is not a single stock Torque 3D object that has a field called "catchPhrase." However, by adding this field to the SimObject it is now stored as long as that object exists.

The other game object variant mentioned previously involves the usage of datablocks. Datablocks contain static information used by a game object with a similar purpose. Datablocks are transmitted from a server to client, which means they cannot be modified while the game is running.

We will cover datablocks in more detail later, but the following syntax shows how to create a game object using a datablock:

```
// create a StaticShape using a datablock
datablock StaticShapeData(ceiling_fan)
{
   category = "Misc";
   shapeFile = "art/shapes/undercity/cfan.dts";
   isInvincible = true;
};

new StaticShape(CistFan)
{
   dataBlock = "ceiling_fan";
   position = "12.5693 35.5857 59.5747";
   rotation = "1 0 0 0";
   scale = "1 1 1";
};
```

Once you have learned about datablocks, the process is quite simple:

1. Create a datablock in script, or using the datablock editor
2. Add a shape to the scene from script or using the World Editor
3. Assign the new object a datablock

Handles vs Names

Every game object added to a level can be accessed by two parameters:

Handle A unique numeric ID generated when the object is created

Name This is an optional parameter given to an object when it is created. You can assign a name to an object from the World Editor, or do so in TorqueScript.

Example:

```
// In this example, CistFan is the name of the object
new StaticShape(CistFan)
{
   dataBlock = "ceiling_fan";
   position = "12.5693 35.5857 59.5747";
   rotation = "1 0 0 0";
   scale = "1 1 1";
};
```

While in the World Editor, you will not be allowed to assign the same name to multiple, separate objects. The editor will ignore the attempt. If you manually name two objects the same thing in script, the game will only load the first object and ignore the second.

Singletons

If you need a global script object with only a single instance, you can use the singleton keyword. Singletons, in TorqueScript, are mostly used for unique shaders, materials, and other client-side only objects.

For example, SSAO (screen space ambient occlusion) is a post-processing effect. The game will only ever need a single instance of the shader, but it needs to be globally accessible on the client. The declaration of the SSAO shader in TorqueScript can be shown below:

```
singleton ShaderData( SSAOShader )
{
   DXVertexShaderFile   = "shaders/common/postFx/postFxV.hlsl";
   DXPixelShaderFile    = "shaders/common/postFx/ssao/SSAO_P.hlsl";
   pixVersion = 3.0;
};
```

Fields

Objects instantiated via script may have data members, referred to as Fields.

Methods

In addition to the creation of stand-alone functions, TorqueScript allows you to create and call methods attached to objects. Some of the more important Console Methods are already written in C++, then exposed to script. You can call these methods by using the dot `.` notation:

```
objHandle.function_name();

objName.function_name();
```

Example:

```
new StaticShape(CistFan)
{
   dataBlock = "ceiling_fan";
   position = "12.5693 35.5857 59.5747";
```

```
   rotation = "1 0 0 0";
   scale = "1 1 1";
};

// Write all the objects methods to the console log
CistFan.dump();

// Get the ID of an object, using the object's name
$objID = CistFan.getID();

// Print the ID to the console
echo("Object ID: ", $objID);

// Get the object's position, using the object's handle
%position = $objID.getPosition();

// Print the position to the console
echo("Object Position: ", %position);
```

The above example shows how you can call an object's method by using its name or a variable containing its handle (unique ID number). Additionally, TorqueScript supports the creation of methods that have no associated C++ counterpart:

```
// function - Is a keyword telling TorqueScript we are defining a new function.
// ClassName::- Is the class type this function is supposed to work with.
// function_name - Is the name of the function we are creating.
// ... - Is any number of additional arguments.
// statements - Your custom logic executed when function is called
// %this- Is a variable that will contain the handle of the 'calling object'.
// return val - The value the function will give back after it has completed. Optional.

function Classname::func_name(%this, [arg0],...,[argn])
{
   statements;
   [return val;]
}
```

At a minimum, Console Methods require that you pass them an object handle. You will often see the first argument named %this. People use this as a hint, but you can name it anything you want. As with Console Functions any number of additional arguments can be specified separated by commas.

As a simple example, let's say there is an object called Samurai, derived from the Player class. It is likely that a specific appearance and play style will be given to the samurai, so custom ConsoleMethods can be written. Here is a sample:

```
function Samurai::sheatheSword(%this)
{
    echo("Katana sheathed");
}
```

When you add a Samurai object to your level via the World Editor, it will be given an ID. Let's pretend the handle (ID number) is 1042. We can call its ConsoleMethod once it is defined, using the period syntax:

```
1042.sheatheSword();

OUTPUT: "Katana sheathed"
```

Notice that no parameters were passed into the function. The `%this` parameter is inherent, and the original function did not require any other parameters.

5.2.8 Packages

The package keyword tells the console that the subsequent block of code is to be declared but not loaded. Packages provide dynamic function-polymorphism in TorqueScript. In short, a function defined in a package will over-ride the prior definition of a same named function when the is activated. When the package is subsequently de-activated, the previous definition of any overridden functions will be re-asserted.

A package has the following syntax:

```
package package_name
{
   function function_definition0()
   {
      // code
   }
   function function_definitionN()
   {
      // code
   }
};
```

Some things to know:

- The same function can be defined in multiple packages.
- Only functions can be packaged.
- Datablocks cannot be packaged.
- Packages 'stack' meaning that deactivating packages activated prior to the currently active (s) will deactivate all packages activated prior to the being deactivated (see example below).
- Functions in a package may activate and deactivate packages.

In order to use the functions in a package, the package must be activated:

```
activatePackage(package_name);
```

Subsequently a package can be deactivated:

```
deactivatePackage(package_name);
```

First, define a function and two packages, each of which provides an alternative definition by the same name:

```
function testFunction()
{
   echo( "testFunction() - unpackaged." );
}
package MyPackage0
{
  function testFunction()
  {
      echo( "testFunction() - MyPackage0." );
  }
};
package MyPackage1
{
  function testFunction()
  {
      echo( "testFunction() - MyPackage1." );
  }
};
```

Now invoke the testFunction() function from the console under three different conditions:

```
==> testFunction();
testFunction() - unpackaged.
==> activatePackage( MyPackage0 );
==> testFunction();
testFunction() - MyPackage0.
==> activatePackage( MyPackage1 );
==> testFunction();
testFunction() - MyPackage1.
==> deactivatePackage( MyPackage0 );  // MyPackage1 is automatically deactivated.
==> testFunction();
testFunction() - unpackaged.
```

5.3 Console Reference

5.3.1 Core

Basic engine and language functionality for TorqueScript.

Console

The basis of the TorqueScript system and command execution.

Functions

void **cls**()
Clears the console output.

void **debugEnumInstances**(string *className*, string *functionName*)
Call the given function for each instance of the given class.

Parameters

- **className** – Name of the class for which to enumerate instances.
- **functionName** – Name of function to call and pass each instance of the given class.

bool **dumpEngineDocs**(string *outputFile*)
Dumps the engine scripting documentation to the specified file overwriting any existing content.

Parameters **outputFile** – The relative or absolute output file path and name.

Returns Returns true if successful.

SimXMLDocument **exportEngineAPIToXML**()
Create a XML document containing a dump of the entire exported engine API.

Returns containing a dump of the engine's export information or NULL if the operation failed.

string **getCategoryOfClass**(string *className*)
Returns the category of the given class.

Parameters **className** – The name of the class.

string **getDescriptionOfClass**(string *className*)
Returns the description string for the named class.

Parameters **className** – The name of the class.

Returns The class description in string format.

bool **isClass** (string *identifier*)
Returns true if the passed identifier is the name of a declared class.

bool **isMemberOfClass** (string *className*, string *superClassName*)
Returns true if the class is derived from the super class. If either class doesn't exist this returns false.

Parameters

- **className** – The class name.
- **superClassName** – The super class to look for.

bool **isValidObjectName** (string *name*)
Return true if the given name makes for a valid object name.

Parameters **name** – Name of object

Returns True if name is allowed, false if denied (usually because it starts with a number, _, or invalid character

SimObject **loadObject** (string *filename*)
Loads a serialized object from a file.

Parameters **Name** – and path to text file containing the object

bool **saveObject** (SimObject *object*, string *filename*)
Serialize the object to a file.

Parameters

- **object** – The object to serialize.
- **filename** – The file name and path.

void **unitTest_runTests** ()
Run unit tests, or just the tests that prefix match against the searchString.

Variables

string $instantGroup
The group that objects will be added to when they are created.

bool $Con::alwaysUseDebugOutput
Determines whether to send output to the platform's "debug" system.

bool $Con::logBufferEnabled
If true, the log buffer will be enabled.

int $Con::objectCopyFailures
If greater than zero then it counts the number of object creation failures based on a missing copy object and does not report an error..

int $Con::printLevel
This is deprecated. It is no longer in use and does nothing.

bool $Con::useTimestamp
If true a timestamp is prepended to every console message.

bool $Con::warnUndefinedVariables
If true, a warning will be displayed in the console whenever a undefined variable is used in script.

Debugging

Functionality to help spot program errors. Also provides profiler functions, helpful in determining performance bottlenecks.

Functions

void **backtrace**()
Prints the scripting call stack to the console log. Used to trace functions called from within functions. Can help discover what functions were called (and not yet exited) before the current point in scripts.

void **debug**()
Drops the engine into the native C++ debugger. This function triggers a debug break and drops the process into the IDE's debugger. If the process is not running with a debugger attached it will generate a runtime error on most platforms.

void **debugDumpAllObjects**()
Dumps all current EngineObject instances to the console.

void **debugv**(string *variableName*)
Logs the value of the given variable to the console. Prints a string of the form " lt variableName gt = lt variable value gt " to the console.

Parameters **variableName** – Name of the local or global variable to print.

Example:

```
%var = 1;
debugv( "%var" ); // Prints "%var = 1"
```

void **dumpAlloc**(int *allocNum*)
Dumps information about the given allocated memory block.

Parameters **allocNum** – Memory block to dump information about.

void **dumpMemSnapshot**(string *fileName*)
Dumps a snapshot of current memory to a file. The total memory used will also be output to the console. This function will attempt to create the file if it does not already exist.

Parameters **fileName** – Name and path of file to save profiling stats to. Must use forward slashes (/)

Example:

```
dumpMemSnapshot( "C:/Torque/ProfilerLogs/profilerlog1.txt" );
```

void **dumpUnflaggedAllocs**(string *fileName*)
Dumps all unflagged memory allocations. Dumps all memory allocations that were made after a call to flagCurrentAllocs() . Helpful when used with flagCurrentAllocs() for detecting memory leaks and analyzing general memory usage.

Parameters **fileName** – Optional file path and location to dump all memory allocations not flagged by flagCurrentAllocs(). If left blank, data will be dumped to the console.

Example:

```
dumpMemSnapshot(); // dumps info to console
dumpMemSnapshot( "C:/Torque/profilerlog1.txt" ); // dumps info to file
```

void **flagCurrentAllocs**()
Flags all current memory allocations. Flags all current memory allocations for exclusion in subsequent calls to dumpUnflaggedAllocs() . Helpful in detecting memory leaks and analyzing memory usage.

void **freeMemoryDump**()
Dumps some useful statistics regarding free memory. Dumps an analysis of 'free chunks' of memory. Does not print how much memory is free.

void **profilerDump**()
Dumps current profiling stats to the console window.

void **profilerDumpToFile**(string *fileName*)
Dumps current profiling stats to a file.

Parameters **fileName** – Name and path of file to save profiling stats to. Must use forward slashes (/). Will attempt to create the file if it does not already exist.

Example:

```
profilerDumpToFile( "C:/Torque/log1.txt" );
```

void **profilerEnable**(bool *enable*)
Enables or disables the profiler. Data is only gathered while the profiler is enabled.

void **profilerMarkerEnable**(string *markerName*, bool *enable*)
Enable or disable a specific profile.

Parameters

- **enable** – Optional paramater to enable or disable the profile.
- **markerName** – Name of a specific marker to enable or disable.

void **profilerReset**()
Resets the profiler, clearing it of all its data. If the profiler is currently running, it will first be disabled. All markers will retain their current enabled/disabled status.

int sizeof(string objectOrClass)
Determines the memory consumption of a class or object.

Parameters **objectOrClass** – The object or class being measured.

Returns Returns the total size of an object in bytes.

void **telnetSetParameters**(int *port*, string *consolePass*, string *listenPass*, bool *remoteEcho*)
Initializes and open the telnet console.

Parameters

- **port** – Port to listen on for console connections (0 will shut down listening).
- **consolePass** – Password for read/write access to console.
- **listenPass** – Password for read access to console.
- **remoteEcho** – [optional] Enable echoing back to the client, off by default.

void **trace**(bool *enable*)
Enable or disable tracing in the script code VM. When enabled, the script code runtime will trace the invocation and returns from all functions that are called and log them to the console. This is helpful in observing the flow of the script program.

Parameters **enable** – New setting for script trace execution, on by default.

void **validateMemory**()
Used to validate memory space for the game.

Variables

int **getAppVersionNumber**
Get the version of the application build, as a string.

string **getAppVersionString**
Get the version of the aplication, as a human readable string.

string **getBuildString**
Get the type of build, "Debug" or "Release".

string **getCompileTimeString**
Get the time of compilation.

string **getEngineName**
Get the name of the engine product that this is running from, as a string.

int **getVersionNumber**
Get the version of the engine build, as a string.

string **getVersionString**
Get the version of the engine build, as a human readable string.

Logging

Functions for logging messages, warnings, and errors to the console.

Classes

ConsoleLogger

Inherit: SimObject

Description A class designed to be used as a console consumer and log the data it receives to a file.

Methods

bool `ConsoleLogger::attach()`
Attaches the logger to the console and begins writing to file.

Example:

```
// Create the logger
// Will automatically start writing to testLogging.txt with normal priority
newConsoleLogger(logger, "testLogging.txt", false);

// Send something to the console, with the logger consumes and writes file
echo("This is logged to the file");

// Stop logging, but do not delete the logger
logger.detach();

echo("This is not logged to the file");

// Attach the logger to the console again
logger.attach();

// Logging has resumedecho("Logging has resumed");
```

bool ConsoleLogger::**detach**()
Detaches the logger from the console and stops writing to file.

Example:

```
// Create the logger
// Will automatically start writing to testLogging.txt with normal priority
newConsoleLogger(logger, "testLogging.txt", false);

// Send something to the console, with the logger consumes and writes to file
echo("This is logged to the file");

// Stop logging, but do not delete the logger
logger.detach();

echo("This is not logged to the file");

// Attach the logger to the console again
logger.attach();

// Logging has resumedecho("Logging has resumed");
```

Fields

LogLevel ConsoleLogger::**level**
Determines the priority level and attention the logged entry gets when recorded.

Enumeration

enum LogLevel
Priority levels for logging entries.

Parameters

- **normal** – Lowest priority level, no highlighting.
- **warning** – Mid level priority, tags and highlights possible issues in blue.
- **error** – Highest priority level, extreme emphasis on this entry. Highlighted in red.

Functions

void **dumpConsoleClasses**(bool *dumpScript*, bool *dumpEngine*)
Dumps all declared console classes to the console.

Parameters

- **dumpScript** – Optional parameter specifying whether or not classes defined in script should be dumped.
- **dumpEngine** – Optional parameter specifying whether or not classes defined in the engine should be dumped.

void **dumpConsoleFunctions**(bool *dumpScript*, bool *dumpEngine*)
Dumps all declared console functions to the console.

Parameters

- **dumpScript** – Optional parameter specifying whether or not functions defined in script should be dumped.
- **dumpEngine** – Optional parameter specitying whether or not functions defined in the engine should be dumped.

void **echo**(string *message*, ...)
Logs a message to the console. Concatenates all given arguments to a single string and prints the string to the console. A newline is added automatically after the text.

Parameters **message** – Any number of string arguments.

void **error**(string *message*, ...)
Logs an error message to the console. Concatenates all given arguments to a single string and prints the string to the console as an error message (in the in-game console, these will show up using a red font by default). A newline is added automatically after the text.

Parameters **message** – Any number of string arguments.

void **log**(string *message*)
Logs a message to the console.

Parameters **message** – The message text.

void **logError**(string *message*)
Logs an error message to the console.

Parameters **message** – The message text.

void **logWarning**(string *message*)
Logs a warning message to the console.

Parameters **message** – The message text.

void **setLogMode**(int *mode*)
Determines how log files are written. Sets the operational mode of the console logging system. Additionally, when changing the log mode and thus opening a new log file, either of the two mode values may be combined by binary OR with 0x4 to cause the logging system to flush all console log messages that had already been issued to the console system into the newly created log file.

Parameters **mode** – Parameter specifying the logging mode. This can be:1: Open and close the console log file for each seperate string of output. This will ensure that all parts get written out to disk and that no parts remain in intermediate buffers even if the process crashes.2: Keep the log file open and write to it continuously. This will make the system operate faster but if the process crashes, parts of the output may not have been written to disk yet and will be missing from the log.

void **warn**(string *message*, ...)
Logs a warning message to the console. Concatenates all given arguments to a single string and prints the string to the console as a warning message (in the in-game console, these will show up using a turquoise font by default). A newline is added automatically after the text.

Parameters **message** – Any number of string arguments.

Messaging

Script classes and functions used for passing messages and events between classes.

Classes

EventManager The EventManager class is a wrapper for the standard messaging system.

Inherit: SimObject

Description It provides functionality for management of event queues, events, and subscriptions. Creating an EventManager is as simple as calling new EventManager and specifying a queue name.

Example:

```
// Create the EventManager.
$MyEventManager = newEventManager() { queue = "MyEventManager"; };

// Create an event.
$MyEventManager.registerEvent( "SomeCoolEvent" );

// Create a listener and subscribe.
$MyListener = newScriptMsgListener() { class = MyListener; };
$MyEventManager.subscribe( $MyListener, "SomeCoolEvent" );

function MyListener::onSomeCoolEvent( %this, %data )
{
     echo( "onSomeCoolEvent Triggered" );
}

// Trigger the event.
$MyEventManager.postEvent( "SomeCoolEvent", "Data" );
```

Methods

void EventManager::**dumpEvents**()
Print all registered events to the console.

void EventManager::**dumpSubscribers**(String *event*)
Print all subscribers to an event to the console.

Parameters **event** – The event whose subscribers are to be printed. If this parameter isn't specified, all events will be dumped.

bool EventManager::**isRegisteredEvent**(String *event*)
Check if an event is registered or not.

Parameters **event** – The event to check.

Returns Whether or not the event exists.

bool EventManager::**postEvent**(String *event*, String *data*)
~Trigger an event.

Parameters

- **event** – The event to trigger.
- **data** – The data associated with the event.

Returns Whether or not the event was dispatched successfully.

bool EventManager::**registerEvent**(String *event*)
Register an event with the event manager.

Parameters **event** – The event to register.

Returns Whether or not the event was registered successfully.

void EventManager::**remove**(SimObject *listener*, String *event*)
Remove a listener from an event.

Parameters

- **listener** – The listener to remove.

- **event** – The event to be removed from.

void EventManager::**removeAll**(SimObject *listener*)
Remove a listener from all events.

Parameters **listener** – The listener to remove.

bool EventManager::**subscribe**(SimObject *listener*, String *event*, String *callback*)
Subscribe a listener to an event.

Parameters

- **listener** – The listener to subscribe.
- **event** – The event to subscribe to.
- **callback** – Optional method name to receive the event notification. If this is not specified, "on[event]" will be used.

Returns Whether or not the subscription was successful.

void EventManager::**unregisterEvent**(String *event*)
Remove an event from the EventManager .

Parameters **event** – The event to remove.

Fields
string EventManager::**queue**
List of events currently waiting.

Message Base class for messages.

Inherit: SimObject

Description Message is the base class for C++ defined messages, and may also be used in script for script defined messages if no C++ subclass is appropriate.

Messages are reference counted and will be automatically deleted when their reference count reaches zero. When you dispatch a message, a reference will be added before the dispatch and freed after the dispatch. This allows for temporary messages with no additional code. If you want to keep the message around, for example to dispatch it to multiple queues, call addReference() before dispatching it and freeReference() when you are done with it. Never delete a Message object directly unless addReference() has not been called or the message has not been dispatched.

Message IDs are pooled similarly to datablocks, with the exception that IDs are reused. If you keep a message for longer than a single dispatch, then you should ensure that you clear any script variables that refer to it after the last freeReference(). If you don't, then it is probable that the object ID will become valid again in the future and could cause hard to track down bugs.

Messages have a unique type to simplify message handling code. For object messages, the type is defined as either the script defined class name or the C++ class name if no script class was defined. The message type may be obtained through the getType() method.

By convention, any data for the message is held in script accessible fields. Messages that need to be handled in C++ as well as script provide the relevant data through persistent fields in a subclass of Message to provide best performance on the C++ side. Script defined messages usually their through dynamic fields, and may be accessed in C++ using the SimObject::getDataField() method.

Methods

void Message::**addReference**()
Increment the reference count for this message.

void Message::**freeReference**()
Decrement the reference count for this message.

string Message::**getType**()
Get message type (script class name or C++ class name if no script defined class).

void Message::**onAdd**()
Script callback when a message is first created and registered.

Example:

```
function Message::onAdd(%this)
{
   // Perform on add code here
}
```

void Message::**onRemove**()
Script callback when a message is deleted.

Example:

```
function Message::onRemove(%this)
{
   // Perform on remove code here
}
```

MessageForwarder Forward messages from one queue to another.

Inherit: ScriptMsgListener

Description MessageForwarder is a script class that can be used to forward messages from one queue to another.

Example:

```
%fwd = newMessageForwarder()
{
   toQueue = "QueueToSendTo";
};

registerMessageListener("FromQueue", %fwd);
```

Where "QueueToSendTo" is the queue you want to forward to, and "FromQueue" is the queue you want to forward from.

Fields

caseString MessageForwarder::**toQueue**
Name of queue to forward to.

ScriptMsgListener Script accessible version of Dispatcher::IMessageListener. Often used in conjunction with EventManager.

Inherit: SimObject

Description The main use of ScriptMsgListener is to allow script to listen formessages. You can subclass ScriptMsgListener in script to receivethe Dispatcher::IMessageListener callbacks.

Alternatively, you can derive from it in C++ instead of SimObject toget an object that implements Dispatcher::IMessageListener with scriptcallbacks. If you need to derive from something other then SimObject,then you will need to implement the Dispatcher::IMessageListenerinterface yourself.

Example:

```
// Create the EventManager.
$MyEventManager = newEventManager() { queue = "MyEventManager"; };

// Create an event.
$MyEventManager.registerEvent( "SomeCoolEvent" );

// Create a listener and subscribe.
$MyListener = newScriptMsgListener() { class = MyListener; };
$MyEventManager.subscribe( $MyListener, "SomeCoolEvent" );

function MyListener::onSomeCoolEvent( %this, %data )
{
   echo( "onSomeCoolEvent Triggered" );
}

// Trigger the event.
$MyEventManager.postEvent( "SomeCoolEvent", "Data" );
```

Methods

void ScriptMsgListener::**onAdd**()

Script callback when a listener is first created and registered.

Example:

```
function ScriptMsgListener::onAdd(%this)
{
   // Perform on add code here
}
```

void ScriptMsgListener::**onAddToQueue**(string *queue*)

Callback for when the listener is added to a queue. The default implementation of onAddToQueue() and onRemoveFromQueue() provide tracking of the queues this listener is added to through the mQueues member. Overrides of onAddToQueue() or onRemoveFromQueue() should ensure they call the parent implementation in any overrides.

Parameters **queue** – The name of the queue that the listener added to

bool ScriptMsgListener::**onMessageObjectReceived**(string *queue*, Message *msg*)

Called when a message object (not just the message data) is passed to a listener.

Parameters

- **queue** – The name of the queue the message was dispatched to
- **msg** – The message object

Returns false to prevent other listeners receiving this message, true otherwise

bool ScriptMsgListener::**onMessageReceived**(string *queue*, string *event*, string *data*)

Called when the listener has received a message.

Parameters

- **queue** – The name of the queue the message was dispatched to
- **event** – The name of the event (function) that was triggered
- **data** – The data (parameters) for the message

Returns false to prevent other listeners receiving this message, true otherwise

void ScriptMsgListener::**onRemove**()
Script callback when a listener is deleted.

Example:

```
function ScriptMsgListener::onRemove(%this)
{
   // Perform on remove code here
}
```

void ScriptMsgListener::**onRemoveFromQueue**(string *queue*)
Callback for when the listener is removed from a queue. The default implementation of onAddToQueue() and onRemoveFromQueue() provide tracking of the queues this listener is added to through the mQueues member. Overrides of onAddToQueue() or onRemoveFromQueue() should ensure they call the parent implementation in any overrides.

Parameters **queue** – The name of the queue that the listener was removed from

Functions

bool **dispatchMessage**(string *queueName*, string *message*, string *data*)
Dispatch a message to a queue.

Parameters

- **queueName** – Queue to dispatch the message to
- **message** – Message to dispatch
- **data** – Data for message

Returns True for success, false for failure

bool **dispatchMessageObject**(string *queueName*, string *message*)
Dispatch a message object to a queue.

Parameters

- **queueName** – Queue to dispatch the message to
- **message** – Message to dispatch

Returns true for success, false for failure

bool **isQueueRegistered**(string *queueName*)
Determines if a dispatcher queue exists.

Parameters **queueName** – String containing the name of queue

bool **registerMessageListener**(string *queueName*, string *listener*)
Registers an event message.

Parameters

- **queueName** – String containing the name of queue to attach listener to
- **listener** – Name of event messenger

void **registerMessageQueue** (string *queueName*)
Registeres a dispatcher queue.

Parameters **queueName** – String containing the name of queue

void **unregisterMessageListener** (string *queueName*, string *listener*)
Unregisters an event message.

Parameters

- **queueName** – String containing the name of queue
- **listener** – Name of event messenger

void **unregisterMessageQueue** (string *queueName*)
Unregisters a dispatcher queue.

Parameters **queueName** – String containing the name of queue

Packages

Functions relating to the control of packages.

Functions

void **activatePackage** (string *packageName*)
Activates an existing package. The activation occurs by updating the namespace linkage of existing functions and methods. If the package is already activated the function does nothing.

void **deactivatePackage** (string *packageName*)
Deactivates a previously activated package. The package is deactivated by removing its namespace linkages to any function or method. If there are any packages above this one in the stack they are deactivated as well. If the package is not on the stack this function does nothing.

string **getFunctionPackage** (string *funcName*)
Provides the name of the package the function belongs to.

Parameters **funcName** – String containing name of the function

Returns The name of the function's package

string **getMethodPackage** (string, string *method*)
Provides the name of the package the method belongs to.

Parameters

- **namespace** – Class or namespace, such as Player
- **method** – Name of the funciton to search for

Returns The name of the method's package

string **getPackageList** ()
Returns a space delimited list of the active packages in stack order.

bool **isPackage** (string *identifier*)
Returns true if the identifier is the name of a declared package.

Scripting

Functions for working with script code.

Classes

ArrayObject Data structure for storing indexed sequences of key/value pairs.

Inherit: SimObject

Description This is a powerful array class providing PHP style arrays in TorqueScript.

The following features are supported:

- array pointers: this allows you to move forwards or backwards through the array as if it was a list, including jumping to the start or end.
- sorting: the array can be sorted in either alphabetic or numeric mode, on the key or the value, and in ascending or descending order
- add/remove elements: elements can be pushed/popped from the start or end of the array, or can be inserted/erased from anywhere in the middle
- removal of duplicates: remove duplicate keys or duplicate values
- searching: search the array and return the index of a particular key or value
- counting: count the number of instaces of a particular value or key in the array, as well as the total number of elements
- advanced features: array append, array crop and array duplicate

Array element keys and values can be strings or numbers.

Methods

void `ArrayObject::add`(string *key*, string *value*)
Adds a new element to the end of an array (same as push_back()).

Parameters

- **key** – Key for the new element
- **value** – Value for the new element

bool `ArrayObject::append`(ArrayObject *target*)
Appends the target array to the array object.

Parameters **target** – ArrayObject to append to the end of this array

int `ArrayObject::count`()
Get the number of elements in the array.

int `ArrayObject::countKey`(string *key*)
Get the number of times a particular key is found in the array.

Parameters **key** – Key value to count

int `ArrayObject::countValue`(string *value*)
Get the number of times a particular value is found in the array.

Parameters **value** – Array element value to count

bool `ArrayObject::crop`(ArrayObject *target*)
Removes elements with matching keys from array.

Parameters **target** – ArrayObject containing keys to remove from this array

bool ArrayObject::**duplicate**(ArrayObject *target*)
Alters array into an exact duplicate of the target array.

Parameters **target** – ArrayObject to duplicate

void ArrayObject::**echo**()
Echos the array contents to the console.

void ArrayObject::**empty**()
Emptys all elements from an array.

void ArrayObject::**erase**(int *index*)
Removes an element at a specific position from the array.

Parameters **index** – 0-based index of the element to remove

int ArrayObject::**getCurrent**()
Gets the current pointer index.

int ArrayObject::**getIndexFromKey**(string *key*)
Search the array from the current position for the key.

Parameters **value** – Array key to search for

Returns Index of the first element found, or -1 if none

int ArrayObject::**getIndexFromValue**(string *value*)
Search the array from the current position for the element.

Parameters **value** – Array value to search for

Returns Index of the first element found, or -1 if none

string ArrayObject::**getKey**(int *index*)
Get the key of the array element at the submitted index.

Parameters **index** – 0-based index of the array element to get

Returns The key associated with the array element at the specified index, or "" if the index is out of range

string ArrayObject::**getValue**(int *index*)
Get the value of the array element at the submitted index.

Parameters **index** – 0-based index of the array element to get

Returns The value of the array element at the specified index, or "" if the index is out of range

void ArrayObject::**insert**(string *key*, string *value*, int *index*)
Adds a new element to a specified position in the array.

•index = 0 will insert an element at the start of the array (same as push_front())

•index = array. count() will insert an element at the end of the array (same as push_back())

Parameters

- **key** – Key for the new element
- **value** – Value for the new element
- **index** – 0-based index at which to insert the new element

int ArrayObject::**moveFirst**()
Moves array pointer to start of array.

Returns Returns the new array pointer

int ArrayObject::**moveLast**()
Moves array pointer to end of array.

Returns Returns the new array pointer

int ArrayObject::**moveNext**()
Moves array pointer to next position.

Returns Returns the new array pointer, or -1 if already at the end

int ArrayObject::**movePrev**()
Moves array pointer to prev position.

Returns Returns the new array pointer, or -1 if already at the start

void ArrayObject::**pop_back**()
Removes the last element from the array.

void ArrayObject::**pop_front**()
Removes the first element from the array.

void ArrayObject::**push_back**(string *key*, string *value*)
Adds a new element to the end of an array.

Parameters

- **key** – Key for the new element
- **value** – Value for the new element

void ArrayObject::**push_front**(string *key*, string *value*)
Adds a new element to the front of an array.

void ArrayObject::**setCurrent**(int *index*)
Sets the current pointer index.

Parameters **index** – New 0-based pointer index

void ArrayObject::**setKey**(string *key*, int *index*)
Set the key at the given index.

Parameters

- **key** – New key value
- **index** – 0-based index of the array element to update

void ArrayObject::**setValue**(string *value*, int *index*)
Set the value at the given index.

Parameters

- **value** – New array element value
- **index** – 0-based index of the array element to update

void ArrayObject::**sort**(bool *ascending*)
Alpha sorts the array by value.

Parameters **ascending** – [optional] True for ascending sort, false for descending sort

void ArrayObject::**sorta**()
Alpha sorts the array by value in ascending order.

void ArrayObject::**sortd**()
Alpha sorts the array by value in descending order.

void ArrayObject::**sortf**(string *functionName*)
Sorts the array by value in ascending order using the given callback function.

Parameters **functionName** – Name of a function that takes two arguments A and B and returns -1 if A is less, 1 if B is less, and 0 if both are equal.

Example:

```
function mySortCallback(%a, %b)
{
   returnstrcmp( %a.name, %b.name );
}

%array.sortf( "mySortCallback" );
```

void ArrayObject::**sortfd**(string *functionName*)
Sorts the array by value in descending order using the given callback function.

Parameters **functionName** – Name of a function that takes two arguments A and B and returns -1 if A is less, 1 if B is less, and 0 if both are equal.

void ArrayObject::**sortfk**(string *functionName*)
Sorts the array by key in ascending order using the given callback function.

Parameters **functionName** – Name of a function that takes two arguments A and B and returns -1 if A is less, 1 if B is less, and 0 if both are equal.

void ArrayObject::**sortfkd**(string *functionName*)
Sorts the array by key in descending order using the given callback function.

Parameters **functionName** – Name of a function that takes two arguments A and B and returns -1 if A is less, 1 if B is less, and 0 if both are equal.

void ArrayObject::**sortk**(bool *ascending*)
Alpha sorts the array by key.

Parameters **ascending** – [optional] True for ascending sort, false for descending sort

void ArrayObject::**sortka**()
Alpha sorts the array by key in ascending order.

void ArrayObject::**sortkd**()
Alpha sorts the array by key in descending order.

void ArrayObject::**sortn**(bool *ascending*)
Numerically sorts the array by value.

Parameters **ascending** – [optional] True for ascending sort, false for descending sort

void ArrayObject::**sortna**()
Numerically sorts the array by value in ascending order.

void ArrayObject::**sortnd**()
Numerically sorts the array by value in descending order.

void ArrayObject::**sortnk**(bool *ascending*)
Numerically sorts the array by key.

Parameters **ascending** – [optional] True for ascending sort, false for descending sort

void ArrayObject::**sortnka**()
Numerical sorts the array by key in ascending order.

void ArrayObject::**sortnkd**()
Numerical sorts the array by key in descending order.

void ArrayObject::**uniqueKey**()
Removes any elements that have duplicated keys (leaving the first instance).

void ArrayObject::**uniqueValue**()
Removes any elements that have duplicated values (leaving the first instance).

Fields

bool ArrayObject::**caseSensitive**
Makes the keys and values case-sensitive. By default, comparison of key and value strings will be case-insensitive.

caseString ArrayObject::**key**
Helper field which allows you to add new key['keyname'] = value pairs.

ScriptGroup Essentially a SimGroup, but with onAdd and onRemove script callbacks.

Inherit: SimGroup

Description Essentially a SimGroup, but with onAdd and onRemove script callbacks.

Example:

```
// First container, SimGroup containing a ScriptGroupnewSimGroup(Scenes)
{
   // Subcontainer, ScriptGroup containing variables
   // related to a cut scene and a starting
   WayPointnewScriptGroup(WelcomeScene)
   {
      class = "Scene";
      pathName = "Pathx";
      description = "A small orc village set in the Hardesty mountains. This town and its surrounding
      pathTime = "0";
      title = "Welcome to Orc Town";

      newWayPoint(start)
      {
         position = "163.873 -103.82 208.354";
         rotation = "0.136165 -0.0544916 0.989186 44.0527";
         scale = "1 1 1";
         dataBlock = "WayPointMarker";
         team = "0";
      };
   };
};
```

Methods

void ScriptGroup::**onAdd**(SimObjectId *ID*)
Called when this ScriptGroup is added to the system.

Parameters **ID** – Unique object ID assigned when created (this in script).

void ScriptGroup::**onRemove**(SimObjectId *ID*)
Called when this ScriptObject is removed from the system.

Parameters **ID** – Unique object ID assigned when created (this in script).

ScriptObject A script-level OOP object which allows binding of a class, superClass and arguments along with declaration of methods.

Inherit: SimObject

Description ScriptObjects are extrodinarily powerful objects that allow defining of any type of data required. They can optionally have a class and a superclass defined for added control of multiple ScriptObjects through a simple class definition.

Example:

```
newScriptObject(Game)
{
   class = "DeathMatchGame";
   superClass = GameCore;
   genre = "Action FPS"; // Note the new, non-Torque variable
};
```

Methods

void ScriptObject::**onAdd**(SimObjectId *ID*)
Called when this ScriptObject is added to the system.

Parameters **ID** – Unique object ID assigned when created (this in script).

void ScriptObject::**onRemove**(SimObjectId *ID*)
Called when this ScriptObject is removed from the system.

Parameters **ID** – Unique object ID assigned when created (this in script).

ScriptTickObject A ScriptObject that responds to tick and frame events.

Inherit: ScriptObject

Description ScriptTickObject is a ScriptObject that adds callbacks for tick and frame events. Use setProcessTicks() to enable or disable the onInterpolateTick() and onProcessTick() callbacks. The callOnAdvanceTime property determines if the onAdvanceTime() callback is called.

Methods

bool ScriptTickObject::**isProcessingTicks**()
Is this object wanting to receive tick notifications. If this object is set to receive tick notifications then its onInterpolateTick() and onProcessTick() callbacks are called.

Returns True if object wants tick notifications

void ScriptTickObject::**onAdvanceTime**(float *timeDelta*)
This is called every frame regardless if the object is set to process ticks, but only if the callOnAdvanceTime property is set to true.

Parameters **timeDelta** – The time delta for this frame.

void ScriptTickObject::**onInterpolateTick**(float *delta*)
This is called every frame, but only if the object is set to process ticks.

Parameters **delta** – The time delta for this frame.

void ScriptTickObject::**onProcessTick**()
Called once every 32ms if this object is set to process ticks.

void ScriptTickObject::**setProcessTicks**(bool *tick*)
Sets this object as either tick processing or not.

Parameters **tick** – This object's onInterpolateTick() and onProcessTick() callbacks are called if set to true.

Fields

bool ScriptTickObject::**callOnAdvanceTime**
Call the onAdvaceTime() callback.

Functions

string **call**(string *functionName*, string *args*, ...)
Apply the given arguments to the specified global function and return the result of the call.

Parameters **functionName** – The name of the function to call. This function must be in the global namespace, i.e. you cannot call a function in a namespace through call. Use eval() for that.

Returns The result of the function call.

Example:

```
function myFunction( %arg )
{
  return ( %arg SPC "World!" );
}

echo( call( "myFunction", "Hello" ) );
// Prints "Hello World!" to the console.
```

bool **compile**(string *fileName*, bool *overrideNoDSO*)
Compile a file to bytecode. This function will read the TorqueScript code in the specified file, compile it to internal bytecode, and, if DSO generation is enabled or overrideNoDDSO is true, will store the compiled code in a .dso file in the current DSO path mirrorring the path of fileName .

Parameters

- **fileName** – Path to the file to compile to bytecode.
- **overrideNoDSO** – If true, force generation of DSOs even if the engine is compiled to not generate write compiled code to DSO files.

Returns True if the file was successfully compiled, false if not.

void **deleteVariables**(string *pattern*)
Undefine all global variables matching the given name pattern .

Parameters **pattern** – A global variable name pattern. Must begin with '$'.

Example:

```
// Define a global variable in the "My" namespace.
$My::Variable = "value";

// Undefine all variable in the "My" namespace.
deleteVariables( "$My::*" );
```

bool **exec**(string *fileName*, bool *noCalls*, bool *journalScript*)
Execute the given script file.

Parameters

- **fileName** – Path to the file to execute
- **noCalls** – Deprecated
- **journalScript** – Deprecated

Returns True if the script was successfully executed, false if not.

Example:

```
// Execute the init.cs script file found in the
// same directory as the current script file.
exec( "./init.cs" );
```

bool **execPrefs**(string *relativeFileName*, bool *noCalls*, bool *journalScript*)
Manually execute a special script file that contains game or editor preferences.

Parameters

- **relativeFileName** – Name and path to file from project folder
- **noCalls** – Deprecated
- **journalScript** – Deprecated

Returns True if script was successfully executed

void export(string pattern, string filename, bool append)
Write out the definitions of all global variables matching the given name pattern . If fileName is not "", the variable definitions are written to the specified file. Otherwise the definitions will be printed to the console. The output are valid TorqueScript statements that can be executed to restore the global variable values.

Parameters

- **pattern** – A global variable name pattern. Must begin with '$'.
- **filename** – Path of the file to which to write the definitions or "" to write the definitions to the console.
- **append** – If true and fileName is not "", then the definitions are appended to the specified file. Otherwise existing contents of the file (if any) will be overwritten.

Example:

```
// Write out all preference variables to a prefs.cs file.
export( "$prefs::*", "prefs.cs" );
```

string **getDSOPath**(string *scriptFileName*)
Get the absolute path to the file in which the compiled code for the given script file will be stored.

Parameters **scriptFileName** – Path to the .cs script file.

Returns The absolute path to the .dso file for the given script file.

string **getVariable**(string *varName*)
Returns the value of the named variable or an empty string if not found. Name of the variable to search for

Returns Value contained by varName, "" if the variable does not exist

bool **isDefined**(string *varName*)
Determines if a variable exists and contains a value.

Parameters **varName** – Name of the variable to search for

Returns True if the variable was defined in script, false if not

Example:

```
isDefined( "$myVar" );
```

bool **isFunction**(string *funcName*)
Determines if a function exists or not.

Parameters **funcName** – String containing name of the function

Returns True if the function exists, false if not

bool **isMethod**(string, string *method*)
Determines if a class/namespace method exists.

Parameters

- **namespace** – Class or namespace, such as Player
- **method** – Name of the function to search for

Returns True if the method exists, false if not

void **setVariable**(string *varName*, string *value*)
Sets the value of the named variable.

Parameters

- **varName** – Name of the variable to locate
- **value** – New value of the variable

Returns True if variable was successfully found and set

File I/O

Functions allowing you to search for files, read them, write them, and access their properties.

Classes

FileDialog Base class responsible for displaying an OS file browser.

Inherit: SimObject

Description FileDialog is a platform agnostic dialog interface for querying the user for file locations. It is designed to be used through the exposed scripting interface.

FileDialog is the base class for Native File Dialog controls in Torque. It provides these basic areas of functionality:

FileDialog is *not* intended to be used directly in script and is only exposed to script to expose generic file dialog attributes.

This base class is usable in TorqueScript, but is does not specify what functionality is intended (open or save?). Its children, OpenFileDialog and SaveFileDialog, do make use of DialogStyle flags and do make use of specific funcationality. These are the preferred classes to use

However, the FileDialog base class does contain the key properties and important method for file browing. The most important function is Execute(). This is used by both SaveFileDialog and OpenFileDialog to initiate the browser.

Example:

```
// NOTE: This is not he preferred class to use, but this still works
// Create the file dialog
%baseFileDialog = newFileDialog()
{
   // Allow browsing of all file typesfilters = "*.*";

   // No default filedefaultFile = ;

   // Set default path relative to projectdefaultPath = "./";

   // Set the titletitle = "Durpa";

   // Allow changing of path you are browsingchangePath = true;
};

 // Launch the file dialog
 %baseFileDialog.Execute();

 // Dont forget to cleanup
 %baseFileDialog.delete();
```

Methods

bool FileDialog::**Execute**()

Launches the OS file browser. After an Execute() call, the chosen file name and path is available in one of two areas. If only a single file selection is permitted, the results will be stored in the fileName attribute. If multiple file selection is permitted, the results will be stored in the files array. The total number of files in the array will be stored in the fileCount attribute.

Returns True if the file was selected was successfully found (opened) or declared (saved).

Example:

```
    // NOTE: This is not he preferred class to use, but this still works
    // Create the file dialog
    %baseFileDialog = newFileDialog()
    {
       // Allow browsing of all file typesfilters = "*.*";

       // No default filedefaultFile = ;

       // Set default path relative to projectdefaultPath = "./";

       // Set the titletitle = "Durpa";

       // Allow changing of path you are browsingchangePath = true;
    };

     // Launch the file dialog
     %baseFileDialog.Execute();

     // Dont forget to cleanup
     %baseFileDialog.delete();


     // A better alternative is to use the
     // derived classes which are specific to file open and save
```

```
// Create a dialog dedicated to opening files
%openFileDlg = newOpenFileDialog()
{
   // Look for jpg image files
   // First part is the descriptor|second part is the extension
   Filters = "Jepg Files|*.jpg";
   // Allow browsing through other folders
   ChangePath = true;

   // Only allow opening of one file at a time
   MultipleFiles = false;
};

// Launch the open file dialog
%result = %openFileDlg.Execute();

// Obtain the chosen file name and pathif ( %result )
{
   %seletedFile = %openFileDlg.file;
}
else
{
   %selectedFile = "";
}
// Cleanup
%openFileDlg.delete();


// Create a dialog dedicated to saving a file
%saveFileDlg = newSaveFileDialog()
{
   // Only allow for saving of COLLADA files
   Filters = "COLLADA Files (*.dae)|*.dae|";

   // Default save path to where the WorldEditor last saved
   DefaultPath = $pref::WorldEditor::LastPath;

   // No default file specified
   DefaultFile = "";

   // Do not allow the user to change to a new directory
   ChangePath = false;

   // Prompt the user if they are going to overwrite an existing file
   OverwritePrompt = true;
};

// Launch the save file dialog
%result = %saveFileDlg.Execute();

// Obtain the file name
%selectedFile = "";
if ( %result )
   %selectedFile = %saveFileDlg.file;

// Cleanup
%saveFileDlg.delete();
```

Fields

bool `FileDialog::changePath`
: True/False whether to set the working directory to the directory returned by the dialog.

string `FileDialog::defaultFile`
: The default file path when the dialog is shown.

string `FileDialog::defaultPath`
: The default directory path when the dialog is shown.

string `FileDialog::fileName`
: The default file name when the dialog is shown.

string `FileDialog::filters`
: The filter string for limiting the types of files visible in the dialog. It makes use of the pipe symbol 'l' as a delimiter. For example: 'All Files|*.*' 'Image Files|*.png;*.jpg|Png Files|*.png|Jepg Files|*.jpg'

string `FileDialog::title`
: The title for the dialog.

FileObject This class is responsible opening, reading, creating, and saving file contents.

Inherit: SimObject

Description FileObject acts as the interface with OS level files. You create a new FileObject and pass into it a file's path and name. The FileObject class supports three distinct operations for working with files:

Before you may work with a file you need to use one of the three above methods on the FileObject.

Example:

```
// Create a file object for writing
%fileWrite = newFileObject();

// Open a file to write to, if it does not exist it will be created
%result = %fileWrite.OpenForWrite("./test.txt");

if ( %result )
{
   // Write a line to the text files
   %fileWrite.writeLine("READ. READ CODE. CODE");
}

// Close the file when finished
%fileWrite.close();

// Cleanup the file object
%fileWrite.delete();


// Create a file object for reading
%fileRead = newFileObject();

// Open a text file, if it exists
%result = %fileRead.OpenForRead("./test.txt");

if ( %result )
{
   // Read in the first line
   %line = %fileRead.readline();
```

```
   // Print the line we just readecho(%line);
}

// Close the file when finished
%fileRead.close();

// Cleanup the file object
%fileRead.delete();
```

Methods

void FileObject::**close**()

Close the file. It is EXTREMELY important that you call this function when you are finished reading or writing to a file. Failing to do so is not only a bad programming practice, but could result in bad data or corrupt files. Remember: Open, Read/Write, Close, Delete...in that order!

Example:

```
// Create a file object for reading
%fileRead = newFileObject();

// Open a text file, if it exists
%fileRead.OpenForRead("./test.txt");

// Peek the first line
%line = %fileRead.peekLine();

// Print the line we just peekedecho(%line);
// If we peek again...
%line = %fileRead.peekLine();

// We will get the same output as the first time
// since the stream did not move forwardecho(%line);

// Close the file when finished
%fileWrite.close();

// Cleanup the file object
%fileWrite.delete();
```

bool FileObject::**isEOF**()

Determines if the parser for this FileObject has reached the end of the file.

Returns True if the parser has reached the end of the file, false otherwise

Example:

```
// Create a file object for reading
%fileRead = newFileObject();

// Open a text file, if it exists
%fileRead.OpenForRead("./test.txt");

// Keep reading until we reach the end of the file
while(!%fileRead.isEOF())
{
   %line = %fileRead.readline();
   echo(%line);
}
```

```
    // Made it to the endecho("Finished reading file");
```

bool FileObject::**openForAppend**(string *filename*)

Open a specified file for writing, adding data to the end of the file. There is no limit as to what kind of file you can write. Any format and data is allowable, not just text. Unlike openForWrite() , which will erase an existing file if it is opened, openForAppend() preserves data in an existing file and adds to it.

Parameters **filename** – Path, name, and extension of file to append to

Returns True if file was successfully opened, false otherwise

Example:

```
    // Create a file object for writing
    %fileWrite = newFileObject();

    // Open a file to write to, if it does not exist it will be created
    // If it does exist, whatever we write will be added to the end
    %result = %fileWrite.OpenForAppend("./test.txt");
```

bool FileObject::**openForRead**(string *filename*)

Open a specified file for reading. There is no limit as to what kind of file you can read. Any format and data contained within is accessible, not just text

Parameters **filename** – Path, name, and extension of file to be read

Returns True if file was successfully opened, false otherwise

Example:

```
    // Create a file object for reading
    %fileRead = newFileObject();

    // Open a text file, if it exists
    %result = %fileRead.OpenForRead("./test.txt");
```

bool FileObject::**openForWrite**(string *filename*)

Open a specified file for writing. There is no limit as to what kind of file you can write. Any format and data is allowable, not just text

Parameters **filename** – Path, name, and extension of file to write to

Returns True if file was successfully opened, false otherwise

Example:

```
    // Create a file object for writing
    %fileWrite = newFileObject();

    // Open a file to write to, if it does not exist it will be created
    %result = %fileWrite.OpenForWrite("./test.txt");
```

string FileObject::**peekLine**()

Read a line from the file without moving the stream position. Emphasis on *line*, as in you cannot parse individual characters or chunks of data. There is no limitation as to what kind of data you can read. Unlike readLine, the parser does not move forward after reading.

Parameters **filename** – Path, name, and extension of file to be read

Returns String containing the line of data that was just peeked

Example:

```
// Create a file object for reading
%fileRead = newFileObject();

// Open a text file, if it exists
%fileRead.OpenForRead("./test.txt");

// Peek the first line
%line = %fileRead.peekLine();

// Print the line we just peekedecho(%line);
// If we peek again...
%line = %fileRead.peekLine();

// We will get the same output as the first time
// since the stream did not move forward
echo(%line);
```

string FileObject::**readLine**()

Read a line from file. Emphasis on *line*, as in you cannot parse individual characters or chunks of data. There is no limitation as to what kind of data you can read.

Returns String containing the line of data that was just read

Example:

```
// Create a file object for reading
%fileRead = newFileObject();

// Open a text file, if it exists
%fileRead.OpenForRead("./test.txt");

// Read in the first line
%line = %fileRead.readline();

// Print the line we just read
echo(%line);
```

void FileObject::**writeLine**(string *text*)

Write a line to the file, if it was opened for writing. There is no limit as to what kind of text you can write. Any format and data is allowable, not just text. Be careful of what you write, as whitespace, current values, and literals will be preserved.

Parameters **text** – The data we are writing out to file.

Returns True if file was successfully opened, false otherwise

Example:

```
// Create a file object for writing
%fileWrite = newFileObject();

// Open a file to write to, if it does not exist it will be created
%fileWrite.OpenForWrite("./test.txt");

// Write a line to the text files
%fileWrite.writeLine("READ. READ CODE. CODE");
```

void FileObject::**writeObject**(SimObject *object*)

Write an object to a text file. Unlike a simple writeLine using specified strings, this function writes an entire object to file, preserving its type, name, and properties. This is similar to the save() functionality of the SimObject

class, but with a bit more control.

Parameters **object** – The SimObject being written to file, properties, name, and all.

Example:

```
// Lets assume this SpawnSphere was created and currently
// exists in the running level
newSpawnSphere(TestSphere)
{
   spawnClass = "Player";
   spawnDatablock = "DefaultPlayerData";
   autoSpawn = "1";
   radius = "5";
   sphereWeight = "1";
   indoorWeight = "1";
   outdoorWeight = "1";
   dataBlock = "SpawnSphereMarker";
   position = "-42.222 1.4845 4.80334";
   rotation = "0 0 -1 108";
   scale = "1 1 1";
   canSaveDynamicFields = "1";
};

// Create a file object for writing
%fileWrite = newFileObject();

// Open a file to write to, if it does not exist it will be created
%fileWrite.OpenForWrite("./spawnSphers.txt");

// Write out the TestSphere
%fileWrite.writeObject(TestSphere);

// Close the text file
%fileWrite.close();

// Cleanup
%fileWrite.delete();
```

void FileObject::**writeObject**(SimObject *object*, string *prepend*)

Write an object to a text file, with some data added first. Unlike a simple writeLine using specified strings, this function writes an entire object to file, preserving its type, name, and properties. This is similar to the save() functionality of the SimObject class, but with a bit more control.

Parameters

- **object** – The SimObject being written to file, properties, name, and all.
- **prepend** – Data or text that is written out before the SimObject.

Example:

```
// Lets assume this SpawnSphere was created and currently
// exists in the running level
newSpawnSphere(TestSphere)
{
   spawnClass = "Player";
   spawnDatablock = "DefaultPlayerData";
   autoSpawn = "1";
   radius = "5";
   sphereWeight = "1";
```

```
        indoorWeight = "1";
        outdoorWeight = "1";
        dataBlock = "SpawnSphereMarker";
        position = "-42.222 1.4845 4.80334";
        rotation = "0 0 -1 108";
        scale = "1 1 1";
        canSaveDynamicFields = "1";
    };

    // Create a file object for writing
    %fileWrite = newFileObject();

    // Open a file to write to, if it does not exist it will be created
    %fileWrite.OpenForWrite("./spawnSphers.txt");

    // Write out the TestSphere, with a prefix
    %fileWrite.writeObject(TestSphere, "$mySphere = ");

    // Close the text file
    %fileWrite.close();

    // Cleanup
    %fileWrite.delete();
```

FileStreamObject A wrapper around StreamObject for parsing text and data from files.

Inherit: StreamObject

Description FileStreamObject inherits from StreamObject and provides some unique methods for working with strings. If you're looking for general file handling, you may want to use FileObject.

Example:

```
// Create a file stream object for reading
%fsObject = newFileStreamObject();

// Open a file for reading
%fsObject.open("./test.txt", "read");

// Get the status and print it
%status = %fsObject.getStatus();
echo(%status);

// Always remember to close a file stream when finished
%fsObject.close();
```

Methods

void `FileStreamObject::close()`

Close the file. You can no longer read or write to it unless you open it again.

Example:

```
    // Create a file stream object for reading
    %fsObject = newFileStreamObject();

    // Open a file for reading
```

```
    %fsObject.open("./test.txt", "read");

    // Always remember to close a file stream when finished
    %fsObject.close();
```

bool `FileStreamObject::open`(string *filename*, string *openMode*)

Open a file for reading, writing, reading and writing, or appending. Using "Read" for the open mode allows you to parse the contents of file, but not making modifications. "Write" will create a new file if it does not exist, or erase the contents of an existing file when opened. Write also allows you to modify the contents of the file. "ReadWrite" will provide the ability to parse data (read it in) and manipulate data (write it out) interchangeably. Keep in mind the stream can move during each operation. Finally, "WriteAppend" will open a file if it exists, but will not clear the contents. You can write new data starting at the end of the files existing contents.

Parameters

- **filename** – Name of file to open
- **openMode** – One of "Read", "Write", "ReadWrite" or "WriteAppend"

Returns True if the file was successfully opened, false if something went wrong

Example:

```
    // Create a file stream object for reading
    %fsObject = newFileStreamObject();

    // Open a file for reading
    %fsObject.open("./test.txt", "read");

    // Get the status and print it
    %status = %fsObject.getStatus();
    echo(%status);

    // Always remember to close a file stream when finished
    %fsObject.close();
```

OpenFileDialog Derived from FileDialog, this class is responsible for opening a file browser with the intention of opening a file.

Inherit: FileDialog

Description The core usage of this dialog is to locate a file in the OS and return the path and name. This does not handle the actual file parsing or data manipulation. That functionality is left up to the FileObject class.

Example:

```
// Create a dialog dedicated to opening files
 %openFileDlg = newOpenFileDialog()
 {
    // Look for jpg image files
    // First part is the descriptor|second part is the extension
    Filters = "Jepg Files|*.jpg";
    // Allow browsing through other folders
    ChangePath = true;

    // Only allow opening of one file at a time
    MultipleFiles = false;
 };
```

```
// Launch the open file dialog
%result = %openFileDlg.Execute();

// Obtain the chosen file name and pathif ( %result )
{
   %seletedFile = %openFileDlg.file;
}
else
{
   %selectedFile = "";
}

// Cleanup
%openFileDlg.delete();
```

Fields

bool OpenFileDialog::**MultipleFiles**
: True/False whether multiple files may be selected and returned or not.

bool OpenFileDialog::**MustExist**
: True/False whether the file returned must exist or not.

OpenFolderDialog OS level dialog used for browsing folder structures.

Inherit: OpenFileDialog

Description This is essentially an OpenFileDialog, but only used for returning directory paths, not files.

Fields

filename OpenFolderDialog::**fileMustExist**
: File that must be in selected folder for it to be valid.

SaveFileDialog Derived from FileDialog, this class is responsible for opening a file browser with the intention of saving a file.

Inherit: FileDialog

Description The core usage of this dialog is to locate a file in the OS and return the path and name. This does not handle the actual file writing or data manipulation. That functionality is left up to the FileObject class.

Example:

```
// Create a dialog dedicated to opening file
 %saveFileDlg = newSaveFileDialog()
 {
    // Only allow for saving of COLLADA files
    Filters        = "COLLADA Files (*.dae)|*.dae|";

    // Default save path to where the WorldEditor last saved
    DefaultPath    = $pref::WorldEditor::LastPath;

    // No default file specified
    DefaultFile    = "";
```

```
   // Do not allow the user to change to a new directory
   ChangePath     = false;

   // Prompt the user if they are going to overwrite an existing file
   OverwritePrompt   = true;
};

// Launch the save file dialog
%saveFileDlg.Execute();

if ( %result )
{
   %seletedFile = %openFileDlg.file;
}
else
{
   %selectedFile = "";
}

// Cleanup
%saveFileDlg.delete();
```

Fields

bool SaveFileDialog::**OverwritePrompt**

True/False whether the dialog should prompt before accepting an existing file name.

SimXMLDocument File I/O object used for creating, reading, and writing XML documents.

Inherit: SimObject

Description A SimXMLDocument is a container of various XML nodes. The Document level may contain a header (sometimes called a declaration), comments and child Elements. Elements may contain attributes, data (or text) and child Elements.

You build new Elements using addNewElement(). This makes the new Element the current one you're working with. You then use setAttribute() to add attributes to the Element. You use addData() or addText() to write to the text area of an Element.

Example:

```
// Thanks to Rex Hiebert for this example
// Given the following XML
// <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
// <DataTables>
//   <table tableName="2DShapes">
//      <rec id="1">Triangle</rec>
//      <rec id="2">Square</rec><rec id="3">Circle</rec>
//    </table>
//   <table tableName="3DShapes">
//      <rec id="1">Pyramid</rec>
//      <rec id="2">Cube</rec>
//      <rec id="3">Sphere</rec>
//   </table>
// </DataTables>

// Using SimXMLDocument by itself
```

```
function readXmlExample(%filename)
{
   %xml = newSimXMLDocument() {};
   %xml.loadFile(%filename);

   %xml.pushChildElement("DataTables");
   %xml.pushFirstChildElement("table");
   while(true)
   {
     echo("TABLE:" SPC %xml.attribute("tableName"));
     %xml.pushFirstChildElement("rec");
     while (true)
     {
       %id = %xml.attribute("id");
       %desc = %xml.getData();
       echo("  Shape" SPC %id SPC %desc);
       if (!%xml.nextSiblingElement("rec")) break;
     }
     %xml.popElement();
     if (!%xml.nextSiblingElement("table")) break;
   }
}

// Thanks to Scott Peal for this example
// Using FileObject in conjunction with SimXMLDocument
// This example uses an XML file with a format of:
// <Models>
//    <Model category="" name="" path="" />
// </Models>
function getModelsInCatagory()
{
   %file = "./Catalog.xml";
   %fo = newFileObject();
   %text = "";

   if(%fo.openForRead(%file))
   {
     while(!%fo.isEOF())
     {
       %text = %text @ %fo.readLine();
       if (!%fo.isEOF()) %text = %text @ "\n";
     }
   }
   else
   {
     echo("Unable to locate the file: " @ %file);
   }

   %fo.delete();

   %xml = newSimXMLDocument() {};
   %xml.parse(%text);
   // "Get" inside of the root element, "Models".
   %xml.pushChildElement(0);

   // "Get" into the first child element
   if (%xml.pushFirstChildElement("Model"))
   {
```

```
     while (true)
     {
       // Here, i read the elements attributes.
       // You might want to save these values in an array or call the %xml.getElementValue()
       // if you have a different XML structure.

       %catagory = %xml.attribute("catagory");
       %name = %xml.attribute("name");
       %path = %xml.attribute("path");

       // now, read the next "Model"
       if (!%xml.nextSiblingElement("Model")) break;
     }
   }
}
```

Methods

void SimXMLDocument::**addComment**(string *comment*)

Add the given comment as a child of the document.

Parameters **comment** – String containing the comment.

Example:

```
// Create a new XML document with a header, a comment and single element.
%x = newSimXMLDocument();
%x.addHeader();
%x.addComment("This is a test comment");
%x.addNewElement("NewElement");
%x.saveFile("test.xml");

// Produces the following file:
// <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
// <!--This is a test comment-->
// <NewElement />
```

void SimXMLDocument::**addData**(string *text*)

Add the given text as a child of current Element. Use getData() to retrieve any text from the current Element. addData() and addText() may be used interchangeably. As there is no difference between data and text, you may also use removeText() to clear any data from the current Element.

Parameters **text** – String containing the text.

Example:

```
// Create a new XML document with a header and single element// with some added data.
%x = newSimXMLDocument();
%x.addHeader();
%x.addNewElement("NewElement");
%x.addData("Some text");
%x.saveFile("test.xml");

// Produces the following file:
// <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
// <NewElement>Some text</NewElement>
```

void SimXMLDocument::**addHeader**()

Add a XML header to a document. Sometimes called a declaration, you typically add a standard header to

the document before adding any elements. SimXMLDocument always produces the following header: lt ?xml version=”1.0” encoding=”utf-8” standalone=”yes” ? gt

Example:

```
// Create a new XML document with just a header and single element.
%x = newSimXMLDocument();
%x.addHeader();
%x.addNewElement("NewElement");
%x.saveFile("test.xml");

// Produces the following file:
// <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
// <NewElement />
```

void SimXMLDocument::**addNewElement**(string *name*)

Create a new element with the given name as child of current Element's parent and push it onto the Element stack making it the current one.

Parameters **name** – XML tag for the new Element.

void SimXMLDocument::**addText**(string *text*)

Add the given text as a child of current Element. Use getText() to retrieve any text from the current Element and removeText() to clear any text. addText() and addData() may be used interchangeably.

Parameters **text** – String containing the text.

Example:

```
// Create a new XML document with a header and single element// with some added text.
%x = newSimXMLDocument();
%x.addHeader();
%x.addNewElement("NewElement");
%x.addText("Some text");
%x.saveFile("test.xml");

// Produces the following file:
// <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
// <NewElement>Some text</NewElement>
```

string SimXMLDocument::**attribute**(string *attributeName*)

Get a string attribute from the current Element on the stack.

Parameters **attributeName** – Name of attribute to retrieve.

Returns The attribute string if found. Otherwise returns an empty string.

bool SimXMLDocument::**attributeExists**(string *attributeName*)

Tests if the requested attribute exists.

Parameters **attributeName** – Name of attribute being queried for.

Returns True if the attribute exists.

float SimXMLDocument::**attributeF32**(string *attributeName*)

Get float attribute from the current Element on the stack.

Parameters **attributeName** – Name of attribute to retrieve.

Returns The value of the given attribute in the form of a float.

int SimXMLDocument::**attributeS32**(string *attributeName*)

Get int attribute from the current Element on the stack.

Parameters **attributeName** – Name of attribute to retrieve.

Returns The value of the given attribute in the form of an integer.

void SimXMLDocument::**clear**()
Set this document to its default state. Clears all Elements from the documents. Equivalent to using reset()

void SimXMLDocument::**clearError**()
Clear the last error description.

string SimXMLDocument::**elementValue**()
Get the Element's value if it exists. Usually returns the text from the Element.

Returns The value from the Element, or an empty string if none is found.

string SimXMLDocument::**firstAttribute**()
Obtain the name of the current Element's first attribute.

Returns String containing the first attribute's name, or an empty string if none is found.

string SimXMLDocument::**getData**()
Gets the text from the current Element. Use addData() to add text to the current Element. getData() and getText() may be used interchangeably. As there is no difference between data and text, you may also use removeText() to clear any data from the current Element.

Returns String containing the text in the current Element.

Example:

```
// Using the following test.xml file as an example:
// <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
// <NewElement>Some data</NewElement>
// Load in the file
%x = newSimXMLDocument();
%x.loadFile("test.xml");

// Make the first Element the current one
%x.pushFirstChildElement("NewElement");

// Store the current Elements data (Some data in this example)
// into result
%result = %x.getData();
echo( %result );
```

string SimXMLDocument::**getErrorDesc**()
Get last error description.

Returns A string of the last error message.

string SimXMLDocument::**getText**()
Gets the text from the current Element. Use addText() to add text to the current Element and removeText() to clear any text. getText() and getData() may be used interchangeably.

Returns String containing the text in the current Element.

Example:

```
// Using the following test.xml file as an example:
// <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
// <NewElement>Some text</NewElement>
// Load in the file
%x = newSimXMLDocument();
%x.loadFile("test.xml");
```

```
    // Make the first Element the current one
    %x.pushFirstChildElement("NewElement");

    // Store the current Elements text (Some text in this example)
    // into result
    %result = %x.getText();
    echo( %result );
```

string SimXMLDocument::**lastAttribute**()
Obtain the name of the current Element's last attribute.

Returns String containing the last attribute's name, or an empty string if none is found.

bool SimXMLDocument::**loadFile**(string *fileName*)
Load in given filename and prepare it for use.

Parameters **fileName** – Name and path of XML document

Returns True if the file was loaded successfully.

string SimXMLDocument::**nextAttribute**()
Get the name of the next attribute for the current Element after a call to firstAttribute() .

Returns String containing the next attribute's name, or an empty string if none is found.

bool SimXMLDocument::**nextSiblingElement**(string *name*)
Put the next sibling Element with the given name on the stack, making it the current one.

Parameters **name** – String containing name of the next sibling.

Returns True if the Element was found and made the current one.

void SimXMLDocument::**parse**(string *xmlString*)
Create a document from a XML string.

Parameters **xmlString** – Valid XML to parse and store as a document.

void SimXMLDocument::**popElement**()
Pop the last Element off the stack.

string SimXMLDocument::**prevAttribute**()
Get the name of the previous attribute for the current Element after a call to lastAttribute() .

Returns String containing the previous attribute's name, or an empty string if none is found.

bool SimXMLDocument::**pushChildElement**(int *index*)
Push the child Element at the given index onto the stack, making it the current one.

Parameters **index** – Numerical index of Element being pushed.

Returns True if the Element was found and made the current one.

bool SimXMLDocument::**pushFirstChildElement**(string *name*)
Push the first child Element with the given name onto the stack, making it the current Element.

Parameters **name** – String containing name of the child Element.

Returns True if the Element was found and made the current one.

Example:

```
    // Using the following test.xml file as an example:
    // <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
    // <NewElement>Some text</NewElement>
    // Load in the file
```

```
%x = newSimXMLDocument();
%x.loadFile("test.xml");

// Make the first Element the current one
%x.pushFirstChildElement("NewElement");

// Store the current Elements text (Some text in this example)
// into result
%result = %x.getText();
echo( %result );
```

void SimXMLDocument::**pushNewElement**(string *name*)
Create a new element with the given name as child of current Element and push it onto the Element stack making it the current one.

Parameters **name** – XML tag for the new Element.

string SimXMLDocument::**readComment**(int *index*)
Gives the comment at the specified index, if any. Unlike addComment() that only works at the document level, readComment() may read comments from the document or any child Element. The current Element (or document if no Elements have been pushed to the stack) is the parent for any comments, and the provided index is the number of comments in to read back.

Parameters **index** – Comment index number to query from the current Element stack

Returns String containing the comment, or an empty string if no comment is found.

void SimXMLDocument::**removeText**()
Remove any text on the current Element. Use getText() to retrieve any text from the current Element and addText() to add text to the current Element. As getData() and addData() are equivalent to getText() and addText() , removeText() will also remove any data from the current Element.

void SimXMLDocument::**reset**()
Set this document to its default state. Clears all Elements from the documents. Equivalent to using clear()

bool SimXMLDocument::**saveFile**(string *fileName*)
Save document to the given file name.

Parameters **fileName** – Path and name of XML file to save to.

Returns True if the file was successfully saved.

void SimXMLDocument::**setAttribute**(string *attributeName*, string *value*)
Set the attribute of the current Element on the stack to the given value.

Parameters

- **attributeName** – Name of attribute being changed
- **value** – New value to assign to the attribute

void SimXMLDocument::**setObjectAttributes**(string *objectID*)
Add the given SimObject's fields as attributes of the current Element on the stack.

Parameters **objectID** – ID of SimObject being copied.

StreamObject Base class for working with streams.

Inherit: SimObject

Description You do not instantiate a StreamObject directly. Instead, it is used as part of a FileStreamObject and ZipObject to support working with uncompressed and compressed files respectively.

Example:

```
// You cannot actually declare a StreamObject
// Instead, use the derived class "FileStreamObject"
%fsObject = FileStreamObject();
```

Methods

bool StreamObject::**copyFrom**(SimObject *other*)

Copy from another StreamObject into this StreamObject .

Parameters **other** – The StreamObject to copy from.

Returns True if the copy was successful.

int StreamObject::**getPosition**()

Gets the position in the stream. The easiest way to visualize this is to think of a cursor in a text file. If you have moved the cursor by five characters, the current position is 5. If you move ahead 10 more characters, the position is now 15. For StreamObject , when you read in the line the position is increased by the number of characters parsed, the null terminator, and a newline.

Returns Number of bytes which stream has parsed so far, null terminators and newlines are included

Example:

```
// Create a file stream object for reading
%fsObject = newFileStreamObject();

// Open a file for reading
// This file contains two lines of text repeated:
// Hello World
// Hello World
%fsObject.open("./test.txt", "read");

// Read in the first line
%line = %fsObject.readLine();

// Get the position of the stream
%position = %fsObject.getPosition();

// Print the current position
// Should be 13, 10 for the words, 1 for the space,
// 1 for the null terminator, and 1 for the newline
echo(%position);

// Always remember to close a file stream when finished
%fsObject.close();
```

string StreamObject::**getStatus**()

Gets a printable string form of the stream's status. OK - Stream is active and no file errors IOError - Something went wrong during read or writing the stream EOS - End of Stream reached (mostly for reads) IllegalCall - An unsupported operation used. Always w/ accompanied by AssertWarn Closed - Tried to operate on a closed stream (or detached filter) UnknownError - Catch all for an error of some kind Invalid - Entire stream is invalid

Returns String containing status constant, one of the following:

Example:

```
// Create a file stream object for reading
%fsObject = newFileStreamObject();

// Open a file for reading
%fsObject.open("./test.txt", "read");

// Get the status and print it
%status = %fsObject.getStatus();
echo(%status);

// Always remember to close a file stream when finished
%fsObject.close();
```

int StreamObject::**getStreamSize**()

Gets the size of the stream. The size is dependent on the type of stream being used. If it is a file stream, returned value will be the size of the file. If it is a memory stream, it will be the size of the allocated buffer.

Returns Size of stream, in bytes

Example:

```
// Create a file stream object for reading
%fsObject = newFileStreamObject();

// Open a file for reading
// This file contains the following two lines:
// HelloWorld
// HelloWorld
%fsObject.open("./test.txt", "read");

// Found out how large the file stream is
// Then print it to the console
// Should be 22
%streamSize = %fsObject.getStreamSize();
echo(%streamSize);

// Always remember to close a file stream when finished
%fsObject.close();
```

bool StreamObject::**isEOF**()

Tests if the stream has reached the end of the file. This is an alternative name for isEOS. Both functions are interchangeable. This simply exists for those familiar with some C++ file I/O standards.

Returns True if the parser has reached the end of the file, false otherwise

Example:

```
// Create a file stream object for reading
%fsObject = newFileStreamObject();

// Open a file for reading
%fsObject.open("./test.txt", "read");

// Keep reading until we reach the end of the file
while(!%fsObject.isEOF())
{
   %line = %fsObject.readLine();
   echo(%line);
}
// Made it to the end
```

```
echo("Finished reading file");

// Always remember to close a file stream when finished
%fsObject.close();
```

bool StreamObject::**isEOS**()

Tests if the stream has reached the end of the file. This is an alternative name for isEOF. Both functions are interchangeable. This simply exists for those familiar with some C++ file I/O standards.

Returns True if the parser has reached the end of the file, false otherwise

Example:

```
// Create a file stream object for reading
%fsObject = newFileStreamObject();

// Open a file for reading
%fsObject.open("./test.txt", "read");

// Keep reading until we reach the end of the file
while(!%fsObject.isEOS())
{
   %line = %fsObject.readLine();
   echo(%line);
}
// Made it to the end
echo("Finished reading file");

// Always remember to close a file stream when finished
%fsObject.close();
```

string StreamObject::**readLine**()

Read a line from the stream. Emphasis on *line*, as in you cannot parse individual characters or chunks of data. There is no limitation as to what kind of data you can read.

Returns String containing the line of data that was just read

Example:

```
// Create a file stream object for reading
// This file contains the following two lines:
// HelloWorld
// HelloWorld
%fsObject = newFileStreamObject();

%fsObject.open("./test.txt", "read");

// Read in the first line
%line = %fsObject.readLine();

// Print the line we just read
echo(%line);

// Always remember to close a file stream when finished
%fsObject.close();
```

String StreamObject::**readLongString**(int *maxLength*)

Read in a string up to the given maximum number of characters.

Parameters **maxLength** – The maximum number of characters to read in.

Returns The string that was read from the stream.

String `StreamObject::readString`()
Read a string up to a maximum of 256 characters.

Returns The string that was read from the stream.

String `StreamObject::readSTString`(bool *caseSensitive*)
Read in a string and place it on the string table.

Parameters **`caseSensitive`** – If false then case will not be taken into account when attempting to match the read in string with what is already in the string table.

Returns The string that was read from the stream.

bool `StreamObject::setPosition`(int *newPosition*)
Gets the position in the stream. The easiest way to visualize this is to think of a cursor in a text file. If you have moved the cursor by five characters, the current position is 5. If you move ahead 10 more characters, the position is now 15. For StreamObject , when you read in the line the position is increased by the number of characters parsed, the null terminator, and a newline. Using setPosition allows you to skip to specific points of the file.

Returns Number of bytes which stream has parsed so far, null terminators and newlines are included

Example:

```
// Create a file stream object for reading
%fsObject = newFileStreamObject();

// Open a file for reading
// This file contains the following two lines:
// 11111111111
// Hello World
%fsObject.open("./test.txt", "read");

// Skip ahead by 12, which will bypass the first line entirely
%fsObject.setPosition(12);

// Read in the next line
%line = %fsObject.readLine();

// Print the line just read in, should be "Hello World"
echo(%line);

// Always remember to close a file stream when finished
%fsObject.close();
```

void `StreamObject::writeLine`(string *line*)
Write a line to the stream, if it was opened for writing. There is no limit as to what kind of data you can write. Any format and data is allowable, not just text. Be careful of what you write, as whitespace, current values, and literals will be preserved.

Parameters **`line`** – The data we are writing out to file.

Example:

```
// Create a file stream
%fsObject = newFileStreamObject();

// Open the file for writing
// If it does not exist, it is created.
// If it does exist, the file is cleared
```

```
    %fsObject.open("./test.txt", "write");

    // Write a line to the file
    %fsObject.writeLine("Hello World");

    // Write another line to the file
    %fsObject.writeLine("Documentation Rocks!");

    // Always remember to close a file stream when finished
    %fsObject.close();
```

void StreamObject::**writeLongString**(int *maxLength*, string *string*)
Write out a string up to the maximum number of characters.

Parameters

- **maxLength** – The maximum number of characters that will be written.
- **string** – The string to write out to the stream.

void StreamObject::**writeString**(string *string*, int *maxLength*)
Write out a string with a default maximum length of 256 characters.

Parameters

- **string** – The string to write out to the stream
- **maxLength** – The maximum string length to write out with a default of 256 characters. This value should not be larger than 256 as it is written to the stream as a single byte.

ZipObject Provides access to a zip file.

Inherit: SimObject

Description A ZipObject add, delete and extract files that are within a zip archive. You may also read and write directly to the files within the archive by obtaining a StreamObject for the file.

Example:

```
// Open a zip archive, creating it if it doesn't exist
%archive = newZipObject();
%archive.openArchive("testArchive.zip", Write);

// Add a file to the archive with the given name
%archive.addFile("./water.png", "water.png");

// Close the archive to save the changes
%archive.closeArchive();
```

Methods

bool ZipObject::**addFile**(string *filename*, string *pathInZip*, bool *replace*)
Add a file to the zip archive.

Parameters

- **filename** – The path and name of the file to add to the zip archive.
- **pathInZip** – The path and name to be given to the file within the zip archive.

- **replace** – If a file already exists within the zip archive at the same location as this new file, this parameter indicates if it should be replaced. By default, it will be replaced.

Returns True if the file was successfully added to the zip archive.

void ZipObject::**closeArchive**()
Close an already opened zip archive.

void ZipObject::**closeFile**(SimObject *stream*)
Close a previously opened file within the zip archive.

Parameters **stream** – The StreamObject of a previously opened file within the zip archive.

bool ZipObject::**deleteFile**(string *pathInZip*)
Deleted the given file from the zip archive.

Parameters **pathInZip** – The path and name of the file to be deleted from the zip archive.

Returns True of the file was successfully deleted.

bool ZipObject::**extractFile**(string *pathInZip*, string *filename*)
Extact a file from the zip archive and save it to the requested location.

Parameters

- **pathInZip** – The path and name of the file to be extracted within the zip archive.
- **filename** – The path and name to give the extracted file.

Returns True if the file was successfully extracted.

String ZipObject::**getFileEntry**(int *index*)
Get information on the requested file within the zip archive. This methods provides five different pieces of information for the requested file:

•filename - The path and name of the file within the zip archive

•uncompressed size

•compressed size

•compression method

•CRC32

Use getFileEntryCount() to obtain the total number of files within the archive.

Parameters **index** – The index of the file within the zip archive. Use getFileEntryCount() to determine the number of files.

Returns A tab delimited list of information on the requested file, or an empty string if the file could not be found.

int ZipObject::**getFileEntryCount**()
Get the number of files within the zip archive. Use getFileEntry() to retrive information on each file within the archive.

Returns The number of files within the zip archive.

bool ZipObject::**openArchive**(string *filename*, string *accessMode*)
Open a zip archive for manipulation. Once a zip archive is opened use the various ZipObject methods for working with the files within the archive. Be sure to close the archive when you are done with it.

Parameters

- **filename** – The path and file name of the zip archive to open.
- **accessMode** – One of read, write or readwrite

Returns True is the archive was successfully opened.

SimObject `ZipObject::openFileForRead`(string *filename*)
Open a file within the zip archive for reading. Be sure to close the file when you are done with it.

Parameters **filename** – The path and name of the file to open within the zip archive.

Returns is returned for working with the file.

SimObject `ZipObject::openFileForWrite`(string *filename*)
Open a file within the zip archive for writing to. Be sure to close the file when you are done with it.

Parameters **filename** – The path and name of the file to open within the zip archive.

Returns is returned for working with the file.

Functions

bool **createPath**(string *path*)
Create the given directory or the path leading to the given filename. If path ends in a trailing slash, then all components in the given path will be created as directories (if not already in place). If path , does not end in a trailing slash, then the last component of the path is taken to be a file name and only the directory components of the path will be created.

Parameters **path** – The path to create.

string **expandFilename**(string *filename*)
Grabs the full path of a specified file.

Parameters **filename** – Name of the local file to locate

Returns String containing the full filepath on disk

string **expandOldFilename**(string *filename*)
Retrofits a filepath that uses old Torque style.

Returns String containing filepath with new formatting

String **fileBase**(string *fileName*)
Get the base of a file name (removes extension).

Parameters **fileName** – Name and path of file to check

Returns String containing the file name, minus extension

String **fileCreatedTime**(string *fileName*)
Returns a platform specific formatted string with the creation time for the file.

Parameters **fileName** – Name and path of file to check

Returns Formatted string (OS specific) containing created time, "9/3/2010 12:33:47 PM" for example

bool **fileDelete**(string *path*)
Delete a file from the hard drive.

Parameters **path** – Name and path of the file to delete

Returns True if file was successfully deleted

String **fileExt**(string *fileName*)
Get the extension of a file.

Parameters **fileName** – Name and path of file

Returns String containing the extension, such as ".exe" or ".cs"

String **fileModifiedTime**(string *fileName*)
Returns a platform specific formatted string with the last modified time for the file.

Parameters **fileName** – Name and path of file to check

Returns Formatted string (OS specific) containing modified time, "9/3/2010 12:33:47 PM" for example

String **fileName**(string *fileName*)
Get the file name of a file (removes extension and path).

Parameters **fileName** – Name and path of file to check

Returns String containing the file name, minus extension and path

String **filePath**(string *fileName*)
Get the path of a file (removes name and extension).

Parameters **fileName** – Name and path of file to check

Returns String containing the path, minus name and extension

int **fileSize**(string *fileName*)
Determines the size of a file on disk.

Parameters **fileName** – Name and path of the file to check

Returns Returns filesize in KB, or -1 if no file

String **getCurrentDirectory**()
Return the current working directory.

Returns The absolute path of the current working directory.

String **getDirectoryList**(string *path*, int *depth*)
Gathers a list of directories starting at the given path.

Parameters

- **path** – String containing the path of the directory
- **depth** – Depth of search, as in how many subdirectories to parse through

Returns Tab delimited string containing list of directories found during search, "" if no files were found

String **getExecutableName**()
Gets the name of the game's executable.

Returns String containing this game's executable name

int **getFileCRC**(string *fileName*)
Provides the CRC checksum of the given file.

Parameters **fileName** – The path to the file.

Returns The calculated CRC checksum of the file, or -1 if the file could not be found.

String **getMainDotCsDir**()
Get the absolute path to the directory that contains the main.cs script from which the engine was started. This directory will usually contain all the game assets and, in a user-side game installation, will usually be read-only.

Returns The path to the main game assets.

String **getWorkingDirectory**()
Reports the current directory.

Returns String containing full file path of working directory

bool **IsDirectory**(string *directory*)
Determines if a specified directory exists or not.

Parameters **directory** – String containing path in the form of "foo/bar"

Returns Returns true if the directory was found.

bool **isFile**(string *fileName*)
Determines if the specified file exists or not.

Parameters **fileName** – The path to the file.

Returns Returns true if the file was found.

bool **isWriteableFileName**(string *fileName*)
Determines if a file name can be written to using File I/O.

Parameters **fileName** – Name and path of file to check

Returns Returns true if the file can be written to.

String **makeFullPath**(string *path*, string *cwd*)
Converts a relative file path to a full path. For example, "./console.log" becomes "C:/Torque/t3d/examples/FPS Example/game/console.log"

Parameters

- **path** – Name of file or path to check
- **cwd** – Optional current working directory from which to build the full path.

Returns String containing non-relative directory of path

String **makeRelativePath**(string *path*, string *to*)
Turns a full or local path to a relative one. For example, "./game/art" becomes "game/art"

Parameters

- **path** – Full path (may include a file) to convert
- **to** – Optional base path used for the conversion. If not supplied the current working directory is used.

Returns String containing relative path

void **openFile**(string *file*)
Open the given file through the system. This will usually open the file in its associated application.

Parameters **file** – Path of the file to open.

void **openFolder**(string *path*)
Open the given folder in the system's file manager.

Parameters **path** – full path to a directory.

String **pathConcat**(string *path*, string *file*)
Combines two separate strings containing a file path and file name together into a single string.

Parameters

- **path** – String containing file path
- **file** – String containing file name

Returns String containing concatenated file name and path

bool **pathCopy**(string *fromFile*, string *toFile*, bool *noOverwrite*)
Copy a file to a new location.

Parameters

- **fromFile** – Path of the file to copy.
- **toFile** – Path where to copy fromFile to.
- **noOverwrite** – If true, then fromFile will not overwrite a file that may already exist at toFile.

Returns True if the file was successfully copied, false otherwise.

bool **setCurrentDirectory**(string *path*)
Set the current working directory.

Parameters **path** – The absolute or relative (to the current working directory) path of the directory which should be made the new working directory.

Returns , false otherwise.

void **startFileChangeNotifications**()
Start watching resources for file changes. Typically this is called during initializeCore().

void **stopFileChangeNotifications**()
Stop watching resources for file changes. Typically this is called during shutdownCore().

Variables

string $Con::File
The currently executing script file.

string $Con::Root
The mod folder for the currently executing script file.

File Searching

Functions for searching files by name patterns.

Functions
String **findFirstFile**(string *pattern*, bool *recurse*)
Returns the first file in the directory system matching the given pattern. Use the corresponding findNextFile() to step through the results. If you're only interested in the number of files returned by the pattern match, use getFileCount() . This function differs from findFirstFileMultiExpr() in that it supports a single search pattern being passed in.

Parameters

- **pattern** – The path and file name pattern to match against.
- **recurse** – If true, the search will exhaustively recurse into subdirectories of the given path and match the given filename pattern.

Returns The path of the first file matched by the search or an empty string if no matching file could be found.

Example:

```
// Execute all .cs files in a subdirectory and its subdirectories.
for( %file = findFirstFile( "subdirectory/*.cs" ); %file !$= ""; %file = findNextFile() )
   exec( %file );
```

String **findFirstFileMultiExpr**(string *pattern*, bool *recurse*)

Returns the first file in the directory system matching the given patterns. Use the corresponding findNextFileMultiExpr() to step through the results. If you're only interested in the number of files returned by the pattern match, use getFileCountMultiExpr() . This function differs from findFirstFile() in that it supports multiple search patterns to be passed in.

Parameters

- **pattern** – The path and file name pattern to match against, such as .cs. Separate multiple patterns with TABs. For example: ".cs" TAB ".dso"
- **recurse** – If true, the search will exhaustively recurse into subdirectories of the given path and match the given filename patterns.

Returns String of the first matching file path, or an empty string if no matching files were found.

Example:

```
// Find all DTS or Collada models
%filePatterns = "*.dts" TAB "*.dae";
%fullPath = findFirstFileMultiExpr( %filePatterns );
while ( %fullPath !$= "" )
{
   echo( %fullPath );
   %fullPath = findNextFileMultiExpr( %filePatterns );
}
```

String **findNextFile**(string *pattern*)

Returns the next file matching a search begun in findFirstFile() .

Parameters **pattern** – The path and file name pattern to match against. This is optional and may be left out as it is not used by the code. It is here for legacy reasons.

Returns The path of the next filename matched by the search or an empty string if no more files match.

Example:

```
// Execute all .cs files in a subdirectory and its subdirectories.
for( %file = findFirstFile( "subdirectory/*.cs" ); %file !$= ""; %file = findNextFile() )
   exec( %file );
```

String **findNextFileMultiExpr**(string *pattern*)

Returns the next file matching a search begun in findFirstFileMultiExpr() .

Parameters **pattern** – The path and file name pattern to match against. This is optional and may be left out as it is not used by the code. It is here for legacy reasons.

Returns String of the next matching file path, or an empty string if no matching files were found.

Example:

```
// Find all DTS or Collada models
%filePatterns = "*.dts" TAB "*.dae";
%fullPath = findFirstFileMultiExpr( %filePatterns );
while ( %fullPath !$= "" )
{
   echo( %fullPath );
```

```
      %fullPath = findNextFileMultiExpr( %filePatterns );
   }
```

int **getFileCount**(string *pattern*, bool *recurse*)

Returns the number of files in the directory tree that match the given patterns. This function differs from getFileCountMultiExpr() in that it supports a single search pattern being passed in. If you're interested in a list of files that match the given pattern and not just the number of files, use findFirstFile() and findNextFile() .

Parameters

- **pattern** – The path and file name pattern to match against.
- **recurse** – If true, the search will exhaustively recurse into subdirectories of the given path and match the given filename pattern counting files in subdirectories.

Returns Number of files located using the pattern

Example:

```
// Count the number of .cs files in a subdirectory and its subdirectories.
getFileCount( "subdirectory/*.cs" );
```

int **getFileCountMultiExpr**(string *pattern*, bool *recurse*)

Returns the number of files in the directory tree that match the given patterns. If you're interested in a list of files that match the given patterns and not just the number of files, use findFirstFileMultiExpr() and findNextFileMultiExpr() .

Parameters

- **pattern** – The path and file name pattern to match against, such as .cs. Separate multiple patterns with TABs. For example: ".cs" TAB ".dso"
- **recurse** – If true, the search will exhaustively recurse into subdirectories of the given path and match the given filename pattern.

Returns Number of files located using the patterns

Example:

```
// Count all DTS or Collada models
%filePatterns = "*.dts" TAB "*.dae";
echo( "Nunmer of shape files:" SPC getFileCountMultiExpr( %filePatterns ) );
```

Math

Functions for dealing with vectors and matrices etc.

Functions

Point3F **getBoxCenter**(Box3F *box*)

Get the center point of an axis-aligned box.

Parameters **b** – A Box3F, in string format using "minExtentX minExtentY minExtentZ maxExtentX maxExtentY maxExtentZ"

Returns Center of the box.

float **getMax**(float *v1*, float *v2*)

Calculate the greater of two specified numbers.

Parameters

- **v1** – Input value.
- **v2** – Input value.

Returns The greater value of the two specified values.

float **getMin**(float *v1*, float *v2*)
Calculate the lesser of two specified numbers.

Parameters

- **v1** – Input value.
- **v2** – Input value.

Returns The lesser value of the two specified values.

float **m2Pi**()
Return the value of 2*PI (full-circle in radians).

Returns The value of 2*PI.

float **mAbs**(float *v*)
Calculate absolute value of specified value.

Parameters **v** – Input Value.

Returns Absolute value of specified value.

float **mAcos**(float *v*)
Calculate the arc-cosine of v.

Parameters **v** – Input Value (in radians).

Returns The arc-cosine of the input value.

float **mAsin**(float *v*)
Calculate the arc-sine of v.

Parameters **v** – Input Value (in radians).

Returns The arc-sine of the input value.

float **mAtan**(float *rise*, float *run*)
Calculate the arc-tangent (slope) of a line defined by rise and run.

Parameters

- **rise** – of line.
- **run** – of line.

Returns The arc-tangent (slope) of a line defined by rise and run.

void **mathInit**(...)
Install the math library with specified extensions. Possible parameters are:

•'DETECT' Autodetect math lib settings.

•'C' Enable the C math routines. C routines are always enabled.

•'FPU' Enable floating point unit routines.

•'MMX' Enable MMX math routines.

•'3DNOW' Enable 3dNow! math routines.

- 'SSE' Enable SSE math routines.

int **mCeil** (float *v*)
Round v up to the nearest integer.

Parameters **v** – Number to convert to integer.

Returns Number converted to integer.

float **mClamp** (float *v*, float *min*, float *max*)
Clamp the specified value between two bounds.

Parameters

- **v** – Input value.
- **min** – Minimum Bound.
- **max** – Maximum Bound.

Returns The specified value clamped to the specified bounds.

float **mCos** (float *v*)
Calculate the cosine of v.

Parameters **v** – Input Value (in radians).

Returns The cosine of the input value.

float **mDegToRad** (float *degrees*)
Convert specified degrees into radians.

Parameters **degrees** – Input Value (in degrees).

Returns The specified degrees value converted to radians.

string **mFloatLength** (float *v*, int *precision*)
Formats the specified number to the given number of decimal places.

Parameters

- **v** – Number to format.
- **precision** – Number of decimal places to format to (1-9).

Returns Number formatted to the specified number of decimal places.

int **mFloor** (float *v*)
Round v down to the nearest integer.

Parameters **v** – Number to convert to integer.

Returns Number converted to integer.

float **mFMod** (float *v*, float *d*)
Calculate the remainder of v/d.

Parameters

- **v** – Input Value.
- **d** – Divisor Value.

Returns The remainder of v/d.

bool **mIsPow2** (int *v*)
Returns whether the value is an exact power of two.

Parameters **v** – Input value.

Returns Whether the specified value is an exact power of two.

float **mLerp** (float *v1*, float *v2*, float *time*)
Calculate linearly interpolated value between two specified numbers using specified normalized time.

Parameters

- **v1** – Interpolate From Input value.
- **v2** – Interpolate To Input value.
- **time** – Normalized time used to interpolate values (0-1).

Returns The interpolated value between the two specified values at normalized time t.

float **mLog** (float *v*)
Calculate the natural logarithm of v.

Parameters **v** – Input Value.

Returns The natural logarithm of the input value.

float **mPi** ()
Return the value of PI (half-circle in radians).

Returns The value of PI.

float **mPow** (float *v*, float *p*)
Calculate b raised to the p-th power.

Parameters

- **v** – Input Value.
- **p** – Power to raise value by.

Returns v raised to the p-th power.

float **mRadToDeg** (float *radians*)
Convert specified radians into degrees.

Parameters **radians** – Input Value (in radians).

Returns The specified radians value converted to degrees.

int **mRound** (float *v*)
Round v to the nearest integer.

Parameters **v** – Number to convert to integer.

Returns Number converted to integer.

float **mSaturate** (float *v*)
Clamp the specified value between 0 and 1 (inclusive).

Parameters **v** – Input value.

Returns The specified value clamped between 0 and 1 (inclusive).

float **mSin** (float *v*)
Calculate the sine of v.

Parameters **v** – Input Value (in radians).

Returns The sine of the input value.

string **mSolveCubic** (float *a*, float *b*, float *c*, float *d*)
Solve a cubic equation (3rd degree polynomial) of form a*x^3 + b*x^2 + c*x + d = 0.

Parameters

- **a** – First Coefficient.
- **b** – Second Coefficient.
- **c** – Third Coefficient.
- **d** – Fourth Coefficient.

Returns A 4-tuple, containing: (sol x0 x1 x2). (sol) is the number of solutions(being 0, 1, 2 or 3), and (x0), (x1) and (x2) are the solutions, if any.

string **mSolveQuadratic** (float *a*, float *b*, float *c*)
Solve a quadratic equation (2nd degree polynomial) of form a*x^2 + b*x + c = 0.

Parameters

- **a** – First Coefficient.
- **b** – Second Coefficient.
- **c** – Third Coefficient.

Returns A triple, containing: (sol x0 x1). (sol) is the number of solutions(being 0, 1, or 2), and (x0) and (x1) are the solutions, if any.

string **mSolveQuartic** (float *a*, float *b*, float *c*, float *d*, float *e*)
Solve a quartic equation (4th degree polynomial) of form a*x^4 + b*x^3 + c*x^2 + d*x + e = 0.

Parameters

- **a** – First Coefficient.
- **b** – Second Coefficient.
- **c** – Third Coefficient.
- **d** – Fourth Coefficient.
- **e** – Fifth Coefficient.

Returns A 5-tuple, containing: (sol x0 x1 x2 c3). (sol) is the number of solutions(being 0, 1, 2, 3 or 4), and (x0), (x1), (x2) and (x3) are the solutions, if any.

float **mSqrt** (float *v*)
Calculate the square-root of v.

Parameters **v** – Input Value.

Returns The square-root of the input value.

float **mTan** (float *v*)
Calculate the tangent of v.

Parameters **v** – Input Value (in radians).

Returns The tangent of the input value.

Vector Math

Functions for working with three-dimensional vectors (VectorF/Point3F).

Functions

VectorF **VectorAdd** (VectorF *a*, VectorF *b*)

Add two vectors.

Parameters

- **a** – The first vector.
- **b** – The second vector.

Returns

.

Example:

```
// VectorAdd( %a, %b );
// The sum of vector a, (ax, ay, az), and vector b, (bx, by, bz) is:
//     a + b = ( ax + bx, ay + by, az + bz )

%a = "1 0 0";
%b = "0 1 0";

// %r = "( 1 + 0, 0 + 1, 0 + 0 )";// %r = "1 1 0";
%r = VectorAdd( %a, %b );
```

VectorF **VectorCross** (VectorF *a*, VectorF *b*)

Calculcate the cross product of two vectors.

Parameters

- **a** – The first vector.
- **b** – The second vector.

Returns

.

Example:

```
// VectorCross( %a, %b );
// The cross product of vector a, (ax, ay, az), and vector b, (bx, by, bz), is
//     a x b = ( ( ay * bz ) - ( az * by ), ( az * bx ) - ( ax * bz ), ( ax * by ) - ( ay * bx )

%a = "1 1 0";
%b = "2 0 1";

// %r = "( ( 1 * 1 ) - ( 0 * 0 ), ( 0 * 2 ) - ( 1 * 1 ), ( 1 * 0 ) - ( 1 * 2 ) )";
// %r = "1 -1 -2";
%r = VectorCross( %a, %b );
```

float **VectorDist** (VectorF *a*, VectorF *b*)

Compute the distance between two vectors.

Parameters

- **a** – The first vector.
- **b** – The second vector.

Returns).

Example:

```
// VectorDist( %a, %b );
// The distance between vector a, (ax, ay, az), and vector b, (bx, by, bz), is
//     a -> b = ||( b - a )||
//            = ||( bx - ax, by - ay, bz - az )||
//            = mSqrt( ( bx - ax ) * ( bx - ax ) + ( by - ay ) * ( by - ay ) + ( bz - az ) * ( b

%a = "1 1 0";
%b = "2 0 1";

// %r = mSqrt( ( 2 - 1 ) * ( 2 - 1) + ( 0 - 1 ) * ( 0 - 1 ) + ( 1 - 0 ) * ( 1 - 0 ) );
// %r = mSqrt( 3 );
%r = VectorDist( %a, %b );
```

float **VectorDot** (VectorF *a*, VectorF *b*)

Compute the dot product of two vectors.

Parameters

- **a** – The first vector.
- **b** – The second vector.

Returns

.

Example:

```
// VectorDot( %a, %b );
// The dot product between vector a, (ax, ay, az), and vector b, (bx, by, bz), is:
//     a . b = ( ax * bx + ay * by + az * bz )

%a = "1 1 0";
%b = "2 0 1";

// %r = "( 1 * 2 + 1 * 0 + 0 * 1 )";
// %r = 2;
%r = VectorDot( %a, %b );
```

float **VectorLen** (VectorF *v*)

Calculate the magnitude of the given vector.

Parameters **v** – A vector.

Returns

.

Example:

```
// VectorLen( %a );
// The length or magnitude of  vector a, (ax, ay, az), is:
//     ||a|| = Sqrt( ax * ax + ay * ay + az * az )

%a = "1 1 0";

// %r = mSqrt( 1 * 1 + 1 * 1 + 0 * 0 );
// %r = mSqrt( 2 );
// %r = 1.414;
%r = VectorLen( %a );
```

VectorF **VectorLerp** (VectorF *a*, VectorF *b*, float *t*)

Linearly interpolate between two vectors by t .

Parameters

- **a** – Vector to start interpolation from.
- **b** – Vector to interpolate to.
- **t** – Interpolation factor (0-1). At zero, a is returned and at one, b is returned. In between, an interpolated vector between a and b is returned.

Returns

.

Example:

```
// VectorLerp( %a, %b );
// The point between vector a, (ax, ay, az), and vector b, (bx, by, bz), which is
// weighted by the interpolation factor, t, is
//     r = a + t * ( b - a )
//       = ( ax + t * ( bx - ax ), ay + t * ( by - ay ), az + t * ( bz - az ) )

%a = "1 1 0";
%b = "2 0 1";
%v = "0.25";

// %r = "( 1 + 0.25 * ( 2 - 1 ), 1 + 0.25 * ( 0 - 1 ), 0 + 0.25 * ( 1 - 0 ) )";
// %r = "1.25 0.75 0.25";
%r = VectorLerp( %a, %b );
```

VectorF **VectorNormalize**(VectorF *v*)

Brings a vector into its unit form, i.e. such that it has the magnitute 1.

Parameters **v** – The vector to normalize.

Returns scaled to length 1.

Example:

```
// VectorNormalize( %a );
// The normalized vector a, (ax, ay, az), is:
//     a^ = a / ||a||
//        = ( ax / ||a||, ay / ||a||, az / ||a|| )

%a = "1 1 0";
%l = 1.414;

// %r = "( 1 / 1.141, 1 / 1.141, 0 / 1.141 )";
// %r = "0.707 0.707 0";
%r = VectorNormalize( %a );
```

MatrixF **VectorOrthoBasis**(AngAxisF *aa*)

Create an orthogonal basis from the given vector.

Parameters **aaf** – The vector to create the orthogonal basis from.

Returns A matrix representing the orthogonal basis.

VectorF **VectorScale**(VectorF *a*, float *scalar*)

Scales a vector by a scalar.

Parameters

- **a** – The vector to scale.

- **scalar** – The scale factor.

Returns

.

Example:

```
// VectorScale( %a, %v );
// Scaling vector a, (ax, ay, az), but the scalar, v, is:
//     a * v = ( ax * v, ay * v, az * v )

%a = "1 1 0";
%v = "2";

// %r = "( 1 * 2, 1 * 2, 0 * 2 )";
// %r = "2 2 0";
%r = VectorScale( %a, %v );
```

VectorF **VectorSub** (VectorF *a*, VectorF *b*)

Subtract two vectors.

Parameters

- **a** – The first vector.
- **b** – The second vector.

Returns

.

Example:

```
// VectorSub( %a, %b );
// The difference of vector a, (ax, ay, az), and vector b, (bx, by, bz) is:
//     a - b = ( ax - bx, ay - by, az - bz )

%a = "1 0 0";
%b = "0 1 0";

// %r = "( 1 - 0, 0 - 1, 0 - 0 )";
// %r = "1 -1 0";
%r = VectorSub( %a, %b );
```

Matrix Math

Functions for working with matrices (MatrixF, AngAxisF, MatrixRotation, MatrixPosition).

Functions

TransformF **MatrixCreate** (VectorF *position*, AngAxisF *orientation*)

Create a transform from the given translation and orientation.

Parameters

- **position** – The translation vector for the transform.
- **orientation** – The axis and rotation that orients the transform.

Returns A transform based on the given position and orientation.

TransformF **MatrixCreateFromEuler**(Point3F *angles*)
a matrix from the given rotations.

Parameters **Vector3F** – X, Y, and Z rotation in *radians*.

Returns A transform based on the given orientation.

Point3F **MatrixMulPoint**(TransformF *transform*, Point3F *point*)
Multiply the given point by the given transform assuming that w=1. This function will multiply the given vector such that translation with take effect.

Parameters

- **transform** – A transform.
- **point** – A vector.

Returns The transformed vector.

TransformF **MatrixMultiply**(TransformF *left*, TransformF *right*)
Multiply the two matrices.

Parameters

- **left** – First transform.
- **right** – Right transform.

Returns Concatenation of the two transforms.

VectorF **MatrixMulVector**(TransformF *transform*, VectorF *vector*)
Multiply the vector by the transform assuming that w=0. This function will multiply the given vector by the given transform such that translation will not affect the vector.

Parameters

- **transform** – A transform.
- **vector** – A vector.

Returns The transformed vector.

Random Numbers

Functions for generating random numbers. Based on a seed, the random number generator produces a sequence of numbers. As a given seed will always produce the same sequence of numbers this can be used to generate re-producible sequences of apparently random numbers. To set the seed, call setRandomSeed().

Functions

float **getRandom**(int *a*, int *b*)
Returns a random number based on parameters passed in.. If no parameters are passed in, getRandom() will return a float between 0.0 and 1.0. If one parameter is passed an integer between 0 and the passed in value will be returned. Two parameters will return an integer between the specified numbers.

Parameters

- **a** – If this is the only parameter, a number between 0 and a is returned. Elsewise represents the lower bound.
- **b** – Upper bound on the random number. The random number will be <= b.

Returns , between 0 and a, or a float between 0.0 and 1.1 depending on usage.

int **getRandomSeed**()
Get the current seed used by the random number generator.

Returns The current random number generator seed value.

void **setRandomSeed**(int *seed*)
Set the current seed for the random number generator. Based on this seed, a repeatable sequence of numbers will be produced by getRandom() .

Parameters **seed** – The seed with which to initialize the randon number generator with. The same seed will always leed tothe same sequence of pseudo-random numbers. If -1, the current timestamp will be used as the seed which is a good basis for randomization.

Strings

Functions for dealing with string values. Since in TorqueScript any value is implicitly also a string, these functions can be used with all values.

Functions

string **collapseEscape**(string *text*)
Replace all escape sequences in text with their respective character codes. This function replaces all escape sequences (n, t, etc) in the given string with the respective characters they represent. The primary use of this function is for converting strings from their literal form into their compiled/translated form, as is normally done by the TorqueScript compiler.

Parameters **text** – A string.

Returns with all escape sequences replaced by their respective character codes.

Example:

```
// Print:
//    str
//    ing
// to the console.  Note how the backslash in the string must be escaped here
// in order to prevent the TorqueScript compiler from collapsing the escape
// sequence in the resulting string.
echo( collapseEscape( "str\ning" ) );
```

void **dumpStringMemStats**()
Dumps information about String memory usage.

bool **endsWith**(string *str*, string *suffix*, bool *caseSensitive*)
Test whether the given string ends with the given suffix.

Parameters

- **str** – The string to test.
- **suffix** – The potential suffix of str.
- **caseSensitive** – If true, the comparison will be case-sensitive; if false, differences in casing will not be taken into account.

Returns True if the last characters in str match the complete contents of suffix; false otherwise.

Example:

```
startsWith( "TEST123", "123" ) // Returns true.
```

string **expandEscape**(string *text*)

Replace all characters in text that need to be escaped for the string to be a valid string literal with their respective escape sequences. All characters in text that cannot appear in a string literal will be replaced by an escape sequence (n, t, etc). The primary use of this function is for converting strings suitable for being passed as string literals to the TorqueScript compiler. expandEscape("str" NL "ing") // Returns "strning".

Parameters **text** – A string

Returns A duplicate of the text parameter with all unescaped characters that cannot appear in string literals replaced by their respective escape sequences.

string **getSubStr**(string *str*, int *start*, int *numChars*)

Return a substring of str starting at start and continuing either through to the end of str (if numChars is -1) or for numChars characters (except if this would exceed the actual source string length).

Parameters

- **str** – The string from which to extract a substring.
- **start** – The offset at which to start copying out characters.
- **numChars** – Optional argument to specify the number of characters to copy. If this is -1, all characters up the end of the input string are copied.

Returns A string that contains the given portion of the input string.

Example:

```
getSubStr( "foobar", 1, 2 ) // Returns "oo".
```

int **getTrailingNumber**(string *str*)

Get the numeric suffix of the given input string.

Parameters **str** – The string from which to read out the numeric suffix.

Returns The numeric value of the number suffix of str or -1 if str has no such suffix.

Example:

```
getTrailingNumber( "test123" ) // Returns 123.
```

bool **isalnum**(string *str*, int *index*)

Test whether the character at the given position is an alpha-numeric character. Alpha-numeric characters are characters that are either alphabetic (a-z, A-Z) or numbers (0-9).

Parameters

- **str** – The string to test.
- **index** – The index of a character in str.

Returns True if the character at the given index in str is an alpha-numeric character; false otherwise.

bool **isspace**(string *str*, int *index*)

Test whether the character at the given position is a whitespace character. Characters such as tab, space, or newline are considered whitespace.

Parameters

- **str** – The string to test.
- **index** – The index of a character in str.

Returns True if the character at the given index in str is a whitespace character; false otherwise.

string **ltrim**(string *str*)
Remove leading whitespace from the string.

Parameters **str** – A string.

Returns A string that is the same as str but with any leading (i.e. leftmost) whitespace removed.

Example:

```
ltrim( "   string  " ); // Returns "string  ".
```

string **nextToken**(string *str*, string *token*, string *delimiters*)
Tokenize a string using a set of delimiting characters. This function first skips all leading charaters in str that are contained in delimiters . From that position, it then scans for the next character in str that is contained in delimiters and stores all characters from the starting position up to the first delimiter in a variable in the current scope called token . Finally, it skips all characters in delimiters after the token and then returns the remaining string contents in str . To scan out all tokens in a string, call this function repeatedly by passing the result it returns each time as the new str until the function returns “”.

Parameters

- **str** – A string.
- **token** – The name of the variable in which to store the current token. This variable is set in the scope in which nextToken is called.
- **delimiters** – A string of characters. Each character is considered a delimiter.

Returns The remainder of str after the token has been parsed out or “” if no more tokens were found in str.

Example:

```
// Prints:
// a
// b
// c
%str = "a   b c";
while ( %str !$= "" )
{
   // First time, stores "a" in the variable %token and sets %str to "b c".
   %str = nextToken( %str, "token", "" );
   echo( %token );
}
```

string **rtrim**(string *str*)
Remove trailing whitespace from the string.

Parameters **str** – A string.

Returns A string that is the same as str but with any trailing (i.e. rightmost) whitespace removed.

Example:

```
rtrim( "   string  " ); // Returns "   string".
```

bool **startsWith**(string *str*, string *prefix*, bool *caseSensitive*)
Test whether the given string begins with the given prefix.

Parameters

- **str** – The string to test.

- **prefix** – The potential prefix of str.
- **caseSensitive** – If true, the comparison will be case-sensitive; if false, differences in casing will not be taken into account.

Returns True if the first characters in str match the complete contents of prefix; false otherwise.

Example:

```
startsWith( "TEST123", "test" ) // Returns true.
```

int **strasc**(string *chr*)
Return the integer character code value corresponding to the first character in the given string.

Parameters **chr** – a (one-character) string.

Returns The UTF32 code value for the first character in the given string.

string **strchr**(string *str*, string *chr*)
Find the first occurrence of the given character in str .

Parameters

- **str** – The string to search.
- **chr** – The character to search for. Only the first character from the string is taken.

Returns The remainder of the input string starting with the given character or the empty string if the character could not be found.

int **strchrpos**(string *str*, string *chr*, int *start*)
Find the first occurrence of the given character in the given string.

Parameters

- **str** – The string to search.
- **chr** – The character to look for. Only the first character of this string will be searched for.
- **start** – The index into str at which to start searching for the given character.

Returns The index of the first occurrence of chr in str or -1 if str does not contain the given character.

Example:

```
strchrpos( "test", "s" ) // Returns 2.
```

int **strcmp**(string *str1*, string *str2*)
Compares two strings using case- sensitive comparison.

Parameters

- **str1** – The first string.
- **str2** – The second string.

Returns 0 if both strings are equal, a value <0 if the first character different in str1 has a smaller character code value than the character at the same position in str2, and a value >1 otherwise.

Example:

```
if( strcmp( %var, "foobar" ) == 0 )
   echo( "%var is equal to foobar" );
```

string **strformat**(string *format*, string *value*)
Format the given value as a string using printf-style formatting.

Parameters

- **format** – A printf-style format string.
- **value** – The value argument matching the given format string.

Example:

```
// Convert the given integer value to a string in a hex notation.
%hex = strformat( "%x", %value );
```

int **stricmp**(string *str1*, string *str2*)

Compares two strings using case- insensitive comparison.

Parameters

- **str1** – The first string.
- **str2** – The second string.

Returns 0 if both strings are equal, a value <0 if the first character different in str1 has a smaller character code value than the character at the same position in str2, and a value >0 otherwise.

Example:

```
if( stricmp( "FOObar", "foobar" ) == 0 )
   echo( "this is always true" );
```

int **strinatcmp**(string *str1*, string *str2*)

Compares two strings using "natural order" case-insensitive comparison. Natural order means that rather than solely comparing single character code values, strings are ordered in a natural way. For example, the string "hello10" is considered greater than the string "hello2" even though the first numeric character in "hello10" actually has a smaller character value than the corresponding character in "hello2". However, since 10 is greater than 2, strnatcmp will put "hello10" after "hello2".

Parameters

- **str1** – The first string.
- **str2** – The second string.

Returns 0 if the strings are equal, a value >0 if str1 comes after str2 in a natural order, and a value <0 if str1 comes before str2 in a natural order.

Example:

```
// Bubble sort 10 elements of %array using natural orderdo
{
   %swapped = false;
   for( %i = 0; %i < 10 - 1; %i ++ )
      if( strnatcmp( %array[ %i ], %array[ %i + 1 ] ) > 0 )
      {
         %temp = %array[ %i ];
         %array[ %i ] = %array[ %i + 1 ];
         %array[ %i + 1 ] = %temp;
         %swapped = true;
      }
}
while( %swapped );
```

string **stripChars**(string *str*, string *chars*)

Remove all occurrences of characters contained in chars from str .

Parameters

- **str** – The string to filter characters out from.
- **chars** – A string of characters to filter out from str.

Returns A version of str with all occurrences of characters contained in chars filtered out.

Example:

```
stripChars( "teststring", "se" ); // Returns "tttring".
```

String **stripTrailingNumber** (string *str*)
Strip a numeric suffix from the given string.

Parameters **str** – The string from which to strip its numeric suffix.

Returns The string str without its number suffix or the original string str if it has no such suffix.

Example:

```
stripTrailingNumber( "test123" ) // Returns "test".
```

bool **strIsMatchExpr** (string *pattern*, string *str*, bool *caseSensitive*)
Match a pattern against a string.

Parameters

- **pattern** – The wildcard pattern to match against. The pattern can include characters, '*' to match any number of characters and '?' to match a single character.
- **str** – The string which should be matched against pattern.
- **caseSensitive** – If true, characters in the pattern are matched in case-sensitive fashion against this string. If false, differences in casing are ignored.

Returns True if str matches the given pattern.

Example:

```
strIsMatchExpr( "f?o*R", "foobar" ) // Returns true.
```

bool **strIsMatchMultipleExpr** (string *patterns*, string *str*, bool *caseSensitive*)
Match a multiple patterns against a single string.

Parameters

- **patterns** – A tab-separated list of patterns. Each pattern can include charaters, '*' to match any number of characters and '?' to match a single character. Each of the patterns is tried in turn.
- **str** – The string which should be matched against patterns.
- **caseSensitive** – If true, characters in the pattern are matched in case-sensitive fashion against this string. If false, differences in casing are ignored.

Returns True if str matches any of the given patterns.

Example:

```
strIsMatchMultipleExpr( "*.cs *.gui *.mis", "mymission.mis" ) // Returns true.
```

int **strlen** (string *str*)
Get the length of the given string in bytes.

Parameters **str** – A string.

Returns The length of the given string in bytes.

string **strlwr**(string *str*)
Return an all lower-case version of the given string.

Parameters **str** – A string.

Returns A version of str with all characters converted to lower-case.

Example:

```
strlwr( "TesT1" ) // Returns "test1"
```

int **strnatcmp**(string *str1*, string *str2*)
Compares two strings using "natural order" case-sensitive comparison. Natural order means that rather than solely comparing single character code values, strings are ordered in a natural way. For example, the string "hello10" is considered greater than the string "hello2" even though the first numeric character in "hello10" actually has a smaller character value than the corresponding character in "hello2". However, since 10 is greater than 2, strnatcmp will put "hello10" after "hello2".

Parameters

- **str1** – The first string.
- **str2** – The second string.

Returns 0 if the strings are equal, a value >0 if str1 comes after str2 in a natural order, and a value <0 if str1 comes before str2 in a natural order.

Example:

```
// Bubble sort 10 elements of %array using natural orderdo
{
   %swapped = false;
   for( %i = 0; %i < 10 - 1; %i ++ )
      if( strnatcmp( %array[ %i ], %array[ %i + 1 ] ) > 0 )
      {
         %temp = %array[ %i ];
         %array[ %i ] = %array[ %i + 1 ];
         %array[ %i + 1 ] = %temp;
         %swapped = true;
      }
}
while( %swapped );
```

int **strpos**(string *haystack*, string *needle*, int *offset*)
Find the start of needle in haystack searching from left to right beginning at the given offset.

Parameters

- **haystack** – The string to search.
- **needle** – The string to search for.

Returns The index at which the first occurrence of needle was found in haystack or -1 if no match was found.

Example:

```
strpos( "b ab", "b", 1 ) // Returns 3.
```

string **strrchr**(string *str*, string *chr*)
Find the last occurrence of the given character in str .

Parameters

- **str** – The string to search.
- **chr** – The character to search for. Only the first character from the string is taken.

Returns The remainder of the input string starting with the given character or the empty string if the character could not be found.

int **strrchrpos** (string *str*, string *chr*, int *start*)
Find the last occurrence of the given character in the given string.

Parameters

- **str** – The string to search.
- **chr** – The character to look for. Only the first character of this string will be searched for.
- **start** – The index into str at which to start searching for the given character.

Returns The index of the last occurrence of chr in str or -1 if str does not contain the given character.

Example:

```
strrchrpos( "test", "t" ) // Returns 3.
```

string **strrepeat** (string *str*, int *numTimes*, string *delimiter*)
Return a string that repeats str numTimes number of times delimiting each occurrence with delimiter .

Parameters

- **str** – The string to repeat multiple times.
- **numTimes** – The number of times to repeat str in the result string.
- **delimiter** – The string to put between each repetition of str.

Returns A string containing str repeated numTimes times.

Example:

```
strrepeat( "a", 5, "b" ) // Returns "abababababa".
```

string **strreplace** (string *source*, string *from*, string *to*)
Replace all occurrences of from in source with to .

Parameters

- **source** – The string in which to replace the occurrences of from.
- **from** – The string to replace in source.
- **to** – The string with which to replace occurrences of .

Returns A string with all occurrences of from in source replaced by to.

Example:

```
strreplace( "aabbccbb", "bb", "ee" ) // Returns "aaeeccee".
```

int **strstr** (string *string*, string *substring*)
Find the start of substring in the given string searching from left to right.

Parameters

- **string** – The string to search.
- **substring** – The string to search for.

Returns The index into string at which the first occurrence of substring was found or -1 if substring could not be found.

Example:

```
strstr( "abcd", "c" ) // Returns 2.
```

string **strupr**(string *str*)
Return an all upper-case version of the given string.

Parameters **str** – A string.

Returns A version of str with all characters converted to upper-case.

Example:

```
strupr( "TesT1" ) // Returns "TEST1"
```

string **trim**(string *str*)
Remove leading and trailing whitespace from the string.

Parameters **str** – A string.

Returns A string that is the same as str but with any leading (i.e. leftmost) and trailing (i.e. rightmost) whitespace removed.

Example:

```
trim( "   string  " ); // Returns "string".
```

Field Manipulators

Functions to deal with whitespace-separated lists of values in strings. TorqueScript extensively uses strings to represent lists of values. The functions in this group simplify working with these lists and allow to easily extract individual values from their strings.

The list strings are segregated into three groups according to the delimiters used to separate invididual values in the strings:

- Strings of words: Elements are separated by newlines (n), spaces, or tabs (t).
- Strings of fields: Elements are sepaerated by newlines (n) or tabs (t).
- Strings of records: Elements are separated by newlines (n).

Aside from the functions here, another useful means to work with strings of words is TorqueScript's foreach$ statement.

Functions
string **firstWord**(string *text*)
Return the first word in text .

Parameters **text** – A list of words separated by newlines, spaces, and/or tabs.

Returns The word at index 0 in text or "" if text is empty.
string **getField**(string *text*, int *index*)
Extract the field at the given index in the newline and/or tab separated list in text . Fields in text must be separated by newlines and/or tabs.

Parameters

- **text** – A list of fields separated by newlines and/or tabs.

- **index** – The zero-based index of the field to extract.

Returns The field at the given index or "" if the index is out of range.

Example:

```
getField( "a b" TAB "c d" TAB "e f", 1 ) // Returns "c d"
```

int **getFieldCount**(string *text*)

Return the number of newline and/or tab separated fields in text .

Parameters **text** – A list of fields separated by newlines and/or tabs.

Returns

.

Example:

```
getFieldCount( "a b" TAB "c d" TAB "e f" ) // Returns 3
```

string **getFields**(string *text*, int *startIndex*, int *endIndex*)

Extract a range of fields from the given startIndex onwards thru endIndex . Fields in text must be separated by newlines and/or tabs.

Parameters

- **text** – A list of fields separated by newlines and/or tabs.
- **startIndex** – The zero-based index of the first field to extract from text.
- **endIndex** – The zero-based index of the last field to extract from text. If this is -1, all fields beginning with startIndex are extracted from text.

Returns The number of newline and/or tab sepearated elements in text.

Example:

```
getFields( "a b" TAB "c d" TAB "e f", 1 ) // Returns "c d" TAB "e f"
```

string **getRecord**(string *text*, int *index*)

Extract the record at the given index in the newline-separated list in text . Records in text must be separated by newlines.

Parameters

- **text** – A list of records separated by newlines.
- **index** – The zero-based index of the record to extract.

Returns The record at the given index or "" if index is out of range.

Example:

```
getRecord( "a b" NL "c d" NL "e f", 1 ) // Returns "c d"
```

int **getRecordCount**(string *text*)

Return the number of newline-separated records in text .

Parameters **text** – A list of records separated by newlines.

Returns The number of newline-sepearated elements in text.

Example:

```
getRecordCount( "a b" NL "c d" NL "e f" ) // Returns 3
```

string **getRecords**(string *text*, int *startIndex*, int *endIndex*)

Extract a range of records from the given startIndex onwards thru endIndex . Records in text must be separated by newlines.

Parameters

- **text** – A list of records separated by newlines.
- **startIndex** – The zero-based index of the first record to extract from text.
- **endIndex** – The zero-based index of the last record to extract from text. If this is -1, all records beginning with startIndex are extracted from text.

Returns A string containing the specified range of records from text or "" if startIndex is out of range or greater than endIndex.

Example:

```
getRecords( "a b" NL "c d" NL "e f", 1 ) // Returns "c d" NL "e f"
```

string **getWord**(string *text*, int *index*)

Extract the word at the given index in the whitespace-separated list in text . Words in text must be separated by newlines, spaces, and/or tabs.

Parameters

- **text** – A whitespace-separated list of words.
- **index** – The zero-based index of the word to extract.

Returns The word at the given index or "" if the index is out of range.

Example:

```
getWord( "a b c", 1 ) // Returns "b"
```

int **getWordCount**(string *text*)

Return the number of whitespace-separated words in text . Words in text must be separated by newlines, spaces, and/or tabs.

Parameters **text** – A whitespace-separated list of words.

Returns

.

Example:

```
getWordCount( "a b c d e" ) // Returns 5
```

string **getWords**(string *text*, int *startIndex*, int *endIndex*)

Extract a range of words from the given startIndex onwards thru endIndex . Words in text must be separated by newlines, spaces, and/or tabs.

Parameters

- **text** – A whitespace-separated list of words.
- **startIndex** – The zero-based index of the first word to extract from text.
- **endIndex** – The zero-based index of the last word to extract from text. If this is -1, all words beginning with startIndex are extracted from text.

Returns A string containing the specified range of words from text or "" if startIndex is out of range or greater than endIndex.

Example:

```
getWords( "a b c d", 1, 2, ) // Returns "b c"
```

string **removeField**(string *text*, int *index*)
Remove the field in text at the given index . Fields in text must be separated by newlines and/or tabs.

Parameters

- **text** – A list of fields separated by newlines and/or tabs.
- **index** – The zero-based index of the field in text.

Returns A new string with the field at the given index removed or the original string if index is out of range.

Example:

```
removeField( "a b" TAB "c d" TAB "e f", 1 ) // Returns "a b" TAB "e f"
```

string **removeRecord**(string *text*, int *index*)
Remove the record in text at the given index . Records in text must be separated by newlines.

Parameters

- **text** – A list of records separated by newlines.
- **index** – The zero-based index of the record in text.

Returns is out of range.

Example:

```
removeRecord( "a b" NL "c d" NL "e f", 1 ) // Returns "a b" NL "e f"
```

string **removeWord**(string *text*, int *index*)
Remove the word in text at the given index . Words in text must be separated by newlines, spaces, and/or tabs.

Parameters

- **text** – A whitespace-separated list of words.
- **index** – The zero-based index of the word in text.

Returns A new string with the record at the given index removed or the original string if index is out of range.

Example:

```
removeWord( "a b c d", 2 ) // Returns "a b d"
```

string **restWords**(string *text*)
Return all but the first word in text .

Parameters **text** – A list of words separated by newlines, spaces, and/or tabs.

Returns Text with the first word removed.

string **setField**(string *text*, int *index*, string *replacement*)
Replace the field in text at the given index with replacement . Fields in text must be separated by newlines and/or tabs.

Parameters

- **text** – A list of fields separated by newlines and/or tabs.
- **index** – The zero-based index of the field to replace.
- **replacement** – The string with which to replace the field.

Returns is out of range.

Example:

```
setField( "a b" TAB "c d" TAB "e f", 1, "g h" ) // Returns "a b" TAB "g h" TAB "e f"
```

string **setRecord**(string *text*, int *index*, string *replacement*)

Replace the record in text at the given index with replacement . Records in text must be separated by newlines.

Parameters

- **text** – A list of records separated by newlines.
- **index** – The zero-based index of the record to replace.
- **replacement** – The string with which to replace the record.

Returns A new string with the field at the given index replaced by replacement or the original string if index is out of range.

Example:

```
setRecord( "a b" NL "c d" NL "e f", 1, "g h" ) // Returns "a b" NL "g h" NL "e f"
```

string **setWord**(string *text*, int *index*, string *replacement*)

Replace the word in text at the given index with replacement . Words in text must be separated by newlines, spaces, and/or tabs.

Parameters

- **text** – A whitespace-separated list of words.
- **index** – The zero-based index of the word to replace.
- **replacement** – The string with which to replace the word.

Returns A new string with the record at the given index replaced by replacement or the original string if index is out of range.

Example:

```
setWord( "a b c d", 2, "f" ) // Returns "a b f d"
```

Utilities

Miscellaneous utility functions.

Functions

int **countBits**(int *v*)

Count the number of bits that are set in the given 32 bit integer.

Parameters **v** – An integer value.

Returns

.

Torque::UUID **generateUUID**()
Generate a new universally unique identifier (UUID).

Returns A newly generated UUID.

5.3.2 GUI

Subsystem to display user interface elements and handle high-level rendering control flow.

3D Controls

Controls to render 3D elements.

Classes

GameTSCtrl The main 3D viewport for a Torque 3D game.

Inherit: GuiTSCtrl

Description The main 3D viewport for a Torque 3D game.

With the exception of a few very niche genres, the bulk of your 3D game viewing will occur in a GameTSCtrl. You typically only need a single GameTSCtrl, unless you are implementing a very complex interface system. In the demos, you can find our example named "PlayGui".

It is recommended that any game GUIs that are not pushed and popped constantly, be contained within your GameTSCtrl. Examples include targeting reticle, standard healthbar, ammo count, etc. This is mostly a design decision, but the way Torque 3D's GUI system works somewhat encourages you to group the controls in this manner:

```
// Example of a GameTSCtrl
// PlayGui is the main TSControl through which the game is viewed
// Also contains a Guis for:
// - A lag icon
// - Showing other shape names
// - Crossahir
%guiContent = new GameTSCtrl(PlayGui)
{
  cameraZRot = "0";
  forceFOV = "0";
  reflectPriority = "1";
  Profile = "GuiContentProfile";
  HorizSizing = "right";
  VertSizing = "bottom";
  position = "0 0";
  Extent = "1024 768";

  new GuiBitmapCtrl(LagIcon)
  {
     bitmap = "art/gui/lagIcon.png";
     // Note: Rest of fields hidden for this example
  };

  new GuiShapeNameHud()
  {
     fillColor = "0 0 0 0.25";
```

```
      frameColor = "0 1 0 1";
      textColor = "0 1 0 1";
      showFill = "0";
      showFrame = "0";

      // Note: Rest of fields hidden for this example
   };

   new GuiCrossHairHud(Reticle)
   {
      damageFillColor = "0 1 0 1";
      damageFrameColor = "1 0.6 0 1";
      damageRect = "50 4";
      damageOffset = "0 10";
      bitmap = "art/gui/weaponHud/blank.png";
      // Note: Rest of fields hidden for this example
   };
};
```

GuiObjectView GUI control which displays a 3D model.

Inherit: GuiTSCtrl

Description GUI control which displays a 3D model.

Model displayed in the control can have other objects mounted onto it, and the light settings can be adjusted.

Example:

```
newGuiObjectView(ObjectPreview)
   {
      shapeFile = "art/shapes/items/kit/healthkit.dts";
      mountedNode = "mount0";
      lightColor = "1 1 1 1";
      lightAmbient = "0.5 0.5 0.5 1";
      lightDirection = "0 0.707 -0.707";
      orbitDiststance = "2";
      minOrbitDiststance = "0.917688";
      maxOrbitDiststance = "5";
      cameraSpeed = "0.01";
      cameraZRot = "0";
      forceFOV = "0";
      reflectPriority = "0";
   };
```

Methods

float GuiObjectView::**getCameraSpeed**()

Return the current multiplier for camera zooming and rotation.

Returns zooming / rotation multiplier value.

Example:

```
// Request the current camera zooming and rotation multiplier value
%multiplier = %thisGuiObjectView.getCameraSpeed();
```

string GuiObjectView::**getModel**()

Return the model displayed in this view.

Returns Name of the displayed model.

Example:

```
// Request the displayed model name from the GuiObjectView object.
%modelName = %thisGuiObjectView.getModel();
```

string GuiObjectView::**getMountedModel**()
Return the name of the mounted model.

Returns Name of the mounted model.

Example:

```
// Request the name of the mounted model from the GuiObjectView object
%mountedModelName = %thisGuiObjectView.getMountedModel();
```

string GuiObjectView::**getMountSkin**(int *param1*, int *param2*)
Return the name of skin used on the mounted model.

Returns Name of the skin used on the mounted model.

Example:

```
// Request the skin name from the model mounted on to the main model in the control
%mountModelSkin = %thisGuiObjectView.getMountSkin();
```

float GuiObjectView::**getOrbitDistance**()
Return the current distance at which the camera orbits the object.

Returns The distance at which the camera orbits the object.

Example:

```
// Request the current orbit distance
%orbitDistance = %thisGuiObjectView.getOrbitDistance();
```

string GuiObjectView::**getSkin**()
Return the name of skin used on the primary model.

Returns Name of the skin used on the primary model.

Example:

```
// Request the name of the skin used on the primary model in the control
%skinName = %thisGuiObjectView.getSkin();
```

void GuiObjectView::**onMouseEnter**()
Called whenever the mouse enters the control.

Example:

```
// The mouse has entered the control, causing the callback to occurGuiObjectView::onMouseEnter(%
   {
      // Code to run when the mouse enters this control
   }
```

void GuiObjectView::**onMouseLeave**()
Called whenever the mouse leaves the control.

Example:

```
// The mouse has left the control, causing the callback to occurGuiObjectView::onMouseLeave(%thi
   {
      // Code to run when the mouse leaves this control
   }
```

void GuiObjectView::**setCameraSpeed**(float *factor*)
Sets the multiplier for the camera rotation and zoom speed.

Parameters **factor** – Multiplier for camera rotation and zoom speed.

Example:

```
// Set the factor value
%factor = "0.75";

// Inform the GuiObjectView object to set the camera speed.
%thisGuiObjectView.setCameraSpeed(%factor);
```

void GuiObjectView::**setLightAmbient**(ColorF *color*)
Set the light ambient color on the sun object used to render the model.

Parameters **color** – Ambient color of sunlight.

Example:

```
// Define the sun ambient color value
%color = "1.0 0.4 0.6";

// Inform the GuiObjectView object to set the sun ambient color to the requested value
%thisGuiObjectView.setLightAmbient(%color);
```

void GuiObjectView::**setLightColor**(ColorF *color*)
Set the light color on the sun object used to render the model.

Parameters **color** – Color of sunlight.

Example:

```
// Set the color value for the sun
%color = "1.0 0.4 0.5";

// Inform the GuiObjectView object to change the sun color to the defined value
%thisGuiObjectView.setLightColor(%color);
```

void GuiObjectView::**setLightDirection**(Point3F *direction*)
Set the light direction from which to light the model.

Parameters **direction** – XYZ direction from which the light will shine on the model

Example:

```
// Set the light direction
%direction = "1.0 0.2 0.4"// Inform the GuiObjectView object to change the light direction to th
%thisGuiObjectView.setLightDirection(%direction);
```

void GuiObjectView::**setModel**(string *shapeName*)
Sets the model to be displayed in this control.

Parameters **shapeName** – Name of the model to display.

Example:

```
// Define the model we want to display
%shapeName = "gideon.dts";

// Tell the GuiObjectView object to display the defined model
%thisGuiObjectView.setModel(%shapeName);
```

void GuiObjectView::**setMount**(string *shapeName*, string *mountNodeIndexOrName*)

Mounts the given model to the specified mount point of the primary model displayed in this control. Detailed description

Parameters

- **shapeName** – Name of the model to mount.
- **mountNodeIndexOrName** – Index or name of the mount point to be mounted to. If index, corresponds to "mountN" in your shape where N is the number passed here.

Example:

```
// Set the shapeName to mount
%shapeName = "GideonGlasses.dts"// Set the mount node of the primary model in the control to mou
%mountNodeIndexOrName = "3";
//OR:
%mountNodeIndexOrName = "Face";

// Inform the GuiObjectView object to mount the shape at the specified node.
%thisGuiObjectView.setMount(%shapeName,%mountNodeIndexOrName);
```

void GuiObjectView::**setMountedModel**(string *shapeName*)

Sets the model to be mounted on the primary model.

Parameters **shapeName** – Name of the model to mount.

Example:

```
// Define the model name to mount
%modelToMount = "GideonGlasses.dts";

// Inform the GuiObjectView object to mount the defined model to the existing model in the contr
%thisGuiObjectView.setMountedModel(%modelToMount);
```

void GuiObjectView::**setMountSkin**(string *skinName*)

Sets the skin to use on the mounted model.

Parameters **skinName** – Name of the skin to set on the model mounted to the main model in the control

Example:

```
// Define the name of the skin
%skinName = "BronzeGlasses";

// Inform the GuiObjectView Control of the skin to use on the mounted model
%thisGuiObjectViewCtrl.setMountSkin(%skinName);
```

void GuiObjectView::**setOrbitDistance**(float *distance*)

Sets the distance at which the camera orbits the object. Clamped to the acceptable range defined in the class by min and max orbit distances. Detailed description

Parameters **distance** – The distance to set the orbit to (will be clamped).

Example:

```
// Define the orbit distance value
%orbitDistance = "1.5";

// Inform the GuiObjectView object to set the orbit distance to the defined value
%thisGuiObjectView.setOrbitDistance(%orbitDistance);
```

void GuiObjectView::**setSeq**(string *indexOrName*)
Sets the animation to play for the viewed object.

Parameters **indexOrName** – The index or name of the animation to play.

Example:

```
// Set the animation index value, or animation sequence name.
%indexVal = "3";
//OR:
%indexVal = "idle";

// Inform the GuiObjectView object to set the animation sequence of the object in the control.
%thisGuiObjectVew.setSeq(%indexVal);
```

void GuiObjectView::**setSkin**(string *skinName*)
Sets the skin to use on the model being displayed.

Parameters **skinName** – Name of the skin to use.

Example:

```
// Define the skin we want to apply to the main model in the control
%skinName = "disco_gideon";

// Inform the GuiObjectView control to update the skin the to defined skin
%thisGuiObjectView.setSkin(%skinName);
```

Fields

string GuiObjectView::**animSequence**
The animation sequence to play on the model.

Point3F GuiObjectView::**cameraRotation**
Set the camera rotation.

float GuiObjectView::**cameraSpeed**
Multiplier for mouse camera operations.

ColorF GuiObjectView::**lightAmbient**
Ambient color of the sunlight used to render the model.

ColorF GuiObjectView::**lightColor**
Diffuse color of the sunlight used to render the model.

Point3F GuiObjectView::**lightDirection**
Direction from which the model is illuminated.

float GuiObjectView::**maxOrbitDiststance**
Minimum distance below which the camera will not zoom in further.

float GuiObjectView::**minOrbitDiststance**
Maxiumum distance to which the camera can be zoomed out.

string GuiObjectView::**mountedNode**
Name of node on primary model to which to mount the secondary shape.

filename GuiObjectView::**mountedShapeFile**
Optional shape file to mount on the primary model (e.g. weapon).

string GuiObjectView::**mountedSkin**
Skin name used on mounted shape file.

float GuiObjectView::**orbitDiststance**
Distance from which to render the model.

filename GuiObjectView::**shapeFile**
The object model shape file to show in the view.

string GuiObjectView::**skin**
The skin to use on the object model.

GuiTSCtrl Abstract base class for controls that render 3D scenes.

Inherit: GuiContainer

Description GuiTSCtrl is the base class for controls that render 3D camera views in Torque. The class itself does not implement a concrete scene rendering. Use GuiObjectView to display invidiual shapes in the Gui and GameTSCtrl to render full scenes.

Methods

float GuiTSCtrl::**calculateViewDistance**(float *radius*)
Given the camera's current FOV, get the distance from the camera's viewpoint at which the given radius will fit in the render area.

Parameters **radius** – Radius in world-space units which should fit in the view.

Returns The distance from the viewpoint at which the given radius would be fully visible.

Point2F GuiTSCtrl::**getWorldToScreenScale**()
Get the ratio between world-space units and pixels.

Returns The amount of world-space units covered by the extent of a single pixel.

Point3F GuiTSCtrl::**project**(Point3F *worldPosition*)
Transform world-space coordinates to screen-space (x, y, depth) coordinates.

Parameters **worldPosition** – The world-space position to transform to screen-space.

Returns The

Point3F GuiTSCtrl::**unproject**(Point3F *screenPosition*)
Transform 3D screen-space coordinates (x, y, depth) to world space. This method can be, for example, used to find the world-space position relating to the current mouse cursor position.

Parameters **screenPosition** – The x/y position on the screen plus the depth from the screen-plane outwards.

Returns The world-space position corresponding to the given screen-space coordinates.

Fields

float GuiTSCtrl::**cameraZRot**
Z rotation angle of camera.

float GuiTSCtrl::**forceFOV**
The vertical field of view in degrees or zero to use the normal camera FOV.

float GuiTSCtrl::**reflectPriority**
The share of the per-frame reflection update work this control's rendering should run. The reflect update priorities of all visible GuiTSCtrls are added together and each control is assigned a share of the per-frame reflection update time according to its percentage of the total priority value.

GuiTSRenderStyles GuiTSCtrl::**renderStyle**
Indicates how this control should render its contents.

Enumeration

enum GuiTSRenderStyles
Style of rendering for a GuiTSCtrl .

Parameters

- **standard** –
- **side** –

Core Controls

Core parts of the Gui System.

Classes

GuiCanvas A canvas on which rendering occurs.

Inherit: GuiControl

Description A canvas on which rendering occurs.

What a GUICanvas Can Contain... A content control is the top level GuiControl for a screen. This GuiControl will be the parent control for all other GuiControls on that particular screen.

A dialog is essentially another screen, only it gets overlaid on top of the current content control, and all input goes to the dialog. This is most akin to the "Open File" dialog box found in most operating systems. When you choose to open a file, and the "Open File" dialog pops up, you can no longer send input to the application, and must complete or cancel the open file request. Torque keeps track of layers of dialogs. The dialog with the highest layer is on top and will get all the input, unless the dialog is modeless, which is a profile option.

Dirty Rectangles The GuiCanvas is based on dirty regions. Every frame the canvas paints only the areas of the canvas that are 'dirty' or need updating. In most cases, this only is the area under the mouse cursor. This is why if you look in guiCanvas.cc the call to glClear is commented out. What you will see is a black screen, except in the dirty regions, where the screen will be painted normally. If you are making an animated GuiControl you need to add your control to the dirty areas of the canvas.

Methods
Point2I GuiCanvas::**clientToScreen**(Point2I *coordinate*)
Translate a coordinate from canvas window-space to screen-space.

Parameters **coordinate** – The coordinate in window-space.

Returns The given coordinate translated to screen-space.

void GuiCanvas::**cursorOff**()
Turns on the mouse off.

Example:

```
Canvas.cursorOff();
```

void GuiCanvas::**cursorOn**()
Turns on the mouse cursor.

Example:

```
Canvas.cursorOn();
```

int GuiCanvas::**findFirstMatchingMonitor**(string *name*)
Find the first monitor index that matches the given name. The actual match algorithm depends on the implementation.

Parameters **name** – The name to search for.

Returns The number of monitors attached to the system, including the default monoitor.

int GuiCanvas::**getContent**()
Get the GuiControl which is being used as the content.

Returns ID of current content control

Example:

```
Canvas.getContent();
```

Point2I GuiCanvas::**getCursorPos**()
Get the current position of the cursor.

Parameters **param** – Description

Returns Screen coordinates of mouse cursor, in format "X Y"

Example:

```
%cursorPos = Canvas.getCursorPos();
```

Point2I GuiCanvas::**getExtent**()
Returns the dimensions of the canvas. Reimplemented from GuiControl .

Returns Width and height of canvas. Formatted as numerical values in a single string "# #"

Example:

```
%extent = Canvas.getExtent();
```

string GuiCanvas::**getMode**(int *modeId*)
Gets information on the specified mode of this device.

Parameters **modeId** – Index of the mode to get data from.

Returns A video mode string given an adapter and mode index.

int GuiCanvas::**getModeCount**()
Gets the number of modes available on this device.

Parameters **param** – Description

Returns The number of video modes supported by the device

Example:

```
%modeCount = Canvas.getModeCount()
```

int GuiCanvas::**getMonitorCount**()
Gets the number of monitors attached to the system.

Returns The number of monitors attached to the system, including the default monoitor.

string GuiCanvas::**getMonitorName**(int *index*)
Gets the name of the requested monitor.

Parameters **index** – The monitor index.

Returns The name of the requested monitor.

RectI GuiCanvas::**getMonitorRect**(int *index*)
Gets the region of the requested monitor.

Parameters **index** – The monitor index.

Returns The rectangular region of the requested monitor.

int GuiCanvas::**getMouseControl**()
Gets the gui control under the mouse.

Returns ID of the gui control, if one was found. NULL otherwise

Example:

```
%underMouse = Canvas.getMouseControl();
```

string GuiCanvas::**getVideoMode**()
Gets the current screen mode as a string. The return string will contain 5 values (width, height, fullscreen, bitdepth, refreshRate). You will need to parse out each one for individual use.

Returns String formatted with screen width, screen height, screen mode, bit depth, and refresh rate.

Example:

```
%screenWidth = getWord(Canvas.getVideoMode(), 0);
%screenHeight = getWord(Canvas.getVideoMode(), 1);
%isFullscreen = getWord(Canvas.getVideoMode(), 2);
%bitdepth = getWord(Canvas.getVideoMode(), 3);
%refreshRate = getWord(Canvas.getVideoMode(), 4);
```

Point2I GuiCanvas::**getWindowPosition**()
Get the current position of the platform window associated with the canvas.

Returns The window position of the canvas in screen-space.

void GuiCanvas::**hideCursor**()
Disable rendering of the cursor.

Example:

```
Canvas.hideCursor();
```

bool GuiCanvas::**isCursorOn**()
Determines if mouse cursor is enabled.

Returns Returns true if the cursor is on.

Example:

```
// Is cursor on?if(Canvas.isCursorOn())
   echo("Canvas cursor is on");
```

bool GuiCanvas::**isCursorShown**()
Determines if mouse cursor is rendering.

Returns Returns true if the cursor is rendering.

Example:

```
// Is cursor rendering?if(Canvas.isCursorShown())
   echo("Canvas cursor is rendering");
```

bool GuiCanvas::**isFullscreen**()
Is this canvas currently fullscreen?

bool GuiCanvas::**isMaximized**()

bool GuiCanvas::**isMinimized**()

void GuiCanvas::**maximizeWindow**()
maximize this canvas' window.

void GuiCanvas::**minimizeWindow**()
minimize this canvas' window.

void GuiCanvas::**popDialog**(GuiControl *ctrl*)
Removes a specific dialog control.

Parameters **ctrl** – Dialog to pop

Example:

```
Canvas.popDialog(RecordingsDlg);
```

void GuiCanvas::**popDialog**()
Removes a dialog at the front most layer.

Example:

```
// Pops whatever is on layer 0
Canvas.popDialog();
```

void GuiCanvas::**popLayer**()
Removes the top most layer of dialogs.

Example:

```
Canvas.popLayer();
```

void GuiCanvas::**popLayer**(S32 *layer*)
Removes a specified layer of dialogs.

Parameters **layer** – Number of the layer to pop

Example:

```
Canvas.popLayer(1);
```

void GuiCanvas::**pushDialog**(GuiControl *ctrl*, int *layer*, bool *center*)
Adds a dialog control onto the stack of dialogs.

Parameters

- **ctrl** – Dialog to add
- **layer** – Layer to put dialog on (optional)
- **center** – True to center dialog on canvas (optional)

Example:

```
Canvas.pushDialog(RecordingsDlg);
```

void GuiCanvas::**renderFront**(bool *enable*)
This turns on/off front-buffer rendering.

Parameters **enable** – True if all rendering should be done to the front buffer

Example:

```
Canvas.renderFront(false);
```

void GuiCanvas::**repaint**(int *elapsedMS*)
Force canvas to redraw. If the elapsed time is greater than the time since the last paint then the repaint will be skipped.

Parameters **elapsedMS** – The optional elapsed time in milliseconds.

Example:

```
Canvas.repaint();
```

void GuiCanvas::**reset**()
Reset the update regions for the canvas.

Example:

```
Canvas.reset();
```

void GuiCanvas::**restoreWindow**()
restore this canvas' window.

Point2I GuiCanvas::**screenToClient**(Point2I *coordinate*)
Translate a coordinate from screen-space to canvas window-space.

Parameters **coordinate** – The coordinate in screen-space.

Returns The given coordinate translated to window-space.

void GuiCanvas::**setContent**(GuiControl *ctrl*)
Set the content of the canvas to a specified control.

Parameters **ctrl** – ID or name of GuiControl to set content to

Example:

```
Canvas.setContent(PlayGui);
```

void GuiCanvas::**setCursor**(GuiCursor *cursor*)
Sets the cursor for the canvas.

Parameters **cursor** – Name of the GuiCursor to use

Example:

```
Canvas.setCursor("DefaultCursor");
```

bool GuiCanvas::**setCursorPos**(Point2I *pos*)
Sets the position of the cursor.

Parameters **pos** – Point, in screenspace for the cursor. Formatted as ("x y")

Example:

```
Canvas.setCursorPos("0 0");
```

bool GuiCanvas::**setCursorPos**(F32 *posX*, F32 *posY*)
Sets the position of the cursor.

Parameters

- **posX** – X-coordinate, in screenspace for the cursor.
- **posY** – Y-coordinate, in screenspace for the cursor.

Example:

```
Canvas.setCursorPos(0,0);
```

void GuiCanvas::**setFocus**()
Claim OS input focus for this canvas' window.

void GuiCanvas::**setVideoMode**(int *width*, int *height*, bool *fullscreen*)
Change the video mode of this canvas. This method has the side effect of setting the $pref::Video::mode to the new values.

Parameters

- **width** – The screen width to set.
- **height** – The screen height to set.
- **fullscreen** – Specify true to run fullscreen or false to run in a window
- **bitDepth** – [optional] The desired bit-depth. Defaults to the current setting. This parameter is ignored if you are running in a window.
- **refreshRate** – [optional] The desired refresh rate. Defaults to the current setting. This parameter is ignored if you are running in a window
- **antialiasLevel** – [optional] The level of anti-aliasing to apply 0 = none

void GuiCanvas::**setWindowPosition**(Point2I *position*)
Set the position of the platform window associated with the canvas.

Parameters **position** – The new position of the window in screen-space.

void GuiCanvas::**setWindowTitle**(string *newTitle*)
Change the title of the OS window.

Parameters **newTitle** – String containing the new name

Example:

```
Canvas.setWindowTitle("Documentation Rocks!");
```

void GuiCanvas::**showCursor**()
Enable rendering of the cursor.

Example:

```
Canvas.showCursor();
```

void GuiCanvas::**toggleFullscreen**()
toggle canvas from fullscreen to windowed mode or back.

Example:

```
    // If we are in windowed mode, the following will put is in fullscreen
    Canvas.toggleFullscreen();
```

Fields

bool GuiCanvas::**alwaysHandleMouseButtons**
    Deal with mouse buttons, even if the cursor is hidden.

int GuiCanvas::**numFences**
    The number of GFX fences to use.

GuiConsole The on-screen, in-game console.

Inherit: GuiArrayCtrl

Description Calls getLog() to get the on-screen console entries, then renders them as needed.

Example:

```
newGuiConsole()
      {
         //Properties not specific to this control have been omitted from this example.
      };
```

Methods

void GuiConsole::**onMessageSelected**(ConsoleLogEntry::Level *level*, string *message*)
    Called when a message in the log is clicked.

    Parameters

    - **level** – Diagnostic level of the message.
    - **message** – Message text.

GuiConsoleEditCtrl

Inherit: GuiTextEditCtrl

Description Text entry element of a GuiConsole.

Example:

```
newGuiConsoleEditCtrl(ConsoleEntry)
{
   profile = "ConsoleTextEditProfile";
   horizSizing = "width";
   vertSizing = "top";
   position = "0 462";
   extent = "640 18";
   minExtent = "8 8";
   visible = "1";
   altCommand = "ConsoleEntry::eval();";
   helpTag = "0";
   maxLength = "255";
   historySize = "40";
   password = "0";
   tabComplete = "0";
```

```
   sinkAllKeyEvents = "1";
   useSiblingScroller = "1";
};
```

Fields
bool GuiConsoleEditCtrl::**useSiblingScroller**

GuiControl Base class for all Gui control objects.

Inherit: SimGroup

Description GuiControl is the basis for the Gui system. It represents an individual control that can be placed on the canvas and with which the mouse and keyboard can potentially interact with.

Control Hierarchies GuiControls are arranged in a hierarchy. All children of a control are placed in their parent's coordinate space, i.e. their coordinates are relative to the upper left corner of their immediate parent. When a control is moved, all its child controls are moved along with it.

Since GuiControl's are SimGroups, hierarchy also implies ownership. This means that if a control is destroyed, all its children are destroyed along with it. It also means that a given control can only be part of a single GuiControl hierarchy. When adding a control to another control, it will automatically be reparented from another control it may have previously been parented to.

Layout System GuiControls have a two-dimensional position and are rectangular in shape.

Event System

Control Profiles Common data accessed by GuiControls is stored in so-called "Control Profiles." This includes font, color, and texture information. By pooling this data in shared objects, the appearance of any number of controls can be changed quickly and easily by modifying only the shared profile object.

If not explicitly assigned a profile, a control will by default look for a profile object that matches its class name. This means that the class GuiMyCtrl, for example, will look for a profile called 'GuiMyProfile'. If this profile cannot be found, the control will fall back to GuiDefaultProfile which must be defined in any case for the Gui system to work.

In addition to its primary profile, a control may be assigned a second profile called 'tooltipProfile' that will be used to render tooltip popups for the control.

Triggered Actions

First Responders At any time, a single control can be what is called the "first responder" on the GuiCanvas is placed on. This control will be the first control to receive keyboard events not bound in the global ActionMap. If the first responder choses to handle a particular keyboard event,

Waking and Sleeping

Visibility and Activeness By default, a GuiControl is active which means that it

Methods

void GuiControl::**addGuiControl**(GuiControl *control*)

Add the given control as a child to this control. This is synonymous to calling SimGroup::addObject.

Parameters **control** – The control to add as a child.

void GuiControl::**clearFirstResponder**(bool *ignored*)

Clear this control from being the first responder in its hierarchy chain.

Parameters **ignored** – Ignored. Supported for backwards-compatibility.

bool GuiControl::**controlIsChild**(GuiControl *control*)

Test whether the given control is a direct or indirect child to this control.

Parameters **control** – The potential child control.

Returns True if the given control is a direct or indirect child to this control.

GuiControl GuiControl::**findHitControl**(int *x*, int *y*)

Find the topmost child control located at the given coordinates.

Parameters

- **x** – The X coordinate in the control's own coordinate space.
- **y** – The Y coordinate in the control's own coordinate space.

Returns The topmost child control at the given coordintes or the control on which the method was called if no matching child could be found.

string GuiControl::**findHitControls**(int *x*, int *y*, int *width*, int *height*)

Find all visible child controls that intersect with the given rectangle.

Parameters

- **x** – The X coordinate of the rectangle's upper left corner in the control's own coordinate space.
- **y** – The Y coordinate of the rectangle's upper left corner in the control's own coordinate space.
- **width** – The width of the search rectangle in pixels.
- **height** – The height of the search rectangle in pixels.

Returns A space-separated list of the IDs of all visible control objects intersecting the given rectangle.

Example:

```
// Lock all controls in the rectangle at x=10 and y=10 and the extent width=100 and height=100.f
   %ctrl.setLocked( true );
```

float GuiControl::**getAspect**()

Get the aspect ratio of the control's extents.

Returns The width of the control divided by its height.

Point2I GuiControl::**getCenter**()

Get the coordinate of the control's center point relative to its parent.

Returns The coordinate of the control's center point in parent-relative coordinates.

Point2I GuiControl::**getExtent**()

Get the width and height of the control. Reimplemented in GuiCanvas .

Returns A point structure containing the width of the control in x and the height in y.

GuiControl GuiControl::**getFirstResponder**()
Get the first responder set on this GuiControl tree.

Returns The first responder set on the control's subtree.

Point2I GuiControl::**getGlobalCenter**()
Get the coordinate of the control's center point in coordinates relative to the root control in its control hierarchy. the center coordinate of the control in root-relative coordinates.

Point2I GuiControl::**getGlobalPosition**()
Get the position of the control relative to the root of the GuiControl hierarchy it is contained in.

Returns The control's current position in root-relative coordinates.

Point2I GuiControl::**getMinExtent**()
Get the minimum allowed size of the control.

Returns The minimum size to which the control can be shrunk.

GuiControl GuiControl::**getParent**()
Get the immediate parent control of the control.

Returns

.

Point2I GuiControl::**getPosition**()
Get the control's current position relative to its parent.

Returns The coordinate of the control in its parent's coordinate space.

GuiCanvas GuiControl::**getRoot**()
Get the canvas on which the control is placed.

Returns

.

bool GuiControl::**isAwake**()
Test whether the control is currently awake. If a control is awake it means that it is part of the GuiControl hierarchy of a GuiCanvas .

Returns Waking and Sleeping

bool GuiControl::**isFirstResponder**()
Test whether the control is the current first responder.

Returns True if the control is the current first responder.

bool GuiControl::**isMouseLocked**()
Indicates if the mouse is locked in this control.

Returns True if the mouse is currently locked.

bool GuiControl::**isVisible**()
Test whether the control is currently set to be visible. Visibility and Activeness

Returns True if the control is currently set to be visible.

void GuiControl::**makeFirstResponder**(bool *isFirst*)

void GuiControl::**onAction**()
Called when the control's associated action is triggered and no 'command' is defined for the control. Triggered Actions

void GuiControl::**onActive**(bool *state*)
Called when the control changes its activeness state, i.e. when going from active to inactive or vice versa.

Parameters **stat** – The new activeness state.

void GuiControl::**onAdd**()
Called when the control object is registered with the system after the control has been created.

void GuiControl::**onControlDragEnter**(GuiControl *control*, Point2I *dropPoint*)
Called when a drag amp drop operation through GuiDragAndDropControl has entered the control. This is only called for topmost visible controls as the GuiDragAndDropControl moves over them.

Parameters

- **control** – The payload of the drag operation.
- **dropPoint** – The point at which the payload would be dropped if it were released now. Relative to the canvas.

void GuiControl::**onControlDragExit**(GuiControl *control*, Point2I *dropPoint*)
Called when a drag amp drop operation through GuiDragAndDropControl has exited the control and moved over a different control. This is only called for topmost visible controls as the GuiDragAndDropControl moves off of them.

Parameters

- **control** – The payload of the drag operation.
- **dropPoint** – The point at which the payload would be dropped if it were released now. Relative to the canvas.

void GuiControl::**onControlDragged**(GuiControl *control*, Point2I *dropPoint*)
Called when a drag amp drop operation through GuiDragAndDropControl is moving across the control after it has entered it. This is only called for topmost visible controls as the GuiDragAndDropControl moves across them.

Parameters

- **control** – The payload of the drag operation.
- **dropPoint** – The point at which the payload would be dropped if it were released now. Relative to the canvas.

void GuiControl::**onControlDropped**(GuiControl *control*, Point2I *dropPoint*)
Called when a drag amp drop operation through GuiDragAndDropControl has completed and is dropping its payload onto the control. This is only called for topmost visible controls as the GuiDragAndDropControl drops its payload on them.

Parameters

- **control** – The control that is being dropped onto this control.
- **dropPoint** – The point at which the control is being dropped. Relative to the canvas.

void GuiControl::**onDialogPop**()
Called when the control is removed as a dialog from the canvas.

void GuiControl::**onDialogPush**()
Called when the control is pushed as a dialog onto the canvas.

void GuiControl::**onGainFirstResponder**()
Called when the control gains first responder status on the GuiCanvas .

void GuiControl::**onLoseFirstResponder**()
Called when the control loses first responder status on the GuiCanvas .

void GuiControl::**onRemove**()
Called when the control object is removed from the system before it is deleted.

void GuiControl::**onSleep**()
Called when the control is put to sleep. Waking and Sleeping

void GuiControl::**onVisible**(bool *state*)
Called when the control changes its visibility state, i.e. when going from visible to invisible or vice versa.

Parameters **state** – The new visibility state.

void GuiControl::**onWake**()
Called when the control is woken up. Waking and Sleeping

bool GuiControl::**pointInControl**(int *x*, int *y*)
Test whether the given point lies within the rectangle of the control.

Parameters

- **x** – X coordinate of the point in parent-relative coordinates.
- **y** – Y coordinate of the point in parent-relative coordinates.

Returns True if the point is within the control, false if not.

void GuiControl::**resize**(int *x*, int *y*, int *width*, int *height*)
Resize and reposition the control using the give coordinates and dimensions. Child controls will resize according to their layout behaviors.

Parameters

- **x** – The new X coordinate of the control in its parent's coordinate space.
- **y** – The new Y coordinate of the control in its parent's coordinate space.
- **width** – The new width to which the control should be resized.
- **height** – The new height to which the control should be resized.

void GuiControl::**setActive**(bool *state*)

void GuiControl::**setCenter**(int *x*, int *y*)
Set the control's position by its center point.

Parameters

- **x** – The X coordinate of the new center point of the control relative to the control's parent.
- **y** – The Y coordinate of the new center point of the control relative to the control's parent.

void GuiControl::**setExtent**(S32 *width*, S32 *height*)
Resize the control to the given dimensions. Child controls will resize according to their layout settings.

Parameters

- **width** – The new width of the control in pixels.
- **height** – The new height of the control in pixels.

void GuiControl::**setExtent**(Point2I *p*)
Resize the control to the given dimensions. Child controls with resize according to their layout settings.

Parameters **p** – The new (width, height) extents of the control.

void GuiControl::**setFirstResponder**()
Make this control the current first responder.

void GuiControl::**setPosition**(int *x*, int *y*)
Position the control in the local space of the parent control.

Parameters

- **x** – The new X coordinate of the control relative to its parent's upper left corner.
- **y** – The new Y coordinate of the control relative to its parent's upper left corner.

void GuiControl::**setPositionGlobal**(int *x*, int *y*)
Set position of the control relative to the root of the GuiControl hierarchy it is contained in.

Parameters

- **x** – The new X coordinate of the control relative to the root's upper left corner.
- **y** – The new Y coordinate of the control relative to the root's upper left corner.

void GuiControl::**setProfile**(GuiControlProfile *profile*)
Set the control profile for the control to use. The profile used by a control determines a great part of its behavior and appearance.

Parameters **profile** – The new profile the control should use. Control Profiles

void GuiControl::**setValue**(string *value*)
Set the value associated with the control.

Parameters **value** – The new value for the control.

void GuiControl::**setVisible**(bool *state*)
Set whether the control is visible or not.

Parameters **state** – The new visiblity flag state for the control. Visibility and Activeness

Fields

string GuiControl::**accelerator**
Key combination that triggers the control's primary action when the control is on the canvas.

bool GuiControl::**active**
Whether the control is enabled for user interaction.

string GuiControl::**altCommand**
Command to execute on the secondary action of the control.

string GuiControl::**command**
Command to execute on the primary action of the control.

Point2I GuiControl::**extent**
The width and height of the control.

string GuiControl::**getValue**

GuiHorizontalSizing GuiControl::**horizSizing**
The horizontal resizing behavior.

int GuiControl::**hovertime**
Time for mouse to hover over control until tooltip is shown (in milliseconds).

bool GuiControl::**isActive**

bool GuiControl::**isContainer**
If true, the control may contain child controls.

string GuiControl::**langTableMod**
Name of string table to use for lookup of internationalized text.

Point2I GuiControl::**minExtent**
The minimum width and height of the control. The control will not be resized smaller than this.

deprecated GuiControl::**modal**

Point2I GuiControl::**position**
The position relative to the parent control.

GuiControlProfile GuiControl::**profile**
The control profile that determines fill styles, font settings, etc.

deprecated GuiControl::**setFirstResponder**

string GuiControl::**tooltip**
String to show in tooltip for this control.

GuiControlProfile GuiControl::**tooltipProfile**
Control profile to use when rendering tooltips for this control.

string GuiControl::**variable**
Name of the variable to which the value of this control will be synchronized.

GuiVerticalSizing GuiControl::**vertSizing**
The vertical resizing behavior.

bool GuiControl::**visible**
Whether the control is visible or hidden.

GuiControlProfile

Inherit: SimObject

Description A collection of properties that determine control behavior and rendering.

Methods

int GuiControlProfile::**getStringWidth**()

Fields

bool GuiControlProfile::**autoSizeHeight**
Automatically adjust height of control to fit contents.

bool GuiControlProfile::**autoSizeWidth**
Automatically adjust width of control to fit contents.

ColorI GuiControlProfile::**bevelColorHL**

ColorI GuiControlProfile::**bevelColorLL**

filename GuiControlProfile::**bitmap**
Texture to use for rendering control.

int GuiControlProfile::**border**
Border type (0=no border).

ColorI GuiControlProfile::**borderColor**
Color to draw border with.

ColorI GuiControlProfile::**borderColorHL**

ColorI GuiControlProfile::**borderColorNA**

int GuiControlProfile::**borderThickness**
Thickness of border in pixels.

bool GuiControlProfile::**canKeyFocus**
Whether the control can have the keyboard focus.

string GuiControlProfile::**category**
Category under which the profile will appear in the editor.

ColorI GuiControlProfile::**cursorColor**
Color to use for the text cursor.

ColorI GuiControlProfile::**fillColor**

ColorI GuiControlProfile::**fillColorHL**

ColorI GuiControlProfile::**fillColorNA**

ColorI GuiControlProfile::**fillColorSEL**

GuiFontCharset GuiControlProfile::**fontCharset**

ColorI GuiControlProfile::**fontColor**
Font color for normal text (same as fontColors[0]).

ColorI GuiControlProfile::**fontColorHL**
Font color for highlighted text (same as fontColors[1]).

ColorI GuiControlProfile::**fontColorLink**
Font color for links in text (same as fontColors[4]).

ColorI GuiControlProfile::**fontColorLinkHL**
Font color for highlighted links in text (same as fontColors[5]).

ColorI GuiControlProfile::**fontColorNA**
Font color when control is not active/disabled (same as fontColors[2]).

ColorI GuiControlProfile::**fontColors**[10]
Font colors to use for different text types/states.

ColorI GuiControlProfile::**fontColorSEL**
Font color for selected text (same as fontColors[3]).

int GuiControlProfile::**fontSize**
Font size in points.

string GuiControlProfile::**fontType**
Name of font family and typeface (e.g. “Arial Bold”).

bool GuiControlProfile::**hasBitmapArray**
If true, ‘bitmap’ is an array of images.

GuiAlignmentType GuiControlProfile::**justify**
Horizontal alignment for text.

bool GuiControlProfile::**modal**

bool GuiControlProfile::**mouseOverSelected**

bool GuiControlProfile::**numbersOnly**
Whether control should only accept numerical data (GuiTextEditCtrl).

bool GuiControlProfile::**opaque**

string GuiControlProfile::**profileForChildren**

bool GuiControlProfile::**returnTab**
Whether to add automatic tab event when return is pressed so focus moves on to next control (GuiTextEditCtrl).

SFXTrack GuiControlProfile::**soundButtonDown**
Sound to play when mouse has been pressed on control.

SFXTrack GuiControlProfile::**soundButtonOver**
Sound to play when mouse is hovering over control.

bool GuiControlProfile::**tab**

Point2I GuiControlProfile::**textOffset**

GuiCursor Acts as a skin for the cursor, where each GuiCursor object can have its own look and click-zone.

Inherit: SimObject

Description GuiCursors act as skins for the cursor in the game, where each individual GuiCursor can have its own defined imagemap, click zone and render offset. This allows a game to easily support a wide range of cursors. The active cursor can de changed for each Canvas using canvasObj.setCursor(GuiCursor);.

Example:

```
newGuiCursor(DefaultCursor)
{
   hotSpot = "1 1";
   renderOffset = "0 0";
   bitmapName = "~/art/gui/images/defaultCursor";
};
```

Fields

filename GuiCursor::**bitmapName**
File name of the bitmap for the cursor.

Point2I GuiCursor::**hotSpot**
The location of the cursor's hot spot (which pixel carries the click).

Point2F GuiCursor::**renderOffset**
Offset of the bitmap, where 0 signifies left edge of the bitmap, 1, the right. Similarly for the Y-component.

GuiFadeinBitmapCtrl A GUI control which renders a black square over a bitmap image. The black square will fade out, then fade back in after a determined time. This control is especially useful for transitions and splash screens.

Inherit: GuiBitmapCtrl

Description A GUI control which renders a black square over a bitmap image. The black square will fade out, then fade back in after a determined time. This control is especially useful for transitions and splash screens.

Example:

```
newGuiFadeinBitmapCtrl()
   {
      fadeinTime = "1000";
      waitTime = "2000";
      fadeoutTime = "1000";
      done = "1";
```

```
      // Additional GUI properties that are not specific to GuiFadeinBitmapCtrl have been omitted fr
   };
```

Methods

void GuiFadeinBitmapCtrl::**click**()

Informs the script level that this object received a Click event from the cursor or keyboard.

Example:

```
GuiFadeInBitmapCtrl::click(%this)
   {
      // Code to run when click occurs
   }
```

void GuiFadeinBitmapCtrl::**onDone**()

Informs the script level that this object has completed is fade cycle.

Example:

```
GuiFadeInBitmapCtrl::onDone(%this)
   {
      // Code to run when the fade cycle completes
   }
```

Fields

bool GuiFadeinBitmapCtrl::**done**

Whether the fade cycle has finished running.

ColorF GuiFadeinBitmapCtrl::**fadeColor**

Color to fade in from and fade out to.

EaseF GuiFadeinBitmapCtrl::**fadeInEase**

Easing curve for fade-in.

int GuiFadeinBitmapCtrl::**fadeInTime**

Milliseconds for the bitmap to fade in.

EaseF GuiFadeinBitmapCtrl::**fadeOutEase**

Easing curve for fade-out.

int GuiFadeinBitmapCtrl::**fadeOutTime**

Milliseconds for the bitmap to fade out.

int GuiFadeinBitmapCtrl::**waitTime**

Milliseconds to wait after fading in before fading out the bitmap.

GuiIconButtonCtrl Draws the bitmap within a special button control. Only a single bitmap is used and the button will be drawn in a highlighted mode when the mouse hovers over it or when it has been clicked.

Inherit: GuiButtonCtrl

Description Draws the bitmap within a special button control. Only a single bitmap is used and the button will be drawn in a highlighted mode when the mouse hovers over it or when it has been clicked.

Example:

```
newGuiIconButtonCtrl(TestIconButton)
{
   buttonMargin = "4 4";
   iconBitmap = "art/gui/lagIcon.png";
   iconLocation = "Center";
   sizeIconToButton = "0";
   makeIconSquare = "1";
   textLocation = "Bottom";
   textMargin = "-2";
   autoSize = "0";
   text = "Lag Icon";
   textID = ""STR_LAG"";
   buttonType = "PushButton";
   profile = "GuiIconButtonProfile";
};
```

Methods

void GuiIconButtonCtrl::**setBitmap**(string *buttonFilename*)

Set the bitmap to use for the button portion of this control.

Parameters **buttonFilename** – Filename for the image

Example:

```
// Define the button filename
%buttonFilename = "pearlButton";

// Inform the GuiIconButtonCtrl control to update its main button graphic to the defined bitmap
%thisGuiIconButtonCtrl.setBitmap(%buttonFilename);
```

Fields

bool GuiIconButtonCtrl::**autoSize**

If true, the text and icon will be automatically sized to the size of the control.

Point2I GuiIconButtonCtrl::**buttonMargin**

Margin area around the button.

filename GuiIconButtonCtrl::**iconBitmap**

Bitmap file for the icon to display on the button.

GuiIconButtonIconLocation GuiIconButtonCtrl::**iconLocation**

Where to place the icon on the control. Options are 0 (None), 1 (Left), 2 (Right), 3 (Center).

bool GuiIconButtonCtrl::**makeIconSquare**

If true, will make sure the icon is square.

bool GuiIconButtonCtrl::**sizeIconToButton**

If true, the icon will be scaled to be the same size as the button.

GuiIconButtonTextLocation GuiIconButtonCtrl::**textLocation**

Where to place the text on the control. Options are 0 (None), 1 (Bottom), 2 (Right), 3 (Top), 4 (Left), 5 (Center).

int GuiIconButtonCtrl::**textMargin**

Margin between the icon and the text.

GuiListBoxCtrl A list of text items.

Inherit: GuiControl

Description A list of text items where each individual entry can have its own text value, text color and associated SimObject.

Example:

```
newGuiListBoxCtrl(GuiMusicPlayerMusicList)
{
   allowMultipleSelections = "true";
   fitParentWidth = "true";
   mirrorSet = "AnotherGuiListBoxCtrl";
   makeNameCallback = "";
   colorBullet = "1";
   //Properties not specific to this control have been omitted from this example.
};
```

Methods

void GuiListBoxCtrl::**addFilteredItem**(string *newItem*)

Checks if there is an item with the exact text of what is passed in, and if so the item is removed from the list and adds that item's data to the filtered list.

Parameters **itemName** – Name of the item that we wish to add to the filtered item list of the GuiListBoxCtrl.

Example:

```
// Define the itemName that we wish to add to the filtered item list.
%itemName = "This Item Name";

// Add the item name to the filtered item list.
%thisGuiListBoxCtrl.addFilteredItem(%filteredItemName);
```

void GuiListBoxCtrl::**clearItemColor**(int *index*)

Removes any custom coloring from an item at the defined index id in the list.

Parameters **index** – Index id for the item to clear any custom color from.

Example:

```
// Define the index id
%index = "4";

// Request the GuiListBoxCtrl object to remove any custom coloring from the defined index entry
%thisGuiListBoxCtrl.clearItemColor(%index);
```

void GuiListBoxCtrl::**clearItems**()

Clears all the items in the listbox.

Example:

```
// Inform the GuiListBoxCtrl object to clear all items from its list.
%thisGuiListBoxCtrl.clearItems();
```

void GuiListBoxCtrl::**clearSelection**()

Sets all currently selected items to unselected. Detailed description

Example:

```
// Inform the GuiListBoxCtrl object to set all of its items to unselected./n%thisGuiListBoxCtrl.
```

void GuiListBoxCtrl::**deleteItem**(int *itemIndex*)

Removes the list entry at the requested index id from the control and clears the memory associated with it.

Parameters **itemIndex** – Index id location to remove the item from.

Example:

```
// Define the index id we want to remove from the list
%itemIndex = "8";

// Inform the GuiListBoxCtrl object to remove the item at the defined index id.
%thisGuiListBoxCtrl.deleteItem(%itemIndex);
```

void GuiListBoxCtrl::**doMirror**()

Informs the GuiListBoxCtrl object to mirror the contents of the GuiListBoxCtrl stored in the mirrorSet field.

Example:

```
\ Inform the object to mirror the object located at %thisGuiListBox.mirrorSet
%thisGuiListBox.doMirror();
```

int GuiListBoxCtrl::**findItemText**(string *findText*, bool *bCaseSensitive*)

Returns index of item with matching text or -1 if none found.

Parameters

- **findText** – Text in the list to find.
- **isCaseSensitive** – If true, the search will be case sensitive.

Returns Index id of item with matching text or -1 if none found.

Example:

```
// Define the text we wish to find in the list.
%findText = "Hickory Smoked Gideon"/n/n// Define if this is a case sensitive search or not.
%isCaseSensitive = "false";

// Ask the GuiListBoxCtrl object what item id in the list matches the requested text.
%matchingId = %thisGuiListBoxCtrl.findItemText(%findText,%isCaseSensitive);
```

int GuiListBoxCtrl::**getItemCount**()

Returns the number of items in the list.

Returns The number of items in the list.

Example:

```
// Request the number of items in the list of the GuiListBoxCtrl object.
%listItemCount = %thisGuiListBoxCtrl.getItemCount();
```

string GuiListBoxCtrl::**getItemObject**(int *index*)

Returns the object associated with an item. This only makes sense if you are mirroring a simset.

Parameters **index** – Index id to request the associated item from.

Returns The object associated with the item in the list.

Example:

```
// Define the index id
%index = "12";

// Request the item from the GuiListBoxCtrl object
%object = %thisGuiListBoxCtrl.getItemObject(%index);
```

string GuiListBoxCtrl::**getItemText**(int *index*)
Returns the text of the item at the specified index.

Parameters **index** – Index id to return the item text from.

Returns The text of the requested index id.

Example:

```
// Define the index id entry to request the text from
%index = "12";

// Request the item id text from the GuiListBoxCtrl object.
%text = %thisGuiListBoxCtrl.getItemText(%index);
```

int GuiListBoxCtrl::**getLastClickItem**()
Request the item index for the item that was last clicked.

Returns Index id for the last clicked item in the list.

Example:

```
// Request the item index for the last clicked item in the list
%lastClickedIndex = %thisGuiListBoxCtrl.getLastClickItem();
```

int GuiListBoxCtrl::**getSelCount**()
Returns the number of items currently selected.

Returns Number of currently selected items.

Example:

```
// Request the number of currently selected items
%selectedItemCount = %thisGuiListBoxCtrl.getSelCount();
```

int GuiListBoxCtrl::**getSelectedItem**()
Returns the selected items index or -1 if none selected. If multiple selections exist it returns the first selected item.

Returns The selected items index or -1 if none selected.

Example:

```
// Request the index id of the currently selected item
%selectedItemId = %thisGuiListBoxCtrl.getSelectedItem();
```

string GuiListBoxCtrl::**getSelectedItems**()
Returns a space delimited list of the selected items indexes in the list.

Returns Space delimited list of the selected items indexes in the list

Example:

```
// Request a space delimited list of the items in the GuiListBoxCtrl object.
%selectionList = %thisGuiListBoxCtrl.getSelectedItems();
```

void GuiListBoxCtrl::**insertItem**(string *text*, int *index*)
Inserts an item into the list at the specified index and returns the index assigned or -1 on error.

Parameters

- **text** – Text item to add.
- **index** – Index id to insert the list item text at.

Returns If successful will return the index id assigned. If unsuccessful, will return -1.

Example:

```
// Define the text to insert
%text = "Secret Agent Gideon";

// Define the index entry to insert the text at
%index = "14";

// In form the GuiListBoxCtrl object to insert the text at the defined index.
%assignedId = %thisGuiListBoxCtrl.insertItem(%text,%index);
```

bool GuiListBoxCtrl::**isObjectMirrored**(string *indexIdString*)

Checks if a list item at a defined index id is mirrored, and returns the result.

Parameters **indexIdString** – Index id of the list to check.

Returns A boolean value on if the list item is mirrored or not.

Example:

```
// Engine has requested of the script level to determine if a list entry is mirrored or not.GuiL
   {
      // Perform code required to check and see if the list item at the index id is mirrored or
   }
```

void GuiListBoxCtrl::**onClearSelection**()

Called whenever a selected item in the list is cleared.

Example:

```
// A selected item is cleared, causing the callback to occur.GuiListBoxCtrl::onClearSelection(%t
   {
      // Code to run whenever a selected item is cleared
   }
```

void GuiListBoxCtrl::**onDeleteKey**()

Called whenever the Delete key on the keyboard has been pressed while in this control.

Example:

```
// The delete key on the keyboard has been pressed while this control is in focus, causing the c
   {
      // Code to call whenever the delete key is pressed
   }
```

void GuiListBoxCtrl::**onDoubleClick**()

Called whenever an item in the list has been double clicked.

Example:

```
// An item in the list is double clicked, causing the callback to occur.GuiListBoxCtrl::onDouble
   {
      // Code to run whenever an item in the control has been double clicked
   }
```

void GuiListBoxCtrl::**onMouseDragged**()

Called whenever the mouse is dragged across the control.

Example:

```
// Mouse is dragged across the control, causing the callback to occur.GuiListBoxCtrl::onMouseDra
   {
      // Code to run whenever the mouse is dragged across the control
   }
```

void GuiListBoxCtrl::**onMouseUp**(string *itemHit*, string *mouseClickCount*)

Called whenever the mouse has previously been clicked down (onMouseDown) and has now been raised on the control. If an item in the list was hit during the click cycle, then the index id of the clicked object along with how many clicks occured are passed into the callback. Detailed description

Parameters

- **itemHit** – Index id for the list item that was hit
- **mouseClickCount** – How many mouse clicks occured on this list item

Example:

```
// Mouse was previously clicked down, and now has been released, causing the callback to occur.G
   {
      // Code to call whenever the mouse has been clicked and released on the control
   }
```

void GuiListBoxCtrl::**onSelect**(string *index*, string *itemText*)

Called whenever an item in the list is selected.

Parameters

- **index** – Index id for the item in the list that was selected.
- **itemText** – Text for the list item at the index that was selected.

Example:

```
// An item in the list is selected, causing the callback to occurGuiListBoxCtrl::onSelect(%this,
   {
      // Code to run whenever an item in the list is selected
   }
```

void GuiListBoxCtrl::**onUnselect**(string *index*, string *itemText*)

Called whenever a selected item in the list has been unselected.

Parameters

- **index** – Index id of the item that was unselected
- **itemText** – Text for the list entry at the index id that was unselected

Example:

```
// A selected item is unselected, causing the callback to occur
GuiListBoxCtrl::onUnSelect(%this, %indexId, %itemText)
   {
      // Code to run whenever a selected list item is unselected
   }
```

void GuiListBoxCtrl::**removeFilteredItem**(string *itemName*)

Removes an item of the entered name from the filtered items list.

Parameters **itemName** – Name of the item to remove from the filtered list.

Example:

```
// Define the itemName that you wish to remove.
%itemName = "This Item Name";

// Remove the itemName from the GuiListBoxCtrl
%thisGuiListBoxCtrl.removeFilteredItem(%itemName);
```

void GuiListBoxCtrl::**setCurSel**(int *indexId*)
Sets the currently selected item at the specified index.

Parameters **indexId** – Index Id to set selected.

Example:

```
// Define the index id that we wish to select.
%selectId = "4";

// Inform the GuiListBoxCtrl object to set the requested index as selected.
%thisGuiListBoxCtrl.setCurSel(%selectId);
```

void GuiListBoxCtrl::**setCurSelRange**(int *indexStart*, int *indexStop*)
Sets the current selection range from index start to stop. If no stop is specified it sets from start index to the end of the list.

Parameters

- **indexStart** – Index Id to start selection.
- **indexStop** – Index Id to end selection.

Example:

```
// Set start id
%indexStart = "3";

// Set end id
%indexEnd = "6";

// Request the GuiListBoxCtrl object to select the defined range.
%thisGuiListBoxCtrl.setCurSelRange(%indexStart,%indexEnd);
```

void GuiListBoxCtrl::**setItemColor**(int *index*, ColorF *color*)
Sets the color of a single list entry at the specified index id.

Parameters

- **index** – Index id to modify the color of in the list.
- **color** – Color value to set the list entry to.

Example:

```
// Define the index id value
%index = "5";

// Define the color value
%color = "1.0 0.0 0.0";

// Inform the GuiListBoxCtrl object to change the color of the requested index
%thisGuiListBoxCtrl.setItemColor(%index,%color);
```

void GuiListBoxCtrl::**setItemText**(int *index*, string *newtext*)
Sets the items text at the specified index.

Parameters

- **index** – Index id to set the item text at.
- **newtext** – Text to change the list item at index id to.

Example:

```
// Define the index id/n%index = "12";// Define the text to set the list item to
%newtext = "Gideons Fancy Goggles";

// Inform the GuiListBoxCtrl object to change the text at the requested index
%thisGuiListBoxCtrl.setItemText(%index,%newText);
```

void GuiListBoxCtrl::**setItemTooltip**(int *index*, string *text*)
Set the tooltip text to display for the given list item.

Parameters

- **index** – Index id to change the tooltip text
- **text** – Text for the tooltip.

Example:

```
// Define the index id
%index = "12";

// Define the tooltip text
%tooltip = "Gideons goggles can see through space and time."// Inform the GuiListBoxCtrl object
%thisGuiListBoxCtrl.setItemToolTip(%index,%tooltip);
```

void GuiListBoxCtrl::**setMultipleSelection**(bool *allowMultSelections*)
Enable or disable multiple selections for this GuiListBoxCtrl object.

Parameters **allowMultSelections** – Boolean variable to set the use of multiple selections or not.

Example:

```
// Define the multiple selection use state.
%allowMultSelections = "true";

// Set the allow  multiple selection state on the GuiListBoxCtrl object.
%thisGuiListBoxCtrl.setMultipleSelection(%allowMultSelections);
```

void GuiListBoxCtrl::**setSelected**(int *index*, bool *setSelected*)
Sets the item at the index specified to selected or not. Detailed description

Parameters

- **index** – Item index to set selected or unselected.
- **setSelected** – Boolean selection state to set the requested item index.

Example:

```
// Define the index
%index = "5";

// Define the selection state
%selected = "true"// Inform the GuiListBoxCtrl object of the new selection state for the request
%thisGuiListBoxCtrl.setSelected(%index,%selected);
```

Fields

bool GuiListBoxCtrl::**allowMultipleSelections**
If true, will allow the selection of multiple items in the listbox.

bool GuiListBoxCtrl::**colorBullet**
If true, colored items will render a colored rectangular bullet next to the item text.

bool GuiListBoxCtrl::**fitParentWidth**
If true, the width of the listbox will match the width of its parent control.

string GuiListBoxCtrl::**makeNameCallback**
A script snippet to control what is displayed in the list for a SimObject . Within this snippet, $ThisControl is bound to the guiListBoxCtrl and $ThisObject to the contained object in question.

string GuiListBoxCtrl::**mirrorSet**
If populated with the name of another GuiListBoxCtrl , then this list box will mirror the contents of the mirrorSet listbox.

GuiMenuBar GUI Control which displays a horizontal bar with individual drop-down menu items. Each menu item may also have submenu items.

Inherit: GuiTickCtrl

Description GUI Control which displays a horizontal bar with individual drop-down menu items. Each menu item may also have submenu items.

Example:

```
newGuiMenuBar(newMenuBar)
{
   Padding = "0";
   //Properties not specific to this control have been omitted from this example.
};

// Add a menu to the menu bar
newMenuBar.addMenu(0,"New Menu");

// Add a menu item to the New Menu
newMenuBar.addMenuItem(0,"New Menu Item",0,"n",-1);

// Add a submenu item to the New Menu Item
newMenuBar.addSubmenuItem(0,1,"New Submenu Item",0,"s",-1);
```

Methods

void GuiMenuBar::**addMenu**(string *menuText*, int *menuId*)
Adds a new menu to the menu bar.

Parameters

- **menuText** – Text to display for the new menu item.
- **menuId** – ID for the new menu item.

Example:

```
    // Define the menu text
    %menuText = "New Menu";

    // Define the menu ID.
```

```
%menuId = "2";

// Inform the GuiMenuBar control to add the new menu
%thisGuiMenuBar.addMenu(%menuText,%menuId);
```

void GuiMenuBar::**addMenuItem**(string *targetMenu*, string *menuItemText*, int *menuItemId*, string *accelerator*, int *checkGroup*)

Adds a menu item to the specified menu. The menu argument can be either the text of a menu or its id.

Parameters

- **menu** – Menu name or menu Id to add the new item to.
- **menuItemText** – Text for the new menu item.
- **menuItemId** – Id for the new menu item.
- **accelerator** – Accelerator key for the new menu item.
- **checkGroup** – Check group to include this menu item in.

Example:

```
// Define the menu we wish to add the item to
%targetMenu = "New Menu";  or  %menu = "4";

// Define the text for the new menu item
%menuItemText = "Menu Item";

// Define the id for the new menu item
%menuItemId = "3";

// Set the accelerator key to toggle this menu item with
%accelerator = "n";

// Define the Check Group that this menu item will be in, if we want it to be in a check group.
%checkGroup = "4";

// Inform the GuiMenuBar control to add the new menu item with the defined fields
%thisGuiMenuBar.addMenuItem(%menu,%menuItemText,%menuItemId,%accelerator,%checkGroup);
```

void GuiMenuBar::**addSubmenuItem**(string *menuTarget*, string *menuItem*, string *submenuItemText*, int *submenuItemId*, string *accelerator*, int *checkGroup*)

Adds a menu item to the specified menu. The menu argument can be either the text of a menu or its id.

Parameters

- **menuTarget** – Menu to affect a submenu in
- **menuItem** – Menu item to affect
- **submenuItemText** – Text to show for the new submenu
- **submenuItemId** – Id for the new submenu
- **accelerator** – Accelerator key for the new submenu
- **checkGroup** – Which check group the new submenu should be in, or -1 for none.

Example:

```
// Define the menuTarget
%menuTarget = "New Menu";  or  %menuTarget = "3";

// Define the menuItem
```

```
%menuItem = "New Menu Item";  or  %menuItem = "5";

// Define the text for the new submenu
%submenuItemText = "New Submenu Item";

// Define the id for the new submenu
%submenuItemId = "4";

// Define the accelerator key for the new submenu
%accelerator = "n";

// Define the checkgroup for the new submenu
%checkgroup = "7";

// Request the GuiMenuBar control to add the new submenu with the defined information
%thisGuiMenuBar.addSubmenuItem(%menuTarget,%menuItem,%submenuItemText,%submenuItemId,%acceleratc
```

void GuiMenuBar::**clearMenuItems**(string *menuTarget*)

Removes all the menu items from the specified menu.

Parameters **menuTarget** – Menu to remove all items from

Example:

```
// Define the menuTarget
%menuTarget = "New Menu";  or %menuTarget = "3";

// Inform the GuiMenuBar control to clear all menu items from the defined menu
%thisGuiMenuBar.clearMenuItems(%menuTarget);
```

void GuiMenuBar::**clearMenus**(int *param1*, int *param2*)

Clears all the menus from the menu bar.

Example:

```
// Inform the GuiMenuBar control to clear all menus from itself.
%thisGuiMenuBar.clearMenus();
```

void GuiMenuBar::**clearSubmenuItems**(string *menuTarget*, string *menuItem*)

Removes all the menu items from the specified submenu.

Parameters

- **menuTarget** – Menu to affect a submenu in
- **menuItem** – Menu item to affect

Example:

```
// Define the menuTarget
%menuTarget = "New Menu";  or %menuTarget = "3";

// Define the menuItem
%menuItem = "New Menu Item";  or  %menuItem = "5";

// Inform the GuiMenuBar to remove all submenu items from the defined menu item
%thisGuiMenuBar.clearSubmenuItems(%menuTarget,%menuItem);
```

void GuiMenuBar::**onMenuItemSelect**(string *menuId*, string *menuText*, string *menuItemId*, string *menuItemText*)

Called whenever an item in a menu is selected.

Parameters

- **menuId** – Index id of the menu which contains the selected menu item
- **menuText** – Text of the menu which contains the selected menu item
- **menuItemId** – Index id of the selected menu item
- **menuItemText** – Text of the selected menu item

Example:

```
// A menu item has been selected, causing the callback to occur.GuiMenuBar::onMenuItemSelect(%th
{
   // Code to run when the callback occurs
}
```

void GuiMenuBar::**onMenuSelect**(string *menuId*, string *menuText*)
Called whenever a menu is selected.

Parameters

- **menuId** – Index id of the clicked menu
- **menuText** – Text of the clicked menu

Example:

```
// A menu has been selected, causing the callback to occur.GuiMenuBar::onMenuSelect(%this,%menuI
{
   // Code to run when the callback occurs
}
```

void GuiMenuBar::**onMouseInMenu**(bool *isInMenu*)
Called whenever the mouse enters, or persists is in the menu.

Parameters **isInMenu** – True if the mouse has entered the menu, otherwise is false.

Example:

```
// Mouse enters or persists within the menu, causing the callback to occur.GuiMenuBar::onMouseIn
{
   // Code to run when the callback occurs
}
```

void GuiMenuBar::**onSubmenuSelect**(string *submenuId*, string *submenuText*)
Called whenever a submenu is selected.

Parameters

- **submenuId** – Id of the selected submenu
- **submenuText** – Text of the selected submenu

Example:

```
GuiMenuBar::onSubmenuSelect(%this,%submenuId,%submenuText)
{
   // Code to run when the callback occurs
}
```

void GuiMenuBar::**removeMenu**(string *menuTarget*)
Removes the specified menu from the menu bar.

Parameters **menuTarget** – Menu to remove from the menu bar

Example:

```
// Define the menuTarget
%menuTarget = "New Menu";  or %menuTarget = "3";

// Inform the GuiMenuBar to remove the defined menu from the menu bar
%thisGuiMenuBar.removeMenu(%menuTarget);
```

void GuiMenuBar::**removeMenuItem**(string *menuTarget*, string *menuItemTarget*)

Removes the specified menu item from the menu.

Parameters

- **menuTarget** – Menu to affect the menu item in
- **menuItem** – Menu item to affect

Example:

```
// Define the menuTarget
%menuTarget = "New Menu";  or %menuTarget = "3";

// Define the menuItem
%menuItem = "New Menu Item";  or  %menuItem = "5";

// Request the GuiMenuBar control to remove the define menu item
%thisGuiMenuBar.removeMenuItem(%menuTarget,%menuItem);
```

void GuiMenuBar::**setCheckmarkBitmapIndex**(int *bitmapindex*)

Sets the menu bitmap index for the check mark image.

Parameters **bitmapIndex** – Bitmap index for the check mark image.

Example:

```
// Define the bitmap index
%bitmapIndex = "2";

// Inform the GuiMenuBar control of the proper bitmap index for the check mark image
%thisGuiMenuBar.setCheckmarkBitmapIndex(%bitmapIndex);
```

void GuiMenuBar::**setMenuBitmapIndex**(string *menuTarget*, int *bitmapindex*, bool *bitmaponly*, bool *drawborder*)

Sets the bitmap index for the menu and toggles rendering only the bitmap.

Parameters

- **menuTarget** – Menu to affect
- **bitmapindex** – Bitmap index to set for the menu
- **bitmaponly** – If true, only the bitmap will be rendered
- **drawborder** – If true, a border will be drawn around the menu.

Example:

```
// Define the menuTarget to affect
%menuTarget = "New Menu";  or %menuTarget = "3";

// Set the bitmap index
%bitmapIndex = "5";

// Set if we are only to render the bitmap or not
```

```
    %bitmaponly = "true";

    // Set if we are rendering a border or not
    %drawborder = "true";

    // Inform the GuiMenuBar of the bitmap and rendering changes
    %thisGuiMenuBar.setMenuBitmapIndex(%menuTarget,%bitmapIndex,%bitmapOnly,%drawBorder);
```

void GuiMenuBar::**setMenuItemBitmap**(string *menuTarget*, string *menuItemTarget*, int *bitmapIndex*)

Sets the specified menu item bitmap index in the bitmap array. Setting the item's index to -1 will remove any bitmap.

Parameters

- **menuTarget** – Menu to affect the menuItem in
- **menuItem** – Menu item to affect
- **bitmapIndex** – Bitmap index to set the menu item to

Example:

```
    // Define the menuTarget
    %menuTarget = "New Menu";  or  %menuTarget = "3";

    // Define the menuItem"
    %menuItem = "New Menu Item";  or %menuItem = "2";

    // Define the bitmapIndex
    %bitmapIndex = "6";

    // Inform the GuiMenuBar control to set the menu item to the defined bitmap
    %thisGuiMenuBar.setMenuItemBitmap(%menuTarget,%menuItem,%bitmapIndex);
```

void GuiMenuBar::**setMenuItemChecked**(string *menuTarget*, string *menuItemTarget*, bool *checked*)

Sets the menu item bitmap to a check mark, which by default is the first element in the bitmap array (although this may be changed with setCheckmarkBitmapIndex()). Any other menu items in the menu with the same check group become unchecked if they are checked.

Parameters

- **menuTarget** – Menu to work in
- **menuItem** – Menu item to affect
- **checked** – Whether we are setting it to checked or not

Returns If not void, return value and description

void GuiMenuBar::**setMenuItemEnable**(string *menuTarget*, string *menuItemTarget*, bool *enabled*)

sets the menu item to enabled or disabled based on the enable parameter. The specified menu and menu item can either be text or ids. Detailed description

Parameters

- **menuTarget** – Menu to work in
- **menuItemTarget** – The menu item inside of the menu to enable or disable
- **enabled** – Boolean enable / disable value.

Example:

```
// Define the menu
%menu = "New Menu";  or  %menu = "4";

// Define the menu item
%menuItem = "New Menu Item";  or %menuItem = "2";

// Define the enabled state
%enabled = "true";

// Inform the GuiMenuBar control to set the enabled state of the requested menu item
%thisGuiMenuBar.setMenuItemEnable(%menu,%menuItme,%enabled);
```

void GuiMenuBar::**setMenuItemSubmenuState**(string *menuTarget*, string *menuItem*, bool *isSubmenu*)

Sets the given menu item to be a submenu.

Parameters

- **menuTarget** – Menu to affect a submenu in
- **menuItem** – Menu item to affect
- **isSubmenu** – Whether or not the menuItem will become a subMenu or not

Example:

```
// Define the menuTarget
%menuTarget = "New Menu";  or %menuTarget = "3";

// Define the menuItem
%menuItem = "New Menu Item";  or  %menuItem = "5";

// Define whether or not the Menu Item is a sub menu or not
%isSubmenu = "true";

// Inform the GuiMenuBar control to set the defined menu item to be a submenu or not.
%thisGuiMenuBar.setMenuItemSubmenuState(%menuTarget,%menuItem,%isSubmenu);
```

void GuiMenuBar::**setMenuItemText**(string *menuTarget*, string *menuItemTarget*, string *newMenuItemText*)

Sets the text of the specified menu item to the new string.

Parameters

- **menuTarget** – Menu to affect
- **menuItem** – Menu item in the menu to change the text at
- **newMenuItemText** – New menu text

Example:

```
// Define the menuTarget
%menuTarget = "New Menu";  or  %menuTarget = "4";

// Define the menuItem
%menuItem = "New Menu Item";  or  %menuItem = "2";

// Define the new text for the menu item
%newMenuItemText = "Very New Menu Item";

// Inform the GuiMenuBar control to change the defined menu item with the new text
%thisGuiMenuBar.setMenuItemText(%menuTarget,%menuItem,%newMenuItemText);
```

void GuiMenuBar::**setMenuItemVisible**(string *menuTarget*, string *menuItemTarget*, bool *isVisible*)

Brief Description. Detailed description

Parameters

- **menuTarget** – Menu to affect the menu item in
- **menuItem** – Menu item to affect
- **isVisible** – Visible state to set the menu item to.

Example:

```
// Define the menuTarget
%menuTarget = "New Menu";  or  %menuTarget = "3";

// Define the menuItem
%menuItem = "New Menu Item";  or  %menuItem = "2";

// Define the visibility state
%isVisible = "true";

// Inform the GuiMenuBarControl of the visibility state of the defined menu item
%thisGuiMenuBar.setMenuItemVisible(%menuTarget,%menuItem,%isVisible);
```

void GuiMenuBar::**setMenuMargins**(int *horizontalMargin*, int *verticalMargin*, int *bitmapToTextSpacing*)

Sets the menu rendering margins: horizontal, vertical, bitmap spacing. Detailed description

Parameters

- **horizontalMargin** – Number of pixels on the left and right side of a menu's text.
- **verticalMargin** – Number of pixels on the top and bottom of a menu's text.
- **bitmapToTextSpacing** – Number of pixels between a menu's bitmap and text.

Example:

```
// Define the horizontalMargin
%horizontalMargin = "5";

// Define the verticalMargin
%verticalMargin = "5";

// Define the bitmapToTextSpacing
%bitmapToTextSpacing = "12";

// Inform the GuiMenuBar control to set its margins based on the defined values.
%thisGuiMenuBar.setMenuMargins(%horizontalMargin,%verticalMargin,%bitmapToTextSpacing);
```

void GuiMenuBar::**setMenuText**(string *menuTarget*, string *newMenuText*)

Sets the text of the specified menu to the new string.

Parameters

- **menuTarget** – Menu to affect
- **newMenuText** – New menu text

Example:

```
// Define the menu to affect%menu = "New Menu";  or %menu = "3";// Define the text to change the
%newMenuText = "Still a New Menu";
```

```
// Inform the GuiMenuBar control to change the defined menu to the defined text
%thisGuiMenuBar.setMenuText(%menu,%newMenuText);
```

void GuiMenuBar::**setMenuVisible**(string *menuTarget*, bool *visible*)

Sets the whether or not to display the specified menu.

Parameters

- **menuTarget** – Menu item to affect
- **visible** – Whether the menu item will be visible or not

Example:

```
// Define the menu to work with
%menuTarget = "New Menu";  or  %menuTarget = "4";

// Define if the menu should be visible or not
%visible = "true";

// Inform the GuiMenuBar control of the new visibility state for the defined menu
%thisGuiMenuBar.setMenuVisible(%menuTarget,%visible);
```

void GuiMenuBar::**setSubmenuItemChecked**(string *menuTarget*, string *menuItemTarget*, string *submenuItemText*, bool *checked*)

Sets the menu item bitmap to a check mark, which by default is the first element in the bitmap array (although this may be changed with setCheckmarkBitmapIndex()). Any other menu items in the menu with the same check group become unchecked if they are checked.

Parameters

- **menuTarget** – Menu to affect a submenu in
- **menuItem** – Menu item to affect
- **submenuItemText** – Text to show for submenu
- **checked** – Whether or not this submenu item will be checked.

Returns If not void, return value and description

Example:

```
// Define the menuTarget
%menuTarget = "New Menu";  or %menuTarget = "3";

// Define the menuItem
%menuItem = "New Menu Item";  or  %menuItem = "5";

// Define the text for the new submenu
%submenuItemText = "Submenu Item";

// Define if this submenu item should be checked or not
%checked = "true";

// Inform the GuiMenuBar control to set the checked state of the defined submenu item
%thisGuiMenuBar.setSubmenuItemChecked(%menuTarget,%menuItem,%submenuItemText,%checked);
```

Fields

int GuiMenuBar::**padding**

Extra padding to add to the bounds of the control.

GuiMLTextCtrl A text control that uses the Gui Markup Language ('ML') tags to dynamically change the text.

Inherit: GuiControl

Description Example of dynamic changes include colors, styles, and/or hyperlinks. These changes can occur without having to use separate text controls with separate text profiles.

Example:

```
newGuiMLTextCtrl(CenterPrintText)
{
    lineSpacing = "2";
    allowColorChars = "0";
    maxChars = "-1";
    deniedSound = "DeniedSoundProfile";
    text = "The Text for This Control.";
    useURLMouseCursor = "true";
    //Properties not specific to this control have been omitted from this example.
};
```

Methods

void GuiMLTextCtrl::**addText** (string *text*, bool *reformat*)

Appends the text in the control with additional text. Also .

Parameters

- **text** – New text to append to the existing text.
- **reformat** – If true, the control will also be visually reset (defaults to true).

Example:

```
    // Define new text to add
    %text = "New Text to Add";

    // Set reformat boolean
    %reformat = "true";

    // Inform the control to add the new text
    %thisGuiMLTextCtrl.addText(%text,%reformat);
```

void GuiMLTextCtrl::**forceReflow**()

Forces the text control to reflow the text after new text is added, possibly resizing the control.

Example:

```
    // Define new text to add
    %newText = "BACON!";

    // Add the new text to the control
    %thisGuiMLTextCtrl.addText(%newText);

    // Inform the GuiMLTextCtrl object to force a reflow to ensure the added text fits properly.
    %thisGuiMLTextCtrl.forceReflow();
```

string GuiMLTextCtrl::**getText**()

Returns the text from the control, including TorqueML characters.

Returns Text string displayed in the control, including any TorqueML characters.

Example:

```
// Get the text displayed in the control
%controlText = %thisGuiMLTextCtrl.getText();
```

void GuiMLTextCtrl::**onResize**(string *width*, string *maxY*)
Called whenever the control size changes.

Parameters

- **width** – The new width value for the control
- **maxY** – The current maximum allowed Y value for the control

Example:

```
// Control size changed, causing the callback to occur.GuiMLTextCtrl::onResize(%this,%width,%max
   {
      // Code to call when the control size changes
   }
```

void GuiMLTextCtrl::**onURL**(string *url*)
Called whenever a URL was clicked on within the control.

Parameters **url** – The URL address that was clicked on.

Example:

```
// A URL address was clicked on in the control, causing the callback to occur.
GuiMLTextCtrl::onUrl(%this,%url)
   {
      // Code to run whenever a URL was clicked on
   }
```

void GuiMLTextCtrl::**scrollToBottom**()
Scroll to the bottom of the text.

Example:

```
// Inform GuiMLTextCtrl object to scroll to its bottom
%thisGuiMLTextCtrl.scrollToBottom();
```

void GuiMLTextCtrl::**scrollToTag**(int *tagID*)
Scroll down to a specified tag. Detailed description

Parameters **tagID** – TagID to scroll the control to

Example:

```
// Define the TagID we want to scroll the control to
%tagId = "4";

// Inform the GuiMLTextCtrl to scroll to the defined TagID
%thisGuiMLTextCtrl.scrollToTag(%tagId);
```

void GuiMLTextCtrl::**scrollToTop**(int *param1*, int *param2*)
Scroll to the top of the text.

Example:

```
// Inform GuiMLTextCtrl object to scroll to its top
%thisGuiMLTextCtrl.scrollToTop();
```

void GuiMLTextCtrl::**setAlpha**(float *alphaVal*)
Sets the alpha value of the control.

Parameters **alphaVal** – n - 1.0 floating value for the alpha

Example:

```
// Define the alphe value
%alphaVal = "0.5";

// Inform the control to update its alpha value.
%thisGuiMLTextCtrl.setAlpha(%alphaVal);
```

bool GuiMLTextCtrl::**setCursorPosition**(int *newPos*)

Change the text cursor's position to a new defined offset within the text in the control.

Parameters **newPos** – Offset to place cursor.

Returns Returns true if the cursor position moved, or false if the position was not changed.

Example:

```
// Define cursor offset position
%position = "23";

// Inform the GuiMLTextCtrl object to move the cursor to the new position.
%thisGuiMLTextCtrl.setCursorPosition(%position);
```

void GuiMLTextCtrl::**setText**(string *text*)

Set the text contained in the control.

Parameters **text** – The text to display in the control.

Example:

```
// Define the text to display
%text = "Nifty Control Text";

// Set the text displayed within the control
%thisGuiMLTextCtrl.setText(%text);
```

Fields

bool GuiMLTextCtrl::**allowColorChars**

If true, the control will allow characters to have unique colors.

SFXTrack GuiMLTextCtrl::**deniedSound**

If the text will not fit in the control, the deniedSound is played.

int GuiMLTextCtrl::**lineSpacing**

The number of blank pixels to place between each line.

int GuiMLTextCtrl::**maxChars**

Maximum number of characters that the control will display.

caseString GuiMLTextCtrl::**text**

Text to display in this control.

bool GuiMLTextCtrl::**useURLMouseCursor**

If true, the mouse cursor will turn into a hand cursor while over a link in the text. This is dependant on the markup language used by the GuiMLTextCtrl

GuiMouseEventCtrl Used to overlaps a 'hot region' where you want to catch inputs with and have specific events occur based on individual callbacks.

Inherit: GuiControl

Description Mouse event callbacks supported by this control are: onMouseUp, onMouseDown, onMouseMove, onMouseDragged, onMouseEnter, onMouseLeave, onRightMouseDown, onRightMouseUp and onRightMouseDragged.

Example:

```
newGuiMouseEventCtrl()
{
   lockMouse = "0";
   //Properties not specific to this control have been omitted from this example.
};
```

Methods

void GuiMouseEventCtrl::**onMouseDown**(U8 *modifier*, Point2I *mousePoint*, U8 *mouseClickCount*)

Callback that occurs whenever the mouse is pressed down while in this control. $EventModifier::RSHIFT $EventModifier::SHIFT $EventModifier::LCTRL $EventModifier::RCTRL $EventModifier::CTRL $EventModifier::CTRL $EventModifier::RALT $EventModifier::ALT

Parameters

- **modifier** – Key that was pressed during this callback. Values are:
- **mousePoint** – X/Y location of the mouse point
- **mouseClickCount** – How many mouse clicks have occured for this event

Example:

```
// Mouse was pressed down in this control, causing the callback
GuiMouseEventCtrl::onMouseDown(%this,%modifier,%mousePoint,%mouseClickCount)
{
   // Code to call when a mouse event occurs.
}
```

void GuiMouseEventCtrl::**onMouseDragged**(U8 *modifier*, Point2I *mousePoint*, U8 *mouseClickCount*)

Callback that occurs whenever the mouse is dragged while in this control. $EventModifier::RSHIFT $EventModifier::SHIFT $EventModifier::LCTRL $EventModifier::RCTRL $EventModifier::CTRL $EventModifier::CTRL $EventModifier::RALT $EventModifier::ALT

Parameters

- **modifier** – Key that was pressed during this callback. Values are:
- **mousePoint** – X/Y location of the mouse point
- **mouseClickCount** – How many mouse clicks have occured for this event

Example:

```
// Mouse was dragged in this control, causing the callback
GuiMouseEventCtrl::onMouseDragged(%this,%modifier,%mousePoint,%mouseClickCount)
{
   // Code to call when a mouse event occurs.
}
```

void GuiMouseEventCtrl::**onMouseEnter**(U8 *modifier*, Point2I *mousePoint*, U8 *mouseClickCount*)

Callback that occurs whenever the mouse enters this control. $EventModifier::RSHIFT $EventModifier::SHIFT $EventModifier::LCTRL $EventModifier::RCTRL $EventModifier::CTRL $EventModifier::CTRL $EventModifier::RALT $EventModifier::ALT

Parameters

- **modifier** – Key that was pressed during this callback. Values are:

- **mousePoint** – X/Y location of the mouse point
- **mouseClickCount** – How many mouse clicks have occured for this event

Example:

```
// Mouse entered this control, causing the callback
GuiMouseEventCtrl::onMouseEnter(%this,%modifier,%mousePoint,%mouseClickCount)
{
   // Code to call when a mouse event occurs.
}
```

void GuiMouseEventCtrl::**onMouseLeave**(U8 *modifier*, Point2I *mousePoint*, U8 *mouseClickCount*)

Callback that occurs whenever the mouse leaves this control. $EventModifier::RSHIFT $EventModifier::SHIFT $EventModifier::LCTRL $EventModifier::RCTRL $EventModifier::CTRL $EventModifier::CTRL $EventModifier::RALT $EventModifier::ALT

Parameters

- **modifier** – Key that was pressed during this callback. Values are:
- **mousePoint** – X/Y location of the mouse point
- **mouseClickCount** – How many mouse clicks have occured for this event

Example:

```
// Mouse left this control, causing the callback
GuiMouseEventCtrl::onMouseLeave(%this,%modifier,%mousePoint,%mouseClickCount)
{
   // Code to call when a mouse event occurs.
}
```

void GuiMouseEventCtrl::**onMouseMove**(U8 *modifier*, Point2I *mousePoint*, U8 *mouseClickCount*)

Callback that occurs whenever the mouse is moved (without dragging) while in this control. $EventModifier::RSHIFT $EventModifier::SHIFT $EventModifier::LCTRL $EventModifier::RCTRL $EventModifier::CTRL $EventModifier::CTRL $EventModifier::RALT $EventModifier::ALT

Parameters

- **modifier** – Key that was pressed during this callback. Values are:
- **mousePoint** – X/Y location of the mouse point
- **mouseClickCount** – How many mouse clicks have occured for this event

Example:

```
// Mouse was moved in this control, causing the callback
GuiMouseEventCtrl::onMouseMove(%this,%modifier,%mousePoint,%mouseClickCount)
{
   // Code to call when a mouse event occurs.
}
```

void GuiMouseEventCtrl::**onMouseUp**(U8 *modifier*, Point2I *mousePoint*, U8 *mouseClickCount*)

Callback that occurs whenever the mouse is released while in this control. $EventModifier::RSHIFT $EventModifier::SHIFT $EventModifier::LCTRL $EventModifier::RCTRL $EventModifier::CTRL $EventModifier::CTRL $EventModifier::RALT $EventModifier::ALT

Parameters

- **modifier** – Key that was pressed during this callback. Values are:
- **mousePoint** – X/Y location of the mouse point

- **mouseClickCount** – How many mouse clicks have occured for this event

Example:

```
// Mouse was released in this control, causing the callback
GuiMouseEventCtrl::onMouseUp(%this,%modifier,%mousePoint,%mouseClickCount)
{
   // Code to call when a mouse event occurs.
}
```

void GuiMouseEventCtrl::**onRightMouseDown**(U8 *modifier*, Point2I *mousePoint*, U8 *mouseClickCount*)

Callback that occurs whenever the right mouse button is pressed while in this control. $EventModifier::RSHIFT $EventModifier::SHIFT $EventModifier::LCTRL $EventModifier::RCTRL $EventModifier::CTRL $EventModifier::CTRL $EventModifier::RALT $EventModifier::ALT

Parameters

- **modifier** – Key that was pressed during this callback. Values are:
- **mousePoint** – X/Y location of the mouse point
- **mouseClickCount** – How many mouse clicks have occured for this event

Example:

```
// Right mouse button was pressed in this control, causing the callback
GuiMouseEventCtrl::onRightMouseDown(%this,%modifier,%mousePoint,%mouseClickCount)
{
   // Code to call when a mouse event occurs.
}
```

void GuiMouseEventCtrl::**onRightMouseDragged**(U8 *modifier*, Point2I *mousePoint*, U8 *mouseClickCount*)

Callback that occurs whenever the mouse is dragged in this control while the right mouse button is pressed. $EventModifier::RSHIFT $EventModifier::SHIFT $EventModifier::LCTRL $EventModifier::RCTRL $EventModifier::CTRL $EventModifier::CTRL $EventModifier::RALT $EventModifier::ALT

Parameters

- **modifier** – Key that was pressed during this callback. Values are:
- **mousePoint** – X/Y location of the mouse point
- **mouseClickCount** – How many mouse clicks have occured for this event

Example:

```
// Right mouse button was dragged in this control, causing the callback
GuiMouseEventCtrl::onRightMouseDragged(%this,%modifier,%mousePoint,%mouseClickCount)
{
   // Code to call when a mouse event occurs.
}
```

void GuiMouseEventCtrl::**onRightMouseUp**(U8 *modifier*, Point2I *mousePoint*, U8 *mouseClickCount*)

Callback that occurs whenever the right mouse button is released while in this control. $EventModifier::RSHIFT $EventModifier::SHIFT $EventModifier::LCTRL $EventModifier::RCTRL $EventModifier::CTRL $EventModifier::CTRL $EventModifier::RALT $EventModifier::ALT

Parameters

- **modifier** – Key that was pressed during this callback. Values are:
- **mousePoint** – X/Y location of the mouse point

- **mouseClickCount** – How many mouse clicks have occured for this event

Example:

```
// Right mouse button was released in this control, causing the callback
GuiMouseEventCtrl::onRightMouseUp(%this,%modifier,%mousePoint,%mouseClickCount)
{
   // Code to call when a mouse event occurs.
}
```

Fields

bool GuiMouseEventCtrl::**lockMouse**

Whether the control should lock the mouse between up and down button events.

GuiTextCtrl GUI control object this displays a single line of text, without TorqueML.

Inherit: GuiContainer

Description GUI control object this displays a single line of text, without TorqueML.

Example:

```
newGuiTextCtrl()
   {
      text = "Hello World";
      textID = ""STR_HELLO"";
      maxlength = "1024";
       //Properties not specific to this control have been omitted from this example.
   };
```

Methods

void GuiTextCtrl::**setText** (string *text*)

Sets the text in the control. Reimplemented in GuiTextEditCtrl , and GuiPopUpMenuCtrlEx .

Parameters **text** – Text to display in the control.

Example:

```
// Set the text to show in the control
%text = "Gideon - Destroyer of World";

// Inform the GuiTextCtrl control to change its text to the defined value
%thisGuiTextCtrl.setText(%text);
```

void GuiTextCtrl::**setTextID** (string *textID*)

Maps the text ctrl to a variable used in localization, rather than raw text.

Parameters **textID** – Name of variable text should be mapped to

Example:

```
// Inform the GuiTextCtrl control of the textID to use
%thisGuiTextCtrl.setTextID("STR_QUIT");
```

Fields

int GuiTextCtrl::**maxLength**

Defines the maximum length of the text. The default is 1024.

caseString GuiTextCtrl::**text**
The text to show on the control.

string GuiTextCtrl::**textID**
Maps the text of this control to a variable used in localization, rather than raw text.

GuiTextEditSliderBitmapCtrl GUI Control which displays a numerical value which can be increased or decreased using a pair of bitmap up/down buttons.

Inherit: GuiTextEditCtrl

Description This control uses the bitmap specified in it's profile (GuiControlProfile::bitmapName). It takes this image and breaks up aspects of it to render the up and down arrows. It is also important to set GuiControlProfile::hasBitmapArray to true on the profile as well.

The bitmap referenced should be broken up into a 1 x 4 grid (using the top left color pixel as a border color between each of the images) in which it will map to the following places:

Example:

```
singleton GuiControlProfile (SliderBitmapGUIProfile)
{
   bitmap = "core/art/gui/images/sliderArray";
   hasBitmapArray = true;
   opaque = false;
};

newGuiTextEditSliderBitmapCtrl()
{
   profile = "SliderBitmapGUIProfile";
   format = "%3.2f";
   range = "-1e+03 1e+03";
   increment = "0.1";
   focusOnMouseWheel = "0";
   bitmap = "";
   //Properties not specific to this control have been omitted from this example.
};
```

Fields

filename GuiTextEditSliderBitmapCtrl::**bitmap**
Unused.

bool GuiTextEditSliderBitmapCtrl::**focusOnMouseWheel**
If true, the control will accept giving focus to the user when the mouse wheel is used.

string GuiTextEditSliderBitmapCtrl::**format**
Character format type to place in the control.

float GuiTextEditSliderBitmapCtrl::**increment**
How far to increment the slider on each step.

Point2F GuiTextEditSliderBitmapCtrl::**range**
Maximum vertical and horizontal range to allow in the control.

GuiTextEditSliderCtrl GUI Control which displays a numerical value which can be increased or decreased using a pair of arrows.

Inherit: GuiTextEditCtrl

Description GUI Control which displays a numerical value which can be increased or decreased using a pair of arrows.

Example:

```
newGuiTextEditSliderCtrl()
{
   format = "%3.2f";
   range = "-1e+03 1e+03";
   increment = "0.1";
   focusOnMouseWheel = "0";
   //Properties not specific to this control have been omitted from this example.
};
```

Fields

bool `GuiTextEditSliderCtrl::`**`focusOnMouseWheel`**
: If true, the control will accept giving focus to the user when the mouse wheel is used.

string `GuiTextEditSliderCtrl::`**`format`**
: Character format type to place in the control.

float `GuiTextEditSliderCtrl::`**`increment`**
: How far to increment the slider on each step.

Point2F `GuiTextEditSliderCtrl::`**`range`**
: Maximum vertical and horizontal range to allow in the control.

Enumeration

`enum GuiAlignmentType`

Parameters

- **`Left`** –
- **`Center`** –
- **`Right`** –
- **`Top`** –
- **`Bottom`** –

`enum GuiFontCharset`

Parameters

- **`ANSI`** –
- **`SYMBOL`** –
- **`SHIFTJIS`** –
- **`HANGEUL`** –
- **`HANGUL`** –
- **`GB2312`** –
- **`CHINESEBIG5`** –
- **`OEM`** –
- **`JOHAB`** –

- **HEBREW** –
- **ARABIC** –
- **GREEK** –
- **TURKISH** –
- **VIETNAMESE** –
- **THAI** –
- **EASTEUROPE** –
- **RUSSIAN** –
- **MAC** –
- **BALTIC** –

enum GuiHorizontalSizing
Horizontal sizing behavior of a GuiControl .

Parameters

- **right** –
- **width** –
- **left** –
- **center** –
- **relative** –
- **windowRelative** –

enum GuiVerticalSizing
Vertical sizing behavior of a GuiControl .

Parameters

- **bottom** –
- **height** –
- **top** –
- **center** –
- **relative** –
- **windowRelative** –

Functions

bool **excludeOtherInstance** (string *appIdentifer*)
Used to exclude/prevent all other instances using the same identifier specified.

Parameters **appIdentifier** – Name of the app set up for exclusive use.

Returns False if another app is running that specified the same appIdentifier

string **StripMLControlChars** (string *inString*)
Strip TorqueML control characters from the specified string, returning a 'clean' version.

Parameters **inString** – String to strip TorqueML control characters from.

Returns Version of the inputted string with all TorqueML characters removed.

Example:

```
// Define the string to strip TorqueML control characters from
%string = "<font:Arial:24>How Now <color:c43c12>Brown <color:000000>Cow";

// Request the stripped version of the string
%strippedString = StripMLControlChars(%string);
```

Variables

GuiControl $ThisControl
The control for which a command is currently being evaluated. Only set during 'command' and altCommand callbacks to the control for which the command or altCommand is invoked.

Button Controls

A collection of various buttons (push buttons, radio buttons, check boxes, etc).

Classes

GuiBitmapButtonCtrl A button that renders its various states (mouse over, pushed, etc.) from separate bitmaps.

Inherit: GuiButtonCtrl

Description A bitmapped button is a push button that uses one or more texture images for rendering its individual states.

To find the individual textures associated with the button, a naming scheme is used. For each state a suffix is appended to the texture file name given in the GuiBitmapButtonCtrl::bitmap field:

If a bitmap for a particular state cannot be found, the default bitmap will be used. To disable all state-based bitmap functionality, set useStates to false which will make the control solely render from the bitmap specified in the bitmap field.

Per-Modifier Button Actions If GuiBitmapButtonCtrl::useModifiers is set to true, per-modifier button actions and textures are enabled. This functionality allows to associate different images and different actions with a button depending on which modifiers are pressed on the keyboard by the user.

When enabled, this functionality alters the texture lookup above by prepending the following strings to the suffixes listed above:

When this functionality is enabled, a new set of callbacks is used:

GuiControl::command or GuiControl::onAction() still work as before when per-modifier functionality is enabled.

Note that modifiers cannot be mixed. If two or more modifiers are pressed, a single one will take precedence over the remaining modifiers. The order of precedence corresponds to the order listed above.

Example:

```
// Create an OK button that will trigger an onOk() call on its parent when clicked:
%okButton = newGuiBitmapButtonCtrl()
{
   bitmap = "art/gui/okButton";
   autoFitExtents = true;
   command = "$ThisControl.getParent().onOk();";
};
```

Methods

void GuiBitmapButtonCtrl::**onAltClick**()
Called when per-modifier functionality is enabled and the user clicks on the button with the ALT key pressed. Per-Modifier Button Actions

void GuiBitmapButtonCtrl::**onCtrlClick**()
Called when per-modifier functionality is enabled and the user clicks on the button with the CTRL key pressed. Per-Modifier Button Actions

void GuiBitmapButtonCtrl::**onDefaultClick**()
Called when per-modifier functionality is enabled and the user clicks on the button without any modifier pressed. Per-Modifier Button Actions

void GuiBitmapButtonCtrl::**onShiftClick**()
Called when per-modifier functionality is enabled and the user clicks on the button with the SHIFT key pressed. Per-Modifier Button Actions

void GuiBitmapButtonCtrl::**setBitmap**(string *path*)
Set the bitmap to show on the button.

Parameters **path** – Path to the texture file in any of the supported formats.

Fields

bool GuiBitmapButtonCtrl::**autoFitExtents**
If true, the control's extents will be set to match the bitmap's extents when setting the bitmap. The bitmap extents will always be taken from the default/normal bitmap (in case the extents of the various bitmaps do not match up.)

filename GuiBitmapButtonCtrl::**bitmap**
Texture file to display on this button. If useStates is false, this will be the file that renders on the control. Otherwise, this will specify the default texture name to which the various state and modifier suffixes are appended to find the per-state and per-modifier (if enabled) textures.

GuiBitmapMode GuiBitmapButtonCtrl::**bitmapMode**
Behavior for fitting the bitmap to the control extents. If set to 'Stretched', the bitmap will be stretched both verticall and horizontally to fit inside the control's extents. If set to 'Centered', the bitmap will stay at its original resolution centered in the control's rectangle (getting clipped if the control is smaller than the texture).

bool GuiBitmapButtonCtrl::**useModifiers**
If true, per-modifier button functionality is enabled. Per-Modifier Button Actions

bool GuiBitmapButtonCtrl::**useStates**
If true, per-mouse state button functionality is enabled. Defaults to true. If you do not use per-state images on this button set this to false to speed up the loading process by inhibiting searches for the individual images.

GuiBitmapButtonTextCtrl An extension of GuiBitmapButtonCtrl that additionally renders a text label on the bitmapped button.

Inherit: GuiBitmapButtonCtrl

Description The text for the label is taken from the GuiButtonBaseCtrl::text property.

For rendering, the label is placed, relative to the control's upper left corner, at the text offset specified in the control's profile (GuiControlProfile::textOffset) and justified according to the profile's setting (GuiControlProfile::justify).

GuiBorderButtonCtrl A push button that renders only a border.

Inherit: GuiButtonBaseCtrl

Description A border button consists of a border rendered along its extents according to the border thickness defined in its profile (GuiControlProfile::border). For the border color, a color is selected from the profile according to current button state:

GuiButtonBaseCtrl The base class for the various button controls.

Inherit: GuiControl

Description This is the base class for the various types of button controls. If no more specific functionality is required than offered by this class, then it can be instantiated and used directly. Otherwise, its subclasses should be used:

Methods

string `GuiButtonBaseCtrl::getText()`
Get the text display on the button's label (if any).

Returns The button's label.

void `GuiButtonBaseCtrl::onClick()`
Called when the primary action of the button is triggered (e.g. by a left mouse click).

void `GuiButtonBaseCtrl::onDoubleClick()`
Called when the left mouse button is double-clicked on the button.

void `GuiButtonBaseCtrl::onMouseDown()`
If useMouseEvents is true, this is called when the left mouse button is pressed on an (active) button.

void `GuiButtonBaseCtrl::onMouseDragged()`
If useMouseEvents is true, this is called when a left mouse button drag is detected, i.e. when the user pressed the left mouse button on the control and then moves the mouse over a certain distance threshold with the mouse button still pressed.

void `GuiButtonBaseCtrl::onMouseEnter()`
If useMouseEvents is true, this is called when the mouse cursor moves over the button (only if the button is the front-most visible control, though).

void `GuiButtonBaseCtrl::onMouseLeave()`
If useMouseEvents is true, this is called when the mouse cursor moves off the button (only if the button had previously received an onMouseEvent() event).

void `GuiButtonBaseCtrl::onMouseUp()`
If useMouseEvents is true, this is called when the left mouse button is release over an (active) button.

void `GuiButtonBaseCtrl::onRightClick()`
Called when the right mouse button is clicked on the button.

void `GuiButtonBaseCtrl::performClick()`
Simulate a click on the button. This method will trigger the button's action just as if the button had been pressed by the user.

void GuiButtonBaseCtrl::**resetState**()
Reset the mousing state of the button. This method should not generally be called.

void GuiButtonBaseCtrl::**setStateOn**(bool *isOn*)
For toggle or radio buttons, set whether the button is currently activated or not. For radio buttons, toggling a button on will toggle all other radio buttons in its group to off. Reimplemented in GuiCheckBoxCtrl .

Parameters **isOn** – If true, the button will be toggled on (if not already); if false, it will be toggled off.

void GuiButtonBaseCtrl::**setText**(string *text*)
Set the text displayed on the button's label.

Parameters **text** – The text to display as the button's text label.

void GuiButtonBaseCtrl::**setTextID**(string *id*)
Set the text displayed on the button's label using a string from the string table assigned to the control. Internationalization

Parameters **id** – Name of the variable that contains the integer string ID. Used to look up string in table.

Fields

GuiButtonType GuiButtonBaseCtrl::**buttonType**
Button behavior type.

int GuiButtonBaseCtrl::**groupNum**
Radio button toggle group number. All radio buttons that are assigned the same groupNum and that are parented to the same control will synchronize their toggle state, i.e. if one radio button is toggled on all other radio buttons in its group will be toggled off. The default group is -1.

caseString GuiButtonBaseCtrl::**text**
Text label to display on button (if button class supports text labels).

string GuiButtonBaseCtrl::**textID**
ID of string in string table to use for text label on button.

bool GuiButtonBaseCtrl::**useMouseEvents**
If true, mouse events will be passed on to script. Default is false.

GuiButtonCtrl The most widely used button class.

Inherit: GuiButtonBaseCtrl

Description GuiButtonCtrl renders seperately of, but utilizes all of the functionality of GuiBaseButtonCtrl. This grants GuiButtonCtrl the versatility to be either of the 3 button types.

Example:

```
// Create a PushButton GuiButtonCtrl that calls randomFunction when clicked
%button = newGuiButtonCtrl()
{
   profile    = "GuiButtonProfile";
   buttonType = "PushButton";
   command    = "randomFunction();";
};
```

GuiCheckBoxCtrl A named checkbox that can be toggled on and off.

Inherit: GuiButtonBaseCtrl

Description A GuiCheckBoxCtrl displays a text label next to a checkbox that can be toggled on and off by the user. Checkboxes are usually used to present boolean choices like, for example, a switch to toggle fullscreen video on and off.

Example:

```
// Create a checkbox that allows to toggle fullscreen on and off.
newGuiCheckBoxCtrl( FullscreenToggle )
{
   text = "Fullscreen";
};

// Set the initial state to match the current fullscreen setting.
FullscreenToggle.setStateOn( Canvas.isFullscreen() );

// Define function to be called when checkbox state is toggled.
function FullscreenToggle::onClick( %this )
{
   Canvas.toggleFullscreen();
}
```

Methods

bool GuiCheckBoxCtrl::**isStateOn**()
Test whether the checkbox is currently checked.

Returns True if the checkbox is currently ticked, false otherwise.

void GuiCheckBoxCtrl::**setStateOn**(bool *newState*)
Set whether the checkbox is ticked or not. Reimplemented from GuiButtonBaseCtrl .

Parameters **newState** – If true the box will be checked, if false, it will be unchecked.

GuiRadioCtrl A button based around the radio concept.

Inherit: GuiCheckBoxCtrl

Description GuiRadioCtrl's functionality is based around GuiButtonBaseCtrl's ButtonTypeRadio type.

A button control with a radio box and a text label. This control is used in groups where multiple radio buttons present a range of options out of which one can be chosen. A radio button automatically signals its siblings when it is toggled on.

Example:

```
// Create a GuiCheckBoxCtrl that calls randomFunction with its current value when clicked.
%radio = newGuiRadioCtrl()
{
   profile = "GuiRadioProfile";
};
```

GuiSwatchButtonCtrl A button that is used to represent color; often used in correlation with a color picker.

Inherit: GuiButtonBaseCtrl

Description A swatch button is a push button that uses its color field to designate the color drawn over an image, on top of a button.

The color itself is a float value stored inside the GuiSwatchButtonCtrl::color field. The texture path that represents the image underlying the color is stored inside the GuiSwatchButtonCtrl::gridBitmap field. The default value assigned toGuiSwatchButtonCtrl::color is "1 1 1 1"(White). The default/fallback image assigned to GuiSwatchButtonCtrl::gridBitmap is "tools/gui/images/transp_grid".

Example:

```
// Create a GuiSwatchButtonCtrl that calls randomFunction with its current color when clicked
%swatchButton = newGuiSwatchButtonCtrl()
{
   profile = "GuiInspectorSwatchButtonProfile";
   command = "randomFunction( $ThisControl.color );";
};
```

Methods

void `GuiSwatchButtonCtrl::setColor` (string *newColor*)

Set the color of the swatch control.

Parameters **`newColor`** – The new color string given to the swatch control in float format "r g b a".

Fields

ColorF `GuiSwatchButtonCtrl::color`

The foreground color of GuiSwatchButtonCtrl .

string `GuiSwatchButtonCtrl::gridBitmap`

The bitmap used for the transparent grid.

Enumeration

`enum GuiButtonType`

Type of button control.

Parameters

- **`PushButton`** – A button that triggers an action when clicked.
- **`ToggleButton`** – A button that is toggled between on and off state.
- **`RadioButton`** – A button placed in groups for presenting choices.

General Controls

A collection of general controls (bitmap, text, popup, etc).

Classes

GuiBitmapCtrl A gui control that is used to display an image.

Inherit: GuiControl

Description The image is stretched to the constraints of the control by default. However, the control can also tile the image as well.

The image itself is stored inside the GuiBitmapCtrl::bitmap field. The boolean value that decides whether the image is stretched or tiled is stored inside the GuiBitmapCtrl::wrap field.

Example:

```
// Create a tiling GuiBitmapCtrl that displays "myImage.png"
%bitmapCtrl = newGuiBitmapCtrl()
{
   bitmap = "myImage.png";
   wrap = "true";
};
```

Methods

void `GuiBitmapCtrl::setBitmap` (String *filename*, bool *resize*)

Assign an image to the control. Child controls with resize according to their layout settings.

Parameters

- **`filename`** – The filename of the image.
- **`resize`** – Optional parameter. If true, the GUI will resize to fit the image.

void `GuiBitmapCtrl::setBitmap` (String *filename*)

Assign an image to the control. Child controls will resize according to their layout settings.

Parameters

- **`filename`** – The filename of the image.
- **`resize`** – A boolean value that decides whether the ctrl refreshes or not.

void `GuiBitmapCtrl::setValue` (int *x*, int *y*)

Set the offset of the bitmap within the control.

Parameters

- **`x`** – The x-axis offset of the image.
- **`y`** – The y-axis offset of the image.

Fields

filename `GuiBitmapCtrl::bitmap`

The bitmap file to display in the control.

bool `GuiBitmapCtrl::wrap`

If true, the bitmap is tiled inside the control rather than stretched to fit.

GuiBubbleTextCtrl A single-line text control that displays its text in a multi-line popup when clicked.

Inherit: GuiTextCtrl

Description This control acts like a GuiTextCtrl (and inherits from it), when clicked it creates a GuiMLTextCtrl roughly where you clicked with the same text in it. This allows you to have a single line text control which upon clicking will display the entire text contained in a multi-line format.

Example:

```
newGuiBubbleTextCtrl(BubbleTextGUI)
{
   text = "This is the first sentence. This second sentence can be sized outside of the default singl
};
```

GuiDirectoryFileListCtrl A control that displays a list of files from within a single directory in the game file system.

Inherit: GuiListBoxCtrl

Description A control that displays a list of files from within a single directory in the game file system.

Example:

```
newGuiDirectoryFileListCtrl()
{
   filePath = "art/shapes";
   fileFilter = "*.dts" TAB "*.dae";
   //Properties not specific to this control have been omitted from this example.
};
```

Methods

string GuiDirectoryFileListCtrl::**getSelectedFile**()
Get the currently selected filename.

Returns The filename of the currently selected file

string GuiDirectoryFileListCtrl::**getSelectedFiles**()
Get the list of selected files.

Returns A space separated list of selected files

void GuiDirectoryFileListCtrl::**reload**()
Update the file list.

void GuiDirectoryFileListCtrl::**setFilter**(string *filter*)
Set the file filter.

Parameters **filter** – Tab-delimited list of file name patterns. Only matched files will be displayed.

bool GuiDirectoryFileListCtrl::**setPath**(string *path*, string *filter*)
Set the search path and file filter.

Parameters

- **path** – Path in game directory from which to list files.
- **filter** – Tab-delimited list of file name patterns. Only matched files will be displayed.

Fields

string GuiDirectoryFileListCtrl::**fileFilter**
Tab-delimited list of file name patterns. Only matched files will be displayed.

string GuiDirectoryFileListCtrl::**filePath**
Path in game directory from which to list files.

GuiMLTextEditCtrl A text entry control that accepts the Gui Markup Language ('ML') tags and multiple lines.

Inherit: GuiMLTextCtrl

Description A text entry control that accepts the Gui Markup Language ('ML') tags and multiple lines.

Example:

```
newGuiMLTextEditCtrl()
   {
      lineSpacing = "2";
      allowColorChars = "0";
      maxChars = "-1";
      deniedSound = "DeniedSoundProfile";
      text = "";
      escapeCommand = "onEscapeScriptFunction();";
     //Properties not specific to this control have been omitted from this example.
   };
```

Fields

string GuiMLTextEditCtrl::**escapeCommand**
Script function to run whenever the 'escape' key is pressed when this control is in focus.

GuiPopUpMenuCtrl A control that allows to select a value from a drop-down list.

Inherit: GuiTextCtrl

Description For a nearly identical GUI with additional features, use GuiPopUpMenuCtrlEx.

Example:

```
newGuiPopUpMenuCtrl()
{
   maxPopupHeight = "200";
   sbUsesNAColor = "0";
   reverseTextList = "0";
   bitmapBounds = "16 16";
   maxLength = "1024";
   position = "56 31";
   extent = "64 64";
   minExtent = "8 2";
   profile = "GuiPopUpMenuProfile";
   tooltipProfile = "GuiToolTipProfile";
};
```

Methods

void GuiPopUpMenuCtrl::**add**(string *name*, int *idNum*, int *scheme*)

void GuiPopUpMenuCtrl::**addScheme**(int *id*, ColorI *fontColor*, ColorI *fontColorHL*, ColorI *fontColorSEL*)

void GuiPopUpMenuCtrl::**changeTextById**(int *id*, string *text*)

void GuiPopUpMenuCtrl::**clearEntry**(S32 *entry*)

int GuiPopUpMenuCtrl::**findText**(string *text*)
Returns the position of the first entry containing the specified text.

string GuiPopUpMenuCtrl::**getTextById**(int *id*)

void GuiPopUpMenuCtrl::**replaceText**(bool *doReplaceText*)

void GuiPopUpMenuCtrl::**setEnumContent**(string, string)
This fills the popup with a classrep's field enumeration type info. More of a helper function than anything. If console access to the field list is added, at least for the enumerated types, then this should go away..

void GuiPopUpMenuCtrl::**setFirstSelected**()

void GuiPopUpMenuCtrl::**setSelected**(int *id*)

Fields
filename GuiPopUpMenuCtrl::**bitmap**
Point2I GuiPopUpMenuCtrl::**bitmapBounds**

void GuiPopUpMenuCtrl::**clear**
Clear the popup list.

void GuiPopUpMenuCtrl::**forceClose**

void GuiPopUpMenuCtrl::**forceOnAction**

int GuiPopUpMenuCtrl::**getSelected**

string GuiPopUpMenuCtrl::**getText**

int GuiPopUpMenuCtrl::**maxPopupHeight**

bool GuiPopUpMenuCtrl::**reverseTextList**

bool GuiPopUpMenuCtrl::**sbUsesNAColor**

void GuiPopUpMenuCtrl::**setNoneSelected**

int GuiPopUpMenuCtrl::**size**
Get the size of the menu - the number of entries in it.

void GuiPopUpMenuCtrl::**sort**
Sort the list alphabetically.

void GuiPopUpMenuCtrl::**sortID**
Sort the list by ID.

GuiPopUpMenuCtrlEx A control that allows to select a value from a drop-down list.

Inherit: GuiTextCtrl

Description This is essentially a GuiPopUpMenuCtrl, but with quite a few more features.

Example:

```
newGuiPopUpMenuCtrlEx()
{
   maxPopupHeight = "200";
   sbUsesNAColor = "0";
   reverseTextList = "0";
   bitmapBounds = "16 16";
   hotTrackCallback = "0";
   extent = "64 64";
   profile = "GuiDefaultProfile";
   tooltipProfile = "GuiToolTipProfile";
};
```

Methods

void GuiPopUpMenuCtrlEx::**add**(string *name*, int *idNum*, int *scheme*)

void GuiPopUpMenuCtrlEx::**add**(string *name*, S32 *idNum*, S32 *scheme*)

Adds an entry to the list.

Parameters

- **name** – String containing the name of the entry
- **idNum** – Numerical value assigned to the name
- **scheme** – Optional ID associated with a scheme for font coloring, highlight coloring, and selection coloring

void GuiPopUpMenuCtrlEx::**addCategory**(string *text*)

Add a category to the list. Acts as a separator between entries, allowing for sub-lists

Parameters **text** – Name of the new category

void GuiPopUpMenuCtrlEx::**addScheme**(int *id*, ColorI *fontColor*, ColorI *fontColorHL*, ColorI *fontColorSEL*)

Create a new scheme and add it to the list of choices for when a new text entry is added.

Parameters

- **id** – Numerical id associated with this scheme
- **fontColor** – The base text font color. Formatted as "Red Green Blue", each a numerical between 0 and 255.
- **fontColorHL** – Color of text when being highlighted. Formatted as "Red Green Blue", each a numerical between 0 and 255.
- **fontColorSel** – Color of text when being selected. Formatted as "Red Green Blue", each a numerical between 0 and 255.

void GuiPopUpMenuCtrlEx::**clear**()

Clear the popup list. Reimplemented from SimSet .

void GuiPopUpMenuCtrlEx::**clearEntry**(S32 *entry*)

int GuiPopUpMenuCtrlEx::**findText**(string *text*)

Returns the id of the first entry containing the specified text or -1 if not found.

Parameters **text** – String value used for the query

Returns Numerical ID of entry containing the text.

void GuiPopUpMenuCtrlEx::**forceClose**()

Manually force this control to collapse and close.

void GuiPopUpMenuCtrlEx::**forceOnAction**()

Manually for the onAction function, which updates everything in this control.

int GuiPopUpMenuCtrlEx::**getSelected**()

Get the current selection of the menu.

Returns Returns the ID of the currently selected entry

string GuiPopUpMenuCtrlEx::**getText**()

Get the. Detailed description

Parameters **param** – Description

Returns Returns current text in string format

Example:

```
// Comment
code();
```

string GuiPopUpMenuCtrlEx::**getTextById**(int *id*)
Get the text of an entry based on an ID.

Parameters **id** – The ID assigned to the entry being queried

Returns String contained by the specified entry, NULL if empty or bad ID

void GuiPopUpMenuCtrlEx::**setNoneSelected**(int *param*)
Clears selection in the menu.

GuiPopUpMenuCtrlEx::**setSelected**(int *id*, bool *scriptCallback*)
brief Manually set an entry as selected int his control

Parameters

- **id** – The ID of the entry to select
- **scripCallback** – Optional boolean that forces the script callback if true

GuiPopUpMenuCtrlEx::**setSelected**(bool *scriptCallback*)
brief Manually set the selection to the first entry

Parameters **scripCallback** – Optional boolean that forces the script callback if true

void GuiPopUpMenuCtrlEx::**setText**(string *text*)
Set the current text to a specified value. Reimplemented from GuiTextCtrl .

Parameters **text** – String containing new text to set

void GuiPopUpMenuCtrlEx::**sort**()
Sort the list alphabetically.

void GuiPopUpMenuCtrlEx::**sortID**()
Sort the list by ID.

Fields

filename GuiPopUpMenuCtrlEx::**bitmap**
File name of bitmap to use.

Point2I GuiPopUpMenuCtrlEx::**bitmapBounds**
Boundaries of bitmap displayed.

string GuiPopUpMenuCtrlEx::**getColorById**
Get color of an entry's box.

Parameters **id** – ID number of entry to query

Returns ColorI in the format of "Red Green Blue Alpha", each of with is a value between 0 - 255

bool GuiPopUpMenuCtrlEx::**hotTrackCallback**
Whether to provide a 'onHotTrackItem' callback when a list item is hovered over.

int GuiPopUpMenuCtrlEx::**maxPopupHeight**
Length of menu when it extends.

void GuiPopUpMenuCtrlEx::**replaceText**
Flag that causes each new text addition to replace the current entry.

Parameters **True** – to turn on replacing, false to disable it

bool GuiPopUpMenuCtrlEx::**reverseTextList**
Reverses text list if popup extends up, instead of down.

bool GuiPopUpMenuCtrlEx::**sbUsesNAColor**
Deprecated.

void GuiPopUpMenuCtrlEx::**setEnumContent**
This fills the popup with a classrep's field enumeration type info. More of a helper function than anything. If console access to the field list is added, at least for the enumerated types, then this should go away.

Parameters

- **class** – Name of the class containing the enum
- **enum** – Name of the enum value to acces

int GuiPopUpMenuCtrlEx::**size**
Get the size of the menu.

Returns Number of entries in the menu

GuiSeparatorCtrl

Inherit: GuiControl

Description A control that renders a horizontal or vertical separator with an optional text label (horizontal only).

Example:

```
newGuiSeparatorCtrl()
{
   profile = "GuiDefaultProfile";
   position = "505 0";
   extent = "10 17";
   minExtent = "10 17";
   canSave = "1";
   visible = "1";
   horizSizing = "left";
};
```

Fields

int GuiSeparatorCtrl::**borderMargin**

string GuiSeparatorCtrl::**caption**
Optional text label to display.

bool GuiSeparatorCtrl::**invisible**

int GuiSeparatorCtrl::**leftMargin**
Left margin of text label.

GuiSeparatorType GuiSeparatorCtrl::**type**
Orientation of separator.

GuiTextEditCtrl A component that places a text entry box on the screen.

Inherit: GuiTextCtrl

Description A component that places a text entry box on the screen.

Fonts and sizes are changed using profiles. The text value can be set or entered by a user.

Example:

```
newGuiTextEditCtrl(MessageHud_Edit)
   {
       text = "Hello World";
       validate = "validateCommand();"escapeCommand = "escapeCommand();";
       historySize = "5";
       tabComplete = "true";
       deniedSound = "DeniedSoundProfile";
       sinkAllKeyEvents = "true";
       password = "true";
       passwordMask = "*";
        //Properties not specific to this control have been omitted from this example.
   };
```

Methods

void GuiTextEditCtrl::**clearSelectedText**()

Unselects all selected text in the control.

Example:

```
// Inform the control to unselect all of its selected text
%thisGuiTextEditCtrl.clearSelectedText();
```

void GuiTextEditCtrl::**forceValidateText**()

Force a validation to occur.

Example:

```
// Inform the control to force a validation of its text.
%thisGuiTextEditCtrl.forceValidateText();
```

int GuiTextEditCtrl::**getCursorPos**()

Returns the current position of the text cursor in the control.

Returns Text cursor position within the control.

Example:

```
// Acquire the cursor position in the control
%position = %thisGuiTextEditCtrl.getCursorPost();
```

string GuiTextEditCtrl::**getText**()

Acquires the current text displayed in this control.

Returns The current text within the control.

Example:

```
// Acquire the value of the text control.
%text = %thisGuiTextEditCtrl.getText();
```

bool GuiTextEditCtrl::**isAllTextSelected**()

Checks to see if all text in the control has been selected.

Returns True if all text in the control is selected, otherwise false.

Example:

```
// Check to see if all text has been selected or not.
%allSelected = %thisGuiTextEditCtrl.isAllTextSelected();
```

void GuiTextEditCtrl::**onReturn**()
Called when the 'Return' or 'Enter' key is pressed.

Example:

```
// Return or Enter key was pressed, causing the callback to occur.GuiTextEditCtrl::onReturn(%thi
   {
      // Code to run when the onReturn callback occurs
   }
```

void GuiTextEditCtrl::**onTabComplete**(string *val*)
Called if tabComplete is true, and the 'tab' key is pressed.

Parameters **val** – Input to mimick the '1' sent by the actual tab key button press.

Example:

```
// Tab key has been pressed, causing the callback to occur.GuiTextEditCtrl::onTabComplete(%this,
   {
      //Code to run when the onTabComplete callback occurs
   }
```

void GuiTextEditCtrl::**onValidate**()
Called whenever the control is validated.

Example:

```
// The control gets validated, causing the callback to occur
GuiTextEditCtrl::onValidated(%this)
   {
      // Code to run when the control is validated
   }
```

void GuiTextEditCtrl::**selectAllText**()
Selects all text within the control.

Example:

```
// Inform the control to select all of its text.
%thisGuiTextEditCtrl.selectAllText();
```

void GuiTextEditCtrl::**setCursorPos**(int *position*)
Sets the text cursor at the defined position within the control.

Parameters **position** – Text position to set the text cursor.

Example:

```
// Define the cursor position
%position = "12";

// Inform the GuiTextEditCtrl control to place the text cursor at the defined position
%thisGuiTextEditCtrl.setCursorPos(%position);
```

void GuiTextEditCtrl::**setText**(string *text*)
Sets the text in the control. Reimplemented from GuiTextCtrl .

Parameters **text** – Text to place in the control.

Example:

```
    // Define the text to display
    %text = "Text!"// Inform the GuiTextEditCtrl to display the defined text
    %thisGuiTextEditCtrl.setText(%text);
```

Fields

SFXTrack GuiTextEditCtrl::**deniedSound**
If the attempted text cannot be entered, this sound effect will be played.

string GuiTextEditCtrl::**escapeCommand**
Script command to be called when the Escape key is pressed.

int GuiTextEditCtrl::**historySize**
How large of a history buffer to maintain.

bool GuiTextEditCtrl::**password**
If true, all characters entered will be stored in the control, however will display as the character stored in passwordMask.

string GuiTextEditCtrl::**passwordMask**
If 'password' is true, this is the character that will be used to mask the characters in the control.

bool GuiTextEditCtrl::**sinkAllKeyEvents**
If true, every key event will act as if the Enter key was pressed.

bool GuiTextEditCtrl::**tabComplete**
If true, when the 'tab' key is pressed, it will act as if the Enter key was pressed on the control.

string GuiTextEditCtrl::**validate**
Script command to be called when the first validater is lost.

GuiTextListCtrl GUI control that displays a list of text. Text items in the list can be individually selected.

Inherit: GuiArrayCtrl

Description GUI control that displays a list of text. Text items in the list can be individually selected.

Example:

```
newGuiTextListCtrl(EndGameGuiList)
      {
         columns = "0 256";
           fitParentWidth = "1";
         clipColumnText = "0";
          //Properties not specific to this control have been omitted from this example.
      };
```

Methods

int GuiTextListCtrl::**addRow**(int *id*, string *text*, int *index*)
Adds a new row at end of the list with the defined id and text. If index is used, then the new row is inserted at the row location of 'index'.

Parameters

- **id** – Id of the new row.
- **text** – Text to display at the new row.
- **index** – Index to insert the new row at. If not used, new row will be placed at the end of the list.

Returns Returns the row index of the new row. If 'index' was defined, then this just returns the number of rows in the list.

Example:

```
// Define the id
%id = "4";

// Define the text to display
%text = "Display Text"// Define the index (optional)
%index = "2"// Inform the GuiTextListCtrl control to add the new row with the defined informatic
%rowIndex = %thisGuiTextListCtrl.addRow(%id,%text,%index);
```

void GuiTextListCtrl::**clear**()

Clear the list.

Example:

```
// Inform the GuiTextListCtrl control to clear its contents
%thisGuiTextListCtrl.clear();
```

void GuiTextListCtrl::**clearSelection**()

Set the selection to nothing.

Example:

```
// Deselect anything that is currently selected
%thisGuiTextListCtrl.clearSelection();
```

int GuiTextListCtrl::**findTextIndex**(string *needle*)

Find needle in the list, and return the row number it was found in.

Parameters **needle** – Text to find in the list.

Returns Row number that the defined text was found in,

Example:

```
// Define the text to find in the list
%needle = "Text To Find";

// Request the row number that contains the defined text to find

%rowNumber = %thisGuiTextListCtrl.findTextIndex(%needle);
```

int GuiTextListCtrl::**getRowId**(int *index*)

Get the row ID for an index.

Parameters **index** – Index to get the RowID at

Returns RowId at the defined index.

Example:

```
// Define the index
%index = "3";

// Request the row ID at the defined index
%rowId = %thisGuiTextListCtrl.getRowId(%index);
```

int GuiTextListCtrl::**getRowNumById**(int *id*)

Get the row number for a specified id.

Parameters **id** – Id to get the row number at

Example:

```
// Define the id
%id = "4";

// Request the row number from the GuiTextListCtrl control at the defined id.
%rowNumber = %thisGuiTextListCtrl.getRowNumById(%id);
```

string GuiTextListCtrl::**getRowText**(int *index*)
Get the text of the row with the specified index.

Parameters **index** – Row index to acquire the text at.

Returns Text at the defined row index.

Example:

```
// Define the row index
%index = "5";

// Request the text from the row at the defined index
%rowText = %thisGuiTextListCtrl.getRowText(%index);
```

string GuiTextListCtrl::**getRowTextById**(int *id*)
Get the text of a row with the specified id.

Returns Row text at the requested row id.

Example:

```
// Define the id
%id = "4";

// Inform the GuiTextListCtrl control to return the text at the defined row id
%rowText = %thisGuiTextListCtrl.getRowTextById(%id);
```

int GuiTextListCtrl::**getSelectedId**()
Get the ID of the currently selected item.

Returns The id of the selected item in the list.

Example:

```
// Acquire the ID of the selected item in the list.
%id = %thisGuiTextListCtrl.getSelectedId();
```

int GuiTextListCtrl::**getSelectedRow**()
Returns the selected row index (not the row ID).

Returns Index of the selected row

Example:

```
// Acquire the selected row index
%rowIndex = %thisGuiTextListCtrl.getSelectedRow();
```

bool GuiTextListCtrl::**isRowActive**(int *rowNum*)
Check if the specified row is currently active or not.

Parameters **rowNum** – Row number to check the active state.

Returns Active state of the defined row number.

Example:

```
// Define the row number
%rowNum = "5";

// Request the active state of the defined row number from the GuiTextListCtrl control.
%rowActiveState = %thisGuiTextListCtrl.isRowActive(%rowNum);
```

void GuiTextListCtrl::**onDeleteKey**(string *id*)
Called when the delete key has been pressed.

Parameters **id** – Id of the selected item in the list

Example:

```
// The delete key was pressed while the GuiTextListCtrl was in focus, causing the callback to oc
   {
      // Code to run when the delete key is pressed
   }
```

void GuiTextListCtrl::**onSelect**(string *cellid*, string *text*)
Called whenever an item in the list is selected.

Parameters

- **cellid** – The ID of the cell that was selected
- **text** – The text in the selected cel

Example:

```
// A cel in the control was selected, causing the callback to occurGuiTextListCtrl::onSelect(%th
   {
      // Code to run when a cel item is selected
   }
```

void GuiTextListCtrl::**removeRow**(int *index*)
Remove a row from the table, based on its index.

Parameters **index** – Row index to remove from the list.

Example:

```
// Define the row index
%index = "4";

// Inform the GuiTextListCtrl control to remove the row at the defined row index
%thisGuiTextListCtrl.removeRow(%index);
```

void GuiTextListCtrl::**removeRowById**(int *id*)
Remove row with the specified id.

Parameters **id** – Id to remove the row entry at

Example:

```
// Define the id
%id = "4";

// Inform the GuiTextListCtrl control to remove the row at the defined id
%thisGuiTextListCtrl.removeRowById(%id);
```

int GuiTextListCtrl::**rowCount**()
Get the number of rows.

Returns Number of rows in the list.

Example:

```
// Get the number of rows in the list
%rowCount = %thisGuiTextListCtrl.rowCount();
```

void GuiTextListCtrl::**scrollVisible**(int *rowNum*)
Scroll so the specified row is visible.

Parameters **rowNum** – Row number to make visible

Example:

```
// Define the row number to make visible
%rowNum = "4";

// Inform the GuiTextListCtrl control to scroll the list so the defined rowNum is visible.
%thisGuiTextListCtrl.scrollVisible(%rowNum);
```

void GuiTextListCtrl::**setRowActive**(int *rowNum*, bool *active*)
Mark a specified row as active/not.

Parameters

- **rowNum** – Row number to change the active state.
- **active** – Boolean active state to set the row number.

Example:

```
// Define the row number
%rowNum = "4";

// Define the boolean active state
%active = "true";

// Informthe GuiTextListCtrl control to set the defined active state at the defined row number.
%thisGuiTextListCtrl.setRowActive(%rowNum,%active);
```

void GuiTextListCtrl::**setRowById**(int *id*, string *text*)
Sets the text at the defined id.

Parameters

- **id** – Id to change.
- **text** – Text to use at the Id.

Example:

```
// Define the id
%id = "4";

// Define the text
%text = "Text To Display";

// Inform the GuiTextListCtrl control to display the defined text at the defined id
%thisGuiTextListCtrl.setRowById(%id,%text);
```

void GuiTextListCtrl::**setSelectedById**(int *id*)
Finds the specified entry by id, then marks its row as selected.

Parameters **id** – Entry within the text list to make selected.

Example:

```
// Define the id
%id = "5";

// Inform the GuiTextListCtrl control to set the defined id entry as selected
%thisGuiTextListCtrl.setSelectedById(%id);
```

void GuiTextListCtrl::**setSelectedRow**(int *rowNum*)
the specified row.

Parameters **rowNum** – Row number to set selected.

Example:

```
// Define the row number to set selected
%rowNum = "4";

%guiTextListCtrl.setSelectedRow(%rowNum);
```

void GuiTextListCtrl::**sort**(int *columnId*, bool *increasing*)
Performs a standard (alphabetical) sort on the values in the specified column.

Parameters

- **columnId** – Column ID to perform the sort on.
- **increasing** – If false, sort will be performed in reverse.

Example:

```
// Define the columnId
%id = "1";

// Define if we are increasing or not
%increasing = "false";

// Inform the GuiTextListCtrl to perform the sort operation
%thisGuiTextListCtrl.sort(%id,%increasing);
```

void GuiTextListCtrl::**sortNumerical**(int *columnID*, bool *increasing*)
Perform a numerical sort on the values in the specified column. Detailed description

Parameters

- **columnId** – Column ID to perform the sort on.
- **increasing** – If false, sort will be performed in reverse.

Example:

```
// Define the columnId
%id = "1";

// Define if we are increasing or not
%increasing = "false";

// Inform the GuiTextListCtrl to perform the sort operation
%thisGuiTextListCtrl.sortNumerical(%id,%increasing);
```

Fields

bool GuiTextListCtrl::**clipColumnText**
If true, text exceeding a column's given width will get clipped.

intList GuiTextListCtrl::**columns**
A vector of column offsets. The number of values determines the number of columns in the table.

bool GuiTextListCtrl::**fitParentWidth**
If true, the width of this control will match the width of its parent.

GuiArrayCtrl

Enumeration

enum GuiSeparatorType
GuiSeparatorCtrl orientations.

Parameters

- **Vertical** –
- **Horizontal** –

Container Controls

A collection of various containers (container, window, scroll, etc).

Classes

GuiAutoScrollCtrl

Inherit: GuiTickCtrl

Description A container that scrolls its child control up over time.

This container can be used to scroll a single child control in either of the four directions.

Example:

```
// Create a GuiAutoScrollCtrl that scrolls a long text of credits.newGuiAutoScrollCtrl( CreditsScroll
{
   position = "0 0";
   extent = Canvas.extent.x SPC Canvas.extent.y;

   scrollDirection = "Up"; // Scroll upwards.startDelay = 4; // Wait 4 seconds before starting to scr
   {
      text = $CREDITS;
   };
};

function CreditsScroller::onComplete( %this )
{
   // Switch back to main menu after credits have rolled.
   Canvas.setContent( MainMenu );
}
```

```
// Start rolling credits.
Canvas.setContent( CreditsScroller );
```

Methods

void GuiAutoScrollCtrl::**onComplete**()
Called when the child control has been scrolled in entirety.

void GuiAutoScrollCtrl::**onReset**()
Called when the child control is reset to its initial position and the cycle starts again.

void GuiAutoScrollCtrl::**onStart**()
Called when the control starts to scroll.

void GuiAutoScrollCtrl::**onTick**()
Called every 32ms on the control.

void GuiAutoScrollCtrl::**reset**()
Reset scrolling.

Fields

int GuiAutoScrollCtrl::**childBorder**
Padding to put around child control (in pixels).

bool GuiAutoScrollCtrl::**isLooping**
If true, the scrolling will reset to the beginning once completing a cycle.

float GuiAutoScrollCtrl::**resetDelay**
Seconds to wait after scrolling completes before resetting and starting over.

GuiAutoScrollDirection GuiAutoScrollCtrl::**scrollDirection**
Direction in which the child control is moved.

bool GuiAutoScrollCtrl::**scrollOutOfSight**
If true, the child control will be completely scrolled out of sight; otherwise it will only scroll until the other end becomes visible.

float GuiAutoScrollCtrl::**scrollSpeed**
Scrolling speed in pixels per second.

float GuiAutoScrollCtrl::**startDelay**
Seconds to wait before starting to scroll.

GuiContainer Brief Desc.

Inherit: GuiControl

Description Brief Desc.

Example:

```
// Comment:
%okButton = new ClassObject()
instantiation
```

Fields

bool GuiContainer::**anchorBottom**

bool GuiContainer::**anchorLeft**

bool GuiContainer::**anchorRight**

bool GuiContainer::**anchorTop**

GuiDockingType GuiContainer::**docking**

RectSpacingI GuiContainer::**margin**

RectSpacingI GuiContainer::**padding**

GuiControlArrayControl Brief Desc.

Inherit: GuiControl

Description Brief Desc.

Example:

```
// Comment:
%okButton = new ClassObject()
instantiation
```

Fields

int GuiControlArrayControl::**colCount**
Number of colums in the array.

intList GuiControlArrayControl::**colSizes**
Size of each individual column.

int GuiControlArrayControl::**colSpacing**
Padding to put between columns.

int GuiControlArrayControl::**rowSize**
Heigth of a row in the array.

int GuiControlArrayControl::**rowSpacing**
Padding to put between rows.

GuiDynamicCtrlArrayControl A container that arranges children into a grid.

Inherit: GuiControl

Description This container maintains a 2D grid of GUI controls. If one is added, deleted, or resized, then the grid is updated. The insertion order into the grid is determined by the internal order of the children (ie. the order of addition). Children are added to the grid by row or column until they fill the assocated GuiDynamicCtrlArrayControl extent (width or height). For example, a GuiDynamicCtrlArrayControl with 15 children, and fillRowFirst set to true may be arranged as follows:

If dynamicSize were set to true in this case, the GuiDynamicCtrlArrayControl height would be calculated to fit the 3 rows of child controls.

Example:

```
newGuiDynamicCtrlArrayControl()
{
   colSize = "128";
   rowSize = "18";
   colSpacing = "2";
   rowSpacing = "2";
```

```
   frozen = "0";
   autoCellSize = "1";
   fillRowFirst = "1";
   dynamicSize = "1";
   padding = "0 0 0 0";
   //Properties not specific to this control have been omitted from this example.
};
```

Methods

void GuiDynamicCtrlArrayControl::**refresh**()
Recalculates the position and size of this control and all its children.

Fields

bool GuiDynamicCtrlArrayControl::**autoCellSize**
When true, the cell size is set to the widest/tallest child control.

int GuiDynamicCtrlArrayControl::**colCount**
Number of columns the child controls have been arranged into. This value is calculated automatically when children are added, removed or resized; writing it directly has no effect.

int GuiDynamicCtrlArrayControl::**colSize**
Width of each column. If autoCellSize is set, this will be calculated automatically from the widest child control.

int GuiDynamicCtrlArrayControl::**colSpacing**
Spacing between columns.

bool GuiDynamicCtrlArrayControl::**dynamicSize**
If true, the width or height of this control will be automatically calculated based on the number of child controls (width if fillRowFirst is false, height if fillRowFirst is true).

bool GuiDynamicCtrlArrayControl::**fillRowFirst**
Controls whether rows or columns are filled first. If true, controls are added to the grid left-to-right (to fill a row); then rows are added top-to-bottom as shown below: If false, controls are added to the grid top-to-bottom (to fill a column); then columns are added left-to-right as shown below:

bool GuiDynamicCtrlArrayControl::**frozen**
When true, the array will not update when new children are added or in response to child resize events. This is useful to prevent unnecessary resizing when adding, removing or resizing a number of child controls.

RectSpacingI GuiDynamicCtrlArrayControl::**padding**
Padding around the top, bottom, left, and right of this control. This reduces the area available for child controls.

int GuiDynamicCtrlArrayControl::**rowCount**
Number of rows the child controls have been arranged into. This value is calculated automatically when children are added, removed or resized; writing it directly has no effect.

int GuiDynamicCtrlArrayControl::**rowSize**
Height of each row. If autoCellSize is set, this will be calculated automatically from the tallest child control.

int GuiDynamicCtrlArrayControl::**rowSpacing**
Spacing between rows.

GuiFrameSetCtrl A gui control allowing a window to be subdivided into panes, each of which displays a gui control child of the GuiFrameSetCtrl.

Inherit: GuiContainer

Description Each gui control child will have an associated FrameDetail through which frame-specific details can be assigned. Frame-specific values override the values specified for the entire frameset.

Note that it is possible to have more children than frames, or more frames than children. In the former case, the extra children will not be visible (they are moved beyond the visible extent of the frameset). In the latter case, frames will be empty. For example, if a frameset had two columns and two rows but only three gui control children they would be assigned to frames as follows:

The last frame would be blank.

Example:

```
newGuiFrameSetCtrl()
{
   columns = "3";
   rows = "2";
   borderWidth = "1";
   borderColor = "128 128 128";
   borderEnable = "dynamic";
   borderMovable = "dynamic";
   autoBalance = "1";
   fudgeFactor = "0";
   //Properties not specific to this control have been omitted from this example.
};
```

Methods

void GuiFrameSetCtrl::**addColumn**()
Add a new column.

void GuiFrameSetCtrl::**addRow**()
Add a new row.

void GuiFrameSetCtrl::**frameBorder**(int *index*, string *state*)
Override the borderEnable setting for this frame.

Parameters

- **index** – Index of the frame to modify
- **state** – New borderEnable state: “on”, “off” or “dynamic”

void GuiFrameSetCtrl::**frameMinExtent**(int *index*, int *width*, int *height*)
Set the minimum width and height for the frame. It will not be possible for the user to resize the frame smaller than this.

Parameters

- **index** – Index of the frame to modify
- **width** – Minimum width in pixels
- **height** – Minimum height in pixels

void GuiFrameSetCtrl::**frameMovable**(int *index*, string *state*)
Override the borderMovable setting for this frame.

Parameters

- **index** – Index of the frame to modify
- **state** – New borderEnable state: “on”, “off” or “dynamic”

void GuiFrameSetCtrl::**framePadding**(int *index*, RectSpacingI *padding*)
Set the padding for this frame. Padding introduces blank space on the inside edge of the frame.

Parameters

- **index** – Index of the frame to modify
- **padding** – Frame top, bottom, left, and right padding

int GuiFrameSetCtrl::**getColumnCount**()
Get the number of columns.

Returns The number of columns

int GuiFrameSetCtrl::**getColumnOffset**(int *index*)
Get the horizontal offset of a column.

Parameters **index** – Index of the column to query

Returns Column offset in pixels

RectSpacingI GuiFrameSetCtrl::**getFramePadding**(int *index*)
Get the padding for this frame.

Parameters **index** – Index of the frame to query

int GuiFrameSetCtrl::**getRowCount**()
Get the number of rows.

Returns The number of rows

int GuiFrameSetCtrl::**getRowOffset**(int *index*)
Get the vertical offset of a row.

Parameters **index** – Index of the row to query

Returns Row offset in pixels

void GuiFrameSetCtrl::**removeColumn**()
Remove the last (rightmost) column.

void GuiFrameSetCtrl::**removeRow**()
Remove the last (bottom) row.

void GuiFrameSetCtrl::**setColumnOffset**(int *index*, int *offset*)
Set the horizontal offset of a column. Note that column offsets must always be in increasing order, and therefore this offset must be between the offsets of the columns either side.

Parameters

- **index** – Index of the column to modify
- **offset** – New column offset

void GuiFrameSetCtrl::**setRowOffset**(int *index*, int *offset*)
Set the vertical offset of a row. Note that row offsets must always be in increasing order, and therefore this offset must be between the offsets of the rows either side.

Parameters

- **index** – Index of the row to modify
- **offset** – New row offset

void GuiFrameSetCtrl::**updateSizes**()
Recalculates child control sizes.

Fields

bool GuiFrameSetCtrl::**autoBalance**
If true, row and column offsets are automatically scaled to match the new extents when the control is resized.

ColorI GuiFrameSetCtrl::**borderColor**
Color of interior borders between cells.

GuiFrameState GuiFrameSetCtrl::**borderEnable**
Controls whether frame borders are enabled. Frames use this value unless overridden for that frame using ctrl.frameBorder(index)

GuiFrameState GuiFrameSetCtrl::**borderMovable**
Controls whether borders can be dynamically repositioned with the mouse by the user. Frames use this value unless overridden for that frame using ctrl.frameMovable(index)

int GuiFrameSetCtrl::**borderWidth**
Width of interior borders between cells in pixels.

intList GuiFrameSetCtrl::**columns**
A vector of column offsets (determines the width of each column).

int GuiFrameSetCtrl::**fudgeFactor**
Offset for row and column dividers in pixels.

intList GuiFrameSetCtrl::**rows**
A vector of row offsets (determines the height of each row).

GuiPaneControl A collapsable pane control.

Inherit: GuiControl

Description This class wraps a single child control and displays a header with caption above it. If you click the header it will collapse or expand (if collapsable is enabled). The control resizes itself based on its collapsed/expanded size. In the GUI editor, if you just want the header you can make collapsable false. The caption field lets you set the caption; it expects a bitmap (from the GuiControlProfile) that contains two images - the first is displayed when the control is expanded and the second is displayed when it is collapsed. The header is sized based on the first image.

Example:

```
newGuiPaneControl()
{
   caption = "Example Pane";
   collapsable = "1";
   barBehindText = "1";
   //Properties not specific to this control have been omitted from this example.
};
```

Methods

void GuiPaneControl::**setCollapsed**(bool *collapse*)
Collapse or un-collapse the control.

Parameters **collapse** – True to collapse the control, false to un-collapse it

Fields

bool GuiPaneControl::**barBehindText**
Whether to draw the bitmapped pane bar behind the header text, too.

string GuiPaneControl::**caption**
Text label to display as the pane header.

string GuiPaneControl::**captionID**
String table text ID to use as caption string (overrides 'caption').

bool GuiPaneControl::**collapsable**
Whether the pane can be collapsed by clicking its header.

GuiPanel The GuiPanel panel is a container that when opaque will draw a left to right gradient using its profile fill and fill highlight colors.

Inherit: GuiContainer

Description The GuiPanel panel is a container that when opaque will draw a left to right gradient using its profile fill and fill highlight colors.

Example:

```
// Mandatory GuiDefaultProfile// Contains the fill color information required by a GuiPanel// Some va
{
   // fill color
   opaque = false;
   fillColor = "242 241 240";
   fillColorHL ="228 228 235";
   fillColorSEL = "98 100 137";
   fillColorNA = "255 255 255 ";
};

newGuiPanel(TestPanel)
{
   position = "45 33";
   extent = "342 379";
   minExtent = "16 16";
   horizSizing = "right";
   vertSizing = "bottom";
   profile = "GuiDefaultProfile"; // Color fill info is in this profileisContainer = "1";
};
```

GuiRolloutCtrl A container that shows a single child with an optional header bar that can be used to collapse and expand the rollout.

Inherit: GuiControl

Description A rollout is a container that can be collapsed and expanded using smooth animation. By default, rollouts will display a header with a caption along the top edge of the control which can be clicked by the user to toggle the collapse state of the rollout.

Rollouts will automatically size themselves to exactly fit around their child control. They will also automatically position their child control in their upper left corner below the header (if present).

Methods

void GuiRolloutCtrl::**collapse**()
Collapse the rollout if it is currently expanded. This will make the rollout's child control invisible.

void GuiRolloutCtrl::**expand**()
Expand the rollout if it is currently collapsed. This will make the rollout's child control visible.

void GuiRolloutCtrl::**instantCollapse**()
Instantly collapse the rollout without animation. To smoothly slide the rollout to collapsed state, use collapse() .

void GuiRolloutCtrl::**instantExpand**()
Instantly expand the rollout without animation. To smoothly slide the rollout to expanded state, use expand() .

bool GuiRolloutCtrl::**isExpanded**()
Determine whether the rollout is currently expanded, i.e. whether the child control is visible. Reimplemented from SimObject .

Returns True if the rollout is expanded, false if not.

void GuiRolloutCtrl::**onCollapsed**()
Called when the rollout is collapsed.

void GuiRolloutCtrl::**onExpanded**()
Called when the rollout is expanded.

void GuiRolloutCtrl::**onHeaderRightClick**()
Called when the user right-clicks on the rollout's header. This is useful for implementing context menus for rollouts.

void GuiRolloutCtrl::**sizeToContents**()
Resize the rollout to exactly fit around its child control. This can be used to manually trigger a recomputation of the rollout size.

void GuiRolloutCtrl::**toggleCollapse**()
Toggle the current collapse state of the rollout. If it is currently expanded, then collapse it. If it is currently collapsed, then expand it.

void GuiRolloutCtrl::**toggleExpanded**(bool *instantly*)
Toggle the current expansion state of the rollout If it is currently expanded, then collapse it. If it is currently collapsed, then expand it.

Parameters **instant** – If true, the rollout will toggle its state without animation. Otherwise, the rollout will smoothly slide into the opposite state.

Fields

bool GuiRolloutCtrl::**autoCollapseSiblings**
Whether to automatically collapse sibling rollouts. If this is true, the rollout will automatically collapse all sibling rollout controls when it is expanded. If this is false, the auto-collapse behavior can be triggered by CTRL (CMD on MAC) clicking the rollout header. CTRL/CMD clicking also works if this is false, in which case the auto-collapsing of sibling controls will be temporarily deactivated.

string GuiRolloutCtrl::**caption**
Text label to display on the rollout header.

bool GuiRolloutCtrl::**clickCollapse**
Whether the rollout can be collapsed by clicking its header.

int GuiRolloutCtrl::**defaultHeight**
Default height of the client area. This is used when no child control has been added to the rollout.

bool GuiRolloutCtrl::**expanded**
The current rollout expansion state.

bool GuiRolloutCtrl::**hideHeader**
Whether to render the rollout header.

RectI GuiRolloutCtrl::**margin**
Margin to put around child control.

GuiScrollCtrl A container that allows to view one or more possibly larger controls inside its area by providing horizontal and/or vertical scroll bars.

Inherit: GuiContainer

Description A container that allows to view one or more possibly larger controls inside its area by providing horizontal and/or vertical scroll bars.

Methods

void `GuiScrollCtrl::computeSizes()`
Refresh sizing and positioning of child controls.

Point2I `GuiScrollCtrl::getScrollPosition()`
Get the current coordinates of the scrolled content.

Returns The current position of the scrolled content.

int `GuiScrollCtrl::getScrollPositionX()`
Get the current X coordinate of the scrolled content.

Returns The current X coordinate of the scrolled content.

int `GuiScrollCtrl::getScrollPositionY()`
Get the current Y coordinate of the scrolled content.

Returns The current Y coordinate of the scrolled content.

void `GuiScrollCtrl::onScroll()`
Called each time the child controls are scrolled by some amount.

void `GuiScrollCtrl::scrollToBottom()`
Scroll all the way to the bottom of the vertical scrollbar and the left of the horizontal bar.

void `GuiScrollCtrl::scrollToObject` (GuiControl *control*)
Scroll the control so that the given child control is visible.

Parameters **`control`** – A child control.

void `GuiScrollCtrl::scrollToTop()`
Scroll all the way to the top of the vertical and left of the horizontal scrollbar.

void `GuiScrollCtrl::setScrollPosition` (int *x*, int *y*)
Set the position of the scrolled content.

Parameters

- **`x`** – Position on X axis.
- **`y`** – Position on y axis.

Fields

Point2I `GuiScrollCtrl::childMargin`
Padding region to put around child contents.

bool `GuiScrollCtrl::constantThumbHeight`

GuiScrollBarBehavior `GuiScrollCtrl::hScrollBar`
When to display the horizontal scrollbar.

bool `GuiScrollCtrl::lockHorizScroll`
Horizontal scrolling not allowed if set.

bool GuiScrollCtrl::**lockVertScroll**
Vertical scrolling not allowed if set.

int GuiScrollCtrl::**mouseWheelScrollSpeed**
Pixels/Tick - if not positive then mousewheel scrolling occurs instantly (like other scrolling).

GuiScrollBarBehavior GuiScrollCtrl::**vScrollBar**
When to display the vertical scrollbar.

bool GuiScrollCtrl::**willFirstRespond**

GuiSpeedometerHud Displays the speed of the current Vehicle based control object.

Inherit: GuiBitmapCtrl

Description This control only works if a server connection exists, and its control object is a Vehicle derived class. If either of these requirements is false, the control is not rendered. The control renders the speedometer needle as a colored quad, rotated to indicate the Vehicle speed as determined by the minAngle, maxAngle, and maxSpeed properties. This control is normally placed on top of a GuiBitmapCtrl representing the speedometer dial.

Example:

```
newGuiSpeedometerHud()
{
   maxSpeed = "100";
   minAngle = "215";
   maxAngle = "0";
   color = "1 0.3 0.3 1";
   center = "130 123";
   length = "100";
   width = "2";
   tail = "0";
   //Properties not specific to this control have been omitted from this example.
};
```

Fields

Point2F GuiSpeedometerHud::**center**
Center of the needle, offset from the GuiSpeedometerHud control top left corner.

ColorF GuiSpeedometerHud::**color**
Color of the needle.

float GuiSpeedometerHud::**length**
Length of the needle from center to end.

float GuiSpeedometerHud::**maxAngle**
Angle (in radians) of the needle when the Vehicle speed is gt = maxSpeed. An angle of 0 points right, 90 points up etc).

float GuiSpeedometerHud::**maxSpeed**
Maximum Vehicle speed (in Torque units per second) to represent on the speedo (Vehicle speeds greater than this are clamped to maxSpeed).

float GuiSpeedometerHud::**minAngle**
Angle (in radians) of the needle when the Vehicle speed is 0. An angle of 0 points right, 90 points up etc).

float GuiSpeedometerHud::**tail**
Length of the needle from center to tail.

float GuiSpeedometerHud::**width**
Width of the needle.

GuiSplitContainer A container that splits its area between two child controls.

Inherit: GuiContainer

Description A GuiSplitContainer can be used to dynamically subdivide an area between two child controls. A splitter bar is placed between the two controls and allows to dynamically adjust the sizing ratio between the two sides. Splitting can be either horizontal (subdividing top and bottom) or vertical (subdividing left and right) depending on orientation.

By using fixedPanel, one of the panels can be chosen to remain at a fixed size (fixedSize).

Example:

```
// Create a vertical splitter with a fixed-size left panel.
%splitter = newGuiSplitContainer()
{
   orientation = "Vertical";
   fixedPanel = "FirstPanel";
   fixedSize = 100;

   newGuiScrollCtrl()
   {
      newGuiMLTextCtrl()
      {
         text = %longText;
      };
   };

   newGuiScrollCtrl()
   {
      newGuiMLTextCtrl()
      {
         text = %moreLongText;
      };
   };
};
```

Fields

GuiSplitFixedPanel GuiSplitContainer::**fixedPanel**
Which (if any) side of the splitter to keep at a fixed size.

int GuiSplitContainer::**fixedSize**
Width of the fixed panel specified by fixedPanel (if any).

GuiSplitOrientation GuiSplitContainer::**orientation**
Whether to split between top and bottom (horizontal) or between left and right (vertical).

Point2I GuiSplitContainer::**splitPoint**
Point on control through which the splitter goes. Changed relatively if size of control changes.

int GuiSplitContainer::**splitterSize**
Width of the splitter bar between the two sides. Default is 2.

GuiStackControl A container that stacks its children horizontally or vertically.

Inherit: GuiControl

Description This container maintains a horizontal or vertical stack of GUI controls. If one is added, deleted, or resized, then the stack is resized to fit. The order of the stack is determined by the internal order of the children (ie. the order of addition).

Example:

```
newGuiStackControl()
{
   stackingType = "Dynamic";
   horizStacking = "Left to Right";
   vertStacking = "Top to Bottom";
   padding = "4";
   dynamicSize = "1";
   dynamicNonStackExtent = "0";
   dynamicPos = "0";
   changeChildSizeToFit = "1";
   changeChildPosition = "1";
   //Properties not specific to this control have been omitted from this example.
};
```

Methods

void GuiStackControl::**freeze**(bool *freeze*)

Prevents control from restacking - useful when adding or removing child controls.

Parameters **freeze** – True to freeze the control, false to unfreeze it

Example:

```
%stackCtrl.freeze(true);
// add controls to stack
%stackCtrl.freeze(false);
```

bool GuiStackControl::**isFrozen**()

Return whether or not this control is frozen.

void GuiStackControl::**updateStack**()

Restack the child controls.

Fields

bool GuiStackControl::**changeChildPosition**

Determines whether to reposition child controls. If true, horizontally stacked children are aligned along the top edge of the stack control. Vertically stacked children are aligned along the left edge of the stack control. If false, horizontally stacked children retain their Y position, and vertically stacked children retain their X position.

bool GuiStackControl::**changeChildSizeToFit**

Determines whether to resize child controls. If true, horizontally stacked children keep their width, but have their height set to the stack control height. Vertically stacked children keep their height, but have their width set to the stack control width. If false, child controls are not resized.

bool GuiStackControl::**dynamicNonStackExtent**

Determines whether to resize the stack control along the non-stack axis (change height for horizontal stacking, change width for vertical stacking). No effect if dynamicSize is false. If true, the stack will be resized to the maximum of the child control widths/heights. If false, the stack will not be resized.

bool GuiStackControl::**dynamicPos**
Determines whether to reposition the stack along the stack axis when it is auto-resized. No effect if dynamicSize is false. If true, the stack will grow left for horizontal stacking, and grow up for vertical stacking. If false, the stack will grow right for horizontal stacking, and grow down for vertical stacking.

bool GuiStackControl::**dynamicSize**
Determines whether to resize the stack control along the stack axis (change width for horizontal stacking, change height for vertical stacking). If true, the stack width/height will be resized to the sum of the child control widths/heights. If false, the stack will not be resized.

GuiHorizontalStackingType GuiStackControl::**horizStacking**
Controls the type of horizontal stacking to use (Left to Right or Right to Left).

int GuiStackControl::**padding**
Distance (in pixels) between stacked child controls.

GuiStackingType GuiStackControl::**stackingType**
Determines the method used to position the child controls.

GuiVerticalStackingType GuiStackControl::**vertStacking**
Controls the type of vertical stacking to use (Top to Bottom or Bottom to Top).

GuiTabBookCtrl A container.

Inherit: GuiContainer

Description A container.

Example:

```
// Create
```

Methods

void GuiTabBookCtrl::**addPage**(string *title*)
Add a new tab page to the control.

Parameters **title** – Title text for the tab page header.

int GuiTabBookCtrl::**getSelectedPage**()
Get the index of the currently selected tab page.

Returns Index of the selected tab page or -1 if no tab page is selected.

void GuiTabBookCtrl::**onTabRightClick**(String *text*, int *index*)
Called when the user right-clicks on a tab page header.

Parameters

- **text** – Text of the page header for the tab that is being selected.
- **index** – Index of the tab page being selected.

void GuiTabBookCtrl::**onTabSelected**(String *text*, int *index*)
Called when a new tab page is selected.

Parameters

- **text** – Text of the page header for the tab that is being selected.
- **index** – Index of the tab page being selected.

void GuiTabBookCtrl::**selectPage**(int *index*)
Set the selected tab page.

Parameters **index** – Index of the tab page.

Fields

bool GuiTabBookCtrl::**allowReorder**
Whether reordering tabs with the mouse is allowed.

int GuiTabBookCtrl::**defaultPage**
Index of page to select on first onWake() call (-1 to disable).

int GuiTabBookCtrl::**frontTabPadding**
X offset of first tab page header.

int GuiTabBookCtrl::**minTabWidth**
Minimum width allocated to a tab page header.

int GuiTabBookCtrl::**selectedPage**
Index of currently selected page.

int GuiTabBookCtrl::**tabHeight**
Height of tab page headers.

int GuiTabBookCtrl::**tabMargin**
Spacing to put between individual tab page headers.

GuiTabPosition GuiTabBookCtrl::**tabPosition**
Where to place the tab page headers.

GuiTabPageCtrl A single page in a GuiTabBookCtrl.

Inherit: GuiTextCtrl

Description A single page in a GuiTabBookCtrl.

Example:

```
newGuiTabPageCtrl()
{
   fitBook = "1";
   //Properties not specific to this control have been omitted from this example.
};
```

Methods

void GuiTabPageCtrl::**select**()
Select this page in its tab book.

Fields

bool GuiTabPageCtrl::**fitBook**
Determines whether to resize this page when it is added to the tab book. If true, the page will be resized according to the tab book extents and tabPosition property.

GuiTreeViewCtrl Hierarchical list of text items with optional icons.

Inherit: GuiArrayCtrl

Description Can also be used to inspect SimObject hierarchies, primarily within editors.

GuiTreeViewCtrls can either display arbitrary user-defined trees or can be used to display SimObject hierarchies where each parent node in the tree is a SimSet or SimGroup and each leaf node is a SimObject.

Each item in the tree has a text and a value. For trees that display SimObject hierarchies, the text for each item is automatically derived from objects while the value for each item is the ID of the respective SimObject. For trees that are not tied to SimObjects, both text and value of each item are set by the user.

Additionally, items in the tree can have icons.

Each item in the tree has a distinct numeric ID that is unique within its tree. The ID of the root item, which is always present on a tree, is 0.

Example:

```
newGuiTreeViewCtrl(DatablockEditorTree)
{
   tabSize = "16";
   textOffset = "2";
   fullRowSelect = "0";
   itemHeight = "21";
   destroyTreeOnSleep = "0";
   MouseDragging = "0";
   MultipleSelections = "1";
   DeleteObjectAllowed = "1";
   DragToItemAllowed = "0";
   ClearAllOnSingleSelection = "1";
   showRoot = "1";
   internalNamesOnly = "0";
   objectNamesOnly = "0";
   compareToObjectID = "0";
   Profile = "GuiTreeViewProfile";
   tooltipprofile = "GuiToolTipProfile";
   hovertime = "1000";
};
```

Methods

void GuiTreeViewCtrl::**addSelection**(int *id*, bool *isLastSelection*)

Add an item/object to the current selection.

Parameters

- **id** – ID of item/object to add to the selection.
- **isLastSelection** – Whether there are more pending items/objects to be added to the selection. If false, the control will defer refreshing the tree and wait until addSelection() is called with this parameter set to true.

bool GuiTreeViewCtrl::**buildIconTable**()

Builds an icon table.

bool GuiTreeViewCtrl::**canRenameObject**(SimObject *object*)

void GuiTreeViewCtrl::**clear**()

Empty tree.

void GuiTreeViewCtrl::**clearFilterText**()

Clear the current item filtering pattern.

void GuiTreeViewCtrl::**clearSelection**()

Unselect all currently selected items.

void GuiTreeViewCtrl::**deleteSelection**()
Delete all items/objects in the current selection.

bool GuiTreeViewCtrl::**editItem**(TreeItemId *item*, string *newText*, string *newValue*)

bool GuiTreeViewCtrl::**expandItem**(TreeItemId *item*, bool *expand*)

int GuiTreeViewCtrl::**findChildItemByName**(int *parentId*, string *childName*)
Get the child item of the given parent item whose text matches childName .

Parameters

- **parentId** – Item ID of the parent in which to look for the child.
- **childName** – Text of the child item to find.

Returns

.

int GuiTreeViewCtrl::**findItemByName**(string *text*)
Get the ID of the item whose text matches the given text .

Parameters **text** – Item text to match.

Returns ID of the item or -1 if no item matches the given text.

int GuiTreeViewCtrl::**findItemByObjectId**(int *id*)
Find item by object id and returns the mId

int GuiTreeViewCtrl::**findItemByValue**(string *value*)
Get the ID of the item whose value matches value .

Parameters **value** – Value text to match.

Returns ID of the item or -1 if no item has the given value.

int GuiTreeViewCtrl::**getChild**(TreeItemId *item*)

string GuiTreeViewCtrl::**getFilterText**()
Get the current filter expression. Only tree items whose text matches this expression are displayed. By default, the expression is empty and all items are shown.

Returns The current filter pattern or an empty string if no filter pattern is currently active.

string GuiTreeViewCtrl::**getItemText**(TreeItemId *item*)

string GuiTreeViewCtrl::**getItemValue**(TreeItemId *item*)

int GuiTreeViewCtrl::**getNextSibling**(TreeItemId *item*)

int GuiTreeViewCtrl::**getParent**(TreeItemId *item*)

int GuiTreeViewCtrl::**getPrevSibling**(TreeItemId *item*)

int GuiTreeViewCtrl::**getSelectedItem**(int *index*)
Return the selected item at the given index.

int GuiTreeViewCtrl::**getSelectedObject**(int *index*)
Return the currently selected SimObject at the given index in inspector mode or -1.

string GuiTreeViewCtrl::**getTextToRoot**(TreeItemId *item*, Delimiter = , none)
gets the text from the current node to the root, concatenating at each branch upward, with a specified delimiter optionally

bool GuiTreeViewCtrl::**handleRenameObject**(string *newName*, SimObject *object*)

void GuiTreeViewCtrl::**hideSelection**(bool *state*)
Call SimObject::setHidden (state) on all objects in the current selection.

Parameters **state** – Visibility state to set objects in selection to.

int GuiTreeViewCtrl::**insertItem**(int *parentId*, string *text*, string *value*, string *icon*, int *normalImage*, int *expandedImage*)
Add a new item to the tree.

Parameters

- **parentId** – Item ID of parent to which to add the item as a child. 0 is root item.
- **text** – Text to display on the item in the tree.
- **value** – Behind-the-scenes value of the item.
- **icon** –
- **normalImage** –
- **expandedImage** –

Returns The ID of the newly added item.

bool GuiTreeViewCtrl::**isItemSelected**(int *id*)
Check whether the given item is currently selected in the tree.

Parameters **id** – Item/object ID.

Returns True if the given item/object is currently selected in the tree.

bool GuiTreeViewCtrl::**isParentItem**(int *id*)
Returns true if the given item contains child items.

bool GuiTreeViewCtrl::**isValidDragTarget**(int *id*, string *value*)

void GuiTreeViewCtrl::**lockSelection**(bool *lock*)
Set whether the current selection can be changed by the user or not.

Parameters **lock** – If true, the current selection is frozen and cannot be changed. If false, the selection may be modified.

bool GuiTreeViewCtrl::**markItem**(TreeItemId *item*, bool *mark*)

void GuiTreeViewCtrl::**moveItemDown**(TreeItemId *item*)

void GuiTreeViewCtrl::**moveItemUp**(TreeItemId *item*)

void GuiTreeViewCtrl::**onAddGroupSelected**(SimGroup *group*)

void GuiTreeViewCtrl::**onAddMultipleSelectionBegin**()

void GuiTreeViewCtrl::**onAddMultipleSelectionEnd**()

void GuiTreeViewCtrl::**onAddSelection**(int *itemOrObjectId*, bool *isLastSelection*)

void GuiTreeViewCtrl::**onBeginReparenting**()

void GuiTreeViewCtrl::**onClearSelection**()

void GuiTreeViewCtrl::**onDefineIcons**()

bool GuiTreeViewCtrl::**onDeleteObject**(SimObject *object*)

void GuiTreeViewCtrl::**onDeleteSelection**()

void GuiTreeViewCtrl::**onDragDropped**()

void GuiTreeViewCtrl::**onEndReparenting**()

void GuiTreeViewCtrl::**onInspect**(int *itemOrObjectId*)

void GuiTreeViewCtrl::**onKeyDown**(int *modifier*, int *keyCode*)

void GuiTreeViewCtrl::**onMouseDragged**()

void GuiTreeViewCtrl::**onMouseUp**(int *hitItemId*, int *mouseClickCount*)

void GuiTreeViewCtrl::**onObjectDeleteCompleted**()

void GuiTreeViewCtrl::**onRemoveSelection**(int *itemOrObjectId*)

void GuiTreeViewCtrl::**onReparent**(int *itemOrObjectId*, int *oldParentItemOrObjectId*, int *newParentItemOrObjectId*)

void GuiTreeViewCtrl::**onRightMouseDown**(int *itemId*, Point2I *mousePos*, SimObject *object*)

void GuiTreeViewCtrl::**onRightMouseUp**(int *itemId*, Point2I *mousePos*, SimObject *object*)

void GuiTreeViewCtrl::**onSelect**(int *itemOrObjectId*)

void GuiTreeViewCtrl::**onUnselect**(int *itemOrObjectId*)

void GuiTreeViewCtrl::**open**(SimSet *obj*, bool *okToEdit*)
Set the root of the tree view to the specified object, or to the root set.

bool GuiTreeViewCtrl::**removeItem**(TreeItemId *item*)

void GuiTreeViewCtrl::**removeSelection**()
Deselects an item.

void GuiTreeViewCtrl::**scrollVisible**(TreeItemId *item*)

int GuiTreeViewCtrl::**scrollVisibleByObjectId**(int *id*)
Show item by object id.returns true if sucessful.

bool GuiTreeViewCtrl::**selectItem**(TreeItemId *item*, bool *select*)

void GuiTreeViewCtrl::**setDebug**(bool *value*)
Enable/disable debug output.

void GuiTreeViewCtrl::**setFilterText**(string *pattern*)
Set the pattern by which to filter items in the tree. Only items in the tree whose text matches this pattern are displayed.

Parameters **pattern** – New pattern based on which visible items in the tree should be filtered. If empty, all items become visible.

void GuiTreeViewCtrl::**setItemImages**(int *id*, int *normalImage*, int *expandedImage*)
Sets the normal and expanded images to show for the given item.

void GuiTreeViewCtrl::**setItemTooltip**(int *id*, string *text*)
Set the tooltip to show for the given item.

void GuiTreeViewCtrl::**showItemRenameCtrl**(TreeItemId *id*)
Show the rename text field for the given item (only one at a time).

void GuiTreeViewCtrl::**sort**(int *parent*, bool *traverseHierarchy*, bool *parentsFirst*, bool *caseSensitive*)
Sorts all items of the given parent (or root). With 'hierarchy', traverses hierarchy.

void GuiTreeViewCtrl::**toggleHideSelection**()
Toggle the hidden state of all objects in the current selection.

void GuiTreeViewCtrl::**toggleLockSelection**()
Toggle the locked state of all objects in the current selection.

Fields

void GuiTreeViewCtrl::**addChildSelectionByValue**
addChildSelectionByValue(TreeItemId parent, value)

void GuiTreeViewCtrl::**buildVisibleTree**
Build the visible tree.

void GuiTreeViewCtrl::**cancelRename**
For internal use.

bool GuiTreeViewCtrl::**canRenameObjects**
If true clicking on a selected item (that is an object and not the root) will allow you to rename it.

bool GuiTreeViewCtrl::**clearAllOnSingleSelection**

bool GuiTreeViewCtrl::**compareToObjectID**

bool GuiTreeViewCtrl::**deleteObjectAllowed**

bool GuiTreeViewCtrl::**destroyTreeOnSleep**
If true, the entire tree item hierarchy is deleted when the control goes to sleep.

bool GuiTreeViewCtrl::**dragToItemAllowed**

bool GuiTreeViewCtrl::**fullRowSelect**

int GuiTreeViewCtrl::**getFirstRootItem**
Get id for root item.

int GuiTreeViewCtrl::**getItemCount**

string GuiTreeViewCtrl::**getSelectedItemList**
returns a space seperated list of mulitple item ids

int GuiTreeViewCtrl::**getSelectedItemsCount**

string GuiTreeViewCtrl::**getSelectedObjectList**
Returns a space sperated list of all selected object ids.

int GuiTreeViewCtrl::**itemHeight**

bool GuiTreeViewCtrl::**mouseDragging**

bool GuiTreeViewCtrl::**multipleSelections**
If true, multiple items can be selected concurrently.

void GuiTreeViewCtrl::**onRenameValidate**
For internal use.

void GuiTreeViewCtrl::**removeAllChildren**
removeAllChildren(TreeItemId parent)

void GuiTreeViewCtrl::**removeChildSelectionByValue**
removeChildSelectionByValue(TreeItemId parent, value)

bool GuiTreeViewCtrl::**renameInternal**
If true then object renaming operates on the internalName rather than the object name.

bool GuiTreeViewCtrl::**showClassNameForUnnamedObjects**
If true, class names will be used as object names for unnamed objects.

bool GuiTreeViewCtrl::**showClassNames**
If true, item text labels for objects will include class names.

bool GuiTreeViewCtrl::**showInternalNames**
If true, item text labels for obje ts will include internal names.

bool GuiTreeViewCtrl::**showObjectIds**
If true, item text labels for objects will include object IDs.

bool GuiTreeViewCtrl::**showObjectNames**
If true, item text labels for objects will include object names.

bool GuiTreeViewCtrl::**showRoot**
If true, the root item is shown in the tree.

int GuiTreeViewCtrl::**tabSize**

int GuiTreeViewCtrl::**textOffset**

bool GuiTreeViewCtrl::**tooltipOnWidthOnly**

bool GuiTreeViewCtrl::**useInspectorTooltips**

GuiWindowCtrl A window with a title bar and an optional set of buttons.

Inherit: GuiContainer

Description The GuiWindowCtrl class implements windows that can be freely placed within the render window. Additionally, the windows can be resized and maximized/minimized.

Example:

```
newGuiWindowCtrl( MyWindow )
{
   text = "My Window"; // The text that is displayed on the title bar.
   resizeWidth = true; // Allow horizontal resizing by user via mouse.
   resizeHeight = true; // Allow vertical resizing by user via mouse.
   canClose = true; // Display a close button in the title bar.
   canMinimize = true; // Display a minimize button in the title bar.
   canMaximize = true; // Display a maximize button in the title bar.
};
```

Methods

static void GuiWindowCtrl::**attach**(GuiWindowCtrl *bottomWindow*, GuiWindowCtrl *topWindow*)
Attach bottomWindow to so that bottomWindow moves along with topWindow when it is dragged.

Parameters

- **bottomWindow** –
- **topWindow** –

void GuiWindowCtrl::**attachTo**(GuiWindowCtrl *window*)

void GuiWindowCtrl::**onClose**()
Called when the close button has been pressed.

void GuiWindowCtrl::**onCollapse**()
Called when the window is collapsed by clicking its title bar.

void GuiWindowCtrl::**onMaximize**()
Called when the window has been maximized.

void GuiWindowCtrl::**onMinimize**()
Called when the window has been minimized.

void GuiWindowCtrl::**onRestore**()
Called when the window is restored from minimized, maximized, or collapsed state.

void GuiWindowCtrl::**selectWindow**()
Bring the window to the front.

void GuiWindowCtrl::**setCollapseGroup**(bool *state*)
Set the window's collapsing state.

void GuiWindowCtrl::**toggleCollapseGroup**()
Toggle the window collapsing.

Fields

bool GuiWindowCtrl::**canClose**
Whether the window has a close button.

bool GuiWindowCtrl::**canCollapse**
Whether the window can be collapsed by clicking its title bar.

bool GuiWindowCtrl::**canMaximize**
Whether the window has a maximize button.

bool GuiWindowCtrl::**canMinimize**
Whether the window has a minimize button.

bool GuiWindowCtrl::**canMove**
Whether the window can be moved by dragging its titlebar.

string GuiWindowCtrl::**closeCommand**
Script code to execute when the window is closed.

bool GuiWindowCtrl::**edgeSnap**
If true, the window will snap to the edges of other windows when moved close to them.

bool GuiWindowCtrl::**resizeHeight**
Whether the window can be resized vertically.

bool GuiWindowCtrl::**resizeWidth**
Whether the window can be resized horizontally.

string GuiWindowCtrl::**text**
Text label to display in titlebar.

Enumeration

enum GuiAutoScrollDirection
Direction in which to scroll the child control.

Parameters

- **Up** – Scroll from bottom towards top.
- **Down** – Scroll from top towards bottom.
- **Left** – Scroll from right towards left.
- **Right** – Scroll from left towards right.

enum GuiDockingType

Parameters

- **None** –
- **Client** –

- **Top** –
- **Bottom** –
- **Left** –
- **Right** –

enum GuiFrameState

Parameters

- **alwaysOn** –
- **alwaysOff** –
- **dynamic** –

enum GuiHorizontalStackingType

Determines how child controls are stacked horizontally.

Parameters

- **Right** – Child controls are positioned in order from left to right (left-most control is first).
- **Left** – Child controls are positioned in order from right to left (right-most control is first).

enum GuiScrollBarBehavior

Display behavior of a scroll bar. Determines when a scrollbar will be visible.

Parameters

- **alwaysOn** – Always visible.
- **alwaysOff** – Never visible.
- **dynamic** – Only visible when actually needed, i.e. when the child control(s) exceed the visible space on the given axis.

enum GuiSplitFixedPanel

Which side of the splitter to keep at a fixed size (if any).

Parameters

- **None** – Allow both childs to resize (default).
- **FirstPanel** – Keep.
- **SecondPanel** –

enum GuiSplitOrientation

Axis along which to divide the container's space.

Parameters

- **Vertical** – Divide vertically placing one child left and one child right.
- **Horizontal** – Divide horizontally placing one child on top and one child below.

enum GuiStackingType

Stacking method used to position child controls.

Parameters

- **Vertical** – Stack children vertically by setting their Y position.
- **Horizontal** – Stack children horizontall by setting their X position.

- **Dynamic** – Automatically switch between Vertical and Horizontal stacking. Vertical stacking is chosen when the stack control is taller than it is wide, horizontal stacking is chosen when the stack control is wider than it is tall.

enum GuiTabPosition
Where the control should put the tab headers for selecting individual pages.

Parameters

- **Top** – Tab headers on top edge.
- **Bottom** – Tab headers on bottom edge.

enum GuiVerticalStackingType
Determines how child controls are stacked vertically.

Parameters

- **Bottom** – Child controls are positioned in order from top to bottom (top-most control is first).
- **Top** – Child controls are positioned in order from bottom to top (bottom-most control is first).

Image and Video Controls

Controls that display images or videos.

Classes

GuiBitmapBorderCtrl A control that renders a skinned border specified in its profile.

Inherit: GuiControl

Description This control uses the bitmap specified in it's profile (GuiControlProfile::bitmapName). It takes this image and breaks up aspects of it to skin the border of this control with. It is also important to set GuiControlProfile::hasBitmapArray to true on the profile as well.

The bitmap referenced should be broken up into a 3 x 3 grid (using the top left color pixel as a border color between each of the images) in which it will map to the following places: 1 = Top Left Corner 2 = Top Right Corner 3 = Top Center 4 = Left Center 5 = Right Center 6 = Bottom Left Corner 7 = Bottom Center 8 = Bottom Right Corner 0 = Nothing

1 2 3 4 5 0 6 7 8

Example:

```
singleton GuiControlProfile (BorderGUIProfile)
{
   bitmap = "core/art/gui/images/borderArray";
   hasBitmapArray = true;
   opaque = false;
};

newGuiBitmapBorderCtrl(BitmapBorderGUI)
{
   profile = "BorderGUIProfile";
   position = "0 0";
   extent = "400 40";
```

```
   visible = "1";
};
```

GuiChunkedBitmapCtrl This is a control that will render a specified bitmap or a bitmap specified in a referenced variable.

Inherit: GuiControl

Description This control allows you to either set a bitmap with the "bitmap" field or with the setBitmap method. You can also choose to reference a variable in the "variable" field such as "$image" and then set "useVariable" to true. This will cause it to synchronize the variable with the bitmap displayed (if the variable holds a valid image). You can then change the variable and effectively changed the displayed image.

Example:

```
$image = "anotherbackground.png";
newGuiChunkedBitmapCtrl(ChunkedBitmap)
{
   bitmap = "background.png";
   variable = "$image";
   useVariable = false;
}

// This will result in the control rendering "background.png"// If we now set the useVariable to true
ChunkedBitmap.useVariable = true;
```

Methods

void `GuiChunkedBitmapCtrl::setBitmap`(string *filename*)

Set the image rendered in this control.

Parameters **`filename`** – The image name you want to set

Example:

```
ChunkedBitmap.setBitmap("images/background.png");
```

Fields

filename `GuiChunkedBitmapCtrl::bitmap`

This is the bitmap to render to the control.

bool `GuiChunkedBitmapCtrl::tile`

This is no longer in use.

bool `GuiChunkedBitmapCtrl::useVariable`

This decides whether to use the "bitmap" file or a bitmap stored in "variable".

GuiTheoraCtrl A control to playing Theora videos.

Inherit: GuiControl

Description This control can be used to play videos in the Theora video format. The videos may include audio in Vorbis format. The codecs for both formats are integrated with the engine and no codecs must be present on the user's machine.

Example:

```
%video = newGuiTheoraCtrl()
{
   theoraFile = "videos/intro.ogv";
   playOnWake = false;
   stopOnSleep = true;
}

Canvas.setContent( %video );
%video.play();
```

Methods

float GuiTheoraCtrl::**getCurrentTime**()
Get the current playback time.

Returns The elapsed playback time in seconds.

bool GuiTheoraCtrl::**isPlaybackDone**()
Test whether the video has finished playing.

Returns True if the video has finished playing, false otherwise.

void GuiTheoraCtrl::**pause**()
Pause playback of the video. If the video is not currently playing, the call is ignored. While stopped, the control displays the last frame.

void GuiTheoraCtrl::**play**()
Start playing the video. If the video is already playing, the call is ignored.

void GuiTheoraCtrl::**setFile**(string *filename*)
Set the video file to play. If a video is already playing, playback is stopped and the new video file is loaded.

Parameters **filename** – The video file to load.

void GuiTheoraCtrl::**stop**()
Stop playback of the video. The next call to play() will then start playback from the beginning of the video. While stopped, the control renders empty with just the background color.

Fields

ColorI GuiTheoraCtrl::**backgroundColor**
Fill color when video is not playing.

bool GuiTheoraCtrl::**matchVideoSize**
Whether to automatically match control extents to the video size.

bool GuiTheoraCtrl::**playOnWake**
Whether to start playing video when control is woken up.

bool GuiTheoraCtrl::**renderDebugInfo**
If true, displays an overlay on top of the video with useful debugging information.

bool GuiTheoraCtrl::**stopOnSleep**
Whether to stop video when control is set to sleep. If this is not set to true, the video will be paused when the control is put to sleep. This is because there is no support for seeking in the video stream in the player backend and letting the time source used to synchronize video (either audio or a raw timer) get far ahead of frame decoding will cause possibly very long delays when the control is woken up again.

filename GuiTheoraCtrl::**theoraFile**
Theora video file to play.

GuiTheoraTranscoder GuiTheoraCtrl::**transcoder**
The routine to use for Y'CbCr to RGB conversion.

Enumeration

enum GuiBitmapMode
Rendering behavior when placing bitmaps in controls.

Parameters

- **Stretched** – Stretch bitmap to fit control extents.
- **Centered** – Center bitmap in control.

enum GuiIconButtonIconLocation

Parameters

- **None** –
- **Left** –
- **Right** –
- **Center** –

enum GuiIconButtonTextLocation

Parameters

- **None** –
- **Bottom** –
- **Right** –
- **Top** –
- **Left** –
- **Center** –

enum GuiTheoraTranscoder
Routine to use for converting Theora's Y'CbCr pixel format to RGB color space.

Parameters

- **Auto** – Automatically detect most appropriate setting.
- **Generic** – Slower but beneric transcoder that can convert all Y'CbCr input formats to RGB or RGBA output.
- **SSE2420RGBA** – Fast SSE2-based transcoder with fixed conversion from 4:2:0 Y'CbCr to RGBA.

Value Controls

Controls that display values and optionally allow to edit them.

Classes

GuiGraphCtrl A control that plots one or more curves in a chart.

Inherit: GuiControl

Description Up to 6 individual curves can be plotted in the graph. Each plotted curve can have its own display style including its own charting style (plotType) and color (plotColor).

The data points on each curve can be added in one of two ways:

Example:

```
// Create a graph that plots a red polyline graph of the FPS counter in a 1 second (1000 millise
{
   plotType[ 0 ] = "PolyLine";
   plotColor[ 0 ] = "1 0 0";
   plotVariable[ 0 ] = "fps::real";
   plotInterval[ 0 ] = 1000;
};
```

Methods

void GuiGraphCtrl::**addAutoPlot**(int *plotId*, string *variable*, int *updateFrequency*)

Sets up the given plotting curve to automatically plot the value of the variable with a frequency of updateFrequency .

Parameters

- **plotId** – Index of the plotting curve. Must be 0<=plotId<6.
- **variable** – Name of the global variable.
- **updateFrequency** – Frequency with which to add new data points to the plotting curve (in milliseconds).

Example:

```
// Plot FPS counter at 1 second intervals.
%graph.addAutoPlot( 0, "fps::real", 1000 );
```

void GuiGraphCtrl::**addDatum**(int *plotId*, float *value*)

Add a data point to the plot's curve.

Parameters

- **plotId** – Index of the plotting curve to which to add the data point. Must be 0<=plotId<6.
- **value** – Value of the data point to add to the curve.

float GuiGraphCtrl::**getDatum**(int *plotId*, int *index*)

Get a data point on the given plotting curve.

Parameters

- **plotId** – Index of the plotting curve from which to fetch the data point. Must be 0<=plotId<6.
- **index** – Index of the data point on the curve.

Returns are out of range.

void GuiGraphCtrl::**matchScale**(int *plotID1*, int *plotID2*, ...)

Set the scale of all specified plots to the maximum scale among them.

Parameters

- **plotID1** – Index of plotting curve.
- **plotID2** – Index of plotting curve.

void GuiGraphCtrl::**removeAutoPlot**(int *plotId*)
Stop automatic variable plotting for the given curve.

Parameters **plotId** – Index of the plotting curve. Must be 0<=plotId<6.

void GuiGraphCtrl::**setGraphType**(int *plotId*, GuiGraphType *graphType*)
Change the charting type of the given plotting curve.

Parameters

- **plotId** – Index of the plotting curve. Must be 0<=plotId<6.
- **graphType** – Charting type to use for the curve.

Fields

float GuiGraphCtrl::**centerY**
Ratio of where to place the center coordinate of the graph on the Y axis. 0.5=middle height of control. This allows to account for graphs that have only positive or only negative data points, for example.

ColorF GuiGraphCtrl::**plotColor**[6]
Color to use for the plotting curves in the graph.

int GuiGraphCtrl::**plotInterval**[6]
Interval between auto-plots of plotVariable for the respective curve (in milliseconds).

GuiGraphType GuiGraphCtrl::**plotType**[6]
Charting type of the plotting curves.

string GuiGraphCtrl::**plotVariable**[6]
Name of the variable to automatically plot on the curves. If empty, auto-plotting is disabled for the respective curve.

GuiProgressBitmapCtrl A horizontal progress bar rendered from a repeating image.

Inherit: GuiTextCtrl

Description This class is used give progress feedback to the user. Unlike GuiProgressCtrl which simply renders a filled rectangle, GuiProgressBitmapCtrl renders the bar using a bitmap.

This bitmap can either be simple, plain image which is then stretched into the current extents of the bar as it fills up or it can be a bitmap array with three entries. In the case of a bitmap array, the first entry in the array is used to render the left cap of the bar and the third entry in the array is used to render the right cap of the bar. The second entry is streched in-between the two caps.

Example:

```
// This example shows one way to break down a long-running computation into phases// and increme
{
   bitmap = "core/art/gui/images/loading";
   extent = "300 50";
   position = "100 100";
};

// Put the control on the canvas.
%wrapper = newGuiControl();
%wrapper.addObject( Progress );
Canvas.pushDialog( %wrapper );

// Start the computation.schedule( 1, 0, "phase1" );
```

```
function phase1()
{
   Progress.setValue( 0 );

   // Perform some computation.//...// Update progress.
   Progress.setValue( 0.25 );

   // Schedule next phase.  Dont call directly so engine gets a change to run refresh.schedule(
}

function phase2()
{
   // Perform some computation.//...// Update progress.
   Progress.setValue( 0.7 );

   // Schedule next phase.  Dont call directly so engine gets a change to run refresh.schedule(
}

function phase3()
{
   // Perform some computation.//...// Update progress.
   Progress.setValue( 0.9 );

   // Schedule next phase.  Dont call directly so engine gets a change to run refresh.schedule(
}

function phase4()
{
   // Perform some computation.//...// Final update of progress.
   Progress.setValue( 1.0 );
}
```

Methods

void GuiProgressBitmapCtrl::**setBitmap**(string *filename*)

Set the bitmap to use for rendering the progress bar.

Parameters **filename** – ~Path to the bitmap file.

Fields

filename GuiProgressBitmapCtrl::**bitmap**

~Path to the bitmap file to use for rendering the progress bar. If the profile assigned to the control already has a bitmap assigned, this property need not be set in which case the bitmap from the profile is used.

GuiProgressCtrl GUI Control which displays a horizontal bar which increases as the progress value of 0.0 - 1.0 increases.

Inherit: GuiTextCtrl

Description GUI Control which displays a horizontal bar which increases as the progress value of 0.0 - 1.0 increases.

Example:

```
newGuiProgressCtrl(JS_statusBar)
    {
          //Properties not specific to this control have been omitted from this example.
     };

// Define the value to set the progress bar%value = "0.5f"// Set the value of the progress bar,
%thisGuiProgressCtrl.setValue(%value);
// Get the value of the progress bar.
%progress = %thisGuiProgressCtrl.getValue();
```

GuiSliderCtrl A control that displays a value between its minimal and maximal bounds using a slider placed on a vertical or horizontal axis.

Inherit: GuiControl

Description A control that displays a value between its minimal and maximal bounds using a slider placed on a vertical or horizontal axis.

A slider displays a value and allows that value to be changed by dragging a thumb control along the axis of the slider. In this way, the value is changed between its allowed minimum and maximum.

To hook up script code to the value changes of a slider, use the command and altCommand properties. command is executed once the thumb is released by the user whereas altCommand is called any time the slider value changes. When changing the slider value from script, however, trigger of altCommand is suppressed by default.

The orientation of a slider is automatically determined from the ratio of its width to its height. If a slider is taller than it is wide, it will be rendered with a vertical orientation. If it is wider than it is tall, it will be rendered with a horizontal orientation.

The rendering of a slider depends on the bitmap in the slider's profile. This bitmap must be a bitmap array comprised of at least five bitmap rectangles. The rectangles are used such that:

Example:

```
// Create a sound source and a slider that changes the volume of the source.

%source = sfxPlayOnce( "art/sound/testing", AudioLoop2D );

new GuiSlider()
{
   // Update the sound source volume when the slider is being dragged and released.command = %sc

   // Limit the range to 0..1 since that is the allowable range for sound volumes.range = "0 1";
};
```

Methods

float `GuiSliderCtrl::getValue()`

Get the current value of the slider based on the position of the thumb.

Returns Slider position (from range.x to range.y).

bool `GuiSliderCtrl::isThumbBeingDragged()`

Returns true if the thumb is currently being dragged by the user. This method is mainly useful for scrubbing type sliders where the slider position is sync'd to a changing value. When the user is dragging the thumb, however, the sync'ing should pause and not get in the way of the user.

void GuiSliderCtrl::**onMouseDragged**()
Called when the left mouse button is dragged across the slider.

void GuiSliderCtrl::**setValue**(float *pos*, bool *doCallback*)
Set position of the thumb on the slider.

Parameters

- **pos** – New slider position (from range.x to range.y)
- **doCallback** – If true, the altCommand callback will be invoked

Fields

Point2F GuiSliderCtrl::**range**
Min and max values corresponding to left and right slider position.

bool GuiSliderCtrl::**snap**
Whether to snap the slider to tick marks.

int GuiSliderCtrl::**ticks**
Spacing between tick marks in pixels. 0=off.

float GuiSliderCtrl::**value**
The value corresponding to the current slider position.

Enumeration

enum GuiGraphType
The charting style of a single plotting curve in a GuiGraphCtrl .

Parameters

- **Bar** – Plot the curve as a bar chart.
- **Filled** – Plot a filled poly graph that connects the data points on the curve.
- **Point** – Plot each data point on the curve as a single dot.
- **PolyLine** – Plot straight lines through the data points of the curve.

Utility Controls

A collection of utility classes that support other GUI controls.

Classes

GuiDragAndDropControl A container control that can be used to implement drag&drop behavior.

Inherit: GuiControl

Description GuiDragAndDropControl is a special control that can be used to allow drag&drop behavior to be implemented where GuiControls may be dragged across the canvas and the dropped on other GuiControls.

To start a drag operation, construct a GuiDragAndDropControl and add the control that should be drag&dropped as a child to it. Note that this must be a single child control. To drag multiple controls, wrap them in a new GuiControl object as a temporary container.

Then, to initiate the drag, add the GuiDragAndDropControl to the canvas and call startDragging(). You can optionally supply an offset to better position the GuiDragAndDropControl on the mouse cursor.

As the GuiDragAndDropControl is then moved across the canvas, it will call the onControlDragEnter(), onControlDragExit(), onControlDragged(), and finally onControlDropped() callbacks on the visible top-most controls that it moves across. onControlDropped() is called when the mouse button is released and the drag operation thus finished.

Example:

```
// The following example implements drag&drop behavior for GuiSwatchButtonCtrl so that// one col
function GuiSwatchButtonCtrl::onMouseDragged( %this )
{
   // First we construct a new temporary swatch button that becomes the payload for our// drag c

   %payload = newGuiSwatchButtonCtrl();
   %payload.assignFieldsFrom( %this );
   %payload.position = "0 0";
   %payload.dragSourceControl = %this; // Remember where the drag originated from so that we don

   %xOffset = getWord( %payload.extent, 0 ) / 2;
   %yOffset = getWord( %payload.extent, 1 ) / 2;

   // Compute the initial position of the GuiDragAndDrop control on the cavas based on the curre

   %cursorpos = Canvas.getCursorPos();
   %xPos = getWord( %cursorpos, 0 ) - %xOffset;
   %yPos = getWord( %cursorpos, 1 ) - %yOffset;

   // Create the drag control.

   %ctrl = newGuiDragAndDropControl()
   {
      canSaveDynamicFields    = "0";
      Profile                 = "GuiSolidDefaultProfile";
      HorizSizing             = "right";
      VertSizing              = "bottom";
      Position                = %xPos SPC %yPos;
      extent                  = %payload.extent;
      MinExtent               = "4 4";
      canSave                 = "1";
      Visible                 = "1";
      hovertime               = "1000";

      // Let the GuiDragAndDropControl delete itself on mouse-up.  When the drag is aborted,// t
      deleteOnMouseUp         = true;

      // To differentiate drags, use the namespace hierarchy to classify them.// This will allow
   };

   // Add the temporary color swatch to the drag control as the payload.
   %ctrl.add( %payload );

   // Start drag by adding the drag control to the canvas and then calling startDragging().

   Canvas.getContent().add( %ctrl );
   %ctrl.startDragging( %xOffset, %yOffset );
}
```

```
//-----------------------------------------------------------------------------/
function GuiSwatchButtonCtrl::onControlDropped( %this, %payload, %position )
{
   // Make sure this is a color swatch drag operation.if( !%payload.parentGroup.isInNamespaceHie
      return;

   // If dropped on same button whence we came from,// do nothing.if( %payload.dragSourceControl
      return;

   // If a swatch button control is dropped onto this control,// copy its color.if( %payload.isM
   {
      // If the swatch button is part of a color-type inspector field,// remember the inspector
         %this.parentGroup.apply( ColorFloatToInt( %payload.color ) );
      elseif( %this.parentGroup.isMemberOfClass( "GuiInspectorTypeColorF" ) )
         %this.parentGroup.apply( %payload.color );
      else
         %this.setColor( %payload.color );
   }
}
```

Methods

void GuiDragAndDropControl::**startDragging**(int *x*, int *y*)

Start the drag operation.

Parameters

- **x** – X coordinate for the mouse pointer offset which the drag control should position itself.
- **y** – Y coordinate for the mouse pointer offset which the drag control should position itself.

Fields

bool GuiDragAndDropControl::**deleteOnMouseUp**

If true, the control deletes itself when the left mouse button is released. If at this point, the drag amp drop control still contains its payload, it will be deleted along with the control.

GuiInputCtrl A control that locks the mouse and reports all keyboard input events to script.

Inherit: GuiMouseEventCtrl

Description This is useful for implementing custom keyboard handling code, and most commonly used in Torque for a menu that allows a user to remap their in-game controls

Example:

```
newGuiInputCtrl(OptRemapInputCtrl)
{
   lockMouse = "0";
   position = "0 0";
   extent = "64 64";
   minExtent = "8 8";
   horizSizing = "center";
   vertSizing = "bottom";
   profile = "GuiInputCtrlProfile";
   visible = "1";
```

```
   active = "1";
   tooltipProfile = "GuiToolTipProfile";
   hovertime = "1000";
   isContainer = "0";
   canSave = "1";
   canSaveDynamicFields = "0";
};
```

Methods

void `GuiInputCtrl::onInputEvent` (string *device*, string *action*, bool *state*)

Callback that occurs when an input is triggered on this control.

Parameters

- **`device`** – The device type triggering the input, such as keyboard, mouse, etc
- **`action`** – The actual event occuring, such as a key or button
- **`state`** – True if the action is being pressed, false if it is being release

GuiMessageVectorCtrl A chat HUD control that displays messages from a MessageVector.

Inherit: GuiControl

Description A chat HUD control that displays messages from a MessageVector.

This renders messages from a MessageVector; the important thing here is that the MessageVector holds all the messages we care about, while we can destroy and create these GUI controls as needed.

Example:

```
// Declare ChatHud, which is what will display the actual chat from a MessageVectornewGuiMessage
   profile = "ChatHudMessageProfile";
   horizSizing = "width";
   vertSizing = "height";
   position = "1 1";
   extent = "252 16";
   minExtent = "8 8";
   visible = "1";
   helpTag = "0";
   lineSpacing = "0";
   lineContinuedIndex = "10";
   matchColor = "0 0 255 255";
   maxColorIndex = "5";
};

// All messages are stored in this HudMessageVector, the actual// MainChatHud only displays the

// Attach the MessageVector to the chat control
chatHud.attach(HudMessageVector);
```

Methods

bool `GuiMessageVectorCtrl::attach` (MessageVector *item*)

Push a line onto the back of the list.

Parameters **`item`** – The GUI element being pushed into the control

Returns Value

Example:

```
// All messages are stored in this HudMessageVector, the actual
// MainChatHud only displays the contents of this vector.
newMessageVector(HudMessageVector);

// Attach the MessageVector to the chat control
chatHud.attach(HudMessageVector);
```

void GuiMessageVectorCtrl::**detach**()
Stop listing messages from the MessageVector previously attached to, if any. Detailed description

Parameters **param** – Description

Example:

```
// Deatch the MessageVector from HudMessageVector
// HudMessageVector will no longer render the text
chatHud.detach();
```

Fields
string GuiMessageVectorCtrl::**allowedMatches**[16]
int GuiMessageVectorCtrl::**lineContinuedIndex**

int GuiMessageVectorCtrl::**lineSpacing**

ColorI GuiMessageVectorCtrl::**matchColor**

int GuiMessageVectorCtrl::**maxColorIndex**

GuiScriptNotifyCtrl A control which adds several reactions to other GUIs via callbacks.

Inherit: GuiControl

Description GuiScriptNotifyCtrl does not exist to render anything. When parented or made a child of other controls, you can toggle flags on or off to make use of its specialized callbacks. Normally these callbacks are used as utility functions by the GUI Editor, or other container classes. However, for very fancy GUI work where controls interact with each other constantly, this is a handy utility to make use of.

Example:

```
// Common member fields left out for sake of examplenewGuiScriptNotifyCtrl()
{
   onChildAdded = "0";
   onChildRemoved = "0";
   onChildResized = "0";
   onParentResized = "0";
};
```

Methods
void GuiScriptNotifyCtrl::**onChildAdded**(SimObjectId *ID*, SimObjectId *childID*)
Called when a child is added to this GUI.

Parameters

- **ID** – Unique object ID assigned when created (this in script).

- **childID** – Unique object ID of child being added.

void GuiScriptNotifyCtrl::**onChildRemoved**(SimObjectId *ID*, SimObjectId *childID*)
Called when a child is removed from this GUI.

Parameters

- **ID** – Unique object ID assigned when created (this in script).
- **childID** – Unique object ID of child being removed.

void GuiScriptNotifyCtrl::**onChildResized**(SimObjectId *ID*, SimObjectId *childID*)
Called when a child is of this GUI is being resized.

Parameters

- **ID** – Unique object ID assigned when created (this in script).
- **childID** – Unique object ID of child being resized.

void GuiScriptNotifyCtrl::**onGainFirstResponder**(SimObjectId *ID*)
Called when this GUI gains focus.

Parameters **ID** – Unique object ID assigned when created (this in script).

void GuiScriptNotifyCtrl::**onLoseFirstResponder**(SimObjectId *ID*)
Called when this GUI loses focus.

Parameters **ID** – Unique object ID assigned when created (this in script).

void GuiScriptNotifyCtrl::**onParentResized**(SimObjectId *ID*)
Called when this GUI's parent is resized.

Parameters **ID** – Unique object ID assigned when created (this in script).

void GuiScriptNotifyCtrl::**onResize**(SimObjectId *ID*)
Called when this GUI is resized.

Parameters **ID** – Unique object ID assigned when created (this in script).

Fields

bool GuiScriptNotifyCtrl::**onChildAdded**
Enables/disables onChildAdded callback.

bool GuiScriptNotifyCtrl::**onChildRemoved**
Enables/disables onChildRemoved callback.

bool GuiScriptNotifyCtrl::**onChildResized**
Enables/disables onChildResized callback.

bool GuiScriptNotifyCtrl::**onGainFirstResponder**
Enables/disables onGainFirstResponder callback.

bool GuiScriptNotifyCtrl::**onLoseFirstResponder**
Enables/disables onLoseFirstResponder callback.

bool GuiScriptNotifyCtrl::**onParentResized**
Enables/disables onParentResized callback.

bool GuiScriptNotifyCtrl::**onResize**
Enables/disables onResize callback.

GuiTickCtrl Brief Description.

Inherit: GuiControl

Description This Gui Control is designed to be subclassed to let people create controls which want to receive update ticks at a constant interval. This class was created to be the Parent class of a control which used a DynamicTexture along with a VectorField to create warping effects much like the ones found in visualization displays for iTunes or Winamp. Those displays are updated at the framerate frequency. This works fine for those effects, however for an application of the same type of effects for things like Gui transitions the framerate-driven update frequency is not desirable because it does not allow the developer to be able to have any idea of a consistent user-experience.

Enter the ITickable interface. This lets the Gui control, in this case, update the dynamic texture at a constant rate of once per tick, even though it gets rendered every frame, thus creating a framerate-independent update frequency so that the effects are at a consistent speed regardless of the specifics of the system the user is on. This means that the screen-transitions will occur in the same time on a machine getting 300fps in the Gui shell as a machine which gets 150fps in the Gui shell.

Methods

void `GuiTickCtrl::setProcessTicks`(bool *tick*)

This will set this object to either be processing ticks or not. This will set this object to either be processing ticks or not.

Parameters `tick` – (optional) True or nothing to enable ticking, false otherwise.

Example:

```
// Turn off ticking for a control, like a MenuBar (declared previously)
%sampleMenuBar.setProcessTicks(false);
```

MessageVector Store a list of chat messages.

Inherit: SimObject

Description This is responsible for managing messages which appear in the chat HUD, not the actual control rendered to the screen

Example:

```
// Declare ChatHud, which is what will display the actual chat from a MessageVectornewGuiMessage
   profile = "ChatHudMessageProfile";
   horizSizing = "width";
   vertSizing = "height";
   position = "1 1";
   extent = "252 16";
   minExtent = "8 8";
   visible = "1";
   helpTag = "0";
   lineSpacing = "0";
   lineContinuedIndex = "10";
   matchColor = "0 0 255 255";
   maxColorIndex = "5";
};

// All messages are stored in this HudMessageVector, the actual// MainChatHud only displays the

// Attach the MessageVector to the chat control
chatHud.attach(HudMessageVector);
```

Methods

void MessageVector::**clear**()

Clear all messages in the vector.

Example:

```
HudMessageVector.clear();
```

bool MessageVector::**deleteLine**(int *deletePos*)

Delete the line at the specified position.

Parameters **deletePos** – Position in the vector containing the line to be deleted

Returns False if deletePos is greater than the number of lines in the current vector

Example:

```
// Delete the first line (index 0) in the vector...
HudMessageVector.deleteLine(0);
```

void MessageVector::**dump**(string *filename*)

Dump the message vector to a file without a header.

Parameters **filename** – Name and path of file to dump text to.

Example:

```
// Dump the entire chat log to a text file
HudMessageVector.dump("./chatLog.txt");
```

void MessageVector::**dump**(string *filename*, string *header*)

Dump the message vector to a file with a header.

Parameters

- **filename** – Name and path of file to dump text to.
- **header** – Prefix information for write out

Example:

```
// Arbitrary header data
%headerInfo = "Ars Moriendi Chat Log";

// Dump the entire chat log to a text file
HudMessageVector.dump("./chatLog.txt", %headerInfo);
```

int MessageVector::**getLineIndexByTag**(int *tag*)

Scan through the vector, returning the line number of the first line that matches the specified tag; else returns -1 if no match was found.

Parameters **tag** – Numerical value assigned to a message when it was added or inserted

Returns Line with matching tag, other wise -1

Example:

```
// Locate a line of text tagged with the value "1", then delete it.
%taggedLine = HudMessageVector.getLineIndexByTag(1);
HudMessageVector.deleteLine(%taggedLine);
```

int MessageVector::**getLineTag**(int *pos*)

Get the tag of a specified line.

Parameters **pos** – Position in vector to grab tag from

Returns Tag value of a given line, if the position is greater than the number of lines return 0

Example:

```
// Remove all lines that do not have a tag value of 1.while( HudMessageVector.getNumLines())
{
   %tag = HudMessageVector.getLineTag(1);
   if(%tag != 1)
      %tag.delete();
   HudMessageVector.popFrontLine();
}
```

string MessageVector::**getLineText** (int *pos*)

Get the text at a specified line.

Parameters **pos** – Position in vector to grab text from

Returns Text at specified line, if the position is greater than the number of lines return “”

Example:

```
// Print a line of text at position 1.
%text = HudMessageVector.getLineText(1);
echo(%text);
```

string MessageVector::**getLineTextByTag** (int *tag*)

Scan through the lines in the vector, returning the first line that has a matching tag.

Parameters **tag** – Numerical value assigned to a message when it was added or inserted

Returns Text from a line with matching tag, other wise “”

Example:

```
// Locate text in the vector tagged with the value "1", then print it
%taggedText = HudMessageVector.getLineTextByTag(1);
echo(%taggedText);
```

int MessageVector::**getNumLines** ()

Get the number of lines in the vector.

Example:

```
// Find out how many lines have been stored in HudMessageVector
%chatLines = HudMessageVector.getNumLines();
echo(%chatLines);
```

bool MessageVector::**insertLine** (int *insertPos*, string *msg*, int *tag*)

Push a line onto the back of the list.

Parameters

- **msg** – Text that makes up the message
- **tag** – Numerical value associated with this message, useful for searching.

Returns False if insertPos is greater than the number of lines in the current vector

Example:

```
// Add the message...
HudMessageVector.insertLine(1, "Hello World", 0);
```

bool MessageVector::**popBackLine**()
Pop a line from the back of the list; destroys the line.

Returns False if there are no lines to pop (underflow), true otherwise

Example:

```
HudMessageVector.popBackLine();
```

bool MessageVector::**popFrontLine**()
Pop a line from the front of the vector, destroying the line.

Returns False if there are no lines to pop (underflow), true otherwise

Example:

```
HudMessageVector.popFrontLine();
```

void MessageVector::**pushBackLine**(string *msg*, int *tag*)
Push a line onto the back of the list.

Parameters

- **msg** – Text that makes up the message
- **tag** – Numerical value associated with this message, useful for searching.

Example:

```
// Add the message...
HudMessageVector.pushBackLine("Hello World", 0);
```

void MessageVector::**pushFrontLine**(string *msg*, int *tag*)
Push a line onto the front of the vector.

Parameters

- **msg** – Text that makes up the message
- **tag** – Numerical value associated with this message, useful for searching.

Example:

```
// Add the message...
HudMessageVector.pushFrontLine("Hello World", 0);
```

Game Controls

GUI controls dedicated to game play systems, such as heads up displays.

Classes

GuiClockHud Basic HUD clock. Displays the current simulation time offset from some base.

Inherit: GuiControl

Description Basic HUD clock. Displays the current simulation time offset from some base.

Example:

```
newGuiClockHud(){
   fillColor = "0.0 1.0 0.0 1.0"; // Fills with a solid green colorframeColor = "1.0 1.0 1.0 1.0"; /
   showFrame = "true";
};
```

Methods

float GuiClockHud::**getTime**()

Returns the current time, in seconds.

Returns timeInseconds Current time, in seconds

Example:

```
// Get the current time from the GuiClockHud control
%timeInSeconds = %guiClockHud.getTime();
```

void GuiClockHud::**setReverseTime**(float *timeInSeconds*)

Sets a time for a countdown clock. Setting the time like this will cause the clock to count backwards from the specified time.

Parameters **timeInSeconds** – Time to set the clock, in seconds (IE: 00:02 would be 120)

void GuiClockHud::**setTime**(float *timeInSeconds*)

Sets the current base time for the clock.

Parameters **timeInSeconds** – Time to set the clock, in seconds (IE: 00:02 would be 120)

Example:

```
// Define the time, in seconds
%timeInSeconds = 120;

// Change the time on the GuiClockHud control
%guiClockHud.setTime(%timeInSeconds);
```

Fields

ColorF GuiClockHud::**fillColor**

Standard color for the background of the control.

ColorF GuiClockHud::**frameColor**

Color for the control's frame.

bool GuiClockHud::**showFill**

If true, draws a background color behind the control.

bool GuiClockHud::**showFrame**

If true, draws a frame around the control.

ColorF GuiClockHud::**textColor**

Color for the text on this control.

GuiCrossHairHud Basic cross hair hud. Reacts to state of control object. Also displays health bar for named objects under the cross hair.

Inherit: GuiBitmapCtrl

Description Basic cross hair hud. Reacts to state of control object. Also displays health bar for named objects under the cross hair.

Uses the base bitmap control to render a bitmap, and decides whether to draw or not depending on the current control object and it's state. If there is ShapeBase object under the cross hair and it's named, then a small health bar is displayed.

Example:

```
newGuiCrossHairHud(){
   damageFillColor = "1.0 0.0 0.0 1.0"; // Fills with a solid red colordamageFrameColor = "1.0 1.0 1.
   damageOffset = "0 -10";
};
```

Fields

ColorF `GuiCrossHairHud::`**`damageFillColor`**
: As the health bar depletes, this color will represent the health loss amount.

ColorF `GuiCrossHairHud::`**`damageFrameColor`**
: Color for the health bar's frame.

Point2I `GuiCrossHairHud::`**`damageOffset`**
: Offset for drawing the damage portion of the health control.

Point2I `GuiCrossHairHud::`**`damageRect`**
: Size for the health bar portion of the control.

GuiGameListMenuCtrl A base class for cross platform menu controls that are gamepad friendly.

Inherit: GuiControl

Description A base class for cross platform menu controls that are gamepad friendly.

This class is used to build row-based menu GUIs that can be easily navigated using the keyboard, mouse or gamepad. The desired row can be selected using the mouse, or by navigating using the Up and Down buttons.

Example:

```
newGuiGameListMenuCtrl()
{
   debugRender = "0";
   callbackOnA = "applyOptions();";
   callbackOnB = "Canvas.setContent(MainMenuGui);";
   callbackOnX = "";
   callbackOnY = "revertOptions();";
   //Properties not specific to this control have been omitted from this example.
};
```

Methods

void `GuiGameListMenuCtrl::`**`activateRow`**`()`
: Activates the current row. The script callback of the current row will be called (if it has one).

void `GuiGameListMenuCtrl::`**`addRow`**`(`string *label*, string *callback*, int *icon*, int *yPad*, bool *useHighlightIcon*, bool *enabled*`)`
: Add a row to the list control.

Parameters

- **`label`** – The text to display on the row as a label.

- **callback** – Name of a script function to use as a callback when this row is activated.
- **icon** – [optional] Index of the icon to use as a marker.
- **yPad** – [optional] An extra amount of height padding before the row. Does nothing on the first row.
- **useHighlightIcon** – [optional] Does this row use the highlight icon?.
- **enabled** – [optional] If this row is initially enabled.

int GuiGameListMenuCtrl::**getRowCount**()
Gets the number of rows on the control.

Returns (int) The number of rows on the control.

string GuiGameListMenuCtrl::**getRowLabel**(int *row*)
Gets the label displayed on the specified row.

Parameters **row** – Index of the row to get the label of.

Returns The label for the row.

int GuiGameListMenuCtrl::**getSelectedRow**()
Gets the index of the currently selected row.

Returns Index of the selected row.

bool GuiGameListMenuCtrl::**isRowEnabled**(int *row*)
Determines if the specified row is enabled or disabled.

Parameters **row** – The row to set the enabled status of.

Returns True if the specified row is enabled. False if the row is not enabled or the given index was not valid.

void GuiGameListMenuCtrl::**onChange**()
Called when the selected row changes.

void GuiGameListMenuCtrl::**setRowEnabled**(int *row*, bool *enabled*)
Sets a row's enabled status according to the given parameters.

Parameters

- **row** – The index to check for validity.
- **enabled** – Indicate true to enable the row or false to disable it.

void GuiGameListMenuCtrl::**setRowLabel**(int *row*, string *label*)
Sets the label on the given row.

Parameters

- **row** – Index of the row to set the label on.
- **label** – Text to set as the label of the row.

void GuiGameListMenuCtrl::**setSelected**(int *row*)
Sets the selected row. Only rows that are enabled can be selected.

Parameters **row** – Index of the row to set as selected.

Fields

string GuiGameListMenuCtrl::**callbackOnA**
Script callback when the 'A' button is pressed. 'A' inputs are Keyboard: A, Return, Space; Gamepad: A, Start.

string GuiGameListMenuCtrl::**callbackOnB**
Script callback when the 'B' button is pressed. 'B' inputs are Keyboard: B, Esc, Backspace, Delete; Gamepad: B, Back.

string GuiGameListMenuCtrl::**callbackOnX**
Script callback when the 'X' button is pressed. 'X' inputs are Keyboard: X; Gamepad: X.

string GuiGameListMenuCtrl::**callbackOnY**
Script callback when the 'Y' button is pressed. 'Y' inputs are Keyboard: Y; Gamepad: Y.

bool GuiGameListMenuCtrl::**debugRender**
Enable debug rendering.

GuiGameListMenuProfile A GuiControlProfile with additional fields specific to GuiGameListMenuCtrl.

Inherit: GuiControlProfile

Description A GuiControlProfile with additional fields specific to GuiGameListMenuCtrl.

Example:

```
newGuiGameListMenuProfile()
{
   hitAreaUpperLeft = "10 2";
   hitAreaLowerRight = "190 18";
   iconOffset = "10 2";
   rowSize = "200 20";
   //Properties not specific to this control have been omitted from this example.
};
```

Fields

Point2I GuiGameListMenuProfile::**hitAreaLowerRight**
Position of the lower right corner of the row hit area (relative to row's top left corner).

Point2I GuiGameListMenuProfile::**hitAreaUpperLeft**
Position of the upper left corner of the row hit area (relative to row's top left corner).

Point2I GuiGameListMenuProfile::**iconOffset**
Offset from the row's top left corner at which to render the row icon.

Point2I GuiGameListMenuProfile::**rowSize**
The base size ("width height") of a row.

GuiGameListOptionsCtrl A control for showing pages of options that are gamepad friendly.

Inherit: GuiGameListMenuCtrl

Description Each row in this control allows the selection of one value from a set of options using the keyboard, gamepad or mouse. The row is rendered as 2 columns: the first column contains the row label, the second column contains left and right arrows (for mouse picking) and the currently selected value.

Methods

void GuiGameListOptionsCtrl::**addRow**(string *label*, string *options*, bool *wrapOptions*, string *callback*, int *icon*, int *yPad*, bool *enabled*)
Add a row to the list control.

Parameters

- **label** – The text to display on the row as a label.
- **options** – A tab separated list of options.
- **wrapOptions** – Specify true to allow options to wrap at each end or false to prevent wrapping.
- **callback** – Name of a script function to use as a callback when this row is activated.
- **icon** – [optional] Index of the icon to use as a marker.
- **yPad** – [optional] An extra amount of height padding before the row. Does nothing on the first row.
- **enabled** – [optional] If this row is initially enabled.

string GuiGameListOptionsCtrl::**getCurrentOption**(int *row*)
Gets the text for the currently selected option of the given row.

Parameters **row** – Index of the row to get the option from.

Returns A string representing the text currently displayed as the selected option on the given row. If there is no such displayed text then the empty string is returned.

bool GuiGameListOptionsCtrl::**selectOption**(int *row*, string *option*)
Set the row's current option to the one specified.

Parameters

- **row** – Index of the row to set an option on.
- **option** – The option to be made active.

Returns True if the row contained the option and was set, false otherwise.

void GuiGameListOptionsCtrl::**setOptions**(int *row*, string *optionsList*)
Sets the list of options on the given row.

Parameters

- **row** – Index of the row to set options on.
- **optionsList** – A tab separated list of options for the control.

GuiGameListOptionsProfile A GuiControlProfile with additional fields specific to GuiGameListOptionsCtrl.

Inherit: GuiGameListMenuProfile

Description A GuiControlProfile with additional fields specific to GuiGameListOptionsCtrl.

Example:

```
newGuiGameListOptionsProfile()
{
   columnSplit = "100";
   rightPad = "4";
   //Properties not specific to this control have been omitted from this example.
};
```

Fields

int GuiGameListOptionsProfile::**columnSplit**
Padding between the leftmost edge of the control, and the row's left arrow.

int GuiGameListOptionsProfile::**rightPad**
Padding between the rightmost edge of the control and the row's right arrow.

GuiHealthBarHud A basic health bar. Shows the damage value of the current PlayerObjectType control object.

Inherit: GuiControl

Description A basic health bar. Shows the damage value of the current PlayerObjectType control object.

This gui displays the damage value of the current PlayerObjectType control object. The gui can be set to pulse if the health value drops below a set value. This control only works if a server connection exists and it's control object is a PlayerObjectType. If either of these requirements is false, the control is not rendered.

Example:

```
newGuiHealthBarHud(){
   fillColor = "0.0 1.0 0.0 1.0"; // Fills with a solid green colorframeColor = "1.0 1.0 1.0 1.0"; /
   pulseThreshold = "0.25";
   showFill = "true";
   showFrame = "true";
   displayEnergy = "false";
};
```

Fields

ColorF GuiHealthBarHud::**damageFillColor**
As the health bar depletes, this color will represent the health loss amount.

bool GuiHealthBarHud::**displayEnergy**
If true, display the energy value rather than the damage value.

ColorF GuiHealthBarHud::**fillColor**
Standard color for the background of the control.

ColorF GuiHealthBarHud::**frameColor**
Color for the control's frame.

int GuiHealthBarHud::**pulseRate**
Speed at which the control will pulse.

float GuiHealthBarHud::**pulseThreshold**
Health level the control must be under before the control will pulse.

bool GuiHealthBarHud::**showFill**
If true, we draw the background color of the control.

bool GuiHealthBarHud::**showFrame**
If true, we draw the frame of the control.

GuiHealthTextHud Shows the health or energy value of the current PlayerObjectType control object.

Inherit: GuiControl

Description Shows the health or energy value of the current PlayerObjectType control object.

This gui can be configured to display either the health or energy value of the current Player Object. It can use an alternate display color if the health or drops below a set value. It can also be set to pulse if the health or energy drops below a set value. This control only works if a server connection exists and it's control object is a PlayerObjectType. If either of these requirements is false, the control is not rendered.

Example:

```
newGuiHealthTextHud(){
   fillColor = "0.0 0.0 0.0 0.5"; // Fills with a transparent black colorframeColor = "1.0 1.0 1.0 1.
   showFrame = "true";
   showTrueValue = "false";
   showEnergy = "false";
   warnThreshold = "50";
   pulseThreshold = "25";
   pulseRate = "500";
   profile = "GuiBigTextProfile";
};
```

Fields

ColorF `GuiHealthTextHud::`**`fillColor`**
: Color for the background of the control.

ColorF `GuiHealthTextHud::`**`frameColor`**
: Color for the control's frame.

int `GuiHealthTextHud::`**`pulseRate`**
: Speed at which the control will pulse.

float `GuiHealthTextHud::`**`pulseThreshold`**
: Health level at which to begin pulsing.

bool `GuiHealthTextHud::`**`showEnergy`**
: If true, display the energy value rather than the damage value.

bool `GuiHealthTextHud::`**`showFill`**
: If true, draw the background.

bool `GuiHealthTextHud::`**`showFrame`**
: If true, draw the frame.

bool `GuiHealthTextHud::`**`showTrueValue`**
: If true, we don't hardcode maxHealth to 100.

ColorF `GuiHealthTextHud::`**`textColor`**
: Color for the text on this control.

ColorF `GuiHealthTextHud::`**`warningColor`**
: Color for the text when health is low.

float `GuiHealthTextHud::`**`warnThreshold`**
: The health level at which to use the warningColor.

GuiShapeNameHud Displays name and damage of ShapeBase objects in its bounds. Must be a child of a GuiTSCtrl and a server connection must be present.

Inherit: GuiControl

Description This control displays the name and damage value of all named ShapeBase objects on the client. The name and damage of objects within the control's display area are overlayed above the object.

This GUI control must be a child of a TSControl, and a server connection and control object must be present. This is a stand-alone control and relies only on the standard base GuiControl.

Example:

```
newGuiShapeNameHud(){
   fillColor = "0.0 1.0 0.0 1.0"; // Fills with a solid green colorframeColor = "1.0 1.0 1.0 1.0"; /
   showFrame = "true";
   labelFillColor = "0.0 1.0 0.0 1.0"; // Fills with a solid green colorlabelFrameColor = "1.0 1.0 1
   showLabelFrame = "true";
   verticalOffset = "0.15";
   distanceFade = "15.0";
};
```

Fields

float GuiShapeNameHud::**distanceFade**
Visibility distance (how far the player must be from the ShapeBase object in focus) for this control to render.

ColorF GuiShapeNameHud::**fillColor**
Standard color for the background of the control.

ColorF GuiShapeNameHud::**frameColor**
Color for the control's frame.

ColorF GuiShapeNameHud::**labelFillColor**
Color for the background of each shape name label.

ColorF GuiShapeNameHud::**labelFrameColor**
Color for the frames around each shape name label.

Point2I GuiShapeNameHud::**labelPadding**
The padding (in pixels) between the label text and the frame.

bool GuiShapeNameHud::**showFill**
If true, we draw the background color of the control.

bool GuiShapeNameHud::**showFrame**
If true, we draw the frame of the control.

bool GuiShapeNameHud::**showLabelFill**
If true, we draw a background for each shape name label.

bool GuiShapeNameHud::**showLabelFrame**
If true, we draw a frame around each shape name label.

ColorF GuiShapeNameHud::**textColor**
Color for the text on this control.

float GuiShapeNameHud::**verticalOffset**
Amount to vertically offset the control in relation to the ShapeBase object in focus.

Functions

void **snapToggle**()
Prevents mouse movement from being processed. In the source, whenever a mouse move event occurs GameTSCtrl::onMouseMove() is called. Whenever snapToggle() is called, it will flag a variable that can prevent

this from happening: gSnapLine. This variable is not exposed to script, so you need to call this function to trigger it.

Example:

```
// Snapping is off by default, so we will toggle
// it on first:
PlayGui.snapToggle();

// Mouse movement should be disabled
// Lets turn it back on
PlayGui.snapToggle();
```

5.3.3 Game

Objects, functions, and variables related to game play elements.

Game

Classes

SimObject Base class for almost all objects involved in the simulation.

Description SimObject is a base class for most of the classes you'll encounter working in Torque. It provides fundamental services allowing "smart" object referencing, creation, destruction, organization, and location. Along with SimEvent, it gives you a flexible event-scheduling system, as well as laying the foundation for the in-game editors, GUI system, and other vital subsystems.

Subclassing You will spend a lot of your time in Torque subclassing, or working with subclasses of, SimObject. SimObject is designed to be easy to subclass.

You should not need to override anything in a subclass except:

- the constructor/destructor
- processArguments()
- onAdd()/onRemove()
- onGroupAdd()/onGroupRemove()
- onNameChange()
- onStaticModified()
- onDeleteNotify()
- onEditorEnable()/onEditorDisable()
- inspectPreApply()/inspectPostApply()
- things from ConsoleObject (see ConsoleObject docs for specifics)

Of course, if you know what you're doing, go nuts! But in most cases, you shouldn't need to touch things not on that list.

When you subclass, you should define a typedef in the class, called Parent, that references the class you're inheriting from:

```
class mySubClass : public SimObject {
    typedef SimObject Parent;
    ...
```

Then, when you override a method, put in:

```
bool mySubClass::onAdd()
{
    if(!Parent::onAdd())
        return false;

    // ... do other things ...
}
```

Of course, you want to replace onAdd with the appropriate method call.

A SimObject's Life Cycle SimObjects do not live apart. One of the primary benefits of using a SimObject is that you can uniquely identify it and easily find it (using its ID). Torque does this by keeping a global hierarchy of SimGroups - a tree - containing every registered SimObject. You can then query for a given object using Sim::findObject() (or SimSet::findObject() if you want to search only a specific set):

```
// Three examples of registering an object.

// Method 1:
AIClient *aiPlayer = new AIClient();
aiPlayer->registerObject();

// Method 2:
ActionMap* globalMap = new ActionMap;
globalMap->registerObject("GlobalActionMap");

// Method 3:
bool reg = mObj->registerObject(id);
```

Registering a SimObject performs these tasks:

- Marks the object as not cleared and not removed.
- Assigns the object a unique SimObjectID if it does not have one already.
- Adds the object to the global name and ID dictionaries so it can be found again.
- Calls the object's onAdd() method. Note: SimObject::onAdd() performs some important initialization steps. See here for details" on how to properly subclass SimObject.
- If onAdd() fails (returns false), it calls unregisterObject().
- Checks to make sure that the SimObject was properly initialized (and asserts if not).

Calling registerObject() and passing an ID or a name will cause the object to be assigned that name and/or ID before it is registered.

Congratulations, you have now registered your object! What now?

Well, hopefully, the SimObject will have a long, useful life. But eventually, it must die.

There are a two ways a SimObject can die.

- First, the game can be shut down. This causes the root SimGroup to be unregistered and deleted. When a SimGroup is unregistered, it unregisters all of its member SimObjects; this results in everything that has been registered with Sim being unregistered, as everything registered with Sim is in the root group.

- Second, you can manually kill it off, either by calling unregisterObject() or by calling deleteObject().

When you unregister a SimObject, the following tasks are performed:

- The object is flagged as removed.
- Notifications are cleaned up.
- If the object is in a group, then it removes itself from the group.
- Delete notifications are sent out.
- Finally, the object removes itself from the Sim globals, and tells Sim to get rid of any pending events for it.

If you call deleteObject(), all of the above tasks are performed, in addition to some sanity checking to make sure the object was previously added properly, and isn't in the process of being deleted. After the object is unregistered, it deallocates itself.

Torque Editors SimObjects are one of the building blocks for the in-game editors. They provide a basic interface for the editor to be able to list the fields of the object, update them safely and reliably, and inform the object things have changed.

This interface is implemented in the following areas:

- onNameChange() is called when the object is renamed.
- onStaticModified() is called whenever a static field is modified.
- inspectPreApply() is called before the object's fields are updated, when changes are being applied.
- inspectPostApply() is called after the object's fields are updated.
- onEditorEnable() is called whenever an editor is enabled (for instance, when you hit F11 to bring up the world editor).
- onEditorDisable() is called whenever the editor is disabled (for instance, when you hit F11 again to close the world editor).

(Note: you can check the variable gEditingMission to see if the mission editor is running; if so, you may want to render special indicators. For instance, the fxFoliageReplicator renders inner and outer radii when the mission editor is runnning.)

The Console SimObject extends ConsoleObject by allowing you to to set arbitrary dynamic fields on the object, as well as statically defined fields. This is done through two methods, setDataField and getDataField, which deal with the complexities of allowing access to two different types of object fields.

Static fields take priority over dynamic fields. This is to be expected, as the role of dynamic fields is to allow data to be stored in addition to the predefined fields.

The fields in a SimObject are like properties (or fields) in a class.

Some fields may be arrays, which is what the array parameter is for; if it's non-null, then it is parsed with dAtoI and used as an index into the array. If you access something as an array which isn't, then you get an empty string.

You don't need to read any further than this. Right now, set/getDataField are called a total of 6 times through the entire Torque codebase. Therefore, you probably don't need to be familiar with the details of accessing them. You may want to look at Con::setData instead. Most of the time you will probably be accessing fields directly, or using the scripting language, which in either case means you don't need to do anything special.

The functions to get/set these fields are very straightforward:

```
setDataField(StringTable->insert("locked", false), NULL, b ? "true" : "false" );
curObject->setDataField(curField, curFieldArray, STR.getStringValue());
setDataField(slotName, array, value);
```

For advanced users: There are two flags which control the behavior of these functions. The first is ModStaticFields, which controls whether or not the DataField functions look through the static fields (defined with addField; see ConsoleObject for details) of the class. The second is ModDynamicFields, which controls dynamically defined fields. They are set automatically by the console constructor code.

Methods

void SimObject::**assignFieldsFrom**(SimObject *fromObject*)
Copy fields from another object onto this one. The objects must be of same type. Everything from the object will overwrite what's in this object; extra fields in this object will remain. This includes dynamic fields.

Parameters **fromObject** – The object from which to copy fields.

void SimObject::**assignPersistentId**()
Assign a persistent ID to the object if it does not already have one.

string SimObject::**call**(string *method*, string *args*, ...)
Dynamically call a method on an object.

Parameters

- **method** – Name of method to call.
- **args** – Zero or more arguments for the method.

Returns The result of the method call.

SimObject SimObject::**clone**()
Create a copy of this object.

Returns An exact duplicate of this object.

SimObject SimObject::**deepClone**()
Create a copy of this object and all its subobjects.

Returns An exact duplicate of this object and all objects it references.

void SimObject::delete()
Delete and remove the object.

void SimObject::**dump**(bool *detailed*)
Dump a description of all fields and methods defined on this object to the console.

Parameters **detailed** – Whether to print detailed information about members.

void SimObject::**dumpClassHierarchy**()
Dump the native C++ class hierarchy of this object's C++ class to the console.

void SimObject::**dumpGroupHierarchy**()
Dump the hierarchy of this object up to RootGroup to the console.

ArrayObject SimObject::**dumpMethods**()
List the methods defined on this object. Each description is a newline-separated vector with the following elements:

•Minimum number of arguments.

•Maximum number of arguments.

•Prototype string.

•Full script file path (if script method).

•Line number of method definition in script (if script method).

•Documentation string (not including prototype). This takes up the remainder of the vector.

Returns populated with (name,description) pairs of all methods defined on the object.

bool SimObject::**getCanSave**()
Get whether the object will be included in saves.

Returns True if the object will be saved; false otherwise.

string SimObject::**getClassName**()
Get the name of the C++ class which the object is an instance of.

Returns The name of the C++ class of the object.

string SimObject::**getClassNamespace**()
Get the name of the class namespace assigned to this object.

Returns The name of the 'class' namespace.

ArrayObject SimObject::**getDebugInfo**()
Return some behind-the-scenes information on the object.

Returns filled with internal information about the object.

int SimObject::**getDeclarationLine**()
Get the line number at which the object is defined in its file.

Returns The line number of the object's definition in script.

string SimObject::**getDynamicField**(int *index*)
Get a value of a dynamic field by index.

Parameters **index** – The index of the dynamic field.

Returns The value of the dynamic field at the given index or "".

int SimObject::**getDynamicFieldCount**()
Get the number of dynamic fields defined on the object.

Returns The number of dynamic fields defined on the object.

string SimObject::**getField**(int *index*)
Retrieve the value of a static field by index.

Parameters **index** – The index of the static field.

Returns The value of the static field with the given index or "".

int SimObject::**getFieldCount**()
Get the number of static fields on the object.

Returns The number of static fields defined on the object.

string SimObject::**getFieldType**(string *fieldName*)
Get the console type code of the given field.

Returns The numeric type code for the underlying console type of the given field.

string SimObject::**getFieldValue**(string *fieldName*, int *index*)
Return the value of the given field on this object.

Parameters

- **fieldName** – The name of the field. If it includes a field index, the index is parsed out.
- **index** – Optional parameter to specify the index of an array field separately.

Returns The value of the given field or "" if undefined.

string SimObject::**getFilename**()
Returns the filename the object is attached to. Reimplemented in CubemapData .

Returns The name of the file the object is associated with; usually the file the object was loaded from.

SimGroup SimObject::**getGroup**()
Get the group that this object is contained in.

Returns object to which the object belongs.

int SimObject::**getId**()
Get the underlying unique numeric ID of the object.

Returns The unique numeric ID of the object.

string SimObject::**getInternalName**()
Get the internal name of the object.

Returns The internal name of the object.

string SimObject::**getName**()
Get the global name of the object.

Returns The global name assigned to the object.

string SimObject::**getSuperClassNamespace**()
Get the name of the superclass namespace assigned to this object.

Returns The name of the 'superClass' namespace.

bool SimObject::**isChildOfGroup**(SimGroup *group*)
Test whether the object belongs directly or indirectly to the given group.

Parameters **group** – The SimGroup object.

Returns True if the object is a child of the given group or a child of a group that the given group is directly or indirectly a child to.

bool SimObject::**isEditorOnly**()
Return true if the object is only used by the editor.

Returns True if this object exists only for the sake of editing.

bool SimObject::**isExpanded**()
Get whether the object has been marked as expanded. (in editor). Reimplemented in GuiRolloutCtrl .

Returns True if the object is marked expanded.

bool SimObject::**isField**(string *fieldName*)
Test whether the given field is defined on this object.

Parameters **fieldName** – The name of the field.

Returns True if the object implements the given field.

bool SimObject::**isInNamespaceHierarchy**(string *name*)
Test whether the namespace of this object is a direct or indirect child to the given namespace.

Parameters **name** – The name of a namespace.

Returns True if the given namespace name is within the namespace hierarchy of this object.

bool SimObject::**isMemberOfClass**(string *className*)
Test whether this object is a member of the specified class.

Parameters **className** – Name of a native C++ class.

Returns True if this object is an instance of the given C++ class or any of its super classes.

bool SimObject::**isMethod**(string *methodName*)
Test whether the given method is defined on this object.

Parameters **The** – name of the method.

Returns True if the object implements the given method.

bool SimObject::**isNameChangeAllowed**()
Get whether this object may be renamed.

Returns True if this object can be renamed; false otherwise.

bool SimObject::**isSelected**()
Get whether the object has been marked as selected. (in editor).

Returns True if the object is currently selected.

bool SimObject::**save**(string *fileName*, bool *selectedOnly*, string *preAppendString*)
Save out the object to the given file.

Parameters

- **fileName** – The name of the file to save to.
- **selectedOnly** – If true, only objects marked as selected will be saved out.
- **preAppendString** – Text which will be preprended directly to the object serialization.
- **True** – on success, false on failure.

int SimObject::**schedule**(float *time*, string *method*, string *args*, ...)
Delay an invocation of a method.

Parameters

- **time** – The number of milliseconds after which to invoke the method. This is a soft limit.
- **method** – The method to call.
- **args** – The arguments with which to call the method.

Returns The numeric ID of the created schedule. Can be used to cancel the call.

void SimObject::**setCanSave**(bool *value*)
Set whether the object will be included in saves.

Parameters **value** – If true, the object will be included in saves; if false, it will be excluded.

void SimObject::**setClassNamespace**(string *name*)
Assign a class namespace to this object.

Parameters **name** – The name of the 'class' namespace for this object.

void SimObject::**setEditorOnly**(bool *value*)
Set/clear the editor-only flag on this object.

Parameters **value** – If true, the object is marked as existing only for the editor.

void SimObject::**setFieldType**(string *fieldName*, string *type*)
Set the console type code for the given field.

Parameters

- **fieldName** – The name of the dynamic field to change to type for.
- **type** – The name of the console type.

bool SimObject::**setFieldValue**(string *fieldName*, string *value*, int *index*)
Set the value of the given field on this object.

Parameters

- **fieldName** – The name of the field to assign to. If it includes an array index, the index will be parsed out.
- **value** – The new value to assign to the field.
- **index** – Optional argument to specify an index for an array field.

Returns True.

void SimObject::**setFilename**(string *fileName*)
Sets the object's file name and path.

Parameters **fileName** – The name of the file to associate this object with.

void SimObject::**setHidden**(bool *value*)
Hide/unhide the object. Reimplemented in ShapeBase .

Parameters **value** – If true, the object will be hidden; if false, the object will be unhidden.

void SimObject::**setInternalName**(string *newInternalName*)
Set the internal name of the object.

Parameters **newInternalName** – The new internal name for the object.

void SimObject::**setIsExpanded**(bool *state*)
Set whether the object has been marked as expanded. (in editor).

Parameters **state** – True if the object is to be marked expanded; false if not.

void SimObject::**setIsSelected**(bool *state*)
Set whether the object has been marked as selected. (in editor).

Parameters **state** – True if object is to be marked selected; false if not.

void SimObject::**setLocked**(bool *value*)
Lock/unlock the object in the editor.

Parameters **value** – If true, the object will be locked; if false, the object will be unlocked.

void SimObject::**setName**(string *newName*)
Set the global name of the object.

Parameters **newName** – The new global name to assign to the object.

void SimObject::**setNameChangeAllowed**(bool *value*)
Set whether this object can be renamed from its first name.

Parameters **value** – If true, renaming is allowed for this object; if false, trying to change the name of the object will generate a console error.

void SimObject::**setSuperClassNamespace**(string *name*)
Assign a superclass namespace to this object.

Parameters **name** – The name of the 'superClass' namespace for this object.

Fields

bool SimObject::**canSave**
Whether the object can be saved out. If false, the object is purely transient in nature.

bool SimObject::**canSaveDynamicFields**
True if dynamic fields (added at runtime) should be saved. Defaults to true.

string SimObject::class
Script class of object.

string SimObject::**className**
Script class of object.

bool SimObject::**hidden**
Whether the object is visible.

string SimObject::**internalName**
Optional name that may be used to lookup this object within a SimSet .

bool SimObject::**locked**
Whether the object can be edited.

string SimObject::**name**
Optional global name of this object.

SimObject SimObject::**parentGroup**
Group hierarchy parent of the object.

pid SimObject::**persistentId**
The universally unique identifier for the object.

string SimObject::**superClass**
Script super-class of object.

SimSet A collection of SimObjects.

Inherit: SimObject

Description It is often necessary to keep track of an arbitrary set of SimObjects. For instance, Torque's networking code needs to not only keep track of the set of objects which need to be ghosted, but also the set of objects which must always be ghosted. It does this by working with two sets. The first of these is the RootGroup (which is actually a SimGroup) and the second is the GhostAlwaysSet, which contains objects which must always be ghosted to the client.

Some general notes on SimSets:

- Membership is not exclusive. A SimObject may be a member of multiple SimSets.
- A SimSet does not destroy subobjects when it is destroyed.
- A SimSet may hold an arbitrary number of objects.

Methods

bool SimSet::**acceptsAsChild**(SimObject *obj*)
Test whether the given object may be added to the set.

Parameters **obj** – The object to test for potential membership.

Returns True if the object may be added to the set, false otherwise.

void SimSet::**add**(SimObject *objects*, ...)
Add the given objects to the set.

Parameters **objects** – The objects to add to the set.

void SimSet::**bringToFront**(SimObject *obj*)
Make the given object the first object in the set.

Parameters **obj** – The object to bring to the frontmost position. Must be contained in the set.

void SimSet::**callOnChildren**(string *method*, string *args*, ...)
Call a method on all objects contained in the set.

Parameters

- **method** – The name of the method to call.
- **args** – The arguments to the method.

void SimSet::**callOnChildrenNoRecurse**(string *method*, string *args*, ...)
Call a method on all objects contained in the set.

Parameters

- **method** – The name of the method to call.
- **args** – The arguments to the method.

void SimSet::**clear**()
Remove all objects from the set. Reimplemented in GuiPopUpMenuCtrlEx .

void SimSet::**deleteAllObjects**()
Delete all objects in the set.

SimObject SimSet::**findObjectByInternalName**(string *internalName*, bool *searchChildren*)
Find an object in the set by its internal name.

Parameters

- **internalName** – The internal name of the object to look for.
- **searchChildren** – If true, SimSets contained in the set will be recursively searched for the object.

Returns The object with the given internal name or 0 if no match was found.

int SimSet::**getCount**()
Get the number of objects contained in the set.

Returns The number of objects contained in the set.

int SimSet::**getFullCount**()
Get the number of direct and indirect child objects contained in the set.

Returns The number of objects contained in the set as well as in other sets contained directly or indirectly in the set.

SimObject SimSet::**getObject**(int *index*)
Get the object at the given index.

Parameters **index** – The object index.

Returns The object at the given index or -1 if index is out of range.

int SimSet::**getObjectIndex**(SimObject *obj*)
Return the index of the given object in this set.

Parameters **obj** – The object for which to return the index. Must be contained in the set.

Returns The index of the object or -1 if the object is not contained in the set.

SimObject SimSet::**getRandom**()
Return a random object from the set.

Returns A randomly selected object from the set or -1 if the set is empty.

bool SimSet::**isMember**(SimObject *obj*)
Test whether the given object belongs to the set.

Parameters **obj** – The object.

Returns True if the object is contained in the set; false otherwise.

void SimSet::**listObjects**()
Dump a list of all objects contained in the set to the console.

void SimSet::**onObjectAdded**(SimObject *object*)
Called when an object is added to the set.

Parameters **object** – The object that was added.

void SimSet::**onObjectRemoved**(SimObject *object*)
Called when an object is removed from the set.

Parameters **object** – The object that was removed.

void SimSet::**pushToBack**(SimObject *obj*)
Make the given object the last object in the set.

Parameters **obj** – The object to bring to the last position. Must be contained in the set.

void SimSet::**remove**(SimObject *objects*, ...)
Remove the given objects from the set.

Parameters **objects** – The objects to remove from the set.

void SimSet::**reorderChild**(SimObject *child1*, SimObject *child2*)
Make sure child1 is ordered right before child2 in the set.

Parameters

- **child1** – The first child. The object must already be contained in the set.
- **child2** – The second child. The object must already be contained in the set.

void SimSet::**sort**(string *callbackFunction*)
Sort the objects in the set using the given comparison function.

Parameters **callbackFunction** – Name of a function that takes two object arguments A and B and returns -1 if A is less, 1 if B is less, and 0 if both are equal.

SimGroup A collection of SimObjects that are owned by the group.

Inherit: SimSet

Description A SimGroup is a stricter form of SimSet. SimObjects may only be a member of a single SimGroup at a time. The SimGroup will automatically enforce the single-group-membership rule (ie. adding an object to a SimGroup will cause it to be removed from its current SimGroup, if any).

Deleting a SimGroup will also delete all SimObjects in the SimGroup.

Example:

```
// Create a SimGroup for particle emittersnewSimGroup(Emitters)
{
   canSaveDynamicFields = "1";

   newParticleEmitterNode(CrystalEmmiter) {
      active = "1";
      emitter = "dustEmitter";
      velocity = "1";
      dataBlock = "GenericSmokeEmitterNode";
      position = "-61.6276 2.1142 4.45027";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      canSaveDynamicFields = "1";
   };

   newParticleEmitterNode(Steam1) {
      active = "1";
      emitter = "SlowSteamEmitter";
      velocity = "1";
      dataBlock = "GenericSmokeEmitterNode";
      position = "-25.0458 1.55289 2.51308";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      canSaveDynamicFields = "1";
   };
};
```

SimDataBlock

Inherit: SimObject

Description

Datablocks and Networking

Client-Side Datablocks

Methods

void SimDataBlock::**reloadOnLocalClient**()

Reload the datablock. This can only be used with a local client configuration.

Functions

bool **addBadWord**(string *badWord*)

Add a string to the bad word filter. The bad word filter is a table containing words which will not be displayed in chat windows. Instead, a designated replacement string will be displayed. There are already a number of bad words automatically defined.

Parameters **badWord** – Exact text of the word to restrict.

Returns True if word was successfully added, false if the word or a subset of it already exists in the table

Example:

```
// In this game, "Foobar" is banned
%badWord = "Foobar";

// Returns true, word was successfully added
addBadWord(%badWord);

// Returns false, word has already been added
addBadWord("Foobar");
```

bool **containerBoxEmpty**(int *mask*, Point3F *center*, float *xRadius*, float *yRadius*, float *zRadius*, bool *useClientContainer*)

See if any objects of the given types are present in box of given extent.

Parameters

- **mask** – Indicates the type of objects we are checking against.
- **center** – Center of box.
- **xRadius** – Search radius in the x-axis. See note above.
- **yRadius** – Search radius in the y-axis. See note above.
- **zRadius** – Search radius in the z-axis. See note above.
- **useClientContainer** – Optionally indicates the search should be within the client container.

Returns true if the box is empty, false if any object is found.

string **containerFindFirst**(int *mask*, Point3F *point*, float *x*, float *y*, float *z*)

Find objects matching the bitmask type within a box centered at point, with extents x, y, z.

Returns

.

string **containerFindNext**()

Get more results from a previous call to containerFindFirst() .

Returns The next object found, or an empty string if nothing else was found.

string **containerRayCast**(Point3F *start*, Point3F *end*, int *mask*, SceneObject *pExempt*, bool *useClientContainer*)

Cast a ray from start to end, checking for collision against items matching mask. If pExempt is specified, then it is temporarily excluded from collision checks (For instance, you might want to exclude the player if said player was firing a weapon.)

Parameters

- **start** – An XYZ vector containing the tail position of the ray.
- **end** – An XYZ vector containing the head position of the ray
- **mask** – A bitmask corresponding to the type of objects to check for
- **pExempt** – An optional ID for a single object that ignored for this raycast
- **useClientContainer** – Optionally indicates the search should be within the client container.

Returns The distance between the start point and the position we hit.

float **containerSearchCurrDist**(bool *useClientContainer*)

Get distance of the center of the current item from the center of the current initContainerRadiusSearch.

Parameters **useClientContainer** – Optionally indicates the search should be within the client container.

Returns distance from the center of the current object to the center of the search

float **containerSearchCurrRadiusDist** (bool *useClientContainer*)
Get the distance of the closest point of the current item from the center of the current initContainerRadiusSearch.

Parameters **useClientContainer** – Optionally indicates the search should be within the client container.

Returns distance from the closest point of the current object to the center of the search

SceneObject **containerSearchNext** (bool *useClientContainer*)
Get next item from a search started with initContainerRadiusSearch() or initContainerTypeSearch() .

Parameters **useClientContainer** – Optionally indicates the search should be within the client container.

Returns the next object found in the search, or null if no more

Example:

```
// print the names of all nearby ShapeBase derived objects
%position = %obj.getPosition;
%radius = 20;
%mask = $TypeMasks::ShapeBaseObjectType;
initContainerRadiusSearch( %position, %radius, %mask );
while ( (%targetObject = containerSearchNext()) != 0 )
{
   echo( "Found: " @ %targetObject.getName() );
}
```

bool **containsBadWords** (string *text*)
Checks to see if text is a bad word. The text is considered to be a bad word if it has been added to the bad word filter.

Parameters **text** – Text to scan for bad words

Returns True if the text has bad word(s), false if it is clean

Example:

```
// In this game, "Foobar" is banned
%badWord = "Foobar";

// Add a banned word to the bad word filteraddBadWord(%badWord);

// Create the base string, can come from anywhere like user chat
%userText = "Foobar";

// Create a string of random letters
%replacementChars = "knqwrtlzs";

// If the text contains a bad word, filter it before printing
// Otherwise print the original text
if(containsBadWords(%userText))
{
   // Filter the string
   %filteredText = filterString(%userText, %replacementChars);

   // Print filtered text
```

```
      echo(%filteredText);
   }
   elseecho(%userText);
```

string **filterString** (string *baseString*, string *replacementChars*)
Replaces the characters in a string with designated text. Uses the bad word filter to determine which characters within the string will be replaced.

Parameters

- **baseString** – The original string to filter.
- **replacementChars** – A string containing letters you wish to swap in the baseString.

Returns The new scrambled string

Example:

```
// Create the base string, can come from anywhere
%baseString = "Foobar";

// Create a string of random letters
%replacementChars = "knqwrtlzs";

// Filter the string
%newString = filterString(%baseString, %replacementChars);

// Print the new string to consoleecho(%newString);
```

String **getOVRHMDChromaticAbCorrection** (int *index*)
Provides the OVR HMD chromatic aberration correction values.

Parameters **index** – The HMD index.

Returns A four component string with the chromatic aberration correction values.

int **getOVRHMDCount** ()
Get the number of HMD devices that are currently connected.

Returns The number of Oculus VR HMD devices that are currently connected.

float **getOVRHMDCurrentIPD** (int *index*)
Physical distance between the user's eye centers.

Parameters **index** – The HMD index.

Returns The current IPD.

Point2I **getOVRHMDDisplayDesktopPos** (int *index*)
Desktop coordinate position of the screen (can be negative; may not be present on all platforms).

Parameters **index** – The HMD index.

Returns Position of the screen.

int **getOVRHMDDisplayDeviceId** (int *index*)
MacOS display ID.

Parameters **index** – The HMD index.

Returns The ID of the HMD display device, if any.

string **getOVRHMDDisplayDeviceName** (int *index*)
Windows display device name used in EnumDisplaySettings/CreateDC.

Parameters **index** – The HMD index.

Returns The name of the HMD display device, if any.

String **getOVRHMDDistortionCoefficients** (int *index*)
Provides the OVR HMD distortion coefficients.

Parameters **index** – The HMD index.

Returns A four component string with the distortion coefficients.

float **getOVRHMDDistortionScale** (int *index*)
Provides the OVR HMD calculated distortion scale.

Parameters **index** – The HMD index.

Returns The calculated distortion scale.

Point2F **getOVRHMDEyeXOffsets** (int *index*)
Provides the OVR HMD eye x offsets in uv coordinates.

Parameters **index** – The HMD index.

Returns A two component string with the left and right eye x offsets.

string **getOVRHMDManufacturer** (int *index*)
Retrieves the HMD manufacturer name.

Parameters **index** – The HMD index.

Returns The manufacturer of the HMD product.

string **getOVRHMDProductName** (int *index*)
Retrieves the HMD product name.

Parameters **index** – The HMD index.

Returns The name of the HMD product.

float **getOVRHMDProfileIPD** (int *index*)
Physical distance between the user's eye centers as defined by the current profile.

Parameters **index** – The HMD index.

Returns The profile IPD.

Point2I **getOVRHMDResolution** (int *index*)
Provides the OVR HMD screen resolution.

Parameters **index** – The HMD index.

Returns A two component string with the screen's resolution.

int **getOVRHMDVersion** (int *index*)
Retrieves the HMD version number.

Parameters **index** – The HMD index.

Returns The version number of the HMD product.

float **getOVRHMDXCenterOffset** (int *index*)
Provides the OVR HMD calculated XCenterOffset.

Parameters **index** – The HMD index.

Returns The calculated XCenterOffset.

float **getOVRHMDYFOV**(int *index*)
Provides the OVR HMD calculated Y FOV.

Parameters **index** – The HMD index.

Returns The calculated Y FOV.

Point3F **getOVRSensorAcceleration**(int *index*)
Get the acceleration values for the given sensor index.

Parameters **index** – The sensor index.

Returns The acceleration values of the Oculus VR sensor, in m/s^2.

Point3F **getOVRSensorAngVelocity**(int *index*)
Get the angular velocity values for the given sensor index.

Parameters **index** – The sensor index.

Returns The angular velocity values of the Oculus VR sensor, in degrees/s.

int **getOVRSensorCount**()
Get the number of sensor devices that are currently connected.

Returns The number of Oculus VR sensor devices that are currently connected.

Point3F **getOVRSensorEulerRotation**(int *index*)
Get the Euler rotation values for the given sensor index.

Parameters **index** – The sensor index.

Returns The Euler rotation values of the Oculus VR sensor, in degrees.

bool **getOVRSensorGravityCorrection**(int *index*)
Get the gravity correction state for the given sensor index.

Parameters **index** – The sensor index.

Returns True if gravity correction (for pitch and roll) is active.

Point3F **getOVRSensorMagnetometer**(int *index*)
Get the magnetometer reading (direction and field strength) for the given sensor index.

Parameters **index** – The sensor index.

Returns The magnetometer reading (direction and field strength) of the Oculus VR sensor, in Gauss.

bool **getOVRSensorMagnetometerCalibrated**(int *index*)
Get the magnetometer calibrated data state for the given sensor index.

Parameters **index** – The sensor index.

Returns True if magnetometer calibration data is available.

float **getOVRSensorPredictionTime**(int *index*)
Get the prediction time set for the given sensor index.

Parameters **index** – The sensor index.

Returns The prediction time of the Oculus VR sensor, given in seconds.

bool **getOVRSensorYawCorrection**(int *index*)
Get the yaw correction state for the given sensor index.

Parameters **index** – The sensor index.

Returns True if yaw correction (using magnetometer calibration data) is active.

Point3F **getRazerHydraControllerPos**(int *controller*)
Get the given Razer Hydra controller's last position.

Parameters **controller** – Controller number to check.

Returns A Point3F containing the last known position.

AngAxisF **getRazerHydraControllerRot**(int *controller*)
Get the given Razer Hydra controller's last rotation.

Parameters **controller** – Controller number to check.

Returns A AngAxisF containing the last known rotation.

TransformF **getRazerHydraControllerTransform**(int *controller*)
Get the given Razer Hydra controller's last transform.

Parameters **controller** – Controller number to check.

Returns A TransformF containing the last known transform.

void **initContainerRadiusSearch**(Point3F *pos*, float *radius*, int *mask*, bool *useClientContainer*)
Start a search for items at the given position and within the given radius, filtering by mask.

Parameters

- **pos** – Center position for the search
- **radius** – Search radius
- **mask** – Bitmask of object types to include in the search
- **useClientContainer** – Optionally indicates the search should be within the client container.

void **initContainerTypeSearch**(int *mask*, bool *useClientContainer*)
Start a search for all items of the types specified by the bitset mask.

Parameters

- **mask** – Bitmask of object types to include in the search
- **useClientContainer** – Optionally indicates the search should be within the client container.

bool **isLeapMotionActive**()
Used to determine if the Leap Motion input device is active. The Leap Motion input device is considered active when the support library has been loaded and the device has been found.

Returns True if the Leap Motion input device is active.

bool **isOculusVRDeviceActive**()
Used to determine if the Oculus VR input device is active. The Oculus VR device is considered active when the library has been initialized and either a real of simulated HMD is present.

Returns True if the Oculus VR input device is active.

bool **isOVRHMDSimulated**(int *index*)
Determines if the requested OVR HMD is simulated or real.

Parameters **index** – The HMD index.

Returns True if the HMD is simulated.

bool **isRazerHydraActive**()
Used to determine if the Razer Hydra input device active. The Razer Hydra input device is considered active when the support library has been loaded and the controller has been found.

Returns True if the Razer Hydra input device is active.

bool **isRazerHydraControllerDocked**(int *controller*)
Used to determine if the given Razer Hydra controller is docked.

Parameters **controller** – Controller number to check.

Returns True if the given Razer Hydra controller is docked. Also returns true if the input device is not found or active.

void **ovrResetAllSensors**()
Resets all Oculus VR sensors. This resets all sensor orientations such that their 'normal' rotation is defined when this function is called. This defines an HMD's forwards and up direction, for example.

void **resetFPSTracker**()
Reset FPS stats (fps::).

void **sceneDumpZoneStates**(bool *updateFirst*)
Dump the current zoning states of all zone spaces in the scene to the console.

Parameters **updateFirst** – If true, zoning states are brought up to date first; if false, the zoning states are dumped as is.

SceneObject **sceneGetZoneOwner**(int *zoneId*)
Return the SceneObject that contains the given zone.

Parameters **zoneId** – ID of zone.

Returns is invalid.

void **setAllSensorPredictionTime**(float *dt*)
Set the prediction time set for all sensors.

Parameters **dt** – The prediction time to set given in seconds. Setting to 0 disables prediction.

bool **setOVRHMDAsGameConnectionDisplayDevice**(GameConnection *conn*)
Sets the first HMD to be a GameConnection's display device.

Parameters **conn** – The GameConnection to set.

Returns display device was set.

void **setOVRHMDCurrentIPD**(int *index*, float *ipd*)
Set the physical distance between the user's eye centers.

Parameters

- **index** – The HMD index.
- **ipd** – The IPD to use.

void **setOVRSensorGravityCorrection**(int *index*, bool *state*)
Set the gravity correction state for the given sensor index.

Parameters

- **index** – The sensor index.
- **state** – The gravity correction state to change to.

void **setOVRSensorYawCorrection**(int *index*, bool *state*)
Set the yaw correction state for the given sensor index.

Parameters

- **index** – The sensor index.

- **state** – The yaw correction state to change to.

void **setSensorPredictionTime** (int *index*, float *dt*)
Set the prediction time set for the given sensor index.

Parameters

- **index** – The sensor index.
- **dt** – The prediction time to set given in seconds. Setting to 0 disables prediction.

bool spawnObject(class [, dataBlock, name, properties, script])
Global function used for spawning any type of object. Note: This is separate from SpawnSphere::spawnObject() . This function is not called off any other class and uses different parameters than the SpawnSphere's function. In the source, SpawnSphere::spawnObject() actually calls this function and passes its properties (spawnClass, spawnDatablock, etc).

Parameters

- **class** – Mandatory field specifying the object class, such as Player or TSStatic.
- **datablock** – Field specifying the object's datablock, optional for objects such as TSStatic, mandatory for game objects like Player.
- **name** – Optional field specifying a name for this instance of the object.
- **properties** – Optional set of parameters applied to the spawn object during creation.
- **script** – Optional command(s) to execute when spawning an object.

Example:

```
// Set the parameters for the spawn function
%objectClass = "Player";
%objectDatablock = "DefaultPlayerData";
%objectName = "PlayerName";
%additionalProperties = "health = \"0\";"; // Note the escape sequence \ in front of quotes
%spawnScript = "echo(\"Player Spawned\");"// Note the escape sequence \ in front of quotes// Spa
%player = spawnObject(%objectClass, %objectDatablock, %objectName, %additionalProperties, %spawn
```

Variables

float $cameraFov
The camera's Field of View.

float $mvBackwardAction
Backwards movement speed for the active player.

bool $mvDeviceIsKeyboardMouse
Boolean state for it the system is using a keyboard and mouse or not.

float $mvDownAction
Downwards movement speed for the active player.

float $mvForwardAction
Forwards movement speed for the active player.

bool $mvFreeLook
Boolean state for if freelook is active or not.

float $mvLeftAction
Left movement speed for the active player.

float $mvPitch
Current pitch value, typically applied through input devices, such as a mouse.

float $mvPitchDownSpeed
Downwards pitch speed.

float $mvPitchUpSpeed
Upwards pitch speed.

float $mvRightAction
Right movement speed for the active player.

float $mvRoll
Current roll value, typically applied through input devices, such as a mouse.

float $mvRollLeftSpeed
Left roll speed.

float $mvRollRightSpeed
Right roll speed.

int $mvTriggerCount0
Used to determine the trigger counts of buttons. Namely used for input actions such as jumping and weapons firing.

int $mvTriggerCount1
Used to determine the trigger counts of buttons. Namely used for input actions such as jumping and weapons firing.

int $mvTriggerCount2
Used to determine the trigger counts of buttons. Namely used for input actions such as jumping and weapons firing.

int $mvTriggerCount3
Used to determine the trigger counts of buttons. Namely used for input actions such as jumping and weapons firing.

int $mvTriggerCount4
Used to determine the trigger counts of buttons. Namely used for input actions such as jumping and weapons firing.

int $mvTriggerCount5
Used to determine the trigger counts of buttons. Namely used for input actions such as jumping and weapons firing.

float $mvUpAction
Upwards movement speed for the active player.

float $mvXAxis_L
Left thumbstick X axis position on a dual-analog gamepad.

float $mvXAxis_R
Right thumbstick X axis position on a dual-analog gamepad.

float $mvYaw
Current yaw value, typically applied through input devices, such as a mouse.

float $mvYawLeftSpeed
Left Yaw speed.

float $mvYawRightSpeed
Right Yaw speed.

float $mvYAxis_L
Left thumbstick Y axis position on a dual-analog gamepad.

float $mvYAxis_R
Right thumbstick Y axis position on a dual-analog gamepad.

int $Ease::Back
Backwards ease for curve movement.

int $Ease::Bounce
Bounce ease for curve movement.

int $Ease::Circular
Circular ease for curve movement.

bool $RazerHydra::CombinedPositionEvents
If true, one position event will be sent that includes one component per argument.

int $Ease::Cubic
Cubic ease for curve movement.

float $pref::Camera::distanceScale
A scale to apply to the normal visible distance, typically used for tuning performance.

int $Ease::Elastic
Elastic ease for curve movement.

bool $pref::enableBadWordFilter
If true, the bad word filter will be enabled.

bool $pref::LeapMotion::EnableDevice
If true, the Leap Motion device will be enabled, if present.

bool $pref::OculusVR::EnableDevice
If true, the Oculus VR device will be enabled, if present.

bool $pref::RazerHydra::EnableDevice
If true, the Razer Hydra device will be enabled, if present.

bool $pref::enablePostEffects
If true, post effects will be eanbled.

int $Ease::Exponential
Exponential ease for curve movement.

bool $OculusVR::GenerateAngleAxisRotationEvents
If true, broadcast sensor rotation events as angled axis.

bool $OculusVR::GenerateEulerRotationEvents
If true, broadcast sensor rotation events as Euler angles about the X, Y and Z axis.

bool $LeapMotion::GenerateIndividualEvents
Indicates that events for each hand and pointable will be created.

bool $OculusVR::GenerateRotationAsAxisEvents
If true, broadcast sensor rotation as axis events.

bool $OculusVR::GenerateSensorRawEvents
If ture, broadcast sensor raw data: acceleration, angular velocity, magnetometer reading.

bool $LeapMotion::GenerateSingleHandRotationAsAxisEvents
If true, broadcast single hand rotation as axis events.

bool $LeapMotion::GenerateWholeFrameEvents
Indicates that a whole frame event should be generated and frames should be buffered.

bool $OculusVR::GenerateWholeFrameEvents
Indicates that a whole frame event should be generated and frames should be buffered.

bool $RazerHydra::GenerateWholeFrameEvents
Indicates that a whole frame event should be generated and frames should be buffered.

int $Ease::In
In ease for curve movement.

int $Ease::InOut
InOut ease for curve movement.

bool $pref::Input::JoystickEnabled
If true, the joystick is currently enabled.

bool $LeapMotion::KeepHandIndexPersistent
Indicates that we track hand IDs and will ensure that the same hand will remain at the same index between frames.

bool $LeapMotion::KeepPointableIndexPersistent
Indicates that we track pointable IDs and will ensure that the same pointable will remain at the same index between frames.

int $Ease::Linear
Linear ease for curve movement.

float $OculusVR::MaximumAxisAngle
The maximum sensor angle when used as an axis event as measured from a vector pointing straight up (in degrees). Should range from 0 to 90 degrees.

float $RazerHydra::MaximumAxisAngle
The maximum controller angle when used as an axis event as measured from a vector pointing straight up (in degrees). Shoud range from 0 to 90 degrees.

int $LeapMotion::MaximumFramesStored
The maximum number of frames to keep when $LeapMotion::GenerateWholeFrameEvents is true.

int $RazerHydra::MaximumFramesStored
The maximum number of frames to keep when $RazerHydra::GenerateWholeFrameEvents is true.

float $LeapMotion::MaximumHandAxisAngle
The maximum hand angle when used as an axis event as measured from a vector pointing straight up (in degrees). Shoud range from 0 to 90 degrees.

int $Ease::Out
Out ease for curve movement.

bool $RazerHydra::ProcessWhenDocked
If true, events will still be sent when a controller is docked.

int $Ease::Quadratic
Quadratic ease for curve movement.

int $Ease::Quartic
Quartic ease for curve movement.

int $Ease::Quintic
Quintic ease for curve movement.

bool $RazerHydra::RotationAsAxisEvents
If true, broadcast controller rotation as axis events.

bool $RazerHydra::SeparatePositionEvents
If true, separate position events will be sent for each component.

int $Ease::Sinusoidal
Sinusoidal ease for curve movement.

bool $pref::OculusVR::UseChromaticAberrationCorrection
If true, Use the chromatic aberration correction version of the Oculus VR barrel distortion shader.

Objects

Objects which can be controlled or directly interact with a user, such as Player, Projectile, Item, etc. Does not include vehicles as they have their own section.

Classes

AIPlayer A Player object not controlled by conventional input, but by an AI engine.

Inherit: Player

Description The AIPlayer provides a Player object that may be controlled from script. You control where the player moves and how fast. You may also set where the AIPlayer is aiming at – either a location or another game object.

The AIPlayer class does not have a datablock of its own. It makes use of the PlayerData datablock to define how it looks, etc. As the AIPlayer is an extension of the Player class it can mount objects and fire weapons, or mount vehicles and drive them.

While the PlayerData datablock is used, there are a number of additional callbacks that are implemented by AIPlayer on the datablock. These are listed here:

void onReachDestination(AIPlayer obj) Called when the player has reached its set destination using the setMoveDestination() method. The actual point at which this callback is called is when the AIPlayer is within the mMoveTolerance of the defined destination.

void onMoveStuck(AIPlayer obj) While in motion, if an AIPlayer has moved less than moveStuckTolerance within a single tick, this callback is called. From here you could choose an alternate destination to get the AIPlayer moving again.

void onTargetEnterLOS(AIPlayer obj) When an object is being aimed at (following a call to setAimObject()) and the targeted object enters the AIPlayer's line of sight, this callback is called. The LOS test is a ray from the AIPlayer's eye position to the center of the target's bounding box. The LOS ray test only checks against interiors, statis shapes, and terrain.

void onTargetExitLOS(AIPlayer obj) When an object is being aimed at (following a call to setAimObject()) and the targeted object leaves the AIPlayer's line of sight, this callback is called. The LOS test is a ray from the AIPlayer's eye position to the center of the target's bounding box. The LOS ray test only checks against interiors, statis shapes, and terrain.

Example:

```
// Create the demo player object
%player = new AiPlayer()
{
   dataBlock = DemoPlayer;
```

```
   path = "";
};
```

Methods

void AIPlayer::**clearAim**()

Use this to stop aiming at an object or a point.

Point3F AIPlayer::**getAimLocation**()

Returns the point the AIPlayer is aiming at. This will reflect the position set by setAimLocation() , or the position of the object that the bot is now aiming at. If the bot is not aiming at anything, this value will change to whatever point the bot's current line-of-sight intercepts.

Returns World space coordinates of the object AI is aiming at. Formatted as "X Y Z".

int AIPlayer::**getAimObject**()

Gets the object the AIPlayer is targeting.

Returns is aiming at.

Point3F AIPlayer::**getMoveDestination**()

Get the AIPlayer's current destination.

Returns Returns a point containing the "x y z" position of the AIPlayer's current move destination. If no move destination has yet been set, this returns "0 0 0".

float AIPlayer::**getMoveSpeed**()

Gets the move speed of an AI object.

Returns A speed multiplier between 0.0 and 1.0.

void AIPlayer::**setAimLocation**(Point3F *target*)

Tells the AIPlayer to aim at the location provided.

Parameters **target** – An "x y z" position in the game world to target.

void AIPlayer::**setAimObject**(GameBase *targetObject*, Point3F *offset*)

Sets the AIPlayer's target object. May optionally set an offset from target location.

Parameters

- **targetObject** – The object to target
- **offset** – Optional three-element offset vector which will be added to the position of the aim object.

Example:

```
    // Without an offset
    %ai.setAimObject(%target);

    // With an offset
    // Cause our AI object to aim at the target
    // offset (0, 0, 1) so you dont aim at the targets feet
    %ai.setAimObject(%target, "0 0 1");
```

void AIPlayer::**setMoveDestination**(Point3F *goal*, bool *slowDown*)

Tells the AI to move to the location provided.

Parameters

- **goal** – Coordinates in world space representing location to move to.

- **slowDown** – A boolean value. If set to true, the bot will slow down when it gets within 5-meters of its move destination. If false, the bot will stop abruptly when it reaches the move destination. By default, this is true.

void AIPlayer::**setMoveSpeed**(float *speed*)
Sets the move speed for an AI object.

Parameters **speed** – A speed multiplier between 0.0 and 1.0. This is multiplied by the AIPlayer's base movement rates (as defined in its PlayerData datablock)

void AIPlayer::**stop**()
Tells the AIPlayer to stop moving.

Fields

float AIPlayer::**mMoveTolerance**
Distance from destination before stopping. When the AIPlayer is moving to a given destination it will move to within this distance of the destination and then stop. By providing this tolerance it helps the AIPlayer from never reaching its destination due to minor obstacles, rounding errors on its position calculation, etc. By default it is set to 0.25.

int AIPlayer::**moveStuckTestDelay**
The number of ticks to wait before testing if the AIPlayer is stuck. When the AIPlayer is asked to move, this property is the number of ticks to wait before the AIPlayer starts to check if it is stuck. This delay allows the AIPlayer to accelerate to full speed without its initial slow start being considered as stuck.

float AIPlayer::**moveStuckTolerance**
Distance tolerance on stuck check. When the AIPlayer is moving to a given destination, if it ever moves less than this tolerance during a single tick, the AIPlayer is considered stuck. At this point the onMoveStuck() callback is called on the datablock.

AITurretShape Provides an AI controlled turret.

Inherit: TurretShape

Description Provides an AI controlled turret.

Uses the AITurretShapeData datablock, which is based on the TurretShapeData datablock for common properties.

AITurretShape builds an AI controlled turret. It uses a state machine and properties as defined in AITurretShapeData to decide how to find targets and what to do with them. As with TurretShape (which AITurretShape derives from) the AITurretShape class provides the base on which ShapeBaseImageData weapons may be mounted.

Overview The AITurretShape functions through the use of a state machine as defined in its AITurretShapeData datablock. It is very similar to how ShapeBaseImageData works. This allows you to customize exactly how the turret behaves while it searches for a target, and what it does once it has a target. But in general, the AI turret goes through a number of stages:

The AI turret usually starts off by scanning for a suitable target. This is done by checking for targets within a pie wedge shaped volume in front of the turret based on where the scanPoint node is placed. The turret takes cover into account when searching for a target so that it doesn't "cheat".

Once a target is acquired the turret attempts to follow it. Usually at this point the turret activates its weapon. If a target is lost due to it going behind cover, the turret will attempt to follow and reacquire the target using its last known position and velocity. The amount of time allowed for this attempt is defined by AITurretShapeData::trackLostTargetTime.

If the target is lost (either by going behind cover or it is dead) the turret returns to its scanning mode to find another victim.

If the AI turret is destroyed then it can go into a special state to show the user that it has been destroyed. As with TurretShape turrets a AITurretShape may respawn after a set amount of time (see TurretShape and TurretShape::doRespawn()).

In addition to AI turrets being placed within a mission, it is also possible for a player to deploy a turret such as throwing one from their inventory. When a turret has been tossed it will be in a Thrown state and usually in an inactive mode. Once the turret comes to rest on the ground it moves into a Deploy state where it may unfold to get ready. Once ready the turret begins the scanning process. As the AI turret's state machine may be customized for your specific circumstances, the way in which turrets are deployed by a player is up to you. An AI turret could be thrown in a fully working state, ready to take out targets before the turret even hits the ground.

Example State Machine Here is an example AITurretShapeData datablock with a defined state machine and the script to support the state machine. This is just one possible example.

Example:

```
//--------------------------------------------------------------------------// AI Turret//--------

datablock AITurretShapeData(AITurret)
{
   category = "Turrets";
   shapeFile = "art/shapes/weapons/Turret/Turret_Legs.DAE";

   maxDamage = 70;
   destroyedLevel = 70;
   explosion = GrenadeExplosion;

   simpleServerCollision = false;

   zRotOnly = false;

   // Rotation settings
   minPitch = 15;
   maxPitch = 80;
   maxHeading = 90;
   headingRate = 50;
   pitchRate = 50;

   // Scan settings
   maxScanPitch = 10;
   maxScanHeading = 30;
   maxScanDistance = 20;
   trackLostTargetTime = 2;

   maxWeaponRange = 30;

   weaponLeadVelocity = 0;

   // Weapon mounting
   numWeaponMountPoints = 1;

   weapon[0] = AITurretHead;
   weaponAmmo[0] = AITurretAmmo;
   weaponAmmoAmount[0] = 10000;

   maxInv[AITurretHead] = 1;
   maxInv[AITurretAmmo] = 10000;
```

```
   // Initial start up state
   stateName[0]                     = "Preactivate";
   stateTransitionOnAtRest[0]       = "Scanning";
   stateTransitionOnNotAtRest[0]    = "Thrown";

   // Scan for targets
   stateName[1]                     = "Scanning";
   stateScan[1]                     = true;
   stateTransitionOnTarget[1]       = "Target";
   stateSequence[1]                 = "scan";
   stateScript[1]                   = "OnScanning";

   // Have a target
   stateName[2]                     = "Target";
   stateTransitionOnNoTarget[2]     = "NoTarget";
   stateTransitionOnTimeout[2]      = "Firing";
   stateTimeoutValue[2]             = 2.0;
   stateScript[2]                   = "OnTarget";

   // Fire at target
   stateName[3]                     = "Firing";
   stateFire[3]                     = true;
   stateTransitionOnNoTarget[3]     = "NoTarget";
   stateScript[3]                   = "OnFiring";

   // Lost target
   stateName[4]                     = "NoTarget";
   stateTransitionOnTimeout[4]      = "Scanning";
   stateTimeoutValue[4]             = 2.0;
   stateScript[4]                   = "OnNoTarget";

   // Player thrown turret
   stateName[5]                     = "Thrown";
   stateTransitionOnAtRest[5]       = "Deploy";
   stateSequence[5]                 = "throw";
   stateScript[5]                   = "OnThrown";

   // Player thrown turret is deploying
   stateName[6]                     = "Deploy";
   stateTransitionOnTimeout[6]      = "Scanning";
   stateTimeoutValue[6]             = 2.5;
   stateSequence[6]                 = "deploy";
   stateScaleAnimation[6]           = true;
   stateScript[6]                   = "OnDeploy";

   // Special state that is set when the turret is destroyed.// This state is set in the onDestroyed
   stateName[7]                     = "Destroyed";
   stateSequence[7]                 = "destroyed";
};

//--------------------------------------------------------------------------// Deployable AI Turre
datablock AITurretShapeData(DeployableTurret : AITurret)
{
   // Mission editor category
   category = "Weapon";

   className = "DeployableTurretWeapon";
```

```
   startLoaded = false;

   // Basic Item properties
   mass = 1.5;
   elasticity = 0.1;
   friction = 0.6;
   simpleServerCollision = false;

   // Dynamic properties defined by the scripts
   PreviewImage = turret.png;
   pickUpName = "a deployable turret";
   description = "Deployable Turret";
   image = DeployableTurretImage;
   reticle = "blank";
   zoomReticle = blank;
};

// ----------------------------------------------------------------------------// AITurretShapeData/

function AITurretShapeData::onAdd(%this, %obj)
{
   Parent::onAdd(%this, %obj);

   %obj.mountable = false;
}

// Player has thrown a deployable turret.  This copies from ItemData::onThrow()
function AITurretShapeData::onThrow(%this, %user, %amount)
{
   // Remove the object from the inventoryif (%amount $= "")
      %amount = 1;
   if (%this.maxInventory !$= "")
      if (%amount > %this.maxInventory)
         %amount = %this.maxInventory;
   if (!%amount)
      return 0;
   %user.decInventory(%this,%amount);

   // Construct the actual object in the world, and add it to// the mission group so its cleaned up w
   %rot = %user.getEulerRotation();
   %obj = newAITurretShape()
   {
      datablock = %this;
      rotation = "0 0 1 " @ getWord(%rot, 2);
      count = 1;
      sourceObject = %user;
      client = %user.client;
      isAiControlled = true;
   };
   MissionGroup.add(%obj);

   // Let the turret know that were a firend
   %obj.addToIgnoreList(%user);

   // We need to add this turret to a list on the client so that if we die,// the turret will still
   %client = %user.client;
   if (%client)
   {
```

```
      if (!%client.ownedTurrets)
      {
         %client.ownedTurrets = newSimSet();
      }

      // Go through the clients owned turret list.  Make sure were// a friend of every turret and eve
      {
         %turret = %client.ownedTurrets.getObject(%i);
         %turret.addToIgnoreList(%obj);
         %obj.addToIgnoreList(%turret);
      }

      // Add ourselves to the clients owned list.
      %client.ownedTurrets.add(%obj);
   }

   return %obj;
}

function AITurretShapeData::onDestroyed(%this, %turret, %lastState)
{
   // This method is invoked by the ShapeBase code whenever the// objects damage state changes.

   %turret.playAudio(0, TurretDestroyed);
   %turret.setAllGunsFiring(false);
   %turret.resetTarget();
   %turret.setTurretState( "Destroyed", true );

   // Set the weapons to destoryedfor(%i = 0; %i < %this.numWeaponMountPoints; %i++)
   {
      %turret.setImageGenericTrigger(%i, 0, true);
   }

   Parent::onDestroyed(%this, %turret, %lastState);
}

function AITurretShapeData::OnScanning(%this, %turret)
{
   //echo("AITurretShapeData::OnScanning: " SPC %this SPC %turret);

   %turret.startScanForTargets();
   %turret.playAudio(0, TurretScanningSound);
}

function AITurretShapeData::OnTarget(%this, %turret)
{
   //echo("AITurretShapeData::OnTarget: " SPC %this SPC %turret);

   %turret.startTrackingTarget();
   %turret.playAudio(0, TargetAquiredSound);
}

function AITurretShapeData::OnNoTarget(%this, %turret)
{
   //echo("AITurretShapeData::OnNoTarget: " SPC %this SPC %turret);

   %turret.setAllGunsFiring(false);
   %turret.recenterTurret();
```

```
   %turret.playAudio(0, TargetLostSound);
}

function AITurretShapeData::OnFiring(%this, %turret)
{
   //echo("AITurretShapeData::OnFiring: " SPC %this SPC %turret);

   %turret.setAllGunsFiring(true);
}

function AITurretShapeData::OnThrown(%this, %turret)
{
   //echo("AITurretShapeData::OnThrown: " SPC %this SPC %turret);

   %turret.playAudio(0, TurretThrown);
}

function AITurretShapeData::OnDeploy(%this, %turret)
{
   //echo("AITurretShapeData::OnDeploy: " SPC %this SPC %turret);// Set the weapons to loadedfor(%i =
   {
      %turret.setImageLoaded(%i, true);
   }

   %turret.playAudio(0, TurretActivatedSound);
}
```

And here is the above example state machine's flow:

Shape File Nodes In addition to the required TurretBase nodes, AITurretShape makes use of additional nodes within the shape file to allow the AI to do its work. The first is the 'scanPoint' node. This is used by the AI to project a pie wedge shaped scanning volume in which to detect possible targets. The scanPoint node is at the apex of the scanning wedge. If the scanPoint node is not present within the shape file then the turret's world transform is used.

The second is the 'aimPoint' node. Once the AI turret has obtained a target the aimPoint is used to point the turret at the target. Specifically, the turret rotates in both pitch and heading such that the aimPoint points at the target. If you're using a weapon that doesn't have its muzzle point on the same plane as its mount point (known as an off-axis weapon) then be sure to place the aimPoint at a z position equivalent to the weapon's muzzle point. This allows for the correct pitch calculation. If the aimPoint is not found on the turret's shape, then the pitch node will be used.

Ignore List AI turrets keep track of an ignore list. This is used by default to stop a player deployed turret from targeting its owner, even when that owner is killed and respawns. But this ignore list could also be used to have the turret ignore team mates, squad members, invisible players, etc. Use AITurretShape::addToIgnoreList() and AITurretShape::removeFromIgnoreList() to manipulate this list. You should also look in scripts/server/turret.cs at AITurretShapeData::onThrow() to see how the ignore list is handled and deployed turrets are kept track of on a per connected client basis.

Methods

void `AITurretShape::activateTurret`()

Activate a turret from a deactive state.

void `AITurretShape::addToIgnoreList`(ShapeBase *obj*)

Adds object to the turret's ignore list. All objects in this list will be ignored by the turret's targeting.

Parameters **`obj`** – The ShapeBase object to ignore.

void AITurretShape::**deactivateTurret**()
Deactivate a turret from an active state.

SimObject AITurretShape::**getTarget**()
Get the turret's current target.

Returns The object that is the target's current target, or 0 if no target.

float AITurretShape::**getWeaponLeadVelocity**()
Get the turret's defined projectile velocity that helps with target leading.

Returns The defined weapon projectile speed, or 0 if leading is disabled.

bool AITurretShape::**hasTarget**()
Indicates if the turret has a target.

Returns True if the turret has a target.

void AITurretShape::**recenterTurret**()
Recenter the turret's weapon.

void AITurretShape::**removeFromIgnoreList**(ShapeBase *obj*)
Removes object from the turret's ignore list. All objects in this list will be ignored by the turret's targeting.

Parameters **obj** – The ShapeBase object to once again allow for targeting.

void AITurretShape::**resetTarget**()
Resets the turret's target tracking. Only resets the internal target tracking. Does not modify the turret's facing.

void AITurretShape::**setAllGunsFiring**(bool *fire*)
Set the firing state of the turret's guns.

Parameters **fire** – Set to true to activate all guns. False to deactivate them.

void AITurretShape::**setGunSlotFiring**(int *slot*, bool *fire*)
Set the firing state of the given gun slot.

Parameters

- **slot** – The gun to modify. Valid range is 0-3 that corresponds to the weapon mount point.
- **fire** – Set to true to activate the gun. False to deactivate it.

void AITurretShape::**setTurretState**(string *newState*, bool *force*)
Set the turret's current state. Normally the turret's state comes from updating the state machine but this method allows you to override this and jump to the requested state immediately.

Parameters

- **newState** – The name of the new state.
- **force** – Is true then force the full processing of the new state even if it is the same as the current state. If false then only the time out value is reset and the state's script method is called, if any.

void AITurretShape::**setWeaponLeadVelocity**(float *velocity*)
Set the turret's projectile velocity to help lead the target. This value normally comes from AITurretShapeData::weaponLeadVelocity but this method allows you to override the datablock value. This can be useful if the turret changes ammunition, uses a different weapon than the default, is damaged, etc.

void AITurretShape::**startScanForTargets**()
Begin scanning for a target.

void AITurretShape::**startTrackingTarget**()
Have the turret track the current target.

void AITurretShape::**stopScanForTargets**()
Stop scanning for targets.

void AITurretShape::**stopTrackingTarget**()
Stop the turret from tracking the current target.

AITurretShapeData object.

Inherit: TurretShapeData

Description Defines properties for an AITurretShape object.

Fields

float AITurretShapeData::**maxScanDistance**
Maximum distance to scan. When combined with maxScanHeading and maxScanPitch this forms a 3D scanning wedge used to initially locate a target.

float AITurretShapeData::**maxScanHeading**
Maximum number of degrees to scan left and right.

float AITurretShapeData::**maxScanPitch**
Maximum number of degrees to scan up and down.

float AITurretShapeData::**maxWeaponRange**
Maximum distance that the weapon will fire upon a target.

int AITurretShapeData::**scanTickFrequency**
How often should we perform a full scan when looking for a target. Expressed as the number of ticks between full scans, but no less than 1.

int AITurretShapeData::**scanTickFrequencyVariance**
Random amount that should be added to the scan tick frequency each scan period. Expressed as the number of ticks to randomly add, but no less than zero.

bool AITurretShapeData::**stateDirection**[31]
Direction of the animation to play in this state. True is forward, false is backward.

bool AITurretShapeData::**stateFire**[31]
The first state with this set to true is the state entered by the client when it receives the 'fire' event.

caseString AITurretShapeData::**stateName**[31]
Name of this state.

bool AITurretShapeData::**stateScaleAnimation**[31]
If true, the timeScale of the stateSequence animation will be adjusted such that the sequence plays for stateTimeoutValue seconds.

bool AITurretShapeData::**stateScan**[31]
Indicates the turret should perform a continuous scan looking for targets.

caseString AITurretShapeData::**stateScript**[31]
Method to execute on entering this state. Scoped to AITurretShapeData .

string AITurretShapeData::**stateSequence**[31]
Name of the sequence to play on entry to this state.

float AITurretShapeData::**stateTimeoutValue**[31]
Time in seconds to wait before transitioning to stateTransitionOnTimeout.

string AITurretShapeData::**stateTransitionOnActivated**[31]
Name of the state to transition to when the turret goes from deactivated to activated.

string AITurretShapeData::**stateTransitionOnAtRest**[31]
Name of the state to transition to when the turret is at rest (static).

string AITurretShapeData::**stateTransitionOnDeactivated**[31]
Name of the state to transition to when the turret goes from activated to deactivated.

string AITurretShapeData::**stateTransitionOnNoTarget**[31]
Name of the state to transition to when the turret loses a target.

string AITurretShapeData::**stateTransitionOnNotAtRest**[31]
Name of the state to transition to when the turret is not at rest (not static).

string AITurretShapeData::**stateTransitionOnTarget**[31]
Name of the state to transition to when the turret gains a target.

string AITurretShapeData::**stateTransitionOnTimeout**[31]
Name of the state to transition to when we have been in this state for stateTimeoutValue seconds.

bool AITurretShapeData::**stateWaitForTimeout**[31]
If false, this state ignores stateTimeoutValue and transitions immediately if other transition conditions are met.

float AITurretShapeData::**trackLostTargetTime**
How long after the turret has lost the target should it still track it. Expressed in seconds.

float AITurretShapeData::**weaponLeadVelocity**
Velocity used to lead target. If value lt = 0, don't lead target.

GameBase Base class for game objects which use datablocks, networking, are editable, and need to process ticks.

Inherit: SceneObject

Description Base class for game objects which use datablocks, networking, are editable, and need to process ticks.

Methods

bool GameBase::**applyImpulse**(Point3F *pos*, VectorF *vel*)
Apply an impulse to this object as defined by a world position and velocity vector.

Parameters

- **pos** – impulse world position
- **vel** – impulse velocity (impulse force F = m * v)

Returns Always true

void GameBase::**applyRadialImpulse**(Point3F *origin*, float *radius*, float *magnitude*)
Applies a radial impulse to the object using the given origin and force.

Parameters

- **origin** – World point of origin of the radial impulse.
- **radius** – The radius of the impulse area.
- **magnitude** – The strength of the impulse.

int GameBase::**getDataBlock**()
Get the datablock used by this object.

Returns is using.

void GameBase::**setControl**(bool *controlled*)
Called when the client controlling the object changes.

Parameters **controlled** – true if a client now controls this object, false if no client controls this object.

bool GameBase::**setDataBlock**(GameBaseData *data*)
Assign this GameBase to use the specified datablock.

Parameters **data** – new datablock to use

Returns true if successful, false if failed.

Fields
bool GameBase::**boundingBox**[static]
Toggles on the rendering of the bounding boxes for certain types of objects in scene.
GameBaseData GameBase::**dataBlock**
Script datablock used for game objects.

GameBaseData objects.

Inherit: SimDataBlock

Description Scriptable, demo-able datablock. Used by GameBase objects.

Methods
void GameBaseData::**onAdd**(GameBase *obj*)
Called when the object is added to the scene.

Parameters **obj** – the GameBase object

Example:

```
datablock GameBaseData(MyObjectData)
{
   category = "Misc";
};

function MyObjectData::onAdd( %this, %obj )
{
   echo( "Added " @ %obj.getName() @ " to the scene." );
}

function MyObjectData::onNewDataBlock( %this, %obj )
{
   echo( "Assign " @ %this.getName() @ " datablock to " %obj.getName() );
}

function MyObjectData::onRemove( %this, %obj )
{
   echo( "Removed " @ %obj.getName() @ " to the scene." );
}

function MyObjectData::onMount( %this, %obj, %mountObj, %node )
{
   echo( %obj.getName() @ " mounted to " @ %mountObj.getName() );
}
```

```
function MyObjectData::onUnmount( %this, %obj, %mountObj, %node )
{
   echo( %obj.getName() @ " unmounted from " @ %mountObj.getName() );
}
```

void GameBaseData::**onMount**(GameBase *obj*, SceneObject *mountObj*, int *node*)
Called when the object is mounted to another object in the scene.

Parameters

- **obj** – the GameBase object being mounted
- **mountObj** – the object we are mounted to
- **node** – the mountObj node we are mounted to

void GameBaseData::**onNewDataBlock**(GameBase *obj*)
Called when the object has a new datablock assigned.

Parameters **obj** – the GameBase object

void GameBaseData::**onRemove**(GameBase *obj*)
Called when the object is removed from the scene.

Parameters **obj** – the GameBase object

void GameBaseData::**onUnmount**(GameBase *obj*, SceneObject *mountObj*, int *node*)
Called when the object is unmounted from another object in the scene.

Parameters

- **obj** – the GameBase object being unmounted
- **mountObj** – the object we are unmounted from
- **node** – the mountObj node we are unmounted from

Fields

caseString GameBaseData::**category**
The group that this datablock will show up in under the "Scripted" tab in the World Editor Library.

Item datablock for common properties.

Inherit: ShapeBase

Description Base Item class. Uses the ItemData datablock for common properties.

Items represent an object in the world, usually one that the player will interact with. One example is a health kit on the group that is automatically picked up when the player comes into contact with it.

Example:

```
// This is the "health patch" dropped by a dying player.
datablock ItemData(HealthKitPatch)
{
   // Mission editor category, this datablock will show up in the// specified category under the "sha
   category = "Health";

   className = "HealthPatch";

   // Basic Item properties
```

```
   shapeFile = "art/shapes/items/patch/healthpatch.dts";
   mass = 2;
   friction = 1;
   elasticity = 0.3;
   emap = true;

   // Dynamic properties used by the scripts
   pickupName = "a health patch";
   repairAmount = 50;
};

%obj = newItem()
{
   dataBlock = HealthKitSmall;
   parentGroup = EWCreatorWindow.objectGroup;
   static = true;
   rotate = true;
};
```

Methods

string Item::**getLastStickyNormal**()

Get the normal of the surface on which the object is stuck.

Returns is stuck.

Example:

```
// Acquire the position where this Item is currently stuck
%stuckPosition = %item.getLastStickPos();
```

string Item::**getLastStickyPos**()

Get the position on the surface on which this Item is stuck.

Returns is stuck.

Example:

```
// Acquire the position where this Item is currently stuck
%stuckPosition = %item.getLastStickPos();
```

bool Item::**isAtRest**()

Is the object at rest (ie, no longer moving)?

Returns True if the object is at rest, false if it is not.

Example:

```
// Query the item on if it is or is not at rest.
%isAtRest = %item.isAtRest();
```

bool Item::**isRotating**()

Is the object still rotating?

Returns True if the object is still rotating, false if it is not.

Example:

```
// Query the item on if it is or is not rotating.
%isRotating = %itemData.isRotating();
```

bool Item::**isStatic**()

Is the object static (ie, non-movable)?

Returns True if the object is static, false if it is not.

Example:

```
// Query the item on if it is or is not static.
%isStatic = %itemData.isStatic();
```

void Item::**onEnterLiquid**(string *objID*, string *waterCoverage*, string *liquidType*)
Informs an Item object that it has entered liquid, along with information about the liquid type.

Parameters

- **objID** – Object ID for this Item object.
- **waterCoverage** – How much coverage of water this Item object has.
- **liquidType** – The type of liquid that this Item object has entered.

void Item::**onLeaveLiquid**(string *objID*, string *liquidType*)
Informs an Item object that it has left a liquid, along with information about the liquid type.

Parameters

- **objID** – Object ID for this Item object.
- **liquidType** – The type of liquid that this Item object has left.

void Item::**onStickyCollision**(string *objID*)
Informs the Item object that it is now sticking to another object. This callback is only called if the ItemData::sticky property for this Item is true.

Parameters **objID** – Object ID this Item object.

bool Item::**setCollisionTimeout**(int *ignoreColObj*)
Temporarily disable collisions against a specific ShapeBase object. This is useful to prevent a player from immediately picking up an Item they have just thrown. Only one object may be on the timeout list at a time. The timeout is defined as 15 ticks.

Parameters **objectID** – ShapeBase object ID to disable collisions against.

Returns object requested could be found, false if it could not.

Example:

```
// Set the ShapeBase Object ID to disable collisions against
%ignoreColObj = %player.getID();

// Inform this Item object to ignore collisions temproarily against the %ignoreColObj.
%item.setCollisionTimeout(%ignoreColObj);
```

Fields

int Item::**maxWarpTicks**[static]
When a warp needs to occur due to the client being too far off from the server, this is the maximum number of ticks we'll allow the client to warp to catch up.

float Item::**minWarpTicks**[static]
Fraction of tick at which instant warp occures on the client.

bool Item::**rotate**
If true, the object will automatically rotate around its Z axis.

bool Item::static
If true, the object is not moving in the world.

ItemData type.

Inherit: ShapeBaseData

Description Stores properties for an individual Item type.

Items represent an object in the world, usually one that the player will interact with. One example is a health kit on the group that is automatically picked up when the player comes into contact with it.

ItemData provides the common properties for a set of Items. These properties include a DTS or DAE model used to render the Item in the world, its physical properties for when the Item interacts with the world (such as being tossed by the player), and any lights that emit from the Item.

Example:

```
datablock ItemData(HealthKitSmall)
{
   category ="Health";
   className = "HealthPatch";
   shapeFile = "art/shapes/items/kit/healthkit.dts";
   gravityMod = "1.0";
   mass = 2;
   friction = 1;
   elasticity = 0.3;
   density = 2;
   drag = 0.5;
   maxVelocity = "10.0";
   emap = true;
   sticky = false;
   dynamicType = "0"
;   lightOnlyStatic = false;
   lightType = "NoLight";
   lightColor = "1.0 1.0 1.0 1.0";
   lightTime = 1000;
   lightRadius = 10.0;
   simpleServerCollision = true;   // Dynamic properties used by the scripts

   pickupName = "a small health kit";
   repairAmount = 50;
};
```

Fields

float `ItemData::elasticity`
A floating-point value specifying how 'bouncy' this ItemData is.

float `ItemData::friction`
A floating-point value specifying how much velocity is lost to impact and sliding friction.

float `ItemData::gravityMod`
Floating point value to multiply the existing gravity with, just for this ItemData .

ColorF `ItemData::lightColor`
Color value to make this light. Example: "1.0,1.0,1.0".

bool `ItemData::lightOnlyStatic`
If true, this ItemData will only cast a light if the Item for this ItemData has a static value of true.

float `ItemData::lightRadius`
Distance from the center point of this ItemData for the light to affect.

int ItemData::**lightTime**
Time value for the light of this ItemData , used to control the pulse speed of the PulsingLight LightType.

ItemLightType ItemData::**lightType**
Type of light to apply to this ItemData . Options are NoLight, ConstantLight, PulsingLight. Default is NoLight.

float ItemData::**maxVelocity**
Maximum velocity that this ItemData is able to move.

bool ItemData::**simpleServerCollision**
Determines if only simple server-side collision will be used (for pick ups). If set to true then only simple, server-side collision detection will be used. This is often the case if the item is used for a pick up object, such as ammo. If set to false then a full collision volume will be used as defined by the shape. The default is true.

bool ItemData::**sticky**
If true, ItemData will 'stick' to any surface it collides with. When an item does stick to a surface, the Item::onStickyCollision() callback is called. The Item has methods to retrieve the world position and normal the Item is stuck to.

Player A client-controlled player character.

Inherit: ShapeBase

Description A client-controlled player character.

The Player object is the main client-controlled object in an FPS, or indeed, any game where the user is in control of a single character. This class (and the associated datablock, PlayerData) allows you to fine-tune the movement, collision detection, animation, and SFX properties of the character. Player derives from ShapeBase, so it is recommended to have a good understanding of that class (and it's parent classes) as well.

Movement The Player class supports the following modes of movement, known as poses:

The acceleration, maximum speed, and bounding box for each mode can be set independently using the PlayerData datablock. The player will automatically switch between swimming and one of the other 4 'dry' modes when entering/exiting the water, but transitions between the non-swimming modes are handled by controller input (such as holding down a key to begin crouching). $mvTriggerCount3 activates crouching, while $mvTriggerCount4 activates being prone.

It is important to set the bounding box correctly for each mode so that collisions with the player remain accurate:

When the player changes pose a new PlayerData callback onPoseChange() is called. This is being used as Armor::onPoseChange() to modify an animation prefix used by ShapeBaseImageData to allow the 1st person arms to change their animation based on pose.

Example:

```
function Armor::onPoseChange(%this, %obj, %oldPose, %newPose)
{
   // Set the script anim prefix to be that of the current pose
   %obj.setImageScriptAnimPrefix( $WeaponSlot, addTaggedString(%newPose) );
}
```

Another feature is being able to lock out poses for the Player at any time. This is done with allowCrouch(), allowSprinting() etc. (there is even allowJumping() and allowJetJumping() which aren't actually poses but states). So if for some game play reason the player should not be allowed to crouch right now, that may be disabled. All poses may be allowed with allowAllPoses() on the Player class.

The pose lock out mechanism is being used by the weapon script system – see Weapon::onUse(). With this system, weapons can prevent the player from going into certain poses. This is used by the deployable turret to lock out sprinting while the turret is the current weapon.

Example:

```
function Weapon::onUse(%data, %obj)
{
   // Default behavior for all weapons is to mount it into the objects weapon// slot, which is curren
   {
      serverPlay3D(WeaponUseSound, %obj.getTransform());

      %obj.mountImage(%data.image, $WeaponSlot);
      if (%obj.client)
      {
         if (%data.description !$= "")
            messageClient(%obj.client, MsgWeaponUsed, \c0%1 selected., %data.description);
         else
            messageClient(%obj.client, MsgWeaponUsed, \c0Weapon selected);
      }

      // If this is a Player class object then allow the weapon to modify allowed posesif (%obj.isIn
      {
         // Start by allowing everything
         %obj.allowAllPoses();

         // Now see what isnt allowed by the weapon

         %image = %data.image;

         if (%image.jumpingDisallowed)
            %obj.allowJumping(false);

         if (%image.jetJumpingDisallowed)
            %obj.allowJetJumping(false);

         if (%image.sprintDisallowed)
            %obj.allowSprinting(false);

         if (%image.crouchDisallowed)
            %obj.allowCrouching(false);

         if (%image.proneDisallowed)
            %obj.allowProne(false);

         if (%image.swimmingDisallowed)
            %obj.allowSwimming(false);
      }
   }
}
```

Sprinting As mentioned above, sprinting is another pose for the Player class. It defines its own force and max speed in the three directions in PlayerData just like most poses, such as crouch. It is activated using $mvTriggerCount5 by default which is often connected to Left Shift. When used this way you could treat it just like a standard run – perhaps with the standard pose used for a walk in a RPG.

But sprinting is special in that you can control if a player's movement while sprinting should be constrained. You can place scale factors on strafing, yaw and pitch. These force the player to move mostly in a straight line (or completely if

you set them to 0) while sprinting by limiting their motion. You can also choose if the player can jump while sprinting. This is all set up in PlayerData.

Just like other poses, you can define which sequences should be played on the player while sprinting. These sequences are:

However, if any of these sequences are not defined for the player, then the standard root, run, back, side and side_right sequences will be used. The idea here is that the ground transform for these sequences will force them to play faster to give the appearance of sprinting. But if you want the player to do something different than just look like they're running faster – such as holding their weapon against their body – then you'll want to make use of the sprint specific sequences.

Sprint also provides two PlayerData callbacks: onStartSprintMotion() and onStopSprintMotion(). The start callback is called when the player is in a sprint pose and starts to move (i.e. presses the W key). The stop callback is called when either the player stops moving, or they stop sprinting. These could be used for anything, but by default they are tied into the ShapeBaseImageData system. See Armor::onStartSprintMotion() and Armor::onStopSprintMotion(). With ShapeBaseImageData supporting four generic triggers that may be used by a weapon's state machine to do something, the first one is triggered to allow weapons to enter a special sprint state that plays a sprint animation sequence and locks out firing. However, you may choose to do something different.

Jumping The Player class supports jumping. While the player is in contact with a surface (and optionally has enough energy as defined by the PlayerData), $mvTriggerCount2 will cause the player to jump.

Jetting The Player class includes a simple jetpack behaviour allowing characters to 'jet' upwards while jumping. The jetting behaviour can be linked to the player's energy level using datablock properties as shown below:

Example:

```
datablock PlayerData( JetPlayer )
{
   ...

   jetJumpForce = 16.0 * 90;
   jetJumpEnergyDrain = 10;
   jetMinJumpEnergy = 25;
   jetMinJumpSpeed = 20;
   jetMaxJumpSpeed = 100;
   jetJumpSurfaceAngle = 78;
}
```

This player will not be able to jet if he has less than 25 units of energy, and 10 units will be subtracted each tick.

If PlayerData::jetJumpFore is greater than zero then $mvTriggerCount1 will activate jetting.

Falling and Landing When the player is falling they transition into the "fall" sequence. This transition doesn't occur until the player has reached a particular speed – you don't want the fall sequence to kick in if they've just gone over a small bump. This speed threshold is set by the PlayerData fallingSpeedThreshold field. By default it is set to -10.0.

When the player lands there are two possible outcomes depending on how the player is set up. With the traditional method the "land" sequence has the player start from a standing position and animates into a crouch. The playback speed of this sequence is scaled based on how hard the player hits the ground. Once the land sequence finishes playing the player does a smooth transition back into the root pose (making them effectively stand up).

Starting with 1.2 there is a new method of handling landing. Here the "land" sequence starts with the player crouching on the ground and animates getting back up. This has a look of the player hitting the ground from a fall and slowly standing back up. This new method is used when the PlayerData landSequenceTime field is given a value greater

than zero. This is the amount of time taken for the player to recover form the landing, and is also how long the land sequence will play for. As this has game play ramifications (the player may have movement constraints when landing) this timing is controlled by the datablock field rather than just the length of time of the land sequence.

Also when using the new land sequence the PlayerData transitionToLand flag indicates if the player should smoothly transition between the fall sequence and the land sequence. If set to false (the default) then there is no transition and the player appears to immediately go from falling to landing, which is usually the case when mirroring real life.

Air Control The player may optionally move itself through the air while jumping or falling. This allows the player to adjust their trajectory while in the air, and is known as air control. The PlayerData::airControl property determines what fraction of the player's normal speed they may move while in the air. By default, air control is disabled (set to 0).

Hard Impacts When the player hits something hard it is possible to trigger an impact (such as handled by Armor::onImpact()). The PlayerData minImpactSpeed is the threshold at which falling damage will be considered an impact. Any speed over this parameter will trigger an onImpact() call on the datablock. This allows for small falls to not cause any damage.

The PlayerData minLateralImpactSpeed is the threshold at which non-falling damage impacts will trigger the callback. This is separate from falling as you may not want a sprinting player that hits a wall to get hurt, but being thrown into a wall by an explosion will.

Dismounting It is possible to have the player mount another object, such as a vehicle, just like any other SceneObject. While mounted, $mvTriggerCount2 will cause the player to dismount.

Triggering a Mounted Object A Player may have other objects mounted to it, with each mounted object assigned to a slot. These Player mounted objects are known as images. See ShapeBase::mountImage(). If there is an image mounted to slot 0, $mvTriggerCount0 will trigger it. If the player dies this trigger is automatically released.

If there is an image mounted to slot 1, $mvTriggerCount1 will trigger it. Otherwise $mvTriggerCount1 will be passed along to the image in slot 0 as an alternate fire state.

Character model The following sequences are used by the Player object to animate the character. Not all of them are required, but a model should have at least the root, run, back and side animations. And please see the section on Sprinting above for how they are handled when not present.

Looping sequence played when player is running sideways right.

Looping sequence played when the player is sprinting and moving sideways. If not present then the side_right sequence is used.

Looping sequence played when player is crouched and moving sideways.

Looping sequence played when player is prone (lying down) and moving backward.

Looping sequence played when player is swimming and moving right.

Looping sequence played when player is jetting.

Sequence to control vertical arm movement (for looking) (start=full up, end=full down).

Sequence played when player is firing a heavy weapon (Based on ShapeBaseImageData).

Mounted Image Controlled 3rd Person Animation A player's 3rd person action animation sequence selection may be modified based on what images are mounted on the player. When mounting a ShapeBaseImageData, the image's imageAnimPrefix field is used to control this. If this is left blank (the default) then nothing happens to the 3rd person player – all of the sequences play as defined. If it is filled with some text (best to keep it to letters and numbers, with no spaces) then that text is added to the action animation sequence name and looked up on the player shape. For example:

A rifle ShapeBaseImageData is mounted to the player in slot 0. The rifle's datablock doesn't have an imageAnimPrefix defined, so the 3rd person player will use the standard action animation sequence names. i.e. "root", "run", "back", "crouch_root", etc.

Now a pistol ShapeBaseImageData is mounted to the player in slot 0. The pistol's datablock has imageAnimPrefix = "pistol". Now the "pistol_" (underscore is added by the system) prefix is added to each of the action animation sequence names when looking up what to play on the player's shape. So the Player class will look for "pistol_root", "pistol_run", "pistol_back", "pistol_crouch_root", etc. If any of these new prefixed names are not found on the player's shape, then we fall back to the standard action animation sequence names, such as "root", "run", etc.

In all of our T3D examples the player only mounts a single image. But Torque allows up to four images to be mounted at a time. When more than one image is mounted then the engine adds all of the prefixes together when searching for the action animation sequence name. If that combined name is not found then the engine starts removing prefixes starting with the highest slot down to the lowest slot. For example, if a player is holding a sword (slot 0) and a shield (slot 1) in each hand that are mounted as separate images (and with imageAnimPrefix's of "sword" and "shield" respectively), then the engine will search for the following names while the player is just standing there:

The first one that is found in the above order will be used.

Another example: If the player has a jet pack (slot 3 with a prefix of "jetpack") and two pistols being used akimbo style (slots 1 and 0, both with a prefix of "laserpistol") with slot 2 left open for a helmet (which is skipped as it doesn't have a prefix), then the following search order would be used:

Again, the first one that is found is used.

A player's 3rd person animation may also be modified by the weapon being used. In T3D 1.1 there are the three recoil sequences that may be triggered on the 3rd person player by the weapon's state. Starting with T3D 1.2 this becomes more generic (while still supporting the existing recoil sequence). When a ShapeBaseImageData state defines a stateShapeSequence, that sequence may be played on the player's shape (the new PlayerData allowImageStateAnimation field must be set to "true" as well). The new ShapeBaseImageData state stateScaleShapeSequence flag may also be used to indicate if this player animation sequence should have its playback rate scaled to the length of the image's state.

What exactly happens on the player depends on what else has been defined. First, there is the sequence name as passed in from the image. Then there is also the imageAnimPrefix as defined by the image. Finally, there is the generic script defined prefix that may be added with ShapeBase::setImageScriptAnimPrefix() – we're using this to pass along the current pose, but it could be used for anything. Time for an example. We want to throw a grenade that we're holding (mounted in slot 0). The weapon's state that does this has stateShapeSequence set to "throw". The grenade image itself has an imageAnimPrefix defined as "fraggrenade". Finally, the player is crouching, so Armor::onPoseChange() sets the script prefix to "crouch". The final search order goes like this:

The first of those sequences that is found is played as a new thread on the 3rd person player. As with recoil, only one of these 3rd person animation threads may be active at a time. If an image in another slot also asks to play a 3rd person sequence, the most recent request is what will play.

1st Person Arms Games that have the player hold a weapon in a 1st person view often let you see the player's arms and hands holding that weapon. Rather than requiring you to build the art for all possible combinations of character arms and weapons, T3D allows you to mix and match shapes and animation sequences.

1st person arms are an optional client-side only effect and are not used on the server. The arms are a separate shape from the normal 3rd person player shape. You reference the arms using the PlayerData "shapeNameFP" array. It is

an array as we support up to four mounted images therefore we support up to four arm shapes. However, for T3D 1.2 our examples only make use of a single set of arms for the first mounting slot as our example soldier holds a single weapon at a time.

As the arms are just regular DAE/DTS files they may get their animation sequences from anywhere. For the included 1.2 art path (see the soldier in the template projects) we decided that their sequences should come from the weapons themselves. This means that the weapons include all of the bones/nodes needed to animate the arms, but none of the arm geometry. If you take a look at art/shapes/actors/Soldier/FP/FP_SoldierArms.cs you'll see the external animation sequence references for each of the possible weapons.

As each weapon may require its own set of animation sequences (i.e. a different idle sequence for a pistol vs. a rifle) starting with T3D 1.2 a new ShapeBaseImageData field now exists: imagePrefixFP. If this field is defined for the mounted image then it is added to the sequence name as given in the current weapon state in the form of "prefix_sequence" (the underscore is added by the system). For example, the Lurker rifle has an imagePrefixFP of "Rifle". The Lurker's Ready state calls the idle sequence, so the arms will attempt to play the "Rifle_idle" sequence and if not found, they will play the "idle" sequence.

The advantage of having the prefix defined within the datablock and not making it part of the sequence names referenced directly in the weapon state machine is that you can do something like this:

Example:

```
datablock ShapeBaseImageData(Pistol1Image)
{
   imageAnimPrefixFP = "Pistol1";
   ...other data here...
   ...weapon state machine here...
};

datablock ShapeBaseImageData(Pistol2Image : Pistol1Image)
{
   imageAnimPrefixFP = "Pistol2";
};
```

You could define a new pistol (Pistol2Image) that uses the exact same state machine as Pistol1Image, but could use a slightly different set of animation sequences with a prefix of "Pistol2".

As was previously discussed with 3rd person animation above, a script-based modifier may also be added when looking up the sequence name for the arms. This is currently used to pass along the player's pose so the arm's idle sequence could have a swimming motion when in the swim pose, for example. And as with images, the arms sequence name look up uses the following order to find a sequence to play, with the first one found being used:

Finally, the arms support an "ambient" sequence that may be used for anything and will always play, if it is defined in the arm's shape.

Example PlayerData Datablock An example of a player datablock appears below:

Example:

```
datablock PlayerData(DefaultPlayerData)
{
   renderFirstPerson = false;

   computeCRC = false;

   // Third person shape
   shapeFile = "art/shapes/actors/Soldier/soldier_rigged.dae";
   cameraMaxDist = 3;
   allowImageStateAnimation = true;
```

```
   // First person arms
   imageAnimPrefixFP = "soldier";
   shapeNameFP[0] = "art/shapes/actors/Soldier/FP/FP_SoldierArms.DAE";

   canObserve = 1;
   cmdCategory = "Clients";

   cameraDefaultFov = 55.0;
   cameraMinFov = 5.0;
   cameraMaxFov = 65.0;

   debrisShapeName = "art/shapes/actors/common/debris_player.dts";
   debris = playerDebris;

   throwForce = 30;

   aiAvoidThis = 1;

   minLookAngle = "-1.2";
   maxLookAngle = "1.2";
   maxFreelookAngle = 3.0;

   mass = 120;
   drag = 1.3;
   maxdrag = 0.4;
   density = 1.1;
   maxDamage = 100;
   maxEnergy =  60;
   repairRate = 0.33;
   energyPerDamagePoint = 75;

   rechargeRate = 0.256;

   runForce = 4320;
   runEnergyDrain = 0;
   minRunEnergy = 0;
   maxForwardSpeed = 8;
   maxBackwardSpeed = 6;
   maxSideSpeed = 6;

   sprintForce = 4320;
   sprintEnergyDrain = 0;
   minSprintEnergy = 0;
   maxSprintForwardSpeed = 14;
   maxSprintBackwardSpeed = 8;
   maxSprintSideSpeed = 6;
   sprintStrafeScale = 0.25;
   sprintYawScale = 0.05;
   sprintPitchScale = 0.05;
   sprintCanJump = true;

   crouchForce = 405;
   maxCrouchForwardSpeed = 4.0;
   maxCrouchBackwardSpeed = 2.0;
   maxCrouchSideSpeed = 2.0;

   maxUnderwaterForwardSpeed = 8.4;
   maxUnderwaterBackwardSpeed = 7.8;
```

```
   maxUnderwaterSideSpeed = 7.8;

   jumpForce = "747";
   jumpEnergyDrain = 0;
   minJumpEnergy = 0;
   jumpDelay = "15";
   airControl = 0.3;

   fallingSpeedThreshold = -6.0;

   landSequenceTime = 0.33;
   transitionToLand = false;
   recoverDelay = 0;
   recoverRunForceScale = 0;

   minImpactSpeed = 10;
   minLateralImpactSpeed = 20;
   speedDamageScale = 0.4;

   boundingBox = "0.65 0.75 1.85";
   crouchBoundingBox = "0.65 0.75 1.3";
   swimBoundingBox = "1 2 2";
   pickupRadius = 1;

   // Damage location details
   boxHeadPercentage       = 0.83;
   boxTorsoPercentage      = 0.49;
   boxHeadLeftPercentage         = 0.30;
   boxHeadRightPercentage        = 0.60;
   boxHeadBackPercentage         = 0.30;
   boxHeadFrontPercentage        = 0.60;

   // Foot Prints
   decalOffset = 0.25;

   footPuffEmitter = "LightPuffEmitter";
   footPuffNumParts = 10;
   footPuffRadius = "0.25";

   dustEmitter = "LightPuffEmitter";

   splash = PlayerSplash;
   splashVelocity = 4.0;
   splashAngle = 67.0;
   splashFreqMod = 300.0;
   splashVelEpsilon = 0.60;
   bubbleEmitTime = 0.4;
   splashEmitter[0] = PlayerWakeEmitter;
   splashEmitter[1] = PlayerFoamEmitter;
   splashEmitter[2] = PlayerBubbleEmitter;
   mediumSplashSoundVelocity = 10.0;
   hardSplashSoundVelocity = 20.0;
   exitSplashSoundVelocity = 5.0;

   // Controls over slope of runnable/jumpable surfaces
   runSurfaceAngle  = 38;
   jumpSurfaceAngle = 80;
   maxStepHeight = 0.35;  //two meters
```

```
   minJumpSpeed = 20;
   maxJumpSpeed = 30;

   horizMaxSpeed = 68;
   horizResistSpeed = 33;
   horizResistFactor = 0.35;

   upMaxSpeed = 80;
   upResistSpeed = 25;
   upResistFactor = 0.3;

   footstepSplashHeight = 0.35;

   // Footstep Sounds
   FootSoftSound        = FootLightSoftSound;
   FootHardSound        = FootLightHardSound;
   FootMetalSound       = FootLightMetalSound;
   FootSnowSound        = FootLightSnowSound;
   FootShallowSound     = FootLightShallowSplashSound;
   FootWadingSound      = FootLightWadingSound;
   FootUnderwaterSound  = FootLightUnderwaterSound;

   FootBubblesSound     = FootLightBubblesSound;
   movingBubblesSound   = ArmorMoveBubblesSound;
   waterBreathSound     = WaterBreathMaleSound;

   impactSoftSound      = ImpactLightSoftSound;
   impactHardSound      = ImpactLightHardSound;
   impactMetalSound     = ImpactLightMetalSound;
   impactSnowSound      = ImpactLightSnowSound;

   impactWaterEasy      = ImpactLightWaterEasySound;
   impactWaterMedium    = ImpactLightWaterMediumSound;
   impactWaterHard      = ImpactLightWaterHardSound;

   groundImpactMinSpeed    = "45";
   groundImpactShakeFreq   = "4.0 4.0 4.0";
   groundImpactShakeAmp    = "1.0 1.0 1.0";
   groundImpactShakeDuration = 0.8;
   groundImpactShakeFalloff = 10.0;

   exitingWater         = ExitingWaterLightSound;

   observeParameters = "0.5 4.5 4.5";
   class = "armor";

   cameraMinDist = "0";
   DecalData = "PlayerFootprint";

   // Allowable Inventory Items
   mainWeapon = Lurker;

   maxInv[Lurker] = 1;
   maxInv[LurkerClip] = 20;

   maxInv[LurkerGrenadeLauncher] = 1;
   maxInv[LurkerGrenadeAmmo] = 20;
```

```
   maxInv[Ryder] = 1;
   maxInv[RyderClip] = 10;

   maxInv[ProxMine] = 5;

   maxInv[DeployableTurret] = 5;

   // available skins (see materials.cs in model folder)
   availableSkins =  "base DarkBlue DarkGreen   LightGreen  Orange   Red   Teal  Violet   Yellow";
};
```

Methods

void Player::**allowAllPoses**()

Allow all poses a chance to occur. This method resets any poses that have manually been blocked from occuring. This includes the regular pose states such as sprinting, crouch, being prone and swimming. It also includes being able to jump and jet jump. While this is allowing these poses to occur it doesn't mean that they all can due to other conditions. We're just not manually blocking them from being allowed.

void Player::**allowCrouching**(bool *state*)

Set if the Player is allowed to crouch. The default is to allow crouching unless there are other environmental concerns that prevent it. This method is mainly used to explicitly disallow crouching at any time.

Parameters **state** – Set to true to allow crouching, false to disable it.

void Player::**allowJetJumping**(bool *state*)

Set if the Player is allowed to jet jump. The default is to allow jet jumping unless there are other environmental concerns that prevent it. This method is mainly used to explicitly disallow jet jumping at any time.

Parameters **state** – Set to true to allow jet jumping, false to disable it.

void Player::**allowJumping**(bool *state*)

Set if the Player is allowed to jump. The default is to allow jumping unless there are other environmental concerns that prevent it. This method is mainly used to explicitly disallow jumping at any time.

Parameters **state** – Set to true to allow jumping, false to disable it.

void Player::**allowProne**(bool *state*)

Set if the Player is allowed to go prone. The default is to allow being prone unless there are other environmental concerns that prevent it. This method is mainly used to explicitly disallow going prone at any time.

Parameters **state** – Set to true to allow being prone, false to disable it.

void Player::**allowSprinting**(bool *state*)

Set if the Player is allowed to sprint. The default is to allow sprinting unless there are other environmental concerns that prevent it. This method is mainly used to explicitly disallow sprinting at any time.

Parameters **state** – Set to true to allow sprinting, false to disable it.

void Player::**allowSwimming**(bool *state*)

Set if the Player is allowed to swim. The default is to allow swimming unless there are other environmental concerns that prevent it. This method is mainly used to explicitly disallow swimming at any time.

Parameters **state** – Set to true to allow swimming, false to disable it.

bool Player::**checkDismountPoint**(Point3F *oldPos*, Point3F *pos*)

Check if it is safe to dismount at this position. Internally this method casts a ray from oldPos to pos to determine if it hits the terrain, an interior object, a water object, another player, a static shape, a vehicle (exluding the one currently mounted), or physical zone. If this ray is in the clear, then the player's bounding box is also checked for a collision at the pos position. If this displaced bounding box is also in the clear, then checkDismountPoint() returns true.

Parameters

- **oldPos** – The player's current position
- **pos** – The dismount position to check

Returns True if the dismount position is clear, false if not

void Player::**clearControlObject**()
Clears the player's current control object. Returns control to the player. This internally calls Player::setControlObject(0).

Example:

```
%player.clearControlObject();
echo(%player.getControlObject()); //<-- Returns 0, player assumes control
%player.setControlObject(%vehicle);
echo(%player.getControlObject()); //<-- Returns %vehicle, player controls the vehicle now.
```

int Player::**getControlObject**()
Get the current object we are controlling.

Returns object we control, or 0 if not controlling an object.

string Player::**getDamageLocation**(Point3F *pos*)
Get the named damage location and modifier for a given world position. the Player object can simulate different hit locations based on a pre-defined set of PlayerData defined percentages. These hit percentages divide up the Player's bounding box into different regions. The diagram below demonstrates how the various PlayerData properties split up the bounding volume:

Returns a string containing two words (space separated strings), where the first is a location and the second is a modifier.

int Player::**getNumDeathAnimations**()
Get the number of death animations available to this player. Death animations are assumed to be named death1-N using consecutive indices.

string Player::**getPose**()
Get the name of the player's current pose. The pose is one of the following:

•Stand - Standard movement pose.

•Sprint - Sprinting pose.

•Crouch - Crouch pose.

•Prone - Prone pose.

•Swim - Swimming pose.

Returns The current pose; one of: "Stand", "Sprint", "Crouch", "Prone", "Swim"

string Player::**getState**()
Get the name of the player's current state. The state is one of the following:

•Dead - The Player is dead.

•Mounted - The Player is mounted to an object such as a vehicle.

•Move - The Player is free to move. The usual state.

•Recover - The Player is recovering from a fall. See PlayerData::recoverDelay

Returns The current state; one of: "Dead", "Mounted", "Move", "Recover"

bool Player::**setActionThread**(string *name*, bool *hold*, bool *fsp*)

Set the main action sequence to play for this player.

The spine nodes for the Player's shape are named as follows:

- Bip01 Pelvis
- Bip01 Spine
- Bip01 Spine1
- Bip01 Spine2
- Bip01 Neck
- Bip01 Head

You cannot use setActionThread() to have the Player play one of the motion determined action animation sequences. These sequences are chosen based on how the Player moves and the Player's current pose. The names of these sequences are:

- root
- run
- side
- side_right
- crouch_root
- crouch_forward
- crouch_backward
- crouch_side
- crouch_right
- prone_root
- prone_forward
- prone_backward
- swim_root
- swim_forward
- swim_backward
- swim_left
- swim_right
- fall
- jump
- standjump
- land
- jet

If the player moves in any direction then the animation sequence set using this method will be cancelled and the chosen mation-based sequence will take over. This makes great for times when the Player cannot move, such as when mounted, or when it doesn't matter if the action sequence changes, such as waving and saluting.

Parameters

- **name** – Name of the action sequence to set
- **hold** – Set to false to get a callback on the datablock when the sequence ends (PlayerData::animationDone()). When set to true no callback is made.
- **fsp** – True if first person and none of the spine nodes in the shape should animate. False will allow the shape's spine nodes to animate.

Returns True if succesful, false if failed

Example:

```
// Place the player in a sitting position after being mounted
%player.setActionThread( "sitting", true, true );
```

bool Player::**setArmThread**(string *name*)
Set the sequence that controls the player's arms (dynamically adjusted to match look direction).

Parameters **name** – Name of the sequence to play on the player's arms.

Returns true if successful, false if failed.

bool Player::**setControlObject**(ShapeBase *obj*)
Set the object to be controlled by this player. It is possible to have the moves sent to the Player object from the GameConnection to be passed along to another object. This happens, for example when a player is mounted to a vehicle. The move commands pass through the Player and on to the vehicle (while the player remains stationary within the vehicle). With setControlObject() you can have the Player pass along its moves to any object. One possible use is for a player to move a remote controlled vehicle. In this case the player does not mount the vehicle directly, but still wants to be able to control it.

Parameters **obj** – Object to control with this player

Returns True if the object is valid, false if not

Fields

int Player::**crouchTrigger**[static]
The move trigger index used for player crouching.

int Player::**extendedMoveHeadPosRotIndex**[static]
The ExtendedMove position/rotation index used for head movements.

int Player::**imageTrigger0**[static]
The move trigger index used to trigger mounted image 0.

int Player::**imageTrigger1**[static]
The move trigger index used to trigger mounted image 1 or alternate fire on mounted image 0.

int Player::**jumpJetTrigger**[static]
The move trigger index used for player jump jetting.

int Player::**jumpTrigger**[static]
The move trigger index used for player jumping.

float Player::**maxImpulseVelocity**[static]
The maximum velocity allowed due to a single impulse.

int Player::**maxPredictionTicks**[static]
Maximum number of ticks to predict on the client from the last known move obtained from the server.

int Player::**maxWarpTicks**[static]
When a warp needs to occur due to the client being too far off from the server, this is the maximum number of ticks we'll allow the client to warp to catch up.

float Player::**minWarpTicks**[static]
Fraction of tick at which instant warp occures on the client.

int Player::**proneTrigger**[static]
The move trigger index used for player prone pose.

bool Player::**renderCollision**[static]
Determines if the player's collision mesh should be rendered. This is mainly used for the tools and debugging.

bool Player::**renderMyItems**[static]
Determines if mounted shapes are rendered or not. Used on the client side to disable the rendering of all Player mounted objects. This is mainly used for the tools or debugging.

bool Player::**renderMyPlayer**[static]
Determines if the player is rendered or not. Used on the client side to disable the rendering of all Player objects. This is mainly for the tools or debugging.

int Player::**sprintTrigger**[static]
The move trigger index used for player sprinting.

int Player::**vehicleDismountTrigger**[static]
The move trigger index used to dismount player.

PlayerData object.

Inherit: ShapeBaseData

Description Defines properties for a Player object.

Methods

void PlayerData::**animationDone**(Player *obj*)
Called on the server when a scripted animation completes.

Parameters **obj** – The Player object

void PlayerData::**doDismount**(Player *obj*)
Called when attempting to dismount the player from a vehicle. It is up to the doDismount() method to actually perform the dismount. Often there are some conditions that prevent this, such as the vehicle moving too fast.

Parameters **obj** – The Player object

void PlayerData::**onEnterLiquid**(Player *obj*, float *coverage*, string *type*)
Called when the player enters a liquid.

Parameters

- **obj** – The Player object
- **coverage** – Percentage of the player's bounding box covered by the liquid
- **type** – The type of liquid the player has entered

void PlayerData::**onEnterMissionArea**(Player *obj*)
Called when the player enters the mission area.

Parameters **obj** – The Player object

void PlayerData::**onLeaveLiquid**(Player *obj*, string *type*)
Called when the player leaves a liquid.

Parameters

- **obj** – The Player object
- **type** – The type of liquid the player has left

void PlayerData::**onLeaveMissionArea**(Player *obj*)
Called when the player leaves the mission area.

Parameters **obj** – The Player object

void PlayerData::**onPoseChange**(Player *obj*, string *oldPose*, string *newPose*)
Called when the player changes poses.

Parameters

- **obj** – The Player object
- **oldPose** – The pose the player is switching from.
- **newPose** – The pose the player is switching to.

void PlayerData::**onStartSprintMotion**(Player *obj*)
Called when the player starts moving while in a Sprint pose.

Parameters **obj** – The Player object

void PlayerData::**onStartSwim**(Player *obj*)
Called when the player starts swimming.

Parameters **obj** – The Player object

void PlayerData::**onStopSprintMotion**(Player *obj*)
Called when the player stops moving while in a Sprint pose.

Parameters **obj** – The Player object

void PlayerData::**onStopSwim**(Player *obj*)
Called when the player stops swimming.

Parameters **obj** – The Player object

Fields

float PlayerData::**airControl**
Amount of movement control the player has when in the air. This is applied as a multiplier to the player's x and y motion.

bool PlayerData::**allowImageStateAnimation**
Allow mounted images to request a sequence be played on the Player . When true a new thread is added to the player to allow for mounted images to request a sequence be played on the player through the image's state machine. It is only optional so that we don't create a TSThread on the player if we don't need to.

Point3F PlayerData::**boundingBox**
Size of the bounding box used by the player for collision. Dimensions are given as "width depth height".

float PlayerData::**boxHeadBackPercentage**
Percentage of the player's bounding box depth that represents the back side of the head. Used when computing the damage location.

float PlayerData::**boxHeadFrontPercentage**
Percentage of the player's bounding box depth that represents the front side of the head. Used when computing the damage location.

float PlayerData::**boxHeadLeftPercentage**
Percentage of the player's bounding box width that represents the left side of the head. Used when computing the damage location.

float PlayerData::**boxHeadPercentage**
Percentage of the player's bounding box height that represents the head. Used when computing the damage location.

float PlayerData::**boxHeadRightPercentage**
Percentage of the player's bounding box width that represents the right side of the head. Used when computing the damage location.

float PlayerData::**boxTorsoPercentage**
Percentage of the player's bounding box height that represents the torso. Used when computing the damage location.

float PlayerData::**bubbleEmitTime**
Time in seconds to generate bubble particles after entering the water.

Point3F PlayerData::**crouchBoundingBox**
Collision bounding box used when the player is crouching.

float PlayerData::**crouchForce**
Force used to accelerate the player when crouching.

DecalData PlayerData::**DecalData**
Decal to place on the ground for player footsteps.

float PlayerData::**decalOffset**
Distance from the center of the model to the right foot. While this defines the distance to the right foot, it is also used to place the left foot decal as well. Just on the opposite side of the player.

ParticleEmitterData PlayerData::**dustEmitter**
Emitter used to generate dust particles.

SFXTrack PlayerData::**exitingWater**
Sound to play when exiting the water with velocity gt = exitSplashSoundVelocity.

float PlayerData::**exitSplashSoundVelocity**
Minimum velocity when leaving the water for the exitingWater sound to play.

float PlayerData::**fallingSpeedThreshold**
Downward speed at which we consider the player falling.

bool PlayerData::**firstPersonShadows**
Forces shadows to be rendered in first person when renderFirstPerson is disabled. Defaults to false.

SFXTrack PlayerData::**FootBubblesSound**
Sound to play when walking in water and coverage equals 1.0 (fully underwater).

SFXTrack PlayerData::**FootHardSound**
Sound to play when walking on a surface with Material footstepSoundId 1.

SFXTrack PlayerData::**FootMetalSound**
Sound to play when walking on a surface with Material footstepSoundId 2.

ParticleEmitterData PlayerData::**footPuffEmitter**
Particle emitter used to generate footpuffs (particles created as the player walks along the ground).

int PlayerData::**footPuffNumParts**
Number of footpuff particles to generate each step. Each foot puff is randomly placed within the defined foot puff radius. This includes having footPuffNumParts set to one.

float PlayerData::**footPuffRadius**
Particle creation radius for footpuff particles. This is applied to each foot puff particle, even if footPuffNumParts is set to one. So set this value to zero if you want a single foot puff placed at exactly the same location under the player each time.

SFXTrack PlayerData::**FootShallowSound**
Sound to play when walking in water and coverage is less than footSplashHeight.

SFXTrack PlayerData::**FootSnowSound**
Sound to play when walking on a surface with Material footstepSoundId 3.

SFXTrack PlayerData::**FootSoftSound**
Sound to play when walking on a surface with Material footstepSoundId 0.

float PlayerData::**footstepSplashHeight**
Water coverage level to choose between FootShallowSound and FootWadingSound.

SFXTrack PlayerData::**FootUnderwaterSound**
Sound to play when walking in water and coverage equals 1.0 (fully underwater).

SFXTrack PlayerData::**FootWadingSound**
Sound to play when walking in water and coverage is less than 1, but gt footSplashHeight.

float PlayerData::**groundImpactMinSpeed**
Minimum falling impact speed to apply damage and initiate the camera shaking effect.

Point3F PlayerData::**groundImpactShakeAmp**
Amplitude of the camera shake effect after falling. This is how much to shake the camera.

float PlayerData::**groundImpactShakeDuration**
Duration (in seconds) of the camera shake effect after falling. This is how long to shake the camera.

float PlayerData::**groundImpactShakeFalloff**
Falloff factor of the camera shake effect after falling. This is how to fade the camera shake over the duration.

Point3F PlayerData::**groundImpactShakeFreq**
Frequency of the camera shake effect after falling. This is how fast to shake the camera.

float PlayerData::**hardSplashSoundVelocity**
Minimum velocity when entering the water for choosing between the impactWaterMedium and impactWaterHard sound to play.

float PlayerData::**horizMaxSpeed**
Maximum horizontal speed.

float PlayerData::**horizResistFactor**
Factor of resistence once horizResistSpeed has been reached.

float PlayerData::**horizResistSpeed**
Horizontal speed at which resistence will take place.

caseString PlayerData::**imageAnimPrefix**
Optional prefix to all mounted image animation sequences in third person. This defines a prefix that will be added when looking up mounted image animation sequences while in third person. It allows for the customization of a third person image based on the type of player.

caseString PlayerData::**imageAnimPrefixFP**
Optional prefix to all mounted image animation sequences in first person. This defines a prefix that will be added when looking up mounted image animation sequences while in first person. It allows for the customization of a first person image based on the type of player.

SFXTrack PlayerData::**impactHardSound**
Sound to play after falling on a surface with Material footstepSoundId 1.

SFXTrack PlayerData::**impactMetalSound**
Sound to play after falling on a surface with Material footstepSoundId 2.

SFXTrack PlayerData::**impactSnowSound**
Sound to play after falling on a surface with Material footstepSoundId 3.

SFXTrack PlayerData::**impactSoftSound**
Sound to play after falling on a surface with Material footstepSoundId 0.

SFXTrack PlayerData::**impactWaterEasy**
Sound to play when entering the water with velocity lt mediumSplashSoundVelocity.

SFXTrack PlayerData::**impactWaterHard**
Sound to play when entering the water with velocity gt = hardSplashSoundVelocity.

SFXTrack PlayerData::**impactWaterMedium**
Sound to play when entering the water with velocity gt = mediumSplashSoundVelocity and lt hardSplashSound-Velocity.

float PlayerData::**jetJumpEnergyDrain**
Energy level drained each time the player jet jumps.

float PlayerData::**jetJumpForce**
Force used to accelerate the player when a jet jump is initiated.

float PlayerData::**jetJumpSurfaceAngle**
Angle from vertical (in degrees) where the player can jet jump.

float PlayerData::**jetMaxJumpSpeed**
Maximum vertical speed before the player can no longer jet jump.

float PlayerData::**jetMinJumpEnergy**
Minimum energy level required to jet jump.

float PlayerData::**jetMinJumpSpeed**
Minimum speed needed to jet jump. If the player's own z velocity is greater than this, then it is used to scale the jet jump speed, up to jetMaxJumpSpeed.

int PlayerData::**jumpDelay**
Delay time in number of ticks ticks between jumps.

float PlayerData::**jumpEnergyDrain**
Energy level drained each time the player jumps.

float PlayerData::**jumpForce**
Force used to accelerate the player when a jump is initiated.

float PlayerData::**jumpSurfaceAngle**
Angle from vertical (in degrees) where the player can jump.

bool PlayerData::**jumpTowardsNormal**
Controls the direction of the jump impulse. When false, jumps are always in the vertical (+Z) direction. When true jumps are in the direction of the ground normal so long as the player is not directly facing the surface. If the player is directly facing the surface, then they will jump straight up.

float PlayerData::**landSequenceTime**
Time of land sequence play back when using new recover system. If greater than 0 then the legacy fall recovery system will be bypassed in favour of just playing the player's land sequence. The time to recover from a fall then becomes this parameter's time and the land sequence's playback will be scaled to match.

float PlayerData::**maxBackwardSpeed**
Maximum backward speed when running.

float PlayerData::**maxCrouchBackwardSpeed**
Maximum backward speed when crouching.

float PlayerData::**maxCrouchForwardSpeed**
Maximum forward speed when crouching.

float PlayerData::**maxCrouchSideSpeed**
Maximum sideways speed when crouching.

float PlayerData::**maxForwardSpeed**
Maximum forward speed when running.

float PlayerData::**maxFreelookAngle**
Defines the maximum left and right angles (in radians) the player can look in freelook mode.

float PlayerData::**maxJumpSpeed**
Maximum vertical speed before the player can no longer jump.

float PlayerData::**maxLookAngle**
Highest angle (in radians) the player can look.

float PlayerData::**maxProneBackwardSpeed**
Maximum backward speed when prone (laying down).

float PlayerData::**maxProneForwardSpeed**
Maximum forward speed when prone (laying down).

float PlayerData::**maxProneSideSpeed**
Maximum sideways speed when prone (laying down).

float PlayerData::**maxSideSpeed**
Maximum sideways speed when running.

float PlayerData::**maxSprintBackwardSpeed**
Maximum backward speed when sprinting.

float PlayerData::**maxSprintForwardSpeed**
Maximum forward speed when sprinting.

float PlayerData::**maxSprintSideSpeed**
Maximum sideways speed when sprinting.

float PlayerData::**maxStepHeight**
Maximum height the player can step up. The player will automatically step onto changes in ground height less than maxStepHeight. The player will collide with ground height changes greater than this.

float PlayerData::**maxTimeScale**
Maximum time scale for action animations. If an action animation has a defined ground frame, it is automatically scaled to match the player's ground velocity. This field limits the maximum time scale used even if the player's velocity exceeds it.

float PlayerData::**maxUnderwaterBackwardSpeed**
Maximum backward speed when underwater.

float PlayerData::**maxUnderwaterForwardSpeed**
Maximum forward speed when underwater.

float PlayerData::**maxUnderwaterSideSpeed**
Maximum sideways speed when underwater.

float PlayerData::**mediumSplashSoundVelocity**
Minimum velocity when entering the water for choosing between the impactWaterEasy and impactWaterMedium sounds to play.

float PlayerData::**minImpactSpeed**
Minimum impact speed to apply falling damage. This field also sets the minimum speed for the onImpact callback to be invoked.

float PlayerData::**minJumpEnergy**
Minimum energy level required to jump.

float PlayerData::**minJumpSpeed**
Minimum speed needed to jump. If the player's own z velocity is greater than this, then it is used to scale the jump speed, up to maxJumpSpeed.

float PlayerData::**minLateralImpactSpeed**
Minimum impact speed to apply non-falling damage. This field also sets the minimum speed for the onLateralImpact callback to be invoked.

float PlayerData::**minLookAngle**
Lowest angle (in radians) the player can look.

float PlayerData::**minRunEnergy**
Minimum energy level required to run or swim.

float PlayerData::**minSprintEnergy**
Minimum energy level required to sprint.

SFXTrack PlayerData::**movingBubblesSound**
Sound to play when in water and coverage equals 1.0 (fully underwater). Note that unlike FootUnderwaterSound, this sound plays even if the player is not moving around in the water.

string PlayerData::**physicsPlayerType**
Specifies the type of physics used by the player. This depends on the physics module used. An example is 'Capsule'.

float PlayerData::**pickupRadius**
Radius around the player to collide with Items in the scene (on server). Internally the pickupRadius is added to the larger side of the initial bounding box to determine the actual distance, to a maximum of 2 times the bounding box size. The initial bounding box is that used for the root pose, and therefore doesn't take into account the change in pose.

Point3F PlayerData::**proneBoundingBox**
Collision bounding box used when the player is prone (laying down).

float PlayerData::**proneForce**
Force used to accelerate the player when prone (laying down).

int PlayerData::**recoverDelay**
Number of ticks for the player to recover from falling.

float PlayerData::**recoverRunForceScale**
Scale factor applied to runForce while in the recover state. This can be used to temporarily slow the player's movement after a fall, or prevent the player from moving at all if set to zero.

bool PlayerData::**renderFirstPerson**
Flag controlling whether to render the player shape in first person view.

float PlayerData::**runEnergyDrain**
Energy value drained each tick that the player is moving. The player will not be able to move when his energy falls below minRunEnergy.

float PlayerData::**runForce**
Force used to accelerate the player when running.

float PlayerData::**runSurfaceAngle**
Maximum angle from vertical (in degrees) the player can run up.

filename PlayerData::**shapeNameFP**[4]
File name of this player's shape that will be used in conjunction with the corresponding mounted image. These optional parameters correspond to each mounted image slot to indicate a shape that is rendered in addition to the mounted image shape. Typically these are a player's arms (or arm) that is animated along with the mounted image's state animation sequences.

SplashData PlayerData::**Splash**
SplashData datablock used to create splashes when the player moves through water.

float PlayerData::**splashAngle**
Maximum angle (in degrees) from pure vertical movement in water to generate splashes.

ParticleEmitterData PlayerData::**splashEmitter**[3]
Particle emitters used to generate splash particles.

float PlayerData::**splashFreqMod**
Multipled by speed to determine the number of splash particles to generate.

float PlayerData::**splashVelEpsilon**
Minimum speed to generate splash particles.

float PlayerData::**splashVelocity**
Minimum velocity when moving through water to generate splashes.

bool PlayerData::**sprintCanJump**
Can the player jump while sprinting.

float PlayerData::**sprintEnergyDrain**
Energy value drained each tick that the player is sprinting. The player will not be able to move when his energy falls below sprintEnergyDrain.

float PlayerData::**sprintForce**
Force used to accelerate the player when sprinting.

float PlayerData::**sprintPitchScale**
Amount to scale pitch motion while sprinting.

float PlayerData::**sprintStrafeScale**
Amount to scale strafing motion vector while sprinting.

float PlayerData::**sprintYawScale**
Amount to scale yaw motion while sprinting.

Point3F PlayerData::**swimBoundingBox**
Collision bounding box used when the player is swimming.

float PlayerData::**swimForce**
Force used to accelerate the player when swimming.

bool PlayerData::**transitionToLand**
When going from a fall to a land, should we transition between the two.

float PlayerData::**upMaxSpeed**
Maximum upwards speed.

float PlayerData::**upResistFactor**
Factor of resistence once upResistSpeed has been reached.

float PlayerData::**upResistSpeed**
Upwards speed at which resistence will take place.

SFXTrack PlayerData::**waterBreathSound**
Sound to play when in water and coverage equals 1.0 (fully underwater). Note that unlike FootUnderwaterSound, this sound plays even if the player is not moving around in the water.

Projectile class for properties of individual projectiles.

Inherit: GameBase

Description Base projectile class. Uses the ProjectileData class for properties of individual projectiles.

Methods

void Projectile::**presimulate**(float *seconds*)
Updates the projectile's positional and collision information. This function will first delete the projectile if it is a server object and is outside it's ProjectileData::lifetime . Also responsible for applying gravity, determining collisions, triggering explosions, emitting trail particles, and calculating bounces if necessary.

Parameters **seconds** – Amount of time, in seconds since the simulation's start, to advance.

Example:

```
// Tell the projectile to process a simulation event, and provide the amount of time// that has
%seconds = 2.0;
%projectile.presimulate(%seconds);
```

Fields

Point3F Projectile::**initialPosition**
Starting position for the projectile.

Point3F Projectile::**initialVelocity**
Starting velocity for the projectile.

int Projectile::**sourceObject**
ID number of the object that fired the projectile.

int Projectile::**sourceSlot**
The sourceObject's weapon slot that the projectile originates from.

ProjectileData Stores properties for an individual projectile type.

Inherit: GameBaseData

Description Stores properties for an individual projectile type.

Example:

```
datablock ProjectileData(GrenadeLauncherProjectile)
{
 projectileShapeName = "art/shapes/weapons/SwarmGun/rocket.dts";
directDamage = 30;
radiusDamage = 30;
damageRadius = 5;
areaImpulse = 2000;
explosion = GrenadeLauncherExplosion;
waterExplosion = GrenadeLauncherWaterExplosion;
decal = ScorchRXDecal;
splash = GrenadeSplash;
```

```
particleEmitter = GrenadeProjSmokeTrailEmitter;
particleWaterEmitter = GrenadeTrailWaterEmitter;
muzzleVelocity = 30;
velInheritFactor = 0.3;
armingDelay = 2000;
lifetime = 10000;
fadeDelay = 4500;
bounceElasticity = 0.4;
bounceFriction = 0.3;
isBallistic = true;
gravityMod = 0.9;
lightDesc = GrenadeLauncherLightDesc;
damageType = "GrenadeDamage";
};
```

Methods

void ProjectileData::**onCollision**(Projectile *proj*, SceneObject *col*, float *fade*, Point3F *pos*, Point3F *normal*)

Called when a projectile collides with another object. This function is only called on server objects.

Parameters

- **proj** – The projectile colliding with SceneObject col.
- **col** – The SceneObject hit by the projectile.
- **fade** – The current fadeValue of the projectile, affects its visibility.
- **pos** – The position of the collision.
- **normal** – The normal of the collision.

void ProjectileData::**onExplode**(Projectile *proj*, Point3F *pos*, float *fade*)

Called when a projectile explodes. This function is only called on server objects.

Parameters

- **proj** – The exploding projectile.
- **pos** – The position of the explosion.
- **fade** – The current fadeValue of the projectile, affects its visibility.

Fields

int ProjectileData::**armingDelay**

Amount of time, in milliseconds, before the projectile will cause damage or explode on impact. This value must be equal to or less than the projectile's lifetime.

float ProjectileData::**bounceElasticity**

Influences post-bounce velocity of a projectile that does not explode on contact. Scales the velocity from a bounce after friction is taken into account. A value of 1.0 will leave it's velocity unchanged while values greater than 1.0 will increase it.

float ProjectileData::**bounceFriction**

Factor to reduce post-bounce velocity of a projectile that does not explode on contact. Reduces bounce velocity by this factor and a multiple of the tangent to impact. Used to simulate surface friction.

DecalData ProjectileData::**decal**

Decal datablock used for decals placed at projectile explosion points.

ExplosionData ProjectileData::**Explosion**

Explosion datablock used when the projectile explodes outside of water.

int ProjectileData::**fadeDelay**
Amount of time, in milliseconds, before the projectile begins to fade out. This value must be smaller than the projectile's lifetime to have an affect.

float ProjectileData::**gravityMod**
Scales the influence of gravity on the projectile. The larger this value is, the more that gravity will affect the projectile. A value of 1.0 will assume "normal" influence upon it. The magnitude of gravity is assumed to be 9.81 m/s/s

float ProjectileData::**impactForce**

bool ProjectileData::**isBallistic**
Detetmines if the projectile should be affected by gravity and whether or not it bounces before exploding.

int ProjectileData::**lifetime**
Amount of time, in milliseconds, before the projectile is removed from the simulation. Used with fadeDelay to determine the transparency of the projectile at a given time. A projectile may exist up to a maximum of 131040ms (or 4095 ticks) as defined by Projectile::MaxLivingTicks in the source code.

LightDescription ProjectileData::**lightDesc**
LightDescription datablock used for lights attached to the projectile.

float ProjectileData::**muzzleVelocity**
Amount of velocity the projectile recieves from the "muzzle" of the gun. Used with velInheritFactor to determine the initial velocity of the projectile. This value is never modified by the engine.

ParticleEmitterData ProjectileData::**ParticleEmitter**
Particle emitter datablock used to generate particles while the projectile is outside of water.

ParticleEmitterData ProjectileData::**particleWaterEmitter**
Particle emitter datablock used to generate particles while the projectile is submerged in water.

filename ProjectileData::**projectileShapeName**
File path to the model of the projectile.

Point3F ProjectileData::**scale**
Scale to apply to the projectile's size.

SFXTrack ProjectileData::**sound**
SFXTrack datablock used to play sounds while in flight.

SplashData ProjectileData::**Splash**
Splash datablock used to create splash effects as the projectile enters or leaves water.

float ProjectileData::**velInheritFactor**
Amount of velocity the projectile recieves from the source that created it. Use an amount between 0 and 1 for the best effect. This value is never modified by the engine.

ExplosionData ProjectileData::**waterExplosion**
Explosion datablock used when the projectile explodes underwater.

ProximityMine A simple proximity mine.

Inherit: Item

Description A simple proximity mine.

Proximity mines can be deployed using the world editor or thrown by an in-game object. Once armed, any Player or Vehicle object that moves within the mine's trigger area will cause it to explode.

Internally, the ProximityMine object transitions through the following states:

The shape used for the mine may optionally define the following sequences:

Example:

```
datablock ProximityMineData( SimpleMine )
{
   // ShapeBaseData fields
   category = "Weapon";
   shapeFile = "art/shapes/weapons/misc/proximityMine.dts";

   // ItemData fields
   sticky = true;

   // ProximityMineData fields
   armingDelay = 0.5;
   armingSound = MineArmedSound;

   autoTriggerDelay = 0;
   triggerOnOwner = true;
   triggerRadius = 5.0;
   triggerSpeed = 1.0;
   triggerDelay = 0.5;
   triggerSound = MineTriggeredSound;
   explosion = RocketLauncherExplosion;

   // dynamic fields
   pickUpName = "Proximity Mines";
   maxInventory = 20;

   damageType = "MineDamage"; // type of damage applied to objects in radius
   radiusDamage = 30;           // amount of damage to apply to objects in radius
   damageRadius = 8;            // search radius to damage objects when exploding
   areaImpulse = 2000;          // magnitude of impulse to apply to objects in radius
};

function ProximityMineData::onTriggered( %this, %obj, %target )
{
   echo( %this.name SPC "triggered by " @ %target.getClassName() );
}

function ProximityMineData::onExplode( %this, %obj, %position )
{
   // Damage objects within the mines damage radiusif ( %this.damageRadius > 0 )
      radiusDamage( %obj.sourceObject, %position, %this.damageRadius, %this.radiusDamage, %this.damag
}

function ProximityMineData::damage( %this, %obj, %position, %source, %amount, %damageType )
{
   // Explode if any damage is applied to the mine
   %obj.schedule(50 + getRandom(50), explode);
}

%obj = newProximityMine()
{
   dataBlock = SimpleMine;
};
```

Methods

void ProximityMine::**explode**()
Manually cause the mine to explode.

ProximityMineData .

Inherit: ItemData

Description Stores common properties for a ProximityMine.

Methods

void ProximityMineData::**onExplode**(ProximityMine *obj*, Point3F *pos*)
Callback invoked when a ProximityMine is about to explode.

Parameters

- **obj** – The ProximityMine object
- **pos** – The position of the mine explosion

void ProximityMineData::**onTriggered**(ProximityMine *obj*, SceneObject *target*)
Callback invoked when an object triggers the ProximityMine .

Parameters

- **obj** – The ProximityMine object
- **target** – The object that triggered the mine

Fields

float ProximityMineData::**armingDelay**
Delay (in seconds) from when the mine is placed to when it becomes active.

SFXTrack ProximityMineData::**armingSound**
Sound to play when the mine is armed (starts at the same time as the armed sequence if defined).

float ProximityMineData::**autoTriggerDelay**
Delay (in seconds) from arming until the mine automatically triggers and explodes, even if no object has entered the trigger area. Set to 0 to disable.

float ProximityMineData::**explosionOffset**
Offset from the mine's origin where the explosion emanates from.Sometimes a thrown mine may be slightly sunk into the ground. This can be just enough to cause the explosion to occur under the ground, especially on flat ground, which can end up blocking the explosion. This offset along the mine's 'up' normal allows you to raise the explosion origin to a better height.

float ProximityMineData::**triggerDelay**
Delay (in seconds) from when the mine is triggered until it explodes.

bool ProximityMineData::**triggerOnOwner**
Controls whether the mine can be triggered by the object that owns it. For example, a player could deploy mines that are only dangerous to other players and not himself.

float ProximityMineData::**triggerRadius**
Distance at which an activated mine will detect other objects and explode.

SFXTrack ProximityMineData::**triggerSound**
Sound to play when the mine is triggered (starts at the same time as the triggered sequence if defined).

float ProximityMineData::**triggerSpeed**
Speed above which moving objects within the trigger radius will trigger the mine.

SceneObject A networkable object that exists in the 3D world.

Inherit: NetObject

Description A networkable object that exists in the 3D world.

The SceneObject class provides the foundation for 3D objects in the Engine. It exposes the functionality for:

You do not typically work with SceneObjects themselves. The SceneObject provides a reference within the game world (the scene), but does not render to the client on its own. The same is true of collision detection beyond that of the bounding box. Instead you use one of the many classes that derrive from SceneObject, such as TSStatic.

Difference Between setHidden() and isRenderEnabled When it comes time to decide if a SceneObject should render or not, there are two methods that can stop the SceneObject from rendering at all. You need to be aware of the differences between these two methods as they impact how the SceneObject is networked from the server to the client.

The first method of manually controlling if a SceneObject is rendered is through its SceneObject::isRenderEnabled property. When set to false the SceneObject is considered invisible but still present within the scene. This means it still takes part in collisions and continues to be networked.

The second method is using the setHidden() method. This will actually remove a SceneObject from the scene and it will no longer be networked from the server to the cleint. Any client-side ghost of the object will be deleted as the server no longer considers the object to be in scope.

Methods

Point3F `SceneObject::getEulerRotation()`
Get Euler rotation of this object.

Returns the orientation of the object in the form of rotations around the X, Y and Z axes in degrees.

VectorF `SceneObject::getForwardVector()`
Get the direction this object is facing.

Returns a vector indicating the direction this object is facing.

TransformF `SceneObject::getInverseTransform()`
Get the object's inverse transform.

Returns the inverse transform of the object

int `SceneObject::getMountedObject` (int *slot*)
Get the object mounted at a particular slot.

Parameters **`slot`** – mount slot index to query

Returns ID of the object mounted in the slot, or 0 if no object.

int `SceneObject::getMountedObjectCount()`
Get the number of objects mounted to us.

Returns the number of mounted objects.

int `SceneObject::getMountedObjectNode` (int *slot*)
Get the mount node index of the object mounted at our given slot.

Parameters **`slot`** – mount slot index to query

Returns index of the mount node used by the object mounted in this slot.

int `SceneObject::getMountNodeObject` (int *node*)
Get the object mounted at our given node index.

Parameters **node** – mount node index to query

Returns ID of the first object mounted at the node, or 0 if none found.

Box3F SceneObject::**getObjectBox**()
Get the object's bounding box (relative to the object's origin).

Returns six fields, two Point3Fs, containing the min and max points of the objectbox.

int SceneObject::**getObjectMount**()
Get the object we are mounted to.

Returns the SimObjectID of the object we're mounted to, or 0 if not mounted.

Point3F SceneObject::**getPosition**()
Get the object's world position. Reimplemented in Camera .

Returns the current world position of the object

VectorF SceneObject::**getRightVector**()
Get the right vector of the object.

Returns a vector indicating the right direction of this object.

Point3F SceneObject::**getScale**()
Get the object's scale.

Returns object scale as a Point3F

TransformF SceneObject::**getTransform**()
Get the object's transform.

Returns the current transform of the object

int SceneObject::**getType**()
Return the type mask for this object.

Returns The numeric type mask for the object.

VectorF SceneObject::**getUpVector**()
Get the up vector of the object.

Returns a vector indicating the up direction of this object.

Box3F SceneObject::**getWorldBox**()
Get the object's world bounding box.

Returns six fields, two Point3Fs, containing the min and max points of the worldbox.

Point3F SceneObject::**getWorldBoxCenter**()
Get the center of the object's world bounding box.

Returns the center of the world bounding box for this object.

bool SceneObject::**isGlobalBounds**()
Check if this object has a global bounds set. If global bounds are set to be true, then the object is assumed to have an infinitely large bounding box for collision and rendering purposes.

Returns true if the object has a global bounds.

bool SceneObject::**isMounted**()
Check if we are mounted to another object.

Returns true if mounted to another object, false if not mounted.

bool SceneObject::**mountObject**(SceneObject *objB*, int *slot*, TransformF *txfm*)
Mount objB to this object at the desired slot with optional transform.

Parameters

- **objB** – Object to mount onto us
- **slot** – Mount slot ID
- **txfm** – (optional) mount offset transform

Returns true if successful, false if failed (objB is not valid)

void SceneObject::**setScale**(Point3F *scale*)
Set the object's scale.

Parameters **scale** – object scale to set

void SceneObject::**setTransform**(TransformF *txfm*)
Set the object's transform (orientation and position).

Parameters **txfm** – object transform to set

void SceneObject::**unmount**()
Unmount us from the currently mounted object if any.

bool SceneObject::**unmountObject**(SceneObject *target*)
Unmount an object from ourselves.

Parameters **target** – object to unmount

Returns true if successful, false if failed

Fields

bool SceneObject::**isRenderEnabled**
Controls client-side rendering of the object.

bool SceneObject::**isSelectionEnabled**
Determines if the object may be selected from wihin the Tools.

int SceneObject::**mountNode**
Node we are mounted to.

pid SceneObject::**mountPID**
PersistentID of object we are mounted to. Unlike the SimObjectID that is determined at run time, the PersistentID of an object is saved with the level/mission and may be used to form a link between objects.

MatrixPosition SceneObject::**mountPos**
Position we are mounted at (object space of our mount object).

MatrixRotation SceneObject::**mountRot**
Rotation we are mounted at (object space of our mount object).

MatrixPosition SceneObject::**position**
Object world position.

MatrixRotation SceneObject::**rotation**
Object world orientation.

Point3F SceneObject::**scale**
Object world scale.

ShapeBase A scriptable, renderable shape.

Inherit: GameBase

Description A scriptable, renderable shape.

ShapeBase is the renderable shape from which most of the scriptable, game objects are derived, including the Player, Vehicle and Item classes. ShapeBase provides collision detection, audio channels, and animation as well as damage (and damage states), energy, and the ability to mount Images and objects.

ShapeBase objects are not normally instantiated in the scene; derived classes such as Player, WheeledVehicle, and, StaticShape are used instead. But ShapeBase (and the associated datablock, ShapeBaseData) may be used to provide functionality common to all derived objects.

A ShapeBase object consists of a DTS or DAE shape file. This file has the following requirements:

Nodes

Sequences Indicating Condition

Detail Levels

Control Object Generally in a Torque game, each client is in control of a single game object (such as a Player in an FPS game, or a WheeledVehicle in a racing game). In a game where the client has control over multiple objects (such as units in an RTS), the control object may be the Camera that determines the client's view of the world (although in general, the client's camera object does not need to be the same as the control object).

The object controlled by the client is important for several reasons:

Energy/Damage ShapeBase includes basic enery and damage systems that may be used by derived classes as required. For example, the Player class uses energy to determine whether the character is capabable of running and jumping, which can be used to mimic the character getting tired and having to rest before continuing. The Player class also uses the damage system PlayerData::onDestroyed callback to trigger a death animation. The Vehicle classes use the current damage level to trigger particle emitters, so a vehicle could progressively generate more smoke as it becomes more damaged.

ShapeBase also includes parameters to 'blow up' the object when it is Destroyed (damage level above ShapeBaseData::destroyedLevel). Blowing up an object can generate an explosion and debris, as well as exclude the object from rendering.

Parameters to control the object's energy and damage functionality can be found in the ShapeBaseData datablock.

Methods

void `ShapeBase::`**`applyDamage`**(float *amount*)

Increment the current damage level by the specified amount.

Parameters **`amount`** – value to add to current damage level

bool `ShapeBase::`**`applyImpulse`**(Point3F *pos*, Point3F *vec*)

Apply an impulse to the object.

Parameters

- **`pos`** – world position of the impulse
- **`vec`** – impulse momentum (velocity * mass)

Returns true

void `ShapeBase::`**`applyRepair`**(float *amount*)

Repair damage by the specified amount. Note that the damage level is only reduced by repairRate per tick, so it may take several ticks for the total repair to complete.

Parameters **`amount`** – total repair value (subtracted from damage level over time)

void ShapeBase::**blowUp**()
Explodes an object into pieces.

bool ShapeBase::**canCloak**()
Check if this object can cloak.

Returns true

void ShapeBase::**changeMaterial**(string *mapTo*, Material *oldMat*, Material *newMat*)
Change one of the materials on the shape. This method changes materials per mapTo with others. The material that is being replaced is mapped to unmapped_mat as a part of this transition.

Parameters

- **mapTo** – the name of the material target to remap (from getTargetName)
- **oldMat** – the old Material that was mapped
- **newMat** – the new Material to map

Example:

```
// remap the first material in the shape
%mapTo = %obj.getTargetName( 0 );
%obj.changeMaterial( %mapTo, 0, MyMaterial );
```

bool ShapeBase::**destroyThread**(int *slot*)
Destroy an animation thread, which prevents it from playing.

Parameters **slot** – thread slot to destroy

Returns true if successful, false if failed

void ShapeBase::**dumpMeshVisibility**()
Print a list of visible and hidden meshes in the shape to the console for debugging purposes.

Point3F ShapeBase::**getAIRepairPoint**()
Get the position at which the AI should stand to repair things. If the shape defines a node called "AIRepairNode", this method will return the current world position of that node, otherwise "0 0 0".

Returns the AI repair position

float ShapeBase::**getCameraFov**()
Returns the vertical field of view in degrees for this object if used as a camera.

Returns ShapeBaseData::cameraDefaultFov

int ShapeBase::**getControllingClient**()
Get the client (if any) that controls this object. The controlling client is the one that will send moves to us to act on.

Returns , or 0 if this object is not controlled by any client.

int ShapeBase::**getControllingObject**()
Get the object (if any) that controls this object.

Returns object, or 0 if this object is not controlled by another object.

float ShapeBase::**getDamageFlash**()
Get the damage flash level.

Returns flash level

float ShapeBase::**getDamageLevel**()
Get the object's current damage level.

Returns damage level

float ShapeBase::**getDamagePercent**()
Get the object's current damage level as a percentage of maxDamage.

Returns damageLevel / datablock.maxDamage

string ShapeBase::**getDamageState**()
Get the object's damage state.

Returns the damage state; one of "Enabled", "Disabled", "Destroyed"

float ShapeBase::**getDefaultCameraFov**()
Returns the default vertical field of view in degrees for this object if used as a camera.

Returns Default FOV

float ShapeBase::**getEnergyLevel**()
Get the object's current energy level.

Returns energy level

float ShapeBase::**getEnergyPercent**()
Get the object's current energy level as a percentage of maxEnergy.

Returns energyLevel / datablock.maxEnergy

Point3F ShapeBase::**getEyePoint**()
Get the position of the 'eye' for this object. If the object model has a node called 'eye', this method will return that node's current world position, otherwise it will return the object's current world position.

Returns the eye position for this object

TransformF ShapeBase::**getEyeTransform**()
Get the 'eye' transform for this object. If the object model has a node called 'eye', this method will return that node's current transform, otherwise it will return the object's current transform.

Returns the eye transform for this object

VectorF ShapeBase::**getEyeVector**()
Get the forward direction of the 'eye' for this object. If the object model has a node called 'eye', this method will return that node's current forward direction vector, otherwise it will return the object's current forward direction vector.

Returns the eye vector for this object

bool ShapeBase::**getImageAltTrigger**(int *slot*)
Get the alt trigger state of the Image mounted in the specified slot.

Parameters **slot** – Image slot to query

Returns the Image's current alt trigger state

bool ShapeBase::**getImageAmmo**(int *slot*)
Get the ammo state of the Image mounted in the specified slot.

Parameters **slot** – Image slot to query

Returns the Image's current ammo state

bool ShapeBase::**getImageGenericTrigger**(int *slot*, int *trigger*)
Get the generic trigger state of the Image mounted in the specified slot.

Parameters

- **slot** – Image slot to query

- **trigger** – Generic trigger number

Returns the Image's current generic trigger state

bool ShapeBase::**getImageLoaded**(int *slot*)
Get the loaded state of the Image mounted in the specified slot.

Parameters **slot** – Image slot to query

Returns the Image's current loaded state

string ShapeBase::**getImageScriptAnimPrefix**(int *slot*)
Get the script animation prefix of the Image mounted in the specified slot.

Parameters **slot** – Image slot to query

Returns the Image's current script animation prefix

int ShapeBase::**getImageSkinTag**(int *slot*)
Get the skin tag ID for the Image mounted in the specified slot.

Parameters **slot** – Image slot to query

Returns the skinTag value passed to mountImage when the image was mounted

string ShapeBase::**getImageState**(int *slot*)
Get the name of the current state of the Image in the specified slot.

Parameters **slot** – Image slot to query

Returns name of the current Image state, or "Error" if slot is invalid

bool ShapeBase::**getImageTarget**(int *slot*)
Get the target state of the Image mounted in the specified slot.

Parameters **slot** – Image slot to query

Returns the Image's current target state

bool ShapeBase::**getImageTrigger**(int *slot*)
Get the trigger state of the Image mounted in the specified slot.

Parameters **slot** – Image slot to query

Returns the Image's current trigger state

string ShapeBase::**getLookAtPoint**(float *distance*, int *typeMask*)
Get the world position this object is looking at. Casts a ray from the eye and returns information about what the ray hits.

Parameters

- **distance** – maximum distance of the raycast
- **typeMask** – typeMask of objects to include for raycast collision testing

Returns look-at information as "Object HitX HitY HitZ [Material]" or empty string for no hit

Example:

```
%lookat = %obj.getLookAtPoint();
echo( "Looking at: " @ getWords( %lookat, 1, 3 ) );
```

float ShapeBase::**getMaxDamage**()
Get the object's maxDamage level.

Returns datablock.maxDamage

string ShapeBase::**getModelFile**()
Get the model filename used by this shape.

Returns the shape filename

int ShapeBase::**getMountedImage**(int *slot*)
Get the Image mounted in the specified slot.

Parameters **slot** – Image slot to query

Returns datablock mounted in the slot, or 0 if no Image is mounted there.

int ShapeBase::**getMountSlot**(ShapeBaseImageData *image*)
Get the first slot the given datablock is mounted to on this object.

Parameters **image** – ShapeBaseImageData datablock to query

Returns index of the first slot the Image is mounted in, or -1 if the Image is not mounted in any slot on this object.

Point3F ShapeBase::**getMuzzlePoint**(int *slot*)
Get the muzzle position of the Image mounted in the specified slot. If the Image shape contains a node called 'muzzlePoint', then the muzzle position is the position of that node in world space. If no such node is specified, the slot's mount node is used instead.

Parameters **slot** – Image slot to query

Returns the muzzle position, or "0 0 0" if the slot is invalid

VectorF ShapeBase::**getMuzzleVector**(int *slot*)
Get the muzzle vector of the Image mounted in the specified slot. If the Image shape contains a node called 'muzzlePoint', then the muzzle vector is the forward direction vector of that node's transform in world space. If no such node is specified, the slot's mount node is used instead. If the correctMuzzleVector flag (correctMuzzleVectorTP in 3rd person) is set in the Image, the muzzle vector is computed to point at whatever object is right in front of the object's 'eye' node.

Parameters **slot** – Image slot to query

Returns the muzzle vector, or "0 1 0" if the slot is invalid

int ShapeBase::**getPendingImage**(int *slot*)
Get the Image that will be mounted next in the specified slot. Calling mountImage when an Image is already mounted does one of two things: This command retrieves the ID of the pending Image (2nd case above).

Parameters **slot** – Image slot to query

Returns datablock, or 0 if none.

float ShapeBase::**getRechargeRate**()
Get the current recharge rate.

Returns the recharge rate (per tick)

float ShapeBase::**getRepairRate**()
Get the per-tick repair amount.

Returns the current value to be subtracted from damage level each tick

string ShapeBase::**getShapeName**()
Get the name of the shape.

Returns the name of the shape

string ShapeBase::**getSkinName**()
Get the name of the skin applied to this shape.

Returns the name of the skin

TransformF ShapeBase::**getSlotTransform**(int *slot*)
Get the world transform of the specified mount slot.

Parameters **slot** – Image slot to query

Returns the mount transform

int ShapeBase::**getTargetCount**()
Get the number of materials in the shape.

Returns the number of materials in the shape.

string ShapeBase::**getTargetName**(int *index*)
Get the name of the indexed shape material.

Parameters **index** – index of the material to get (valid range is 0 - getTargetCount()-1).

Returns the name of the indexed material.

VectorF ShapeBase::**getVelocity**()
Get the object's current velocity. Reimplemented in Camera .

Returns the current velocity

float ShapeBase::**getWhiteOut**()
Get the white-out level.

Returns white-out level

bool ShapeBase::**hasImageState**(int *slot*, string *state*)
Check if the given state exists on the mounted Image.

Parameters

- **slot** – Image slot to query
- **state** – Image state to check for

Returns true if the Image has the requested state defined.

bool ShapeBase::**isCloaked**()
Check if this object is cloaked.

Returns true if cloaked, false if not

bool ShapeBase::**isDestroyed**()
Check if the object is in the Destroyed damage state.

Returns true if damage state is "Destroyed", false if not

bool ShapeBase::**isDisabled**()
Check if the object is in the Disabled or Destroyed damage state.

Returns true if damage state is not "Enabled", false if it is

bool ShapeBase::**isEnabled**()
Check if the object is in the Enabled damage state.

Returns true if damage state is "Enabled", false if not

bool ShapeBase::**isHidden**()
Check if the object is hidden.

Returns true if the object is hidden, false if visible.

bool ShapeBase::**isImageFiring**(int *slot*)
Check if the current Image state is firing.

Parameters **slot** – Image slot to query

Returns true if the current Image state in this slot has the 'stateFire' flag set.

bool ShapeBase::**isImageMounted**(ShapeBaseImageData *image*)
Check if the given datablock is mounted to any slot on this object.

Parameters **image** – ShapeBaseImageData datablock to query

Returns true if the Image is mounted to any slot, false otherwise.

bool ShapeBase::**mountImage**(ShapeBaseImageData *image*, int *slot*, bool *loaded*, string *skinTag*)
Mount a new Image.

Parameters

- **image** – the Image to mount
- **slot** – Image slot to mount into (valid range is 0 - 3)
- **loaded** – initial loaded state for the Image
- **skinTag** – tagged string to reskin the mounted Image

Returns true if successful, false if failed

Example:

```
%player.mountImage( PistolImage, 1 );
%player.mountImage( CrossbowImage, 0, false );
%player.mountImage( RocketLauncherImage, 0, true, blue );
```

bool ShapeBase::**pauseThread**(int *slot*)
Pause an animation thread. If restarted using playThread, the animation will resume from the paused position.

Parameters **slot** – thread slot to stop

Returns true if successful, false if failed

bool ShapeBase::**playAudio**(int *slot*, SFXTrack *track*)
Attach a sound to this shape and start playing it.

Parameters

- **slot** – Audio slot index for the sound (valid range is 0 - 3)
- **track** – SFXTrack to play

Returns true if the sound was attached successfully, false if failed

bool ShapeBase::**playThread**(int *slot*, string *name*)
Start a new animation thread, or restart one that has been paused or stopped.

Parameters

- **slot** – thread slot to play. Valid range is 0 - 3)
- **name** – name of the animation sequence to play in this slot. If not specified, the paused or stopped thread in this slot will be resumed.

Returns true if successful, false if failed

Example:

```
%obj.playThread( 0, "ambient" );      // Play the ambient sequence in slot 0
%obj.setThreadTimeScale( 0, 0.5 );    // Play at half-speed
%obj.pauseThread( 0 );                // Pause the sequence
%obj.playThread( 0 );                 // Resume playback
%obj.playThread( 0, "spin" );         // Replace the sequence in slot 0
```

void ShapeBase::**setAllMeshesHidden**(bool *hide*)
Set the hidden state on all the shape meshes. This allows you to hide all meshes in the shape, for example, and then only enable a few.

Parameters **hide** – new hidden state for all meshes

void ShapeBase::**setCameraFov**(float *fov*)
Set the vertical field of view in degrees for this object if used as a camera.

Parameters **fov** – new FOV value

void ShapeBase::**setCloaked**(bool *cloak*)
Set the cloaked state of this object. When an object is cloaked it is not rendered.

Parameters **cloak** – true to cloak the object, false to uncloak

void ShapeBase::**setDamageFlash**(float *level*)
Set the damage flash level. Damage flash may be used as a postfx effect to flash the screen when the client is damaged.

Parameters **level** – flash level (0-1)

void ShapeBase::**setDamageLevel**(float *level*)
Set the object's current damage level.

Parameters **level** – new damage level

bool ShapeBase::**setDamageState**(string *state*)
Set the object's damage state.

Parameters **state** – should be one of "Enabled", "Disabled", "Destroyed"

Returns true if successful, false if failed

void ShapeBase::**setDamageVector**(Point3F *vec*)
Set the damage direction vector. Currently this is only used to initialise the explosion if this object is blown up.

Parameters **vec** – damage direction vector

Example:

```
%obj.setDamageVector( "0 0 1" );
```

void ShapeBase::**setEnergyLevel**(float *level*)
Set this object's current energy level.

Parameters **level** – new energy level

void ShapeBase::**setHidden**(bool *show*)
Add or remove this object from the scene. When removed from the scene, the object will not be processed or rendered. Reimplemented from SimObject .

Parameters **show** – False to hide the object, true to re-show it

bool ShapeBase::**setImageAltTrigger**(int *slot*, bool *state*)
Set the alt trigger state of the Image mounted in the specified slot.

Parameters

- **slot** – Image slot to modify
- **state** – new alt trigger state for the Image

Returns the Image's new alt trigger state

bool ShapeBase::**setImageAmmo**(int *slot*, bool *state*)
Set the ammo state of the Image mounted in the specified slot.

Parameters

- **slot** – Image slot to modify
- **state** – new ammo state for the Image

Returns the Image's new ammo state

int ShapeBase::**setImageGenericTrigger**(int *slot*, int *trigger*, bool *state*)
Set the generic trigger state of the Image mounted in the specified slot.

Parameters

- **slot** – Image slot to modify
- **trigger** – Generic trigger number
- **state** – new generic trigger state for the Image

Returns the Image's new generic trigger state or -1 if there was a problem.

bool ShapeBase::**setImageLoaded**(int *slot*, bool *state*)
Set the loaded state of the Image mounted in the specified slot.

Parameters

- **slot** – Image slot to modify
- **state** – new loaded state for the Image

Returns the Image's new loaded state

void ShapeBase::**setImageScriptAnimPrefix**(int *slot*, string *prefix*)
Set the script animation prefix for the Image mounted in the specified slot. This is used to further modify the prefix used when deciding which animation sequence to play while this image is mounted.

Parameters

- **slot** – Image slot to modify
- **prefix** – The prefix applied to the image

bool ShapeBase::**setImageTarget**(int *slot*, bool *state*)
Set the target state of the Image mounted in the specified slot.

Parameters

- **slot** – Image slot to modify
- **state** – new target state for the Image

Returns the Image's new target state

bool ShapeBase::**setImageTrigger**(int *slot*, bool *state*)
Set the trigger state of the Image mounted in the specified slot.

Parameters

- **slot** – Image slot to modify

- **state** – new trigger state for the Image

Returns the Image's new trigger state

void ShapeBase::**setInvincibleMode**(float *time*, float *speed*)
Setup the invincible effect. This effect is used for HUD feedback to the user that they are invincible.

Parameters

- **time** – duration in seconds for the invincible effect
- **speed** – speed at which the invincible effect progresses

void ShapeBase::**setMeshHidden**(string *name*, bool *hide*)
Set the hidden state on the named shape mesh.

Parameters

- **name** – name of the mesh to hide/show
- **hide** – new hidden state for the mesh

void ShapeBase::**setRechargeRate**(float *rate*)
Set the recharge rate. The recharge rate is added to the object's current energy level each tick, up to the maxEnergy level set in the ShapeBaseData datablock.

Parameters **rate** – the recharge rate (per tick)

void ShapeBase::**setRepairRate**(float *rate*)
Set amount to repair damage by each tick. Note that this value is separate to the repairRate field in ShapeBaseData . This value will be subtracted from the damage level each tick, whereas the ShapeBaseData field limits how much of the applyRepair value is subtracted each tick. Both repair types can be active at the same time.

Parameters **rate** – value to subtract from damage level each tick (must be > 0)

void ShapeBase::**setShapeName**(string *name*)
Set the name of this shape.

Parameters **name** – new name for the shape

void ShapeBase::**setSkinName**(string *name*)
Apply a new skin to this shape. 'Skinning' the shape effectively renames the material targets, allowing different materials to be used on different instances of the same model.

Parameters **name** – name of the skin to apply

bool ShapeBase::**setThreadDir**(int *slot*, bool *fwd*)
Set the playback direction of an animation thread.

Parameters

- **slot** – thread slot to modify
- **fwd** – true to play the animation forwards, false to play backwards

Returns true if successful, false if failed

bool ShapeBase::**setThreadPosition**(int *slot*, float *pos*)
Set the position within an animation thread.

Parameters

- **slot** – thread slot to modify
- **pos** – position within thread

Returns true if successful, false if failed

bool ShapeBase::**setThreadTimeScale**(int *slot*, float *scale*)
Set the playback time scale of an animation thread.

Parameters

- **slot** – thread slot to modify
- **scale** – new thread time scale (1=normal speed, 0.5=half speed etc)

Returns true if successful, false if failed

bool ShapeBase::**setVelocity**(Point3F *vel*)
Set the object's velocity.

Parameters **vel** – new velocity for the object

Returns true

void ShapeBase::**setWhiteOut**(float *level*)
Set the white-out level. White-out may be used as a postfx effect to brighten the screen in response to a game event.

Parameters **level** – flash level (0-1)

void ShapeBase::**startFade**(int *time*, int *delay*, bool *fadeOut*)
Fade the object in or out without removing it from the scene. A faded out object is still in the scene and can still be collided with, so if you want to disable collisions for this shape after it fades out use setHidden to temporarily remove this shape from the scene.

Parameters

- **time** – duration of the fade effect in ms
- **delay** – delay in ms before the fade effect begins
- **fadeOut** – true to fade-out to invisible, false to fade-in to full visibility

bool ShapeBase::**stopAudio**(int *slot*)
Stop a sound started with playAudio.

Parameters **slot** – audio slot index (started with playAudio)

Returns true if the sound was stopped successfully, false if failed

bool ShapeBase::**stopThread**(int *slot*)
Stop an animation thread. If restarted using playThread, the animation will start from the beginning again.

Parameters **slot** – thread slot to stop

Returns true if successful, false if failed

bool ShapeBase::**unmountImage**(int *slot*)
Unmount the mounted Image in the specified slot.

Parameters **slot** – Image slot to unmount

Returns true if successful, false if failed

float ShapeBase::**validateCameraFov**(float *fov*)
Called on the server when the client has requested a FOV change. When the client requests that its field of view should be changed (because they want to use a sniper scope, for example) this new FOV needs to be validated by the server. This method is called if it exists (it is optional) to validate the requested FOV, and modify it if necessary. This could be as simple as checking that the FOV falls within a correct range, to making sure that the FOV matches the capabilities of the current weapon. Following this method, ShapeBase ensures that the given FOV still falls within the datablock's cameraMinFov and cameraMaxFov. If that is good enough for your purposes, then you do not need to define the validateCameraFov() callback for your ShapeBase .

Parameters **fov** – The FOV that has been requested by the client.

Returns The FOV as validated by the server.

Fields

bool ShapeBase::**isAIControlled**

Is this object AI controlled. If True then this object is considered AI controlled and not player controlled.

string ShapeBase::**skin**

The skin applied to the shape. 'Skinning' the shape effectively renames the material targets, allowing different materials to be used on different instances of the same model. Using getSkinName() and setSkinName() is equivalent to reading and writing the skin field directly. Any material targets that start with the old skin name have that part of the name replaced with the new skin name. The initial old skin name is "base". For example, if a new skin of "blue" was applied to a model that had material targets base_body and face , the new targets would be blue_body and face . Note that face was not renamed since it did not start with the old skin name of "base". To support models that do not use the default "base" naming convention, you can also specify the part of the name to replace in the skin field itself. For example, if a model had a material target called shapemat , we could apply a new skin "shape=blue", and the material target would be renamed to bluemat (note "shape" has been replaced with "blue"). Multiple skin updates can also be applied at the same time by separating them with a semicolon. For example: "base=blue;face=happy_face". Material targets are only renamed if an existing Material maps to that name, or if there is a diffuse texture in the model folder with the same name as the new target.

ShapeBaseData object.

Inherit: GameBaseData

Description Defines properties for a ShapeBase object.

Methods

bool ShapeBaseData::**checkDeployPos**(TransformF *txfm*)

Check if there is the space at the given transform is free to spawn into. The shape's bounding box volume is used to check for collisions at the given world transform. Only interior and static objects are checked for collision.

Parameters **txfm** – Deploy transform to check

Returns True if the space is free, false if there is already something in the way.

TransformF ShapeBaseData::**getDeployTransform**(Point3F *pos*, Point3F *normal*)

Helper method to get a transform from a position and vector (suitable for use with setTransform).

Parameters

- **pos** – Desired transform position
- **normal** – Vector of desired direction

Returns The deploy transform

void ShapeBaseData::**onCollision**(ShapeBase *obj*, SceneObject *collObj*, VectorF *vec*, float *len*)

Called when we collide with another object.

Parameters

- **obj** – The ShapeBase object
- **collObj** – The object we collided with
- **vec** – Collision impact vector

- **len** – Length of the impact vector

void ShapeBaseData::**onDamage**(ShapeBase *obj*, float *delta*)
Called when the object is damaged.

Parameters

- **obj** – The ShapeBase object
- **obj** – The ShapeBase object
- **delta** – The amount of damage received.

void ShapeBaseData::**onDestroyed**(ShapeBase *obj*, string *lastState*)
Called when the object damage state changes to Destroyed.

Parameters

- **obj** – The ShapeBase object
- **lastState** – The previous damage state

void ShapeBaseData::**onDisabled**(ShapeBase *obj*, string *lastState*)
Called when the object damage state changes to Disabled.

Parameters

- **obj** – The ShapeBase object
- **lastState** – The previous damage state

void ShapeBaseData::**onEnabled**(ShapeBase *obj*, string *lastState*)
Called when the object damage state changes to Enabled.

Parameters

- **obj** – The ShapeBase object
- **lastState** – The previous damage state

void ShapeBaseData::**onEndSequence**(ShapeBase *obj*, int *slot*)
Called when a thread playing a non-cyclic sequence reaches the end of the sequence.

Parameters

- **obj** – The ShapeBase object
- **slot** – Thread slot that finished playing

void ShapeBaseData::**onForceUncloak**(ShapeBase *obj*, string *reason*)
Called when the object is forced to uncloak.

Parameters

- **obj** – The ShapeBase object
- **reason** – String describing why the object was uncloaked

void ShapeBaseData::**onImpact**(ShapeBase *obj*, SceneObject *collObj*, VectorF *vec*, float *len*)
Called when we collide with another object beyond some impact speed. The Player class makes use of this callback when a collision speed is more than PlayerData::minImpactSpeed .

Parameters

- **obj** – The ShapeBase object
- **collObj** – The object we collided with
- **vec** – Collision impact vector

- **len** – Length of the impact vector

void ShapeBaseData::**onTrigger**(ShapeBase *obj*, int *index*, bool *state*)
Called when a move trigger input changes state.

Parameters

- **obj** – The ShapeBase object
- **index** – Index of the trigger that changed
- **state** – New state of the trigger

Fields

bool ShapeBaseData::**cameraCanBank**
If the derrived class supports it, allow the camera to bank.

float ShapeBaseData::**cameraDefaultFov**
The default camera vertical FOV in degrees.

float ShapeBaseData::**cameraMaxDist**
The maximum distance from the camera to the object. Used when computing a custom camera transform for this object.

float ShapeBaseData::**cameraMaxFov**
The maximum camera vertical FOV allowed in degrees.

float ShapeBaseData::**cameraMinDist**
The minimum distance from the camera to the object. Used when computing a custom camera transform for this object.

float ShapeBaseData::**cameraMinFov**
The minimum camera vertical FOV allowed in degrees.

bool ShapeBaseData::**computeCRC**
If true, verify that the CRC of the client's shape model matches the server's CRC for the shape model when loaded by the client.

string ShapeBaseData::**cubeReflectorDesc**
References a ReflectorDesc datablock that defines performance and quality properties for dynamic reflections.

DebrisData ShapeBaseData::**Debris**
Debris to generate when this shape is blown up.

filename ShapeBaseData::**debrisShapeName**
The DTS or DAE model to use for auto-generated breakups.

float ShapeBaseData::**density**
Shape density. Used when computing buoyancy when in water.

float ShapeBaseData::**destroyedLevel**
Damage level above which the object is destroyed. When the damage level increases above this value, the object damage state is set to "Destroyed".

float ShapeBaseData::**disabledLevel**
Damage level above which the object is disabled. Currently unused.

float ShapeBaseData::**drag**
Drag factor. Reduces velocity of moving objects.

ExplosionData ShapeBaseData::**Explosion**
Explosion to generate when this shape is blown up.

bool ShapeBaseData::**firstPersonOnly**
Flag controlling whether the view from this object is first person only.

bool ShapeBaseData::**inheritEnergyFromMount**
Flag controlling whether to manage our own energy level, or to use the energy level of the object we are mounted to.

bool ShapeBaseData::**isInvincible**
Invincible flag; when invincible, the object cannot be damaged or repaired.

float ShapeBaseData::**mass**
Shape mass. Used in simulation of moving objects.

float ShapeBaseData::**maxDamage**
Maximum damage level for this object.

float ShapeBaseData::**maxEnergy**
Maximum energy level for this object.

bool ShapeBaseData::**mountedImagesBank**
Do mounted images bank along with the camera?

bool ShapeBaseData::**observeThroughObject**
Observe this object through its camera transform and default fov. If true, when this object is the camera it can provide a custom camera transform and FOV (instead of the default eye transform).

bool ShapeBaseData::**renderWhenDestroyed**
Whether to render the shape when it is in the "Destroyed" damage state.

float ShapeBaseData::**repairRate**
Rate at which damage is repaired in damage units/tick. This value is subtracted from the damage level until it reaches 0.

bool ShapeBaseData::**shadowEnable**
Enable shadows for this shape (currently unused, shadows are always enabled).

float ShapeBaseData::**shadowMaxVisibleDistance**
Maximum distance at which shadow is visible (currently unused).

float ShapeBaseData::**shadowProjectionDistance**
Maximum height above ground to project shadow. If the object is higher than this no shadow will be rendered.

int ShapeBaseData::**shadowSize**
Size of the projected shadow texture (must be power of 2).

float ShapeBaseData::**shadowSphereAdjust**
Scalar applied to the radius of spot shadows (initial radius is based on the shape bounds but can be adjusted with this field).

filename ShapeBaseData::**shapeFile**
The DTS or DAE model to use for this object.

ExplosionData ShapeBaseData::**underwaterExplosion**
Explosion to generate when this shape is blown up underwater.

bool ShapeBaseData::**useEyePoint**
Flag controlling whether the client uses this object's eye point to view from.

ShapeBaseImageData object.

Inherit: GameBaseData

Description Represents geometry to be mounted to a ShapeBase object.

Unlike other datablocks, ShapeBaseImageData does not have a base class associated with it. Instead, this datablock is an abstraction of geometry that can only be mounted to a ShapeBase object and is only used by a ShapBase object.

The most common use for ShapeBaseImageData objects (referred to as Images hereafter) is for weapons carried by a Player or Vehicle object, and much of the functionality provided by the Image is aimed at that use-case. Images include a powerful state machine to control animations, sounds, script callbacks, and state transitions. This state system is downloaded to the client so that clients can predict state changes and animate accordingly.

The following example - a grenade launcher weapon - demonstates the flexibility of the system. The weapon includes states and transitions to handle the normal ready->fire->reload->ready loop as well as noammo->dryfire for firing when the weapon is out of ammo.

Example:

```
datablock ShapeBaseImageData( GrenadeLauncherImage )
{
   // Basic properties
   shapefile = "art/shapes/weapons/ramrifle/base.dts";

   // Specify mount point & offset for 3rd person, and eye offset// for first person rendering.mountP

   offset = "0.0 0.0 0.1";
   eyeOffset = "0.25 0.4 -0.4";

   // Add the WeaponImage namespace as a parent, WeaponImage namespace// provides some hooks into the

   // Projectile and Ammo.
   item = GrenadeLauncher;
   ammo = GrenadeLauncherAmmo;
   projectile = GrenadeLauncherProjectile;
   wetProjectile = GrenadeWetProjectile;
   projectileType = Projectile;

   // Shell casingscasing = GrenadeLauncherShellCasing;
   shellExitDir = "1.0 0.3 1.0";
   shellExitOffset = "0.15 -0.56 -0.1";
   shellExitVariance = 15.0;
   shellVelocity = 3.0;

   // Let there be light - NoLight, ConstantLight, PulsingLight, WeaponFireLight.lightType = "WeaponF

   lightColor = "1.0 1.0 0.9";
   lightDuration = 200;
   lightRadius = 20;


   // Initial start up statestateName[0] = "Preactivate";
   stateTransitionOnLoaded[0] = "Activate";
   stateTransitionOnNoAmmo[0] = "NoAmmo";

   // Activating the gun.// Called when the weapon is first mounted and there is ammo.stateName[1] =

   stateTransitionOnTimeout[1] = "Ready";
   stateTimeoutValue[1] = 0.6;
   stateSequence[1] = "Activate";

   // Ready to fire, just waiting for the triggerstateName[2] = "Ready";
   stateTransitionOnNoAmmo[2] = "NoAmmo";
   stateTransitionOnTriggerDown[2] = "CheckWet";
   stateSequence[2] = "Ready";
```

```
   // Fire the weapon. Calls the fire script which does the actual work.stateName[3] = "Fire";
   stateTransitionOnTimeout[3] = "PostFire";
   stateTimeoutValue[3] = 0.4;
   stateFire[3] = true;
   stateAllowImageChange[3] = false;
   stateSequence[3] = "Fire";
   stateScript[3] = "onFire";
   stateSound[3] = GrenadeLauncherFireSound;

   // Check ammostateName[4] = "PostFire";
   stateTransitionOnAmmo[4] = "Reload";
   stateTransitionOnNoAmmo[4] = "NoAmmo";

   // Play the reload animation, and transition back into Ready statestateName[5] = "Reload";
   stateTransitionOnTimeout[5] = "Ready";
   stateTimeoutValue[5] = 0.2;
   stateAllowImageChange[5] = false;
   stateSequence[5] = "Reload";
   stateEjectShell[5] = false; // set to true to enable shell casing ejectstateSound[5] = GrenadeLaun

   // No ammo in the weapon, just idle until something shows up.// Play the dry fire sound if the tri
   stateTransitionOnAmmo[6] = "Reload";
   stateSequence[6] = "NoAmmo";
   stateTransitionOnTriggerDown[6] = "DryFire";

   // No ammo dry firestateName[7] = "DryFire";
   stateTimeoutValue[7] = 1.0;
   stateTransitionOnTimeout[7] = "NoAmmo";
   stateSound[7] = GrenadeLauncherFireEmptySound;

   // Check if wetstateName[8] = "CheckWet";
   stateTransitionOnWet[8] = "WetFire";
   stateTransitionOnNotWet[8] = "Fire";

   // Wet firestateName[9] = "WetFire";
   stateTransitionOnTimeout[9] = "PostFire";
   stateTimeoutValue[9] = 0.4;
   stateFire[9] = true;
   stateAllowImageChange[9] = false;
   stateSequence[9] = "Fire";
   stateScript[9] = "onWetFire";
   stateSound[9] = GrenadeLauncherFireSound;
};
```

Images are mounted into a slot on the target ShapeBase derrived object as shown below.

Example:

```
$WeaponSlot = 0;

...

// Use a weapon by mounting it onto the given ShapeBase derrived object.// %data is the weapon whose
function Weapon::onUse( %data, %obj )
{
   // Default behavior for all weapons is to mount it into the objects weapon// slot, as defined by $
   {
      // The requested weapon is not mounted on $WeaponSlot so mount it now.
      %obj.mountImage( %data.image, $WeaponSlot );
```

```
    }
}
```

Weapon Shape Nodes The DTS or DAE model used for the Image has the following requirements:

Weapon Muzzle Flash When the Image is used as a weapon, a sequence can be added to display a muzzle flash when the weapon is fired (if stateSequenceRandomFlash is set to true for the firing state). The name of the muzzle flash sequence is the same as the state sequence (eg. fire), but with '_vis' appended (eg. fire_vis).

In the example below, the muzzle flash is made up of three quads; one facing the player, and two crossed quads pointing out of the weapon so viewers perpendicular to the player will also see the flash.

The visibility of the muzzle flash mesh is animated on for 1 frame then off for 1 frame as shown below, but any Torque supported animation method could be used as well. For example, the node the quads are attached to could be rotated or scaled, or the mesh Material could be animated (UV or frame) to provide further variation.

First Person Shape [Optional] The ShapeBaseImageData supports an optional shape that is displayed when the player is in a first person view. This shape is defined using the shapeFileFP property. You also must set an eyeOffset or make use of an eye mount node for this shape to be used in a first person view.

Having this second shape defined provides for more flexibility between 3rd person (and what other players see) and 1st person views. In a typical first person shooter the 3rd person weapon is not as detailed and supports a limited number of animation sequences. Just enough for the other players in the game to get a sense of what the player is doing. Then the 1st person weapon has a lot more detail, such as moving parts, etc. It may also have some arms and hands incldued that are animated when reloading the weapon and other actions. Only the player holding the weapon sees all of this.

There are a number of things to keep in mind if you make use of shapeFileFP:

Animation Sequence Transitions Starting with T3D 1.2 control is now given over transitioning from one image state's sequence to another state's sequence. The new state "stateSequenceTransitionIn" and "stateSequenceTransitionOut" flags dictate if the current state's sequence should be transitioned into when changing to the current state, or transitioned out of when switching to a new state. However, there are times when you don't want to do an animation sequence transition regardless of which state you are coming from. An example of this is the traditional "Fire" state. A Fire state should play immediately and not be transitioned into. In these cases a state may set the "stateSequenceNeverTransition" flag. With that set a state's sequence will begin to play immediately.

Animation Sequence Selection When it comes to choosing what sequence to play on the mounted image there are now some new rules. Under 1.1 when an image state requested a named sequence that is found on the mounted image and played – its action sequence. This still occurs under 1.2. However, it is now possible to modify the name of the sequence to play based on some prefixes. PlayerData now has two additional, optional fields: imageAnimPrefix and imageAnimPrefixFP. Just like how these same fields on ShapeBaseImageData can modify when sequences are played on the player based on what is mounted (see Player class documentation), these two PlayerData fields can modify what sequence is played on the mounted image based on the mounting player. This becomes especially useful when combined with 1st person arms – although here we're just talking about weapons/mounted images.

Let's suppose we have two types of player: Soldier and Alien. We may want each type of player to use the same weapon slightly differently (or even radically differently, such as the Alien holding the weapon upside down). We use the "Soldier" anim prefix in the soldier's datablock and the "Alien" prefix in the alien's datablock. Now when looking up the sequence for a weapon's fire state – usually called "fire" by convention – the appropriate prefix is added first. And if that prefixed sequence is not found, then we fall back to the standard sequence name. So the soldier's sequence name search looks like this:

and the alien's sequence name search looks like this:

This gives the artist greater control over how the weapons look. And because there are separate prefixes possible on PlayerData for 1st person and 3rd person you can mix and match as appropriate. So you could set a prefix for 1st person, but leave it blank for 3rd person (don't do anything special in 3rd person).

Another way that an image state's sequence name could be modified is through the new ShapeBase::setImageScriptAnimPrefix() method. This allows you to insert an additional prefix into the name look up. The current scripts pass along the player's current pose name, but anything could be passed in based on game play. This can be even more useful with the 1st person arms. You could then have a weapon idle state when swimming that moves the weapon (and 1st person arms) in a gentle swim motion. When you combine the PlayerData prefix with the script anim prefix and finally with the image state sequence name, you end up with the following sequence name search pattern (the first found is used):

Whenever ShapeBase::setImageScriptAnimPrefix() is called there is a transition from the currently playing state sequence into the new script prefixed animation sequence. In our example, this allows for a transition from walking to swimming for the weapon. The new ShapeBaseImageData scriptAnimationTransitionTime controls how long to take for this transition.

eyeMount Node [Optional] As with 1.1 the placement of the 1st person image may be set with the eyeOffset parameter. Now with 1.2 the 1st person image may be placed based on a node in the 1st person DTS/DAE shape, the "eyeMount" node. When the ShapeBaseImageData's useEyeNode parameter is made true, the image is effectively mounted to the 3rd person player's "eye" node, locking it into place. This allows the artist in their 3D application to precisely place the 1st person weapon in view when their 3D app's camera is placed on the eyeMount node and has the same field of view as Torque. This is very handy when animating the 1st person weapon, especially with 1st person arms.

Also with 1.2 an image that is placed with the eyeMount node may have an "eye" node defined. When found the player's camera is mounted to the image's "eye" node rather than the 3rd person player's "eye" node. This allows for animating the camera such as during a fire sequence.

Allowing for this much control does have a potential down side. In order for a weapon to fire correctly on the server it needs to have its muzzle point at the correct location. If a weapon's root pose (without animation) doesn't have its muzzle point at roughly the same location as when the weapon is fired, then the new ShapeBaseImageData "animateOnServer" flag should be set. When set the server will perform all state machine animation to ensure the muzzle point is at the correct location when required. This puts an extra strain on the server. If care is taken when building the weapons such that the root pose is close enough to the fire pose, then you can safely leave the "animateOnServer" flag off and not have to worry about the extra server load.

Special State Triggers Starting with 1.2 there are now a number of new triggers that may be set for a ShapeBaseImageData's state machine to react to. These provide greater game play control over an image's state flow. The first are the "stateTransitionOnMotion" and "stateTransitionOnNoMotion" triggers. This trigger occurs whenever the mounting ShapeBase (usually a Player) has x, y or z motion applied through the Move structure. From a Player perspective this means whenever the user moves their player forwards, backwards or strafes. That has been used to provide weapons a slight bobbing appearance (using an animation sequence) when the weapon is idle. Fire and Reload states don't usually make use of these triggers to keep those actions solid.

There has always been a target trigger for ShapeBaseImageData but under 1.1 it was not possible to set it, nor was it used. Starting with 1.2 you can now set the target trigger in script using ShapeBase::setImageTargetState() and use stateTransitionOnTarget and stateTransitionOnNoTarget for whatever game play reasons are required.

Finally, there are four new generic triggers that may be set from script and used for whatever purpose the game play imposes. These are "stateTransitionGeneric0In", "stateTransitionGeneric1In", etc. and "stateTransitionGeneric0Out", "stateTransition1Out" etc. The FPS Tutorial weapons use the first generic trigger to indicate that the player is sprinting and switch to a Sprint state to prevent firing of the weapon. Other possible uses are for iron sights.

Special States The client and server move through a ShapeBaseImageData's state machine independantly according to various triggers, timeouts, etc. The client is not normally told to move to a specific state when the server does. However, there are three instances where the client is told by the server to immediately jump to a given state. This ensures that the client's experience matches the server at key moments. As such, only one of each of these states may exist in a single ShapeBaseImageData state machine at a time.

The fire state is the first such state. It is indicated by setting the state's "stateFire" flag to true. This is the state immediately jumped to when the weapon begins to fire.

The alternate fire state is the second forced jump point (new in 1.2). It is indicated by setting the state's "stateAlternateFire" flag to true. Not all weapons have an alternate fire state. In fact most games treat a weapon's alternate fire as a separate weapon altogether.

The reload state is the last special state (new in 1.2). It is indicated by setting the state's "stateReload" flag to true. This state is triggered if the weapon makes use of the new 1.2 ammo clip system and the weapon is reloading a clip, either automatically or manually triggered by the client.

Methods

void `ShapeBaseImageData::onMount` (ShapeBase *obj*, int *slot*, float *dt*)

Called when the Image is first mounted to the object.

Parameters

- **obj** – object that this Image has been mounted to
- **slot** – Image mount slot on the object
- **dt** – time remaining in this Image update

void `ShapeBaseImageData::onUnmount` (ShapeBase *obj*, int *slot*, float *dt*)

Called when the Image is unmounted from the object.

Parameters

- **obj** – object that this Image has been unmounted from
- **slot** – Image mount slot on the object
- **dt** – time remaining in this Image update

Fields

bool `ShapeBaseImageData::accuFire`

Flag to control whether the Image's aim is automatically converged with the crosshair. Currently unused.

bool `ShapeBaseImageData::animateAllShapes`

Indicates that all shapes should be animated in sync. When multiple shapes are defined for this image datablock, each of them are automatically animated in step with each other. This allows for easy switching between between shapes when some other condition changes, such as going from first person to third person, and keeping their look consistent. If you know that you'll never switch between shapes on the fly, such as players only being allowed in a first person view, then you could set this to false to save some calculations. There are other circumstances internal to the engine that determine that only the current shape should be animated rather than all defined shapes. In those cases, this property is ignored.

bool `ShapeBaseImageData::animateOnServer`

Indicates that the image should be animated on the server. In most cases you'll want this set if you're using useEyeNode. You may also want to set this if the muzzlePoint is animated while it shoots. You can set this to false even if these previous cases are true if the image's shape is set up in the correct position and orientation in the 'root' pose and none of the nodes are animated at key times, such as the muzzlePoint essentially remaining at the same position at the start of the fire state (it could animate just fine after the projectile is away as the muzzle vector is only calculated at the start of the state). You'll also want to set this to true if you're animating

the camera using the image's 'eye' node – unless the movement is very subtle and doesn't need to be reflected on the server.

Point3F ShapeBaseImageData::**camShakeAmp**
Amplitude of the camera shaking effect.

Point3F ShapeBaseImageData::**camShakeFreq**
Frequency of the camera shaking effect.

DebrisData ShapeBaseImageData::**casing**
DebrisData datablock to use for ejected casings.

bool ShapeBaseImageData::**cloakable**
Whether this Image can be cloaked. Currently unused.

bool ShapeBaseImageData::**computeCRC**
If true, verify that the CRC of the client's Image matches the server's CRC for the Image when loaded by the client.

bool ShapeBaseImageData::**correctMuzzleVector**
Flag to adjust the aiming vector to the eye's LOS point when in 1st person view.

bool ShapeBaseImageData::**correctMuzzleVectorTP**
Flag to adjust the aiming vector to the camera's LOS point when in 3rd person view.

bool ShapeBaseImageData::**emap**
Whether to enable environment mapping on this Image.

MatrixPosition ShapeBaseImageData::**eyeOffset**
"X Y Z" translation offset from the ShapeBase model's eye node. When in first person view, this is the offset from the eye node to place the gun. This gives the gun a fixed point in space, typical of a lot of FPS games.

MatrixRotation ShapeBaseImageData::**eyeRotation**
"X Y Z ANGLE" rotation offset from the ShapeBase model's eye node. When in first person view, this is the rotation from the eye node to place the gun.

bool ShapeBaseImageData::**firstPerson**
Set to true to render the image in first person.

caseString ShapeBaseImageData::**imageAnimPrefix**
Passed along to the mounting shape to modify animation sequences played in third person. [optional].

caseString ShapeBaseImageData::**imageAnimPrefixFP**
Passed along to the mounting shape to modify animation sequences played in first person. [optional].

float ShapeBaseImageData::**lightBrightness**
Brightness of the light this Image emits. Only valid for WeaponFireLight.

ColorF ShapeBaseImageData::**lightColor**
The color of light this Image emits.

int ShapeBaseImageData::**lightDuration**
Duration in SimTime of Pulsing and WeaponFire type lights.

float ShapeBaseImageData::**lightRadius**
Radius of the light this Image emits.

ShapeBaseImageLightType ShapeBaseImageData::**lightType**
The type of light this Image emits.

float ShapeBaseImageData::**mass**
Mass of this Image. This is added to the total mass of the ShapeBase object.

int ShapeBaseImageData::**maxConcurrentSounds**
Maximum number of sounds this Image can play at a time. Any value lt = 0 indicates that it can play an infinite number of sounds.

float ShapeBaseImageData::**minEnergy**
Minimum Image energy for it to be operable.

int ShapeBaseImageData::**mountPoint**
Mount node # to mount this Image to. This should correspond to a mount# node on the ShapeBase derived object we are mounting to.

MatrixPosition ShapeBaseImageData::**offset**
"X Y Z" translation offset from this Image's mountPoint node to attach to. Defaults to "0 0 0". ie. attach this Image's mountPoint node to the ShapeBase model's mount# node without any offset.

ProjectileData ShapeBaseImageData::**Projectile**
The projectile fired by this Image.

MatrixRotation ShapeBaseImageData::**rotation**
"X Y Z ANGLE" rotation offset from this Image's mountPoint node to attach to. Defaults to "0 0 0". ie. attach this Image's mountPoint node to the ShapeBase model's mount# node without any additional rotation.

float ShapeBaseImageData::**scriptAnimTransitionTime**
The amount of time to transition between the previous sequence and new sequence when the script prefix has changed. When setImageScriptAnimPrefix() is used on a ShapeBase that has this image mounted, the image will attempt to switch to the new animation sequence based on the given script prefix. This is the amount of time it takes to transition from the previously playing animation sequence tothe new script prefix-based animation sequence.

bool ShapeBaseImageData::**shakeCamera**
Flag indicating whether the camera should shake when this Image fires.

filename ShapeBaseImageData::**shapeFile**
The DTS or DAE model to use for this Image.

filename ShapeBaseImageData::**shapeFileFP**
The DTS or DAE model to use for this Image when in first person. This is an optional parameter that also requires either eyeOffset or useEyeNode to be set. If none of these conditions is met then shapeFile will be used for all cases. Typically you set a first person image for a weapon that includes the player's arms attached to it for animating while firing, reloading, etc. This is typical of many FPS games.

Point3F ShapeBaseImageData::**shellExitDir**
Vector direction to eject shell casings.

float ShapeBaseImageData::**shellExitVariance**
Variance (in degrees) from the shellExitDir vector to eject casings.

float ShapeBaseImageData::**shellVelocity**
Speed at which to eject casings.

bool ShapeBaseImageData::**stateAllowImageChange**[31]
If false, other Images will temporarily be blocked from mounting while the state machine is executing the tasks in this state. For instance, if we have a rocket launcher, the player shouldn't be able to switch out while firing. So, you'd set stateAllowImageChange to false in firing states, and true the rest of the time.

bool ShapeBaseImageData::**stateAlternateFire**[31]
The first state with this set to true is the state entered by the client when it receives the 'altFire' event.

bool ShapeBaseImageData::**stateDirection**[31]
Direction of the animation to play in this state. True is forward, false is backward.

bool ShapeBaseImageData::**stateEjectShell**[31]
If true, a shell casing will be ejected in this state.

ParticleEmitterData ShapeBaseImageData::**stateEmitter**[31]
Emitter to generate particles in this state (from muzzle point or specified node).

string ShapeBaseImageData::**stateEmitterNode**[31]
Name of the node to emit particles from.

float ShapeBaseImageData::**stateEmitterTime**[31]
How long (in seconds) to emit particles on entry to this state.

float ShapeBaseImageData::**stateEnergyDrain**[31]
Amount of energy to subtract from the Image in this state. Energy is drained at stateEnergyDrain units/tick as long as we are in this state.

bool ShapeBaseImageData::**stateFire**[31]
The first state with this set to true is the state entered by the client when it receives the 'fire' event.

bool ShapeBaseImageData::**stateIgnoreLoadedForReady**[31]
If set to true, and both ready and loaded transitions are true, the ready transition will be taken instead of the loaded transition. A state is 'ready' if pressing the fire trigger in that state would transition to the fire state.

ShapeBaseImageLoadedState ShapeBaseImageData::**stateLoadedFlag**[31]
Set the loaded state of the Image.

- •IgnoreLoaded: Don't change Image loaded state.
- •Loaded: Set Image loaded state to true.
- •NotLoaded: Set Image loaded state to false.

caseString ShapeBaseImageData::**stateName**[31]
Name of this state.

ShapeBaseImageRecoilState ShapeBaseImageData::**stateRecoil**[31]
Type of recoil sequence to play on the ShapeBase object on entry to this state.

- •NoRecoil: Do not play a recoil sequence.
- •LightRecoil: Play the light_recoil sequence.
- •MediumRecoil: Play the medium_recoil sequence.
- •HeavyRecoil: Play the heavy_recoil sequence.

bool ShapeBaseImageData::**stateReload**[31]
The first state with this set to true is the state entered by the client when it receives the 'reload' event.

bool ShapeBaseImageData::**stateScaleAnimation**[31]
If true, the timeScale of the stateSequence animation will be adjusted such that the sequence plays for stateTimeoutValue seconds.

bool ShapeBaseImageData::**stateScaleAnimationFP**[31]
If true, the timeScale of the first person stateSequence animation will be adjusted such that the sequence plays for stateTimeoutValue seconds.

bool ShapeBaseImageData::**stateScaleShapeSequence**[31]
Indicates if the sequence to be played on the mounting shape should be scaled to the length of the state.

caseString ShapeBaseImageData::**stateScript**[31]
Method to execute on entering this state. Scoped to this image class name, then ShapeBaseImageData . The script callback function takes the same arguments as the onMount callback.

string ShapeBaseImageData::**stateSequence**[31]
Name of the sequence to play on entry to this state.

bool ShapeBaseImageData::**stateSequenceNeverTransition**[31]
Never allow a transition to this sequence. Often used for a fire sequence.

bool ShapeBaseImageData::**stateSequenceRandomFlash**[31]
If true, the muzzle flash sequence will be played while in this state. The name of the muzzle flash sequence is the same as stateSequence, with "_vis" at the end.

bool ShapeBaseImageData::**stateSequenceTransitionIn**[31]
Do we transition to the state's sequence when we enter the state?

bool ShapeBaseImageData::**stateSequenceTransitionOut**[31]
Do we transition to the new state's sequence when we leave the state?

float ShapeBaseImageData::**stateSequenceTransitionTime**[31]
The time to transition in or out of a sequence.

string ShapeBaseImageData::**stateShapeSequence**[31]
Name of the sequence that is played on the mounting shape.

SFXTrack ShapeBaseImageData::**stateSound**[31]
Sound to play on entry to this state.

ShapeBaseImageSpinState ShapeBaseImageData::**stateSpinThread**[31]
Controls how fast the 'spin' animation sequence will be played in this state.

- •Ignore: No change to the spin sequence.
- •Stop: Stops the spin sequence at its current position.
- •SpinUp: Increase spin sequence timeScale from 0 (on state entry) to 1 (after stateTimeoutValue seconds).
- •SpinDown: Decrease spin sequence timeScale from 1 (on state entry) to 0 (after stateTimeoutValue seconds).
- •FullSpeed: Resume the spin sequence playback at its current position with timeScale=1.

float ShapeBaseImageData::**stateTimeoutValue**[31]
Time in seconds to wait before transitioning to stateTransitionOnTimeout.

string ShapeBaseImageData::**stateTransitionGeneric0In**[31]
Name of the state to transition to when the generic trigger 0 state changes to true.

string ShapeBaseImageData::**stateTransitionGeneric0Out**[31]
Name of the state to transition to when the generic trigger 0 state changes to false.

string ShapeBaseImageData::**stateTransitionGeneric1In**[31]
Name of the state to transition to when the generic trigger 1 state changes to true.

string ShapeBaseImageData::**stateTransitionGeneric1Out**[31]
Name of the state to transition to when the generic trigger 1 state changes to false.

string ShapeBaseImageData::**stateTransitionGeneric2In**[31]
Name of the state to transition to when the generic trigger 2 state changes to true.

string ShapeBaseImageData::**stateTransitionGeneric2Out**[31]
Name of the state to transition to when the generic trigger 2 state changes to false.

string ShapeBaseImageData::**stateTransitionGeneric3In**[31]
Name of the state to transition to when the generic trigger 3 state changes to true.

string ShapeBaseImageData::**stateTransitionGeneric3Out**[31]
Name of the state to transition to when the generic trigger 3 state changes to false.

string ShapeBaseImageData::**stateTransitionOnAltTriggerDown**[31]
Name of the state to transition to when the alt trigger state of the Image changes to false (alt fire button up).

string ShapeBaseImageData::**stateTransitionOnAltTriggerUp**[31]
Name of the state to transition to when the alt trigger state of the Image changes to true (alt fire button down).

string ShapeBaseImageData::**stateTransitionOnAmmo**[31]
Name of the state to transition to when the ammo state of the Image changes to true.

string ShapeBaseImageData::**stateTransitionOnLoaded**[31]
Name of the state to transition to when the loaded state of the Image changes to 'Loaded'.

string ShapeBaseImageData::**stateTransitionOnMotion**[31]
Name of the state to transition to when the Player moves.

string ShapeBaseImageData::**stateTransitionOnNoAmmo**[31]
Name of the state to transition to when the ammo state of the Image changes to false.

string ShapeBaseImageData::**stateTransitionOnNoMotion**[31]
Name of the state to transition to when the Player stops moving.

string ShapeBaseImageData::**stateTransitionOnNoTarget**[31]
Name of the state to transition to when the Image loses a target.

string ShapeBaseImageData::**stateTransitionOnNotLoaded**[31]
Name of the state to transition to when the loaded state of the Image changes to 'Empty'.

string ShapeBaseImageData::**stateTransitionOnNotWet**[31]
Name of the state to transition to when the Image exits the water.

string ShapeBaseImageData::**stateTransitionOnTarget**[31]
Name of the state to transition to when the Image gains a target.

string ShapeBaseImageData::**stateTransitionOnTimeout**[31]
Name of the state to transition to when we have been in this state for stateTimeoutValue seconds.

string ShapeBaseImageData::**stateTransitionOnTriggerDown**[31]
Name of the state to transition to when the trigger state of the Image changes to false (fire button released).

string ShapeBaseImageData::**stateTransitionOnTriggerUp**[31]
Name of the state to transition to when the trigger state of the Image changes to true (fire button down).

string ShapeBaseImageData::**stateTransitionOnWet**[31]
Name of the state to transition to when the Image enters the water.

bool ShapeBaseImageData::**stateWaitForTimeout**[31]
If false, this state ignores stateTimeoutValue and transitions immediately if other transition conditions are met.

bool ShapeBaseImageData::**useEyeNode**
Mount image using image's eyeMount node and place the camera at the image's eye node (or at the eyeMount node if the eye node is missing). When in first person view, if an 'eyeMount' node is present in the image's shape, this indicates that the image should mount eyeMount node to Player eye node for image placement. The Player's camera should also mount to the image's eye node to inherit any animation (or the eyeMount node if the image doesn't have an eye node).

bool ShapeBaseImageData::**useRemainderDT**
If true, allow multiple timeout transitions to occur within a single tick (useful if states have a very small timeout).

bool ShapeBaseImageData::**usesEnergy**
Flag indicating whether this Image uses energy instead of ammo. The energy level comes from the ShapeBase object we're mounted to.

SpawnSphere This class is used for creating any type of game object, assigning it a class, datablock, and other properties when it is spawned.

Inherit: MissionMarker

Description Torque 3D uses a simple spawn system, which can be easily modified to spawn any kind of object (of any class). Each new level already contains at least one SpawnSphere, which is represented by a green octahedron in stock Torque 3D. The spawnClass field determines the object type, such as Player, AIPlayer, etc. The spawnDataBlock field applies the pre-defined datablock to each spawned object instance. The really powerful feature of this class is provided by the spawnScript field which allows you to define a simple script (multiple lines) that will be executed once the object has been spawned.

Example:

```
// Define an SpawnSphere that essentially performs the following each time an object is spawned
//$SpawnObject = new Player()
//{
//   dataBlock = "DefaultPlayerData";
//   name = "Bob";
//   lifeTotal = 3;
//};
//echo("Spawned a Player: " @ $SpawnObject);
newSpawnSphere(DefaultSpawnSphere)
{
   spawnClass = "Player";
   spawnDatablock = "DefaultPlayerData";
   spawnScript = "echo(\"Spawned a Player: \" @ $SpawnObject);"; // embedded quotes must be escaped w
   dataBlock = "SpawnSphereMarker";
   position = "-0.77266 -19.882 17.8153";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   canSave = "1";
   canSaveDynamicFields = "1";
};

// Because autoSpawn is set to true in the above example, the following lines
// of code will execute AFTER the Player object has been spawned.
echo("Object Spawned");
echo("Hello World");
```

Methods

void SpawnSphere::**onAdd**(int *objectId*)

Called when the SpawnSphere is being created.

Parameters **objectId** – The unique SimObjectId generated when SpawnSphere is created (%this in script)

bool SpawnSphere::**spawnObject**(string *additionalProps*)

Dynamically create a new game object with a specified class, datablock, and optional properties. This is called on the actual SpawnSphere , not to be confused with the Sim::spawnObject() global function

Parameters **additionalProps** – Optional set of semiconlon delimited parameters applied to the spawn object during creation.

Example:

```
// Use the SpawnSphere::spawnObject function to create a game object// No additional properties
%player = DefaultSpawnSphere.spawnObject();
```

Fields

bool SpawnSphere::**autoSpawn**
Flag to spawn object as soon as SpawnSphere is created, true to enable or false to disable.

float SpawnSphere::**indoorWeight**
Deprecated.

float SpawnSphere::**outdoorWeight**
Deprecated.

float SpawnSphere::**radius**
Deprecated.

string SpawnSphere::**spawnClass**
Object class to create (eg. Player , AIPlayer , Debris etc).

string SpawnSphere::**spawnDatablock**
Predefined datablock assigned to the object when created.

string SpawnSphere::**spawnProperties**
String containing semicolon (;) delimited properties to set when the object is created.

string SpawnSphere::**spawnScript**
Command to execute immediately after spawning an object. New object id is stored in $SpawnObject. Max 255 characters.

bool SpawnSphere::**spawnTransform**
Flag to set the spawned object's transform to the SpawnSphere's transform.

float SpawnSphere::**sphereWeight**
Deprecated.

StaticShape The most basic 3D shape with a datablock available in Torque 3D.

Inherit: ShapeBase

Description The most basic 3D shape with a datablock available in Torque 3D.

When it comes to placing 3D objects in the scene, you technically have two options:

1. TSStatic objects
2. ShapeBase derived objects

Since ShapeBase and ShapeBaseData are not meant to be instantiated in script, you will use one of its child classes instead. Several game related objects are derived from ShapeBase: Player, Vehicle, Item, and so on.

When you need a 3D object with datablock capabilities, you will use an object derived from ShapeBase. When you need an object with extremely low overhead, and with no other purpose than to be a 3D object in the scene, you will use TSStatic.

The most basic child of ShapeBase is StaticShape. It does not introduce any of the additional functionality you see in Player, Item, Vehicle or the other game play heavy classes. At its core, it is comparable to a TSStatic, but with a datbalock. Having a datablock provides a location for common variables as well as having access to various ShapeBaseData, GameBaseData and SimDataBlock callbacks.

Example:

```
// Create a StaticShape using a datablock
datablock StaticShapeData(BasicShapeData)
{
   shapeFile = "art/shapes/items/kit/healthkit.dts";
```

```
   testVar = "Simple string, not a stock variable";
};

newStaticShape()
{
   dataBlock = "BasicShapeData";
   position = "0.0 0.0 0.0";
   rotation = "1 0 0 0";
   scale = "1 1 1";
};
```

StaticShapeData derrived shape datablock available in Torque 3D.

Inherit: ShapeBaseData

Description The most basic ShapeBaseData derrived shape datablock available in Torque 3D.

When it comes to placing 3D objects in the scene, you effectively have two options:

1. TSStatic objects
2. ShapeBase derived objects

Since ShapeBase and ShapeBaseData are not meant to be instantiated in script, you will use one of its child classes instead. Several game related objects are derived from ShapeBase: Player, Vehicle, Item, and so on.

When you need a 3D object with datablock capabilities, you will use an object derived from ShapeBase. When you need an object with extremely low overhead, and with no other purpose than to be a 3D object in the scene, you will use TSStatic.

The most basic child of ShapeBase is StaticShape. It does not introduce any of the additional functionality you see in Player, Item, Vehicle or the other game play heavy classes. At its core, it is comparable to a TSStatic, but with a datbalock. Having a datablock provides a location for common variables as well as having access to various ShapeBaseData, GameBaseData and SimDataBlock callbacks.

Example:

```
// Create a StaticShape using a datablock
datablock StaticShapeData(BasicShapeData)
{
   shapeFile = "art/shapes/items/kit/healthkit.dts";
   testVar = "Simple string, not a stock variable";
};

newStaticShape()
{
   dataBlock = "BasicShapeData";
   position = "0.0 0.0 0.0";
   rotation = "1 0 0 0";
   scale = "1 1 1";
};
```

Fields

int `StaticShapeData::dynamicType`

An integer value which, if speficied, is added to the value retured by getType(). This allows you to extend the type mask for a StaticShape that uses this datablock. Type masks are used for container queries, etc.

bool StaticShapeData::**noIndividualDamage**
Deprecated.

Trigger .

Inherit: GameBase

Description A Trigger is a volume of space that initiates script callbacks when objects pass through the Trigger.

TriggerData provides the callbacks for the Trigger when an object enters, stays inside or leaves the Trigger's volume.

Methods

int Trigger::**getNumObjects**()
Get the number of objects that are within the Trigger's bounds.

int Trigger::**getObject**(int *index*)
Retrieve the requested object that is within the Trigger's bounds.

Parameters **index** – Index of the object to get (range is 0 to getNumObjects()-1)

Returns The SimObjectID of the object, or -1 if the requested index is invalid.

void Trigger::**onAdd**(int *objectId*)
Called when the Trigger is being created.

Parameters **objectId** – the object id of the Trigger being created

void Trigger::**onRemove**(int *objectId*)
Called just before the Trigger is deleted.

Parameters **objectId** – the object id of the Trigger being deleted

Fields

string Trigger::**enterCommand**
The command to execute when an object enters this trigger. Object id stored in %obj. Maximum 1023 characters.

string Trigger::**leaveCommand**
The command to execute when an object leaves this trigger. Object id stored in %obj. Maximum 1023 characters.

floatList Trigger::**polyhedron**
Defines a non-rectangular area for the trigger. Rather than the standard rectangular bounds, this optional parameter defines a quadrilateral trigger area. The quadrilateral is defined as a corner point followed by three vectors representing the edges extending from the corner.

string Trigger::**tickCommand**
The command to execute while an object is inside this trigger. Maximum 1023 characters.

TriggerData objects.

Inherit: GameBaseData

Description Defines shared properties for Trigger objects.

The primary focus of the TriggerData datablock is the callbacks it provides when an object is within or leaves the Trigger bounds.

Methods

void TriggerData::**onEnterTrigger**(Trigger *trigger*, GameBase *obj*)

Called when an object enters the volume of the Trigger instance using this TriggerData .

Parameters

- **trigger** – the Trigger instance whose volume the object entered
- **obj** – the object that entered the volume of the Trigger instance

void TriggerData::**onLeaveTrigger**(Trigger *trigger*, GameBase *obj*)

Called when an object leaves the volume of the Trigger instance using this TriggerData .

Parameters

- **trigger** – the Trigger instance whose volume the object left
- **obj** – the object that left the volume of the Trigger instance

void TriggerData::**onTickTrigger**(Trigger *trigger*)

Called every tickPeriodMS number of milliseconds (as specified in the TriggerData) whenever one or more objects are inside the volume of the trigger. The Trigger has methods to retrieve the objects that are within the Trigger's bounds if you want to do something with them in this callback.

Parameters **trigger** – the Trigger instance whose volume the object is inside

Fields

bool TriggerData::**clientSide**

Forces Trigger callbacks to only be called on clients.

int TriggerData::**tickPeriodMS**

Time in milliseconds between calls to onTickTrigger() while at least one object is within a Trigger's bounds.

TSShapeConstructor An object used to modify a DTS or COLLADA shape model after it has been loaded by Torque.

Inherit: SimObject

Description An object used to modify a DTS or COLLADA shape model after it has been loaded by Torque.

TSShapeConstructor is a special object used to modify a DTS or COLLADA shape model after it has been loaded by Torque, but before it is used by any other object.

It is often used to share animations from DSQ files between shapes with a common skeleton.

It may also be used to 'Torquify' a model that is missing the nodes and/or sequences required to function as a particular Torque object. A model used for a Player character for example should have an eye and a cam node, but these might not be present in a model not specifically created for Torque. TSShapeConstructor allows the missing nodes to be added and positioned so that the shape does not need to be re-worked or re-exported by an artist.

TSShapeConstructor also includes features to aid in loading COLLADA models, such as allowing the <up_axis> and <unit> elements to be overridden, and can also apply a user specified prefix to the names of COLLADA materials as shown below. Prefixing material names is useful to avoid name clashes, particularly for 3D apps like Google SketchUp that export models with generic material names like "material0". These options are most easily accessed using the COLLADA import gui, which will be displayed automatically the first time a COLLADA model is imported into Torque.

Settings from the import gui are automatically saved to a TSShapeConstructor script in the same folder as the model.

To create your own TSShapeConstructor object, simply create a TorqueScript file in the same folder as your DTS or COLLADA model, with the same filename but .cs extension. For example, if your model file was called myShape.dts, you would create a file called myShape.cs. Some example appear below:

The name of the TSShapeConstructor object (MyShapeDae and MyShape2Dae in the code samples above) is up to you, but you should choose a name that does not conflict with other objects or datablocks. A common convention for TSShapeConstructor objects is the name of the shape file. eg. MyShapeDae for a file called myshape.dae.

When Torque loads a DTS (.dts) or COLLADA (.dae) file, it first looks in the same folder for a TorqueScript file with the same filename (but .cs extension) in order to create the TSShapeConstructor object. Such scripts are executed automatically by Torque 3D, so there is no need to manually call exec("myShape.cs") from another script. Also, you should avoid adding other object and datablock declarations to this script because it will be executed every time the model is loaded, which may cause unexpected results if the datablocks already exist.

After Torque has loaded the model from the DTS or COLLADA file, it executes the TSShapeConstructor onLoad method to apply the desired set of changes to the shape. It should be noted that the changes are applied to the loaded model in memory rather than to the DTS or COLLADA file itself. This means the model can be re-exported to DTS or COLLADA without overwriting the TSShapeConstructor changes. TSShapeConstructor should be thought of as a post-export processing step, and is intended to be used alongside existing object and datablock setups.

Note that DSQ sequences may still be specified within the TSShapeConstructor object as normal:

Note that most of the features in TSShapeConstructor are far more easily accessible in the Shape Editor tool. This tool uses TSShapeConstructor 'under-the-hood' to edit the nodes, sequences and details of a shape, and the changes are saved to a TSShapeConstructor script object.

Shape Terminology The following definitions should be understood before reading the TSShapeConstructor examples and function reference:

Example 1: Adding a Collision Mesh To an Existing Shape Imagine you have a model that you want to add to the scene as a StaticShape, but it is missing a collision mesh. TSShapeConstructor makes it simple to modify an existing DTS shape to add a collision (or line-of-sight collision) detail level.

First, define the StaticShapeData datablock as normal. Create a script called myShape.cs in the art/datablocks folder, and define the datablock:

We need to tell Torque to execute this script so add exec("./myShape.cs"); to art/datablocks/datablockExec.cs.

Now we define the TSShapeConstructor object by creating a new script called myShape.cs in the art/shapes/myShape folder:

This script will add a box mesh with the same center and dimensions as the original model using the "Col" detail level at size -1. The negative detail size means that the mesh will not be rendered in-game, and the use of the special "Col" name means that this mesh will be detected as a collision mesh by the Torque engine.

When a Torque mission is started, the following steps occur:

Example 2: Adding a Mesh From an Existing DTS File The image below shows a boulder.dts shape in the Torque Show Tool Pro (TSTPro). The circled items indicate the geometry and material that will be copied into a different shape using TSShapeConstructor.

The following shows how to include the boulder1 mesh in another shape:

The output of the dumpShape command is shown below:

Note that the new detail ("detail128"), object ("test") and material ("MossyRock02") have been added to the normal rock1.dts shape.

Example 3: Auto-loading animations Instead of manually specifying all of the animations to load, it's easy to write some TorqueScript that will scan a folder for any matching animations and add them to the shape automatically. Imagine that we have the following shape (DTS) and sequence (DSQ) files:

The following script will scan the animations folder and add the sequences to the shape.

Example 4: Splitting COLLADA animations Many COLLADA exporters do not support the <animation_clip> element, meaning that animated models imported into Torque appear to have only a single sequence containing all of the animations. TSShapeConstructor can be used to split this combined animation into individual sequences. This is most easily done using the Shape Editor tool, but can also be done manually as follows:

Example 5: LOD using separate files In the past, using LOD required the artist to export all detail levels into a single DTS file. Using TSShapeConstructor, we can combine separate model files together. In fact, we can even use the folder-scanning approach from Example 3 to automatically construct the shape detail levels using all of the model files in the folder!

Note that the detail level models must contain the same object name, and for skinned models, the skin must be applied to the same skeleton for this script to work.

Imagine that we have the following shape (DAE) files:

Example 6: Add lights to the scene Although most often used to modify a shape before it is used, TSShapeConstructor can also be used as a general purpose interface to a 3D shape. For example, a 3D modeling package could be used to layout positions for lights in the scene. On import, the shape hierarchy might look like this:

The following code demonstrates how to create a TSShapeConstructor object on-demand in order to access the 3D shape data. This example adds lights to the current scene at position of the lightX nodes in the shape:

The original shape can be placed anywhere in the scene, then AddLights is called to create and place a PointLight at each node.

Example 7: Rigid-body Player Character Using the addNode and addMesh functions, it is possible to create a rigid-body (ie. non-skinned) player model compatible with the default animations, completely from TorqueScript!

The default player skeleton node transforms were obtained by adding the following code to the TSShapeConstructor onLoad function for a shape that already contained the default skeleton:

The contents of the console can then be copied and pasted into a new script. The script below shows the player model creation process: first pick a dummy dts file (rock1.dts in this case), and delete its existing nodes and meshes. Then create the default player skeleton. Finally, some box meshes are added at certain nodes to build up a rigid-body player character.

This produces the following shape:

Methods

bool `TSShapeConstructor::addCollisionDetail`(int *size*, string *type*, string *target*, int *depth*, float *merge*, float *concavity*, int *maxVerts*, float *boxMaxError*, float *sphereMaxError*, float *capsuleMaxError*)

Autofit a mesh primitive or set of convex hulls to the shape geometry. Hulls may optionally be converted to boxes, spheres and/or capsules based on their volume.

Parameters

- **`size`** – size for this detail level
- **`type`** – one of: box, sphere, capsule, 10-dop x, 10-dop y, 10-dop z, 18-dop, 26-dop, convex hulls. See the Shape Editor documentation for more details about these types.

- **target** – geometry to fit collision mesh(es) to; either "bounds" (for the whole shape), or the name of an object in the shape
- **depth** – maximum split recursion depth (hulls only)
- **merge** – volume % threshold used to merge hulls together (hulls only)
- **concavity** – volume % threshold used to detect concavity (hulls only)
- **maxVerts** – maximum number of vertices per hull (hulls only)
- **boxMaxError** – max % volume difference for a hull to be converted to a box (hulls only)
- **sphereMaxError** – max % volume difference for a hull to be converted to a sphere (hulls only)
- **capsuleMaxError** – max % volume difference for a hull to be converted to a capsule (hulls only)

Returns true if successful, false otherwise

Example:

```
%this.addCollisionDetail( -1, "box", "bounds" );
%this.addCollisionDetail( -1, "convex hulls", "bounds", 4, 30, 30, 32, 0, 0, 0 );
%this.addCollisionDetail( -1, "convex hulls", "bounds", 4, 30, 30, 32, 50, 50, 50 );
```

int TSShapeConstructor::**addImposter**(int *size*, int *equatorSteps*, int *polarSteps*, int *dl*, int *dim*, bool *includePoles*, float *polarAngle*)

Add (or edit) an imposter detail level to the shape. If the shape already contains an imposter detail level, this command will simply change the imposter settings

Parameters

- **size** – size of the imposter detail level
- **equatorSteps** – defines the number of snapshots to take around the equator. Imagine the object being rotated around the vertical axis, then a snapshot taken at regularly spaced intervals.
- **polarSteps** – defines the number of snapshots taken between the poles (top and bottom), at each equator step. eg. At each equator snapshot, snapshots are taken at regular intervals between the poles.
- **dl** – the detail level to use when generating the snapshots. Note that this is an array index rather than a detail size. So if an object has detail sizes of: 200, 150, and 40, then setting dl to 1 will generate the snapshots using detail size 150.
- **dim** – defines the size of the imposter images in pixels. The larger the number, the more detailed the billboard will be.
- **includePoles** – flag indicating whether to include the "pole" snapshots. ie. the views from the top and bottom of the object.
- **polar_angle** – if pole snapshots are active (includePoles is true), this parameter defines the camera angle (in degrees) within which to render the pole snapshot. eg. if polar_angle is set to 25 degrees, then the snapshot taken at the pole (looking directly down or up at the object) will be rendered when the camera is within 25 degrees of the pole.

Returns true if successful, false otherwise

Example:

```
%this.addImposter( 2, 4, 0, 0, 64, false, 0 );
%this.addImposter( 2, 4, 2, 0, 64, true, 10 );   // this command would edit the existing imposte
```

bool TSShapeConstructor::**addMesh**(string *meshName*, string *srcShape*, string *srcMesh*)

Add geometry from another DTS or DAE shape file into this shape. Any materials required by the source mesh are also copied into this shape.

Parameters

- **meshName** – full name (object name + detail size) of the new mesh. If no detail size is present at the end of the name, a value of 2 is used. An underscore before the number at the end of the name will be interpreted as a negative sign. eg. "MyMesh_4" will be interpreted as "MyMesh-4".
- **srcShape** – name of a shape file (DTS or DAE) that contains the mesh
- **srcMesh** – the full name (object name + detail size) of the mesh to copy from the DTS/DAE file into this shape

Returns true if successful, false otherwise

Example:

```
%this.addMesh( "ColMesh-1", "./collision.dts", "ColMesh", "Col-1" );
%this.addMesh( "SimpleShape10", "./testShape.dae", "MyMesh2",  );
```

bool TSShapeConstructor::**addNode**(string *name*, string *parentName*, TransformF *txfm*, bool *isWorld*)

Add a new node.

Parameters

- **name** – name for the new node (must not already exist)
- **parentName** – name of an existing node to be the parent of the new node. If empty (""), the new node will be at the root level of the node hierarchy.
- **txfm** – (optional) transform string of the form: "pos.x pos.y pos.z rot.x rot.y rot.z rot.angle"
- **isworld** – (optional) flag to set the local-to-parent or the global transform. If false, or not specified, the position and orientation are treated as relative to the node's parent.

Returns true if successful, false otherwise

Example:

```
%this.addNode( "Nose", "Bip01 Head", "0 2 2 0 0 1 0" );
%this.addNode( "myRoot", "", "0 0 4 0 0 1 1.57" );
%this.addNode( "Nodes", "Bip01 Head", "0 2 0 0 0 1 0", true );
```

bool TSShapeConstructor::**addPrimitive**(string *meshName*, string *type*, string *params*, TransformF *txfm*, string *nodeName*)

Add a new mesh primitive to the shape.

Parameters

- **meshName** – full name (object name + detail size) of the new mesh. If no detail size is present at the end of the name, a value of 2 is used. An underscore before the number at the end of the name will be interpreted as a negative sign. eg. "MyMesh_4" will be interpreted as "MyMesh-4".
- **type** – one of: "box", "sphere", "capsule"
- **params** – mesh primitive parameters: for box: "size_x size_y size_z", for sphere: "radius", for capsule: "height radius"
- **txfm** – local transform offset from the node for this mesh

- **nodeName** – name of the node to attach the new mesh to (will change the object's node if adding a new mesh to an existing object)

Returns true if successful, false otherwise

Example:

```
%this.addMesh( "Box4", "box", "2 4 2", "0 2 0 0 0 1 0", "eye" );
%this.addMesh( "Sphere256", "sphere", "2", "0 0 0 0 0 1 0", "root" );
%this.addMesh( "MyCapsule-1", "capsule", "2 5", "0 0 2 0 0 1 0", "base01" );
```

bool TSShapeConstructor::**addSequence**(string *source*, string *name*, int *start*, int *end*, bool *padRot*, bool *padTrans*)

Add a new sequence to the shape.

Parameters

- **source** – the name of an existing sequence, or the name of a DTS or DAE shape or DSQ sequence file. When the shape file contains more than one sequence, the desired sequence can be specified by appending the name to the end of the shape file. eg. "myShape.dts run" would select the "run" sequence from the "myShape.dts" file.
- **name** – name of the new sequence
- **start** – (optional) first frame to copy. Defaults to 0, the first frame in the sequence.
- **end** – (optional) last frame to copy. Defaults to -1, the last frame in the sequence.
- **padRot** – (optional) copy root-pose rotation keys for non-animated nodes. This is useful if the source sequence data has a different root-pose to the target shape, such as if one character was in the T pose, and the other had arms at the side. Normally only nodes that are actually rotated by the source sequence have keyframes added, but setting this flag will also add keyframes for nodes that are not animated, but have a different root-pose rotation to the target shape root pose.
- **padTrans** – (optional) copy root-pose translation keys for non-animated nodes. This is useful if the source sequence data has a different root-pose to the target shape, such as if one character was in the T pose, and the other had arms at the side. Normally only nodes that are actually moved by the source sequence have keyframes added, but setting this flag will also add keyframes for nodes that are not animated, but have a different root-pose position to the target shape root pose.

Returns true if successful, false otherwise

Example:

```
%this.addSequence( "./testShape.dts ambient", "ambient" );
%this.addSequence( "./myPlayer.dae run", "run" );
%this.addSequence( "./player_look.dsq", "look", 0, -1 );     // start to end
%this.addSequence( "walk", "walk_shortA", 0, 4 );            // start to frame 4
%this.addSequence( "walk", "walk_shortB", 4, -1 );           // frame 4 to end
```

bool TSShapeConstructor::**addTrigger**(string *name*, int *keyframe*, int *state*)

Add a new trigger to the sequence.

Parameters

- **name** – name of the sequence to modify
- **keyframe** – keyframe of the new trigger
- **state** – of the new trigger

Returns true if successful, false otherwise

Example:

```
%this.addTrigger( "walk", 3, 1 );
%this.addTrigger( "walk", 5, -1 );
```

void TSShapeConstructor::**dumpShape**(string *filename*)

Dump the shape hierarchy to the console or to a file. Useful for reviewing the result of a series of construction commands.

Parameters **filename** – Destination filename. If not specified, dump to console.

Example:

```
%this.dumpShape();               // dump to console
%this.dumpShape( "./dump.txt" ); // dump to file
```

Box3F TSShapeConstructor::**getBounds**()

Get the bounding box for the shape.

Returns Bounding box "minX minY minZ maxX maxY maxZ"

int TSShapeConstructor::**getDetailLevelCount**()

Get the total number of detail levels in the shape.

Returns the number of detail levels in the shape

int TSShapeConstructor::**getDetailLevelIndex**(int *size*)

Get the index of the detail level with a given size.

Parameters **size** – size of the detail level to lookup

Returns index of the detail level with the desired size, or -1 if no such detail exists

Example:

```
if ( %this.getDetailLevelSize( 32 ) == -1 )
   echo( "Error: This shape does not have a detail level at size 32" );
```

string TSShapeConstructor::**getDetailLevelName**(int *index*)

Get the name of the indexed detail level.

Parameters **index** – detail level index (valid range is 0 - getDetailLevelCount()-1)

Returns the detail level name

Example:

```
// print the names of all detail levels in the shape
%count = %this.getDetailLevelCount();
for ( %i = 0; %i < %count; %i++ )
   echo( %i SPC %this.getDetailLevelName( %i ) );
```

int TSShapeConstructor::**getDetailLevelSize**(int *index*)

Get the size of the indexed detail level.

Parameters **index** – detail level index (valid range is 0 - getDetailLevelCount()-1)

Returns the detail level size

Example:

```
// print the sizes of all detail levels in the shape
%count = %this.getDetailLevelCount();
for ( %i = 0; %i < %count; %i++ )
   echo( "Detail" @ %i @ " has size " @ %this.getDetailLevelSize( %i ) );
```

int TSShapeConstructor::**getImposterDetailLevel**()
Get the index of the imposter (auto-billboard) detail level (if any).

Returns imposter detail level index, or -1 if the shape does not use imposters.

string TSShapeConstructor::**getImposterSettings**(int *index*)
Get the settings used to generate imposters for the indexed detail level.

Parameters **index** – index of the detail level to query (does not need to be an imposter detail level

Returns 1 if this detail level generates imposters, 0 otherwise

Example:

```
// print the imposter detail level settings
%index = %this.getImposterDetailLevel();
if ( %index != -1 )
   echo( "Imposter settings: " @ %this.getImposterSettings( %index ) );
```

int TSShapeConstructor::**getMeshCount**(string *name*)
Get the number of meshes (detail levels) for the specified object.

Parameters **name** – name of the object to query

Returns the number of meshes for this object.

Example:

```
%count = %this.getMeshCount( "SimpleShape" );
```

string TSShapeConstructor::**getMeshMaterial**(string *name*)
Get the name of the material attached to a mesh. Note that only the first material used by the mesh is returned.

Parameters **name** – full name (object name + detail size) of the mesh to query

Returns mapTo field)

Example:

```
echo( "Mesh material is " @ %this.sgetMeshMaterial( "SimpleShape128" ) );
```

string TSShapeConstructor::**getMeshName**(string *name*, int *index*)
Get the name of the indexed mesh (detail level) for the specified object.

Parameters

- **name** – name of the object to query
- **index** – index of the mesh (valid range is 0 - getMeshCount()-1)

Returns the mesh name.

Example:

```
// print the names of all meshes in the shape
%objCount = %this.getObjectCount();
for ( %i = 0; %i < %objCount; %i++ )
{
   %objName = %this.getObjectName( %i );
   %meshCount = %this.getMeshCount( %objName );
   for ( %j = 0; %j < %meshCount; %j++ )
      echo( %this.getMeshName( %objName, %j ) );
}
```

int TSShapeConstructor::**getMeshSize**(string *name*, int *index*)
Get the detail level size of the indexed mesh for the specified object.

Parameters

- **name** – name of the object to query
- **index** – index of the mesh (valid range is 0 - getMeshCount()-1)

Returns the mesh detail level size.

Example:

```
// print sizes for all detail levels of this object
%objName = "trunk";
%count = %this.getMeshCount( %objName );
for ( %i = 0; %i < %count; %i++ )
   echo( %this.getMeshSize( %objName, %i ) );
```

string TSShapeConstructor::**getMeshType**(string *name*)
Get the display type of the mesh.

Parameters **name** – name of the mesh to query

Returns a normal 3D mesh

Example:

```
echo( "Mesh type is " @ %this.getMeshType( "SimpleShape128" ) );
```

int TSShapeConstructor::**getNodeChildCount**(string *name*)
Get the number of children of this node.

Parameters **name** – name of the node to query.

Returns the number of child nodes.

Example:

```
%count = %this.getNodeChildCount( "Bip01 Pelvis" );
```

string TSShapeConstructor::**getNodeChildName**(string *name*, int *index*)
Get the name of the indexed child node.

Parameters

- **name** – name of the parent node to query.
- **index** – index of the child node (valid range is 0 - getNodeChildName()-1).

Returns the name of the indexed child node.

Example:

```
function dumpNode( %shape, %name, %indent )
{
   echo( %indent @ %name );
   %count = %shape.getNodeChildCount( %name );
   for ( %i = 0; %i < %count; %i++ )
      dumpNode( %shape, %shape.getNodeChildName( %name, %i ), %indent @ "" );
}

function dumpShape( %shape )
{
   // recursively dump node hierarchy
```

```
      %count = %shape.getNodeCount();
      for ( %i = 0; %i < %count; %i++ )
      {
         // dump top level nodes
         %name = %shape.getNodeName( %i );
         if ( %shape.getNodeParentName( %name ) $=  )
            dumpNode( %shape, %name, "" );
      }
   }
```

int TSShapeConstructor::**getNodeCount**()
Get the total number of nodes in the shape.

Returns the number of nodes in the shape.

Example:

```
%count = %this.getNodeCount();
```

int TSShapeConstructor::**getNodeIndex**(string *name*)
Get the index of the node.

Parameters **name** – name of the node to lookup.

Returns the index of the named node, or -1 if no such node exists.

Example:

```
// get the index of Bip01 Pelvis node in the shape
%index = %this.getNodeIndex( "Bip01 Pelvis" );
```

string TSShapeConstructor::**getNodeName**(int *index*)
Get the name of the indexed node.

Parameters **index** – index of the node to lookup (valid range is 0 - getNodeCount()-1).

Returns the name of the indexed node, or “” if no such node exists.

Example:

```
// print the names of all the nodes in the shape
%count = %this.getNodeCount();
for (%i = 0; %i < %count; %i++)
   echo(%i SPC %this.getNodeName(%i));
```

int TSShapeConstructor::**getNodeObjectCount**(string *name*)
Get the number of geometry objects attached to this node.

Parameters **name** – name of the node to query.

Returns the number of attached objects.

Example:

```
%count = %this.getNodeObjectCount( "Bip01 Head" );
```

string TSShapeConstructor::**getNodeObjectName**(string *name*, int *index*)
Get the name of the indexed object.

Parameters

- **name** – name of the node to query.
- **index** – index of the object (valid range is 0 - getNodeObjectCount()-1).

Returns the name of the indexed object.

Example:

```
// print the names of all objects attached to the node
%count = %this.getNodeObjectCount( "Bip01 Head" );
for ( %i = 0; %i < %count; %i++ )
   echo( %this.getNodeObjectName( "Bip01 Head", %i ) );
```

string TSShapeConstructor::**getNodeParentName**(string *name*)

Get the name of the node's parent. If the node has no parent (ie. it is at the root level), return an empty string.

Parameters **name** – name of the node to query.

Returns the name of the node's parent, or "" if the node is at the root level

Example:

```
echo( "Bip01 Pelvis parent = " @ %this.getNodeParentName( "Bip01 Pelvis ") );
```

TransformF TSShapeConstructor::**getNodeTransform**(string *name*, bool *isWorld*)

Get the base (ie. not animated) transform of a node.

Parameters

- **name** – name of the node to query.
- **isWorld** – true to get the global transform, false (or omitted) to get the local-to-parent transform.

Returns the node transform in the form "pos.x pos.y pos.z rot.x rot.y rot.z rot.angle".

Example:

```
%ret = %this.getNodeTransform( "mount0" );
%this.setNodeTransform( "mount4", %ret );
```

int TSShapeConstructor::**getObjectCount**()

Get the total number of objects in the shape.

Returns the number of objects in the shape.

Example:

```
%count = %this.getObjectCount();
```

int TSShapeConstructor::**getObjectIndex**(string *name*)

Get the index of the first object with the given name.

Parameters **name** – name of the object to get.

Returns the index of the named object.

Example:

```
%index = %this.getObjectIndex( "Head" );
```

string TSShapeConstructor::**getObjectName**(int *index*)

Get the name of the indexed object.

Parameters **index** – index of the object to get (valid range is 0 - getObjectCount()-1).

Returns the name of the indexed object.

Example:

```
// print the names of all objects in the shape
%count = %this.getObjectCount();
for ( %i = 0; %i < %count; %i++ )
   echo( %i SPC %this.getObjectName( %i ) );
```

string TSShapeConstructor::**getObjectNode**(string *name*)
Get the name of the node this object is attached to.

Parameters **name** – name of the object to get.

Returns the name of the attached node, or an empty string if this object is not attached to a node (usually the case for skinned meshes).

Example:

```
echo( "Hand is attached to " @ %this.getObjectNode( "Hand" ) );
```

string TSShapeConstructor::**getSequenceBlend**(string *name*)
Get information about blended sequences.

Parameters **name** – name of the sequence to query

Returns a boolean flag indicating whether this sequence is a blend

Example:

```
%blendData = %this.getSequenceBlend( "look" );
if ( getField( %blendData, 0 ) )
   echo( "look is a blend, reference: " @ getField( %blendData, 1 ) );
```

int TSShapeConstructor::**getSequenceCount**()
Get the total number of sequences in the shape.

Returns the number of sequences in the shape

bool TSShapeConstructor::**getSequenceCyclic**(string *name*)
Check if this sequence is cyclic (looping).

Parameters **name** – name of the sequence to query

Returns true if this sequence is cyclic, false if not

Example:

```
if ( !%this.getSequenceCyclic( "ambient" ) )
   error( "ambient sequence is not cyclic!" );
```

int TSShapeConstructor::**getSequenceFrameCount**(string *name*)
Get the number of keyframes in the sequence.

Parameters **name** – name of the sequence to query

Returns number of keyframes in the sequence

Example:

```
echo( "Run has " @ %this.getSequenceFrameCount( "run" ) @ " keyframes" );
```

string TSShapeConstructor::**getSequenceGroundSpeed**(string *name*)
Get the ground speed of the sequence.

Parameters **name** – name of the sequence to query

Returns string of the form: “trans.x trans.y trans.z rot.x rot.y rot.z”

Example:

```
%speed = VectorLen( getWords( %this.getSequenceGroundSpeed( "run" ), 0, 2 ) );
   echo( "Run moves at " @ %speed @ " units per frame" );
```

int TSShapeConstructor::**getSequenceIndex**(string *name*)

Find the index of the sequence with the given name.

Parameters **name** – name of the sequence to lookup

Returns index of the sequence with matching name, or -1 if not found

Example:

```
// Check if a given sequence exists in the shapeif ( %this.getSequenceIndex( "walk" ) == -1 )
   echo( "Could not find walk sequence" );
```

string TSShapeConstructor::**getSequenceName**(int *index*)

Get the name of the indexed sequence.

Parameters **index** – index of the sequence to query (valid range is 0 - getSequenceCount()-1)

Returns the name of the sequence

Example:

```
// print the name of all sequences in the shape
%count = %this.getSequenceCount();
for ( %i = 0; %i < %count; %i++ )
   echo( %i SPC %this.getSequenceName( %i ) );
```

float TSShapeConstructor::**getSequencePriority**(string *name*)

Get the priority setting of the sequence.

Parameters **name** – name of the sequence to query

Returns priority value of the sequence

string TSShapeConstructor::**getSequenceSource**(string *name*)

Get information about where the sequence data came from. For example, whether it was loaded from an external DSQ file.

Parameters **name** – name of the sequence to query

Returns the source of the animation data, such as the path to a DSQ file, or the name of an existing sequence in the shape. This field will be empty for sequences already embedded in the DTS or DAE file.

Example:

```
// print the source for the walk animationecho( "walk source:" SPC getField( %this.getSequenceSo
```

int TSShapeConstructor::**getTargetCount**()

Get the number of materials in the shape.

Returns the number of materials in the shape.

Example:

```
%count = %this.getTargetCount();
```

string TSShapeConstructor::**getTargetName**(int *index*)

Get the name of the indexed shape material.

Parameters **index** – index of the material to get (valid range is 0 - getTargetCount()-1).

Returns the name of the indexed material.

Example:

```
%count = %this.getTargetCount();
for ( %i = 0; %i < %count; %i++ )
   echo( "Target " @ %i @ ": " @ %this.getTargetName( %i ) );
```

string TSShapeConstructor::**getTrigger**(string *name*, int *index*)
Get information about the indexed trigger.

Parameters

- **name** – name of the sequence to query
- **index** – index of the trigger (valid range is 0 - getTriggerCount()-1)

Returns string of the form "frame state"

Example:

```
// print all triggers in the sequence
%count = %this.getTriggerCount( "back" );
for ( %i = 0; %i < %count; %i++ )
   echo( %i SPC %this.getTrigger( "back", %i ) );
```

int TSShapeConstructor::**getTriggerCount**(string *name*)
Get the number of triggers in the specified sequence.

Parameters **name** – name of the sequence to query

Returns number of triggers in the sequence

void TSShapeConstructor::**notifyShapeChanged**()
Notify game objects that this shape file has changed, allowing them to update internal data if needed.

void TSShapeConstructor::**onLoad**()
Called immediately after the DTS or DAE file has been loaded; before the shape data is available to any other object (StaticShape , Player etc). This is where you should put any post-load commands to modify the shape in-memory such as addNode, renameSequence etc.

void TSShapeConstructor::**onUnload**()
Called when the DTS or DAE resource is flushed from memory. Not normally required, but may be useful to perform cleanup.

bool TSShapeConstructor::**removeDetailLevel**(int *index*)
Remove the detail level (including all meshes in the detail level).

Parameters **size** – size of the detail level to remove

Returns true if successful, false otherwise

Example:

```
%this.removeDetailLevel( 2 );
```

bool TSShapeConstructor::**removeImposter**()
Remove the imposter detail level (if any) from the shape.

Returns true if successful, false otherwise

bool TSShapeConstructor::**removeMesh**(string *name*)
Remove a mesh from the shape. If all geometry is removed from an object, the object is also removed.

Parameters **name** – full name (object name + detail size) of the mesh to remove

Returns true if successful, false otherwise

Example:

```
%this.removeMesh( "SimpleShape128" );
```

bool TSShapeConstructor::**removeNode**(string *name*)
Remove a node from the shape. The named node is removed from the shape, including from any sequences that use the node. Child nodes and objects attached to the node are re-assigned to the node's parent.

Parameters **name** – name of the node to remove.

Returns true if successful, false otherwise.

Example:

```
%this.removeNode( "Nose" );
```

bool TSShapeConstructor::**removeObject**(string *name*)
Remove an object (including all meshes for that object) from the shape.

Parameters **name** – name of the object to remove.

Returns true if successful, false otherwise.

Example:

```
// clear all objects in the shape
%count = %this.getObjectCount();
for ( %i = %count-1; %i >= 0; %i-- )
   %this.removeObject( %this.getObjectName(%i) );
```

bool TSShapeConstructor::**removeSequence**(string *name*)
Remove the sequence from the shape.

Parameters **name** – name of the sequence to remove

Returns true if successful, false otherwise

bool TSShapeConstructor::**removeTrigger**(string *name*, int *keyframe*, int *state*)
Remove a trigger from the sequence.

Parameters

- **name** – name of the sequence to modify
- **keyframe** – keyframe of the trigger to remove
- **state** – of the trigger to remove

Returns true if successful, false otherwise

Example:

```
%this.removeTrigger( "walk", 3, 1 );
```

bool TSShapeConstructor::**renameDetailLevel**(string *oldName*, string *newName*)
Rename a detail level.

Parameters

- **oldName** – current name of the detail level
- **newName** – new name of the detail level

Returns true if successful, false otherwise

Example:

```
%this.renameDetailLevel( "detail-1", "collision-1" );
```

bool TSShapeConstructor::**renameNode**(string *oldName*, string *newName*)
Rename a node.

Parameters

- **oldName** – current name of the node
- **newName** – new name of the node

Returns true if successful, false otherwise

Example:

```
%this.renameNode( "Bip01 L Hand", "mount5" );
```

bool TSShapeConstructor::**renameObject**(string *oldName*, string *newName*)
Rename an object.

Parameters

- **oldName** – current name of the object
- **newName** – new name of the object

Returns true if successful, false otherwise

Example:

```
%this.renameObject( "MyBox", "Box" );
```

bool TSShapeConstructor::**renameSequence**(string *oldName*, string *newName*)
Rename a sequence.

Parameters

- **oldName** – current name of the sequence
- **newName** – new name of the sequence

Returns true if successful, false otherwise

Example:

```
%this.renameSequence( "walking", "walk" );
```

void TSShapeConstructor::**saveShape**(string *filename*)
Save the shape (with all current changes) to a new DTS file.

Parameters **filename** – Destination filename.

Example:

```
%this.saveShape( "./myShape.dts" );
```

bool TSShapeConstructor::**setBounds**(Box3F *bbox*)
Set the shape bounds to the given bounding box.

Parameters **Bounding** – box "minX minY minZ maxX maxY maxZ"

Returns true if successful, false otherwise

int TSShapeConstructor::**setDetailLevelSize**(int *index*, int *newSize*)
Change the size of a detail level.

Parameters

- **index** – index of the detail level to modify
- **newSize** – new size for the detail level

Returns new index for this detail level

Example:

```
%this.setDetailLevelSize( 2, 256 );
```

bool TSShapeConstructor::**setMeshMaterial**(string *meshName*, string *matName*)
Set the name of the material attached to the mesh.

Parameters

- **meshName** – full name (object name + detail size) of the mesh to modify
- **matName** – name of the material to attach. This could be the base name of the diffuse texture (eg. "test_mat" for "test_mat.jpg"), or the name of a Material object already defined in script.

Returns true if successful, false otherwise

Example:

```
// set the mesh material
%this.setMeshMaterial( "SimpleShape128", "test_mat" );
```

bool TSShapeConstructor::**setMeshSize**(string *name*, int *size*)
Change the detail level size of the named mesh.

Parameters

- **name** – full name (object name + current size) of the mesh to modify
- **size** – new detail level size

Returns true if successful, false otherwise.

Example:

```
%this.setMeshSize( "SimpleShape128", 64 );
```

bool TSShapeConstructor::**setMeshType**(string *name*, string *type*)
Set the display type for the mesh.

Parameters

- **name** – full name (object name + detail size) of the mesh to modify
- **type** – the new type for the mesh: "normal", "billboard" or "billboardzaxis"

Returns true if successful, false otherwise

Example:

```
// set the mesh to be a billboard
%this.setMeshType( "SimpleShape64", "billboard" );
```

bool TSShapeConstructor::**setNodeParent**(string *name*, string *parentName*)
Set the parent of a node.

Parameters

- **name** – name of the node to modify

- **parentName** – name of the parent node to set (use "" to move the node to the root level)

Returns true if successful, false if failed

Example:

```
%this.setNodeParent( "Bip01 Pelvis", "start01" );
```

bool TSShapeConstructor::**setNodeTransform**(string *name*, TransformF *txfm*, bool *isWorld*)
Set the base transform of a node. That is, the transform of the node when in the root (not-animated) pose.

Parameters

- **name** – name of the node to modify
- **txfm** – transform string of the form: "pos.x pos.y pos.z rot.x rot.y rot.z rot.angle"
- **isworld** – (optional) flag to set the local-to-parent or the global transform. If false, or not specified, the position and orientation are treated as relative to the node's parent.

Returns true if successful, false otherwise

Example:

```
%this.setNodeTransform( "mount0", "0 0 1 0 0 1 0" );
%this.setNodeTransform( "mount0", "0 0 0 0 0 1 1.57" );
%this.setNodeTransform( "mount0", "1 0 0 0 0 1 0", true );
```

bool TSShapeConstructor::**setObjectNode**(string *objName*, string *nodeName*)
Set the node an object is attached to. When the shape is rendered, the object geometry is rendered at the node's current transform.

Parameters

- **objName** – name of the object to modify
- **nodeName** – name of the node to attach the object to

Returns true if successful, false otherwise

Example:

```
%this.setObjectNode( "Hand", "Bip01 LeftHand" );
```

bool TSShapeConstructor::**setSequenceBlend**(string *name*, bool *blend*, string *blendSeq*, int *blendFrame*)
Mark a sequence as a blend or non-blend. A blend sequence is one that will be added on top of any other playing sequences. This is done by storing the animated node transforms relative to a reference frame, rather than as absolute transforms.

Parameters

- **name** – name of the sequence to modify
- **blend** – true to make the sequence a blend, false for a non-blend
- **blendSeq** – the name of the sequence that contains the blend reference frame
- **blendFrame** – the reference frame in the blendSeq sequence

Returns true if successful, false otherwise

Example:

```
%this.setSequenceBlend( "look", true, "root", 0 );
```

bool TSShapeConstructor::**setSequenceCyclic**(string *name*, bool *cyclic*)
Mark a sequence as cyclic or non-cyclic.

Parameters

- **name** – name of the sequence to modify
- **cyclic** – true to make the sequence cyclic, false for non-cyclic

Returns true if successful, false otherwise

Example:

```
%this.setSequenceCyclic( "ambient", true );
%this.setSequenceCyclic( "shoot", false );
```

bool TSShapeConstructor::**setSequenceGroundSpeed**(string *name*, Point3F *transSpeed*, Point3F *rotSpeed*)
Set the translation and rotation ground speed of the sequence. The ground speed of the sequence is set by generating ground transform keyframes. The ground translational and rotational speed is assumed to be constant for the duration of the sequence. Existing ground frames for the sequence (if any) will be replaced.

Parameters

- **name** – name of the sequence to modify
- **transSpeed** – translational speed (trans.x trans.y trans.z) in Torque units per frame
- **rotSpeed** – (optional) rotational speed (rot.x rot.y rot.z) in radians per frame. Default is "0 0 0"

Returns true if successful, false otherwise

Example:

```
%this.setSequenceGroundSpeed( "run", "5 0 0" );
%this.setSequenceGroundSpeed( "spin", "0 0 0", "4 0 0" );
```

bool TSShapeConstructor::**setSequencePriority**(string *name*, float *priority*)
Set the sequence priority.

Parameters

- **name** – name of the sequence to modify
- **priority** – new priority value

Returns true if successful, false otherwise

void TSShapeConstructor::**writeChangeSet**()
Write the current change set to a TSShapeConstructor script file. The name of the script file is the same as the model, but with .cs extension. eg. myShape.cs for myShape.dts or myShape.dae.

Fields

bool TSShapeConstructor::**adjustCenter**
Translate COLLADA model on import so the origin is at the center. No effect for DTS files.

bool TSShapeConstructor::**adjustFloor**
Translate COLLADA model on import so origin is at the (Z axis) bottom of the model. No effect for DTS files. This can be used along with adjustCenter to have the origin at the center of the bottom of the model.

string TSShapeConstructor::**alwaysImport**
TAB separated patterns of nodes to import even if in neverImport list. No effect for DTS files. Torque allows unwanted nodes in COLLADA (.dae) files to to be ignored during import. This field contains a TAB separated

list of patterns to match node names. Any node that matches one of the patterns in the list will always be imported, even if it also matches the neverImport list

Example:

```
singleton TSShapeConstructor(MyShapeDae)
{
   baseShape = "./myShape.dae";
   alwaysImport = "mount*" TAB "eye";
   neverImport = "*-PIVOT";
}
```

string TSShapeConstructor::**alwaysImportMesh**
TAB separated patterns of meshes to import even if in neverImportMesh list. No effect for DTS files. Torque allows unwanted meshes in COLLADA (.dae) files to to be ignored during import. This field contains a TAB separated list of patterns to match mesh names. Any mesh that matches one of the patterns in the list will always be imported, even if it also matches the neverImportMesh list

Example:

```
singleton TSShapeConstructor(MyShapeDae)
{
   baseShape = "./myShape.dae";
   alwaysImportMesh = "body*" TAB "armor" TAB "bounds";
   neverImportMesh = "*-dummy";
}
```

filename TSShapeConstructor::**baseShape**
Specifies the path to the DTS or DAE file to be operated on by this object. Since the TSShapeConstructor script must be in the same folder as the DTS or DAE file, it is recommended to use a relative path so that the shape and script files can be copied to another location without having to modify the path.

bool TSShapeConstructor::**forceUpdateMaterials**
Forces update of the materials.cs file in the same folder as the COLLADA (.dae) file, even if Materials already exist. No effect for DTS files. Normally only Materials that are not already defined are written to materials.cs.

bool TSShapeConstructor::**ignoreNodeScale**
Ignore lt scale gt elements inside COLLADA lt node gt s. No effect for DTS files. This field is a workaround for certain exporters that generate bad node scaling, and is not usually required.

TSShapeConstructorLodType TSShapeConstructor::**lodType**
Control how the COLLADA (.dae) importer interprets LOD in the model. No effect for DTS files. Set to one of the following values:

string TSShapeConstructor::**matNamePrefix**
Prefix to apply to all material map names in the COLLADA (.dae) file. No effect for DTS files. This field is useful to avoid material name clashes for exporters that generate generic material names like “texture0” or “material1”.

string TSShapeConstructor::**neverImport**
TAB separated patterns of nodes to ignore on loading. No effect for DTS files. Torque allows unwanted nodes in COLLADA (.dae) files to to be ignored during import. This field contains a TAB separated list of patterns to match node names. Any node that matches one of the patterns in the list will not be imported (unless it matches the alwaysImport list.

string TSShapeConstructor::**neverImportMesh**
TAB separated patterns of meshes to ignore on loading. No effect for DTS files. Torque allows unwanted meshes in COLLADA (.dae) files to to be ignored during import. This field contains a TAB separated list of patterns to match mesh names. Any mesh that matches one of the patterns in the list will not be imported (unless it matches the alwaysImportMesh list.

filename TSShapeConstructor::**sequence**
Legacy method of adding sequences to a DTS or DAE shape after loading.

Example:

```
singleton TSShapeConstructor(MyShapeDae)
{
   baseShape = "./myShape.dae";
   sequence = "../anims/root.dae root";
   sequence = "../anims/walk.dae walk";
   sequence = "../anims/jump.dsq jump";
}
```

int TSShapeConstructor::**singleDetailSize**
Sets the detail size when lodType is set to SingleSize. No effect otherwise, and no effect for DTS files.

float TSShapeConstructor::**unit**
Override the lt unit gt element in the COLLADA (.dae) file. No effect for DTS files. COLLADA (.dae) files usually contain a lt unit gt element that indicates the 'real world' units that the model is described in. It means you can work in sensible and meaningful units in your modeling app. For example, if you were modeling a small object like a cup, it might make sense to work in inches (1 MAX unit = 1 inch), but if you were modeling a building, it might make more sense to work in feet (1 MAX unit = 1 foot). If you export both models to COLLADA, T3D will automatically scale them appropriately. 1 T3D unit = 1 meter, so the cup would be scaled down by 0.0254, and the building scaled down by 0.3048, given them both the correct scale relative to each other. Omit the field or set to -1 to use the value in the .dae file (1.0 if the lt unit gt element is not present)

TSShapeConstructorUpAxis TSShapeConstructor::**upAxis**
Override the lt up_axis gt element in the COLLADA (.dae) file. No effect for DTS files. Set to one of the following values:

TSStatic A static object derived from a 3D model file and placed within the game world.

Inherit: SceneObject

Description A static object derived from a 3D model file and placed within the game world.

TSStatic is the most basic 3D shape in Torque. Unlike StaticShape it doesn't make use of a datablock. It derrives directly from SceneObject. This makes TSStatic extremely light weight, which is why the Tools use this class when you want to drop in a DTS or DAE object.

While a TSStatic doesn't provide any motion – it stays were you initally put it – it does allow for a single ambient animation sequence to play when the object is first added to the scene.

Example:

```
newTSStatic(Team1Base) {
   shapeName = "art/shapes/desertStructures/station01.dts";
   playAmbient = "1";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useCustomAmbientLighting = "0";
   customAmbientLighting = "0 0 0 1";
   collisionType = "Visible Mesh";
   decalType = "Collision Mesh";
   allowPlayerStep = "1";
   renderNormals = "0";
   forceDetail = "-1";
   position = "315.18 -180.418 244.313";
   rotation = "0 0 1 195.952";
```

```
   scale = "1 1 1";
   isRenderEnabled = "true";
   canSaveDynamicFields = "1";
};
```

Methods

void TSStatic::**changeMaterial**(string *mapTo*, Material *oldMat*, Material *newMat*)

Change one of the materials on the shape. This method changes materials per mapTo with others. The material that is being replaced is mapped to unmapped_mat as a part of this transition.

Parameters

- **mapTo** – the name of the material target to remap (from getTargetName)
- **oldMat** – the old Material that was mapped
- **newMat** – the new Material to map

Example:

```
// remap the first material in the shape
%mapTo = %obj.getTargetName( 0 );
%obj.changeMaterial( %mapTo, 0, MyMaterial );
```

string TSStatic::**getModelFile**()

Get the model filename used by this shape.

Returns the shape filename

Example:

```
// Acquire the model filename used on this shape.
%modelFilename = %obj.getModelFile();
```

int TSStatic::**getTargetCount**()

Get the number of materials in the shape.

Returns the number of materials in the shape.

string TSStatic::**getTargetName**(int *index*)

Get the name of the indexed shape material.

Parameters **index** – index of the material to get (valid range is 0 - getTargetCount()-1).

Returns the name of the indexed material.

Fields

bool TSStatic::**allowPlayerStep**

Allow a Player to walk up sloping polygons in the TSStatic (based on the collisionType). When set to false, the slightest bump will stop the player from walking on top of the object.

TSMeshType TSStatic::**collisionType**

The type of mesh data to use for collision queries.

TSMeshType TSStatic::**decalType**

The type of mesh data used to clip decal polygons against.

int TSStatic::**forceDetail**

Forces rendering to a particular detail level.

bool TSStatic::**meshCulling**
Enables detailed culling of meshes within the TSStatic . Should only be used with large complex shapes like buildings which contain many submeshes.

bool TSStatic::**originSort**
Enables translucent sorting of the TSStatic by its origin instead of the bounds.

bool TSStatic::**playAmbient**
Enables automatic playing of the animation sequence named "ambient" (if it exists) when the TSStatic is loaded.

float TSStatic::**renderNormals**
Debug rendering mode shows the normals for each point in the TSStatic's mesh.

filename TSStatic::**shapeName**
Path and filename of the model file (.DTS, .DAE) to use for this TSStatic .

string TSStatic::**skin**
The skin applied to the shape. 'Skinning' the shape effectively renames the material targets, allowing different materials to be used on different instances of the same model. Any material targets that start with the old skin name have that part of the name replaced with the new skin name. The initial old skin name is "base". For example, if a new skin of "blue" was applied to a model that had material targets base_body and face , the new targets would be blue_body and face . Note that face was not renamed since it did not start with the old skin name of "base". To support models that do not use the default "base" naming convention, you can also specify the part of the name to replace in the skin field itself. For example, if a model had a material target called shapemat , we could apply a new skin "shape=blue", and the material target would be renamed to bluemat (note "shape" has been replaced with "blue"). Multiple skin updates can also be applied at the same time by separating them with a semicolon. For example: "base=blue;face=happy_face". Material targets are only renamed if an existing Material maps to that name, or if there is a diffuse texture in the model folder with the same name as the new target.

TurretShape Base turret class.

Inherit: Item

Description Base turret class.

Uses the TurretShapeData datablock for common properties.

The TurretShape class provides a player mountable turret. It also forms the base for AITurretShape, an AI controlled turret. It is based on the Item class, which allows turrets to be treated as any other Item by the Player, such as throwing smaller turrets. When used directly, TurretShape takes input moves from the player's GameConnection to rotate the turret and trigger its weapons.

A turret consists of two components. There is the TurretShape object (or AITurretShape), and then there are one or more ShapeBaseImageData objects that are mounted to the turret. The TurretShape provides the weapons platform that rotates towards a target. The ShapeBaseImageData provides the actual weapon that fires at the target. Any standard ShapeBaseImageData weapon may be used.

Shape File Nodes The shape file used for the TurretShape needs to have a number of defined nodes. The first node is named 'heading'. The heading node is special in that it is controlled by the TurretShape code. This means that it should not be animated by the artist, nor should it have anything but the default transform applied to it. This doesn't stop the heading node's parent or its children from being animated, however.

The second special node is named 'pitch'. The pitch node is also controlled by the TurretShape code so it too should not be animated within the shape file. Typically the pitch node will be a child of the heading node, although it need not be a direct child. The pitch node is also optional if you don't want the TurretShape to pitch towards its target. In this

case you may be doing something special with the mounted weapon to have its projectiles automatically aim towards the target.

The next set of nodes are weaponMount0 through weaponMount3. These provide up to four mounting points for weapons on the turret. Typically these are children of the pitch nodes, although they need not be direct children of that node. You do not need to have all of these weapon mount point nodes defined within the shape. Only as many as you need for the weapons. The mounted ShapeBaseImageData weapons' mountPoint node will mount to these nodes.

There are four optional nodes named pitch0 through pitch3 that may be used in special cases. These nodes are also controlled by the TurretShape code and have the same restrictions. Their rotation exactly matches that of the standard pitch node. These exist for mounted weapons that may not all rotate about the same x axis. For example, a turret may have two sets of weapons, one mounted above the other. These two sets of weapons could all share the same point of rotation (the pitch node) which means they'll rotate as a group. Or the top weapons could be attached to the pitch node while the bottom weapons could be attached to the pitch0 node. This makes the two sets of weapons rotate about their own centers and provides an entirely different look.

You could also use these additional pitchN nodes to animate some non-weapon attachments on the turret, such as a radar dish or targeting scope. TurretShape also supports four optional heading0 through heading3 nodes that operate in the same way as the pitchN nodes.

Weapon Mounting TurretShape weapon mounting is done within the TurretShapeData::onAdd() script method. This method makes use of datablock fields that are only defined in script and are not passed along to the client. The first field is numWeaponMountPoints that defines the number of weapons that will be mounted and the number of weaponMountN nodes to expect within the turret's shape file.

The other fields that are required to mount weapons are the weapon[], weaponAmmo[] and weaponAmmoAmount[] arrays – one of each per weapon to mount. The weapon[] array points to an ItemData datablock that defines the weapon (just like any Player weapon). The weaponAmmo[] array points to an ItemData datablock that defines the ammo to use for the weapon. Finally, the weaponAmmoAmount[] array is the quantity of ammo the turret has for that weapon.

As turrets use the same inventory system as players, you also need to define the maximum number of weapons and ammo that the turret may possess. Here is an example of setting up three weapons and their ammo for a turret (a TurretShapeData fragment):

Example:

```
// Weapon mounting
   numWeaponMountPoints = 3;

   weapon[0] = TurretWeapon;
   weaponAmmo[0] = BulletAmmo;
   weaponAmmoAmount[0] = 10000;

   weapon[1] = TurretWeaponB;
   weaponAmmo[1] = BulletAmmo;
   weaponAmmoAmount[1] = 10000;

   weapon[2] = TurretWeapon;
   weaponAmmo[2] = BulletAmmo;
   weaponAmmoAmount[2] = 10000;

   maxInv[TurretWeaponB] = 1;
   maxInv[TurretWeapon] = 2;
   maxInv[BulletAmmo] = 10000;
```

Mounted Weapon States There are a couple of things to be aware of so that an turret's mounted weapons play along with the turret's states, especially for AI turrets. Setting TurretShapeData::startLoaded to true indicates that all mounted weapons will start loaded when their state machines start up. A static turret placed with the World Editor would normally begin this way. Setting TurretShapeData::startLoaded to false causes all mounted weapons to not start in a loaded state. This can be used to have the mounted weapons begin in some folded state when a deployable turret is thrown by the player. When a thrown turret comes to rest and begins to deploy, all mounted weapons are automatically set to the loaded state so they may also unfold, start up, or show some other method that the weapon is becoming ready to fight. This could also be used for player mountable turrets so that the weapons come to life when a player mounts the turret.

The default scripts for AITurretShapeData also fires the first image generic trigger on all mounted weapons when the turret is destroyed. This shows up as stateTransitionGeneric0In within a weapon image's state machine. This allows for all weapons to show that they are destroyed or shutting down. Something similar could be done for general TurretShapeData turrets.

Weapons can also feed back to the turret they are mounted on. TurretShape supports the standard ShapeBaseImageData stateRecoil and will play the indicated animation, if available. You can also use ShapeBaseImageData's stateShapeSequence field to play a generic sequence on the turret at any time from a mounted weapon.

Player Mounting Turrets act very similar to vehicles when it comes to player mounting. By default colliding with a turret causes the player to mount it, if the turret is free.

When it comes to firing the turret's weapons there are a number of methods that are triggered based on the weaponLinkType on the TurretShapeData datablock. Setting this field to FireTogether causes all weapons to fire at once based on the input from trigger 0. Using GroupedFire will make weaponMount0 and weaponMount2 mounted weapons fire on trigger 0, and weaponMount1 and weaponMount3 mounted weapons fire on trigger 1. Finally, IndividualFire will have each weaponMountN mounted weapons fire based on their own trigger (0 through 3). This provides exact control over which turret weapon will fire when there are multiple weapons mounted.

The player mounting callbacks are done using the TurretBaseData datablock on the server, and in a special case on the TurretBase object on the client. The server side makes use of the standard TurretBaseData::onMountObject() and TurretBaseData::onUnmountObject() callbacks. See those for more information.

When a player mounted turret is destroyed the TurretShapeData::damage() method will automatically kill all mounted players. To modify this behaviour – such as only dismounting players from a destroyed turret – you'll need to create your own damage() method for your turret's datablock.

On the client side the special turretMountCallback() callback function is called for the TurretShape object that is being mounted. This callback receives the SimObjectID of the turret object, the SimObjectID of the player doing the mounting or unmounting, and a Boolean set to true if mounting and false if unmounting. As this callback is made on the client, it allows the client to set up any action maps, make HUD changes, etc.

Example:

```
// ----------------------------------------------------------------------------// Turret Support// --
function turretMountCallback(%turret, %player, %mounted)
{
   echo ( "\c4turretMountCallback ->" @ %mounted );

   if (%mounted)
   {
      // Push the action map
      turretMap.push();
   }
   else
   {
      // Pop the action map
      turretMap.pop();
```

```
    }
}
```

Turret Destruction When a turret is destroyed the default TurretBaseData::onDestroyed() method is called. This causes the turret to sit in a Dead state for TurretBase::DestroyedFadeDelay milliseconds, and then the turret will fade away. If the turret is marked to respawn – TurretShape::doRespawn() returns true – then the turret is respawned after TurretShape::RespawnTime milliseconds. By default all turrets placed in the World Editor are marked to respawn.

Turret Optional Animation Sequences If present in the TurretShape's shape, the optional 'heading' and 'pitch' sequences will be played as the turret rotates. These sequences are given a timeline position that corresponds to the turret's rotation within its minimum and maximum ranges. These sequences could be used to rotate wheels or gears on the turret as it rotates, for example.

Methods

bool `TurretShape::doRespawn()`
Does the turret respawn after it has been destroyed.

Returns True if the turret respawns.

bool `TurretShape::getAllowManualFire()`
Get if the turret is allowed to fire through moves.

Returns True if the turret is allowed to fire through moves.

bool `TurretShape::getAllowManualRotation()`
Get if the turret is allowed to rotate through moves.

Returns True if the turret is allowed to rotate through moves.

string `TurretShape::getState()`
Get the name of the turret's current state. The state is one of the following:

- Dead - The TurretShape is destroyed.
- Mounted - The TurretShape is mounted to an object such as a vehicle.
- Ready - The TurretShape is free to move. The usual state.

Returns The current state; one of: "Dead", "Mounted", "Ready"

Point3F `TurretShape::getTurretEulerRotation()`
Get Euler rotation of this turret's heading and pitch nodes.

Returns the orientation of the turret's heading and pitch nodes in the form of rotations around the X, Y and Z axes in degrees.

void `TurretShape::setAllowManualFire`(bool *allow*)
Set if the turret is allowed to fire through moves.

Parameters **allow** – If true then the turret may be fired through moves.

void `TurretShape::setAllowManualRotation`(bool *allow*)
Set if the turret is allowed to rotate through moves.

Parameters **allow** – If true then the turret may be rotated through moves.

void `TurretShape::setTurretEulerRotation`(Point3F *rot*)
Set Euler rotation of this turret's heading and pitch nodes in degrees.

Parameters **rot** – The rotation in degrees. The pitch is the X component and the heading is the Z component. The Y component is ignored.

Fields

bool TurretShape::**respawn**
Respawn the turret after it has been destroyed. If true, the turret will respawn after it is destroyed.

TurretShapeData object.

Inherit: ItemData

Description Defines properties for a TurretShape object.

Methods

void TurretShapeData::**onMountObject**(TurretShape *turret*, SceneObject *obj*, int *node*)
Informs the TurretShapeData object that a player is mounting it.

Parameters

- **turret** – The TurretShape object.
- **obj** – The player that is mounting.
- **node** – The node the player is mounting to.

void TurretShapeData::**onStickyCollision**(TurretShape *obj*)
Informs the TurretData object that it is now sticking to another object. This callback is only called if the TurretData::sticky property for this Turret is true.

Parameters **obj** – The Turret object that is colliding.

void TurretShapeData::**onUnmountObject**(TurretShape *turret*, SceneObject *obj*)
Informs the TurretShapeData object that a player is unmounting it.

Parameters

- **turret** – The TurretShape object.
- **obj** – The player that is unmounting.

Fields

float TurretShapeData::**cameraOffset**
Vertical (Z axis) height of the camera above the turret.

float TurretShapeData::**headingRate**
Degrees per second rotation. A value of 0 means no rotation is allowed. A value less than 0 means the rotation is instantaneous.

float TurretShapeData::**maxHeading**
Maximum number of degrees to rotate from center. A value of 180 or more degrees indicates the turret may rotate completely around.

float TurretShapeData::**maxPitch**
Maximum number of degrees to rotate up from straight ahead.

float TurretShapeData::**minPitch**
Minimum number of degrees to rotate down from straight ahead.

float TurretShapeData::**pitchRate**
Degrees per second rotation. A value of 0 means no rotation is allowed. A value less than 0 means the rotation is instantaneous.

bool TurretShapeData::**startLoaded**
Does the turret's mounted weapon(s) start in a loaded state. True indicates that all mounted weapons start in a loaded state.

TurretShapeFireLinkType TurretShapeData::**weaponLinkType**
Set how the mounted weapons are linked and triggered.

•FireTogether: All weapons fire under trigger 0.

•GroupedFire: Weapon mounts 0,2 fire under trigger 0, mounts 1,3 fire under trigger 1.

•IndividualFire: Each weapon mount fires under its own trigger 0-3.

bool TurretShapeData::**zRotOnly**
Should the turret allow only z rotations. True indicates that the turret may only be rotated on its z axis, just like the Item class. This keeps the turret always upright regardless of the surface it lands on.

Enumeration

enum ItemLightType
The type of light the Item has.

Parameters

- **NoLight** – The item has no light attached.
- **ConstantLight** – The item has a constantly emitting light attached.
- **PulsingLight** – The item has a pulsing light attached.

enum PlayerPose
The pose of the Player .

Parameters

- **Stand** – Standard movement pose.
- **Sprint** – Sprinting pose.
- **Crouch** – Crouch pose.
- **Prone** – Prone pose.
- **Swim** – Swimming pose.

enum ShapeBaseImageLightType
The type of light to attach to this ShapeBaseImage.

Parameters

- **NoLight** – No light is attached.
- **ConstantLight** – A constant emitting light is attached.
- **SpotLight** – A spotlight is attached.
- **PulsingLight** – A pusling light is attached.
- **WeaponFireLight** – Light emits when the weapon is fired, then dissipates.

enum ShapeBaseImageLoadedState
The loaded state of this ShapeBaseImage.

Parameters

- **Ignore** – Ignore the loaded state.
- **Loaded** – ShapeBaseImage is loaded.
- **Empty** – ShapeBaseImage is not loaded.

enum ShapeBaseImageRecoilState
What kind of recoil this ShapeBaseImage should emit when fired.

Parameters

- **NoRecoil** – No recoil occurs.
- **LightRecoil** – A light recoil occurs.
- **MediumRecoil** – A medium recoil occurs.
- **HeavyRecoil** – A heavy recoil occurs.

enum ShapeBaseImageSpinState
How the spin animation should be played.

Parameters

- **Ignore** – No changes to the spin sequence.
- **Stop** – Stops the spin sequence at its current position.
- **SpinUp** – Increase spin sequence timeScale from 0 (on state entry) to 1 (after stateTimeoutValue seconds).
- **SpinDown** – Decrease spin sequence timeScale from 1 (on state entry) to 0 (after stateTimeoutValue seconds).
- **FullSpeed** – Resume the spin sequence playback at its current position with timeScale = 1.

enum TSMeshType
Type of mesh data available in a shape.

Parameters

- **None** – No mesh data.
- **Bounds** – Bounding box of the shape.
- **Mesh** – Specifically desingated “collision” meshes.
- **Mesh** – Rendered mesh polygons.

enum TurretShapeFireLinkType
How the weapons are linked to triggers for this TurretShape .

Parameters

- **FireTogether** – All weapons fire under trigger 0.
- **GroupedFire** – Weapon mounts 0,2 fire under trigger 0, mounts 1,3 fire under trigger 1.
- **IndividualFire** – Each weapon mount fires under its own trigger 0-3.

Variables

float $SB::CloakSpeed
Time to cloak, in seconds.

float $SB::DFDec
Speed to reduce the damage flash effect per tick.

float $SB::FullCorrectionDistance
Distance at which a weapon's muzzle vector is fully corrected to match where the player is looking. When a weapon image has correctMuzzleVector set and the Player is in 1st person, the muzzle vector from the weapon is modified to match where the player is looking. Beyond the FullCorrectionDistance the muzzle vector is always corrected. Between FullCorrectionDistance and the player, the weapon's muzzle vector is adjusted so that the closer the aim point is to the player, the closer the muzzle vector is to the true (non-corrected) one.

bool Trigger::**renderTriggers**[static, inherited]
Forces all Trigger's to render. Used by the Tools and debug render modes.

float $SB::WODec
Speed to reduce the whiteout effect per tick.

Special Effects

Classes responsible for special effect objects, such as Explosion, Debris, Particles, etc.

Classes

Debris datablock for properties of individual debris objects.

Inherit: GameBase

Description Base debris class. Uses the DebrisData datablock for properties of individual debris objects.

Debris is typically made up of a shape and up to two particle emitters. In most cases Debris objects are not created directly. They are usually produced automatically by other means, such as through the Explosion class. When an explosion goes off, its ExplosionData datablock determines what Debris to emit.

Example:

```
datablock ExplosionData(GrenadeLauncherExplosion)
{
   // Assiging debris data
   debris = GrenadeDebris;

   // Adjust how debris is ejected
   debrisThetaMin = 10;
   debrisThetaMax = 60;
   debrisNum = 4;
   debrisNumVariance = 2;
   debrisVelocity = 25;
   debrisVelocityVariance = 5;

   // Note: other ExplosionData properties are not listed for this example
};
```

Methods

bool Debris::**init**(string *inputPosition*, string *inputVelocity*)

Manually set this piece of debris at the given position with the given velocity. Usually you do not manually create Debris objects as they are generated through other means, such as an Explosion . This method exists when you do manually create a Debris object and want to have it start moving.

Parameters

- **inputPosition** – Position to place the debris.
- **inputVelocity** – Velocity to move the debris after it has been placed.

Returns Always returns true.

Example:

```
// Define the position
%position = "1.0 1.0 1.0";

// Define the velocity
%velocity = "1.0 0.0 0.0";

// Inform the debris object of its new position and velocity
%debris.init(%position,%velocity);
```

Fields

float Debris::**lifetime**

Length of time for this debris object to exist. When expired, the object will be deleted. The initial lifetime value comes from the DebrisData datablock.

DebrisData Stores properties for an individual debris type.

Inherit: GameBaseData

Description Stores properties for an individual debris type.

DebrisData defines the base properties for a Debris object. Typically you'll want a Debris object to consist of a shape and possibly up to two particle emitters. The DebrisData datablock provides the definition for these items, along with physical properties and how a Debris object will react to other game objects, such as water and terrain.

Example:

```
datablock DebrisData(GrenadeDebris)
{
   shapeFile = "art/shapes/weapons/ramrifle/debris.dts";
   emitters[0] = GrenadeDebrisFireEmitter;
   elasticity = 0.4;
   friction = 0.25;
   numBounces = 3;
   bounceVariance = 1;
   explodeOnMaxBounce = false;
   staticOnMaxBounce = false;
   snapOnMaxBounce = false;
   minSpinSpeed = 200;
   maxSpinSpeed = 600;
   lifetime = 4;
   lifetimeVariance = 1.5;
   velocity = 15;
```

```
   velocityVariance = 5;
   fade = true;
   useRadiusMass = true;
   baseRadius = 0.3;
   gravModifier = 1.0;
   terminalVelocity = 20;
   ignoreWater = false;
};
```

Fields

float DebrisData::**baseRadius**
Radius at which the standard elasticity and friction apply. Only used when useRaduisMass is true.

int DebrisData::**bounceVariance**
Allowed variance in the value of numBounces. Must be less than numBounces.

float DebrisData::**elasticity**
A floating-point value specifying how 'bouncy' this object is. Must be in the range of -10 to 10.

ParticleEmitterData DebrisData::**emitters**[2]
List of particle emitters to spawn along with this debris object. These are optional. You could have Debris made up of only a shape.

bool DebrisData::**explodeOnMaxBounce**
If true, this debris object will explode after it has bounced max times. Be sure to provide an ExplosionData datablock for this to take effect.

ExplosionData DebrisData::**Explosion**
ExplosionData to spawn along with this debris object. This is optional as not all Debris explode.

bool DebrisData::**fade**
If true, this debris object will fade out when destroyed. This fade occurs over the last second of the Debris' lifetime.

float DebrisData::**friction**
A floating-point value specifying how much velocity is lost to impact and sliding friction. Must be in the range of -10 to 10.

float DebrisData::**gravModifier**
How much gravity affects debris.

bool DebrisData::**ignoreWater**
If true, this debris object will not collide with water, acting as if the water is not there.

float DebrisData::**lifetime**
Amount of time until this debris object is destroyed. Must be in the range of 0 to 1000.

float DebrisData::**lifetimeVariance**
Allowed variance in the value of lifetime. Must be less than lifetime.

float DebrisData::**maxSpinSpeed**
Maximum speed that this debris object will rotate. Must be in the range of -10000 to 10000.

float DebrisData::**minSpinSpeed**
Minimum speed that this debris object will rotate. Must be in the range of -10000 to 1000, and must be less than maxSpinSpeed.

int DebrisData::**numBounces**
How many times to allow this debris object to bounce until it either explodes, becomes static or snaps (defined in explodeOnMaxBounce, staticOnMaxBounce, snapOnMaxBounce). Must be within the range of 0 to 10000.

filename DebrisData::**shapeFile**
Object model to use for this debris object. This shape is optional. You could have Debris made up of only particles.

bool DebrisData::**snapOnMaxBounce**
If true, this debris object will snap into a resting position on the last bounce.

bool DebrisData::**staticOnMaxBounce**
If true, this debris object becomes static after it has bounced max times.

float DebrisData::**terminalVelocity**
Max velocity magnitude.

string DebrisData::**texture**
Texture imagemap to use for this debris object. Not used any more.

bool DebrisData::**useRadiusMass**
Use mass calculations based on radius. Allows for the adjustment of elasticity and friction based on the Debris size.

float DebrisData::**velocity**
Speed at which this debris object will move.

float DebrisData::**velocityVariance**
Allowed variance in the value of velocity. Must be less than velocity.

DecalData A datablock describing an individual decal.

Inherit: SimDataBlock

Description A datablock describing an individual decal.

The textures defined by the decal Material can be divided into multiple rectangular sub-textures as shown below, with a different sub-texture selected by all decals using the same DecalData (via frame) or each decal instance (via randomize).

Example of a Decal imagemap

Example:

```
datablock DecalData(BulletHoleDecal)
{
   material = "DECAL_BulletHole";
   size = "5.0";
   lifeSpan = "50000";
   randomize = "1";
   texRows = "2";
   texCols = "2";
   clippingAngle = "60";
};
```

Methods

void DecalData::**postApply**()
Recompute the imagemap sub-texture rectangles for this DecalData .

Example:

```
// Inform the decal object to reload its imagemap and frame data.
%decalData.texRows = 4;
%decalData.postApply();
```

Fields

float DecalData::**clippingAngle**
The angle in degrees used to clip geometry that faces away from the decal projection direction.

float DecalData::**fadeEndPixelSize**
LOD value - size in pixels at which decals of this type are fully faded out. This should be a smaller value than fadeStartPixelSize .

float DecalData::**fadeStartPixelSize**
LOD value - size in pixels at which decals of this type begin to fade out. This should be a larger value than fadeEndPixelSize . However, you may also set this to a negative value to disable lod-based fading.

int DecalData::**fadeTime**
Time (in milliseconds) over which to fade out the decal before deleting it at the end of its lifetime.

int DecalData::**frame**
Index of the texture rectangle within the imagemap to use for this decal.

int DecalData::**lifeSpan**
Time (in milliseconds) before this decal will be automatically deleted.

string DecalData::**Material**
Material to use for this decal.

bool DecalData::**randomize**
If true, a random frame from the imagemap is selected for each instance of the decal.

char DecalData::**renderPriority**
Default renderPriority for decals of this type (determines draw order when decals overlap).

float DecalData::**size**
Width and height of the decal in meters before scale is applied.

int DecalData::**texCols**
Number of columns in the supplied imagemap. Use texRows and texCols if the imagemap frames are arranged in a grid; use textureCoords to manually specify UV coordinates for irregular sized frames.

int DecalData::**texRows**
Number of rows in the supplied imagemap. Use texRows and texCols if the imagemap frames are arranged in a grid; use textureCoords to manually specify UV coordinates for irregular sized frames.

int DecalData::**textureCoordCount**
Number of individual frames in the imagemap (maximum 16).

RectF DecalData::**textureCoords**[16]
An array of RectFs (topleft.x topleft.y extent.x extent.y) representing the UV coordinates for each frame in the imagemap.

DecalManager The object that manages all of the decals in the active mission.

Inherit: SceneObject

Description The object that manages all of the decals in the active mission.

Explosion object.

Inherit: GameBase

Description The emitter for an explosion effect, with properties defined by a ExplosionData object.

The object will initiate the explosion effects automatically after being added to the simulation.

Example:

```
datablock ExplosionData( GrenadeSubExplosion )
{
   offset = 0.25;
   emitter[0] = GrenadeExpSparkEmitter;

   lightStartRadius = 4.0;
   lightEndRadius = 0.0;
   lightStartColor = "0.9 0.7 0.7";
   lightEndColor = "0.9 0.7 0.7";
   lightStartBrightness = 2.0;
   lightEndBrightness = 0.0;
};

datablock ExplosionData( GrenadeLauncherExplosion )
{
   soundProfile = GrenadeLauncherExplosionSound;
   lifeTimeMS = 400; // Quick flash, short burn, and moderate dispersal// Volume particles
   particleEmitter = GrenadeExpFireEmitter;
   particleDensity = 75;
   particleRadius = 2.25;

   // Point emission
   emitter[0] = GrenadeExpDustEmitter;
   emitter[1] = GrenadeExpSparksEmitter;
   emitter[2] = GrenadeExpSmokeEmitter;

   // Sub explosion objects
   subExplosion[0] = GrenadeSubExplosion;

   // Camera Shaking
   shakeCamera = true;
   camShakeFreq = "10.0 11.0 9.0";
   camShakeAmp = "15.0 15.0 15.0";
   camShakeDuration = 1.5;
   camShakeRadius = 20;

   // Exploding debris
   debris = GrenadeDebris;
   debrisThetaMin = 10;
   debrisThetaMax = 60;
   debrisNum = 4;
   debrisNumVariance = 2;
   debrisVelocity = 25;
   debrisVelocityVariance = 5;

   lightStartRadius = 4.0;
   lightEndRadius = 0.0;
   lightStartColor = "1.0 1.0 1.0";
   lightEndColor = "1.0 1.0 1.0";
```

```
   lightStartBrightness = 4.0;
   lightEndBrightness = 0.0;
   lightNormalOffset = 2.0;
};

function createExplosion()
{
   // Create a new explosion - it will explode automatically
   %pos = "0 0 100";
   %obj = newExplosion()
   {
      position = %pos;
      dataBlock = GrenadeLauncherExplosion;
   };
}

// schedule an explosionschedule(1000, 0, createExplosion);
```

ExplosionData : particleEmitters, debris, lighting and camera shake effects.

Inherit: GameBaseData

Description Defines the attributes of an Explosion: particleEmitters, debris, lighting and camera shake effects.

Fields

Point3F `ExplosionData::`**`camShakeAmp`**
: Amplitude of camera shaking, defined in the "X Y Z" axes. Set any value to 0 to disable shaking in that axis.

float `ExplosionData::`**`camShakeDuration`**
: Duration (in seconds) to shake the camera.

float `ExplosionData::`**`camShakeFalloff`**
: Falloff value for the camera shake.

Point3F `ExplosionData::`**`camShakeFreq`**
: Frequency of camera shaking, defined in the "X Y Z" axes.

float `ExplosionData::`**`camShakeRadius`**
: Radial distance that a camera's position must be within relative to the center of the explosion to be shaken.

DebrisData `ExplosionData::`**`Debris`**
: List of DebrisData objects to spawn with this explosion.

int `ExplosionData::`**`debrisNum`**
: Number of debris objects to create.

int `ExplosionData::`**`debrisNumVariance`**
: Variance in the number of debris objects to create (must be from 0 - debrisNum).

float `ExplosionData::`**`debrisPhiMax`**
: Maximum reference angle, from the vertical plane, to eject debris from.

float `ExplosionData::`**`debrisPhiMin`**
: Minimum reference angle, from the vertical plane, to eject debris from.

float `ExplosionData::`**`debrisThetaMax`**
: Maximum angle, from the horizontal plane, to eject debris from.

float ExplosionData::**debrisThetaMin**
Minimum angle, from the horizontal plane, to eject debris from.

float ExplosionData::**debrisVelocity**
Velocity to toss debris at.

float ExplosionData::**debrisVelocityVariance**
Variance in the debris initial velocity (must be gt = 0).

int ExplosionData::**delayMS**
Amount of time, in milliseconds, to delay the start of the explosion effect from the creation of the Explosion object.

int ExplosionData::**delayVariance**
Variance, in milliseconds, of delayMS.

ParticleEmitterData ExplosionData::**emitter**[4]
List of additional ParticleEmitterData objects to spawn with this explosion.

Point3F ExplosionData::**explosionScale**
"X Y Z" scale factor applied to the explosionShape model at the start of the explosion.

filename ExplosionData::**explosionShape**
Optional DTS or DAE shape to place at the center of the explosion. The ambient animation of this model will be played automatically at the start of the explosion.

bool ExplosionData::**faceViewer**
Controls whether the visual effects of the explosion always face the camera.

int ExplosionData::**lifetimeMS**
Lifetime, in milliseconds, of the Explosion object.

int ExplosionData::**lifetimeVariance**
Variance, in milliseconds, of the lifetimeMS of the Explosion object.

float ExplosionData::**lightEndBrightness**
Final brightness of the PointLight created by this explosion.

ColorF ExplosionData::**lightEndColor**
Final color of the PointLight created by this explosion.

float ExplosionData::**lightEndRadius**
Final radius of the PointLight created by this explosion.

float ExplosionData::**lightNormalOffset**
Distance (in the explosion normal direction) of the PointLight position from the explosion center.

float ExplosionData::**lightStartBrightness**
Initial brightness of the PointLight created by this explosion. Brightness is linearly interpolated from lightStartBrightness to lightEndBrightness over the lifetime of the explosion.

ColorF ExplosionData::**lightStartColor**
Initial color of the PointLight created by this explosion. Color is linearly interpolated from lightStartColor to lightEndColor over the lifetime of the explosion.

float ExplosionData::**lightStartRadius**
Initial radius of the PointLight created by this explosion. Radius is linearly interpolated from lightStartRadius to lightEndRadius over the lifetime of the explosion.

float ExplosionData::**offset**
Offset distance (in a random direction) of the center of the explosion from the Explosion object position. Most often used to create some variance in position for subExplosion effects.

int ExplosionData::**particleDensity**
Density of the particle cloud created at the start of the explosion.

ParticleEmitterData ExplosionData::**ParticleEmitter**
Emitter used to generate a cloud of particles at the start of the explosion. Explosions can generate two different particle effects. The first is a single burst of particles at the start of the explosion emitted in a spherical cloud using particleEmitter. The second effect spawns the list of ParticleEmitters given by the emitter[] field. These emitters generate particles in the normal way throughout the lifetime of the explosion.

float ExplosionData::**particleRadius**
Radial distance from the explosion center at which cloud particles are emitted.

float ExplosionData::**playSpeed**
Time scale at which to play the explosionShape ambient sequence.

bool ExplosionData::**shakeCamera**
Controls whether the camera shakes during this explosion.

Point3F ExplosionData::**sizes**[4]
"X Y Z" size keyframes used to scale the explosionShape model. The explosionShape (if defined) will be scaled using the times/sizes keyframes over the lifetime of the explosion.

SFXTrack ExplosionData::**soundProfile**
Non-looping sound effect that will be played at the start of the explosion.

ExplosionData ExplosionData::**subExplosion**[5]
List of additional ExplosionData objects to create at the start of the explosion.

float ExplosionData::**times**[4]
Time keyframes used to scale the explosionShape model. Values should be in increasing order from 0.0 - 1.0, and correspond to the life of the Explosion where 0 is the beginning and 1 is the end of the explosion lifetime.

ForestWindEmitter Object responsible for simulating wind in a level.

Inherit: SceneObject

Description When placed in the level, a ForestWindEmitter will cause tree branches to bend and sway, leaves to flutter, and create vertical bending on the tree's trunk.

Example:

```
// The following is a full declaration of a wind emitternewForestWindEmitter()
{
   position = "497.739 765.821 102.395";
   windEnabled = "1";
   radialEmitter = "1";
   strength = "1";
   radius = "3";
   gustStrength = "0.5";
   gustFrequency = "1";
   gustYawAngle = "10";
   gustYawFrequency = "4";
   gustWobbleStrength = "2";
   turbulenceStrength = "1";
   turbulenceFrequency = "2";
   hasMount = "0";
   scale = "3 3 3";
   canSave = "1";
   canSaveDynamicFields = "1";
```

```
   rotation = "1 0 0 0";
};
```

Methods

void ForestWindEmitter::**attachToObject** (int *objectID*)

Mounts the wind emitter to another scene object.

Parameters **objectID** – Unique ID of the object wind emitter should attach to

Example:

```
// Wind emitter previously created and named %windEmitter// Going to attach it to the player, ma
%windEmitter.attachToObject(%player);
```

Fields

float ForestWindEmitter::**gustFrequency**

The frequency of gusting in seconds.

float ForestWindEmitter::**gustStrength**

The maximum strength of a gust.

float ForestWindEmitter::**gustWobbleStrength**

The amount of random wobble added to gust and turbulence vectors.

float ForestWindEmitter::**gustYawAngle**

The amount of degrees the wind direction can drift (both positive and negative).

float ForestWindEmitter::**gustYawFrequency**

The frequency of wind yaw drift, in seconds.

bool ForestWindEmitter::**hasMount**

Determines if the emitter is mounted to another object.

bool ForestWindEmitter::**radialEmitter**

Determines if the emitter is a global direction or local radial emitter.

float ForestWindEmitter::**radius**

The radius of the emitter for local radial emitters.

float ForestWindEmitter::**strength**

The strength of the wind force.

float ForestWindEmitter::**turbulenceFrequency**

The frequency of gust turbulence, in seconds.

float ForestWindEmitter::**turbulenceStrength**

The strength of gust turbulence.

bool ForestWindEmitter::**windEnabled**

Determines if the emitter will be counted in wind calculations.

LightAnimData A datablock which defines and performs light animation, such as rotation, brightness fade, and colorization.

Inherit: SimDataBlock

Description A datablock which defines and performs light animation, such as rotation, brightness fade, and colorization.

Example:

```
datablock LightAnimData( SubtlePulseLightAnim )
{
   brightnessA = 0.5;
   brightnessZ = 1;
   brightnessPeriod = 1;
   brightnessKeys = "aza";
   brightnessSmooth = true;
};
```

Fields

float LightAnimData::**brightnessA**
The value of the A key in the keyframe sequence.

string LightAnimData::**brightnessKeys**
The keyframe sequence encoded into a string where characters from A to Z define a position between the two animation values.

float LightAnimData::**brightnessPeriod**
The animation time for keyframe sequence.

bool LightAnimData::**brightnessSmooth**
If true the transition between keyframes will be smooth.

float LightAnimData::**brightnessZ**
The value of the Z key in the keyframe sequence.

float LightAnimData::**colorA**[3]
The value of the A key in the keyframe sequence.

string LightAnimData::**colorKeys**[3]
The keyframe sequence encoded into a string where characters from A to Z define a position between the two animation values.

float LightAnimData::**colorPeriod**[3]
The animation time for keyframe sequence.

bool LightAnimData::**colorSmooth**[3]
If true the transition between keyframes will be smooth.

float LightAnimData::**colorZ**[3]
The value of the Z key in the keyframe sequence.

float LightAnimData::**offsetA**[3]
The value of the A key in the keyframe sequence.

string LightAnimData::**offsetKeys**[3]
The keyframe sequence encoded into a string where characters from A to Z define a position between the two animation values.

float LightAnimData::**offsetPeriod**[3]
The animation time for keyframe sequence.

bool LightAnimData::**offsetSmooth**[3]
If true the transition between keyframes will be smooth.

float LightAnimData::**OffsetZ**[3]
The value of the Z key in the keyframe sequence.

float LightAnimData::**rotA**[3]
The value of the A key in the keyframe sequence.

string LightAnimData::**rotKeys**[3]
The keyframe sequence encoded into a string where characters from A to Z define a position between the two animation values.

float LightAnimData::**rotPeriod**[3]
The animation time for keyframe sequence.

bool LightAnimData::**rotSmooth**[3]
If true the transition between keyframes will be smooth.

float LightAnimData::**rotZ**[3]
The value of the Z key in the keyframe sequence.

Lightning An emitter for lightning bolts.

Inherit: GameBase

Description An emitter for lightning bolts.

Lightning strike events are created on the server and transmitted to all clients to render the bolt. The strike may be followed by a random thunder sound. Player or Vehicle objects within the Lightning strike range can be hit and damaged by bolts.

Methods

void Lightning::**applyDamage**(Point3F *hitPosition*, Point3F *hitNormal*, SceneObject *hitObject*)
Informs an object that it was hit by a lightning bolt and needs to take damage.

Parameters

- **hitPosition** – World position hit by the lightning bolt.
- **hitNormal** – Surface normal at hitPosition.
- **hitObject** – Player or Vehicle object that was hit.

Example:

```
function Lightning::applyDamage( %this, %hitPosition, %hitNormal, %hitObject )
{
   // apply damage to the player
   %hitObject.applyDamage( 25 );
}
```

void Lightning::**strikeObject**(int *id*)
Creates a LightningStrikeEvent which strikes a specific object.

void Lightning::**strikeRandomPoint**()
Creates a LightningStrikeEvent which attempts to strike and damage a random object in range of the Lightning object.

Example:

```
// Generate a damaging lightning strike effect on all clients
%lightning.strikeRandomPoint();
```

void Lightning::**warningFlashes**()
Creates a LightningStrikeEvent that triggers harmless lightning bolts on all clients. No objects will be damaged by these bolts.

Example:

```
// Generate a harmless lightning strike effect on all clients
%lightning.warningFlashes();
```

Fields

float Lightning::**boltStartRadius**
Radial distance from the center of the Lightning object for the start point of the bolt. The actual start point will be a random point within this radius.

float Lightning::**chanceToHitTarget**
Percentage chance (0-1) that a given lightning bolt will hit something.

ColorF Lightning::**color**
Color to blend the strike texture with.

ColorF Lightning::**fadeColor**
Color to blend the strike texture with when the bolt is fading away. Bolts fade away automatically shortly after the strike occurs.

float Lightning::**strikeRadius**
Horizontal size (XY plane) of the search box used to find and damage Player or Vehicle objects within range of the strike. Only the object at highest altitude with a clear line of sight to the bolt will be hit.

int Lightning::**strikesPerMinute**
Number of lightning strikes to perform per minute. Automatically invokes strikeRandomPoint() at regular intervals.

float Lightning::**strikeWidth**
Width of a lightning bolt.

bool Lightning::**useFog**
Controls whether lightning bolts are affected by fog when they are rendered.

LightningData emitter object.

Inherit: GameBaseData

Description Common data for a Lightning emitter object.

Fields

SFXTrack LightningData::**strikeSound**
Sound profile to play when a lightning strike occurs.

string LightningData::**strikeTextures**[8]
List of textures to use to render lightning strikes.

SFXTrack LightningData::**thunderSounds**[8]
List of thunder sound effects to play. A random one of these sounds will be played shortly after each strike occurs.

LightningStrikeEvent Network event that triggers a lightning strike on the client when it is received. Description

---

Network event that triggers a lightning strike on the client when it is received.

This event is sent to all clients when the warningFlashes(), strikeRandomPoint() or strikeObject() methods are invoked on the Lightning object on the server.

ParticleData Contains information for how specific particles should look and react including particle colors, particle imagemap, acceleration value for individual particles and spin information.

Inherit: SimDataBlock

Description Contains information for how specific particles should look and react including particle colors, particle imagemap, acceleration value for individual particles and spin information.

Example:

```
datablock ParticleData( GLWaterExpSmoke )
{
   textureName = "art/shapes/particles/smoke";
   dragCoefficient = 0.4;
   gravityCoefficient = -0.25;
   inheritedVelFactor = 0.025;
   constantAcceleration = -1.1;
   lifetimeMS = 1250;
   lifetimeVarianceMS = 0;
   useInvAlpha = false;
   spinSpeed = 1;
   spinRandomMin = -200.0;
   spinRandomMax = 200.0;

   colors[0] = "0.1 0.1 1.0 1.0";
   colors[1] = "0.4 0.4 1.0 1.0";
   colors[2] = "0.4 0.4 1.0 0.0";

   sizes[0] = 2.0;
   sizes[1] = 6.0;
   sizes[2] = 2.0;

   times[0] = 0.0;
   times[1] = 0.5;
   times[2] = 1.0;
};
```

Methods

void `ParticleData::reload`()

Reloads this particle.

Example:

```
// Get the editors current particle
%particle = PE_ParticleEditor.currParticle

// Change a particle value
%particle.setFieldValue( %propertyField, %value );

// Reload it
%particle.reload();
```

Fields

bool `ParticleData::animateTexture`

If true, allow the particle texture to be an animated sprite.

string ParticleData::**animTexFrames**
A list of frames and/or frame ranges to use for particle animation if animateTexture is true. Each frame token must be separated by whitespace. A frame token must be a positive integer frame number or a range of frame numbers separated with a '-'. The range separator, '-', cannot have any whitspace around it. Ranges can be specified to move through the frames in reverse as well as forward (eg. 19-14). Frame numbers exceeding the number of tiles will wrap.

Example:

```
animTexFrames = "0-16 20 19 18 17 31-21";
```

string ParticleData::**animTexName**
Texture file to use for this particle if animateTexture is true. Deprecated. Use textureName instead.

Point2I ParticleData::**animTexTiling**
The number of frames, in rows and columns stored in textureName (when animateTexture is true). A maximum of 256 frames can be stored in a single texture when using animTexTiling. Value should be "NumColumns NumRows", for example "4 4".

ColorF ParticleData::**colors**[4]
Particle RGBA color keyframe values. The particle color will linearly interpolate between the color/time keys over the lifetime of the particle.

float ParticleData::**constantAcceleration**
Constant acceleration to apply to this particle.

float ParticleData::**dragCoefficient**
Particle physics drag amount.

int ParticleData::**framesPerSec**
If animateTexture is true, this defines the frames per second of the sprite animation.

float ParticleData::**gravityCoefficient**
Strength of gravity on the particles.

float ParticleData::**inheritedVelFactor**
Amount of emitter velocity to add to particle initial velocity.

int ParticleData::**lifetimeMS**
Time in milliseconds before this particle is destroyed.

int ParticleData::**lifetimeVarianceMS**
Variance in lifetime of particle, from 0 - lifetimeMS.

float ParticleData::**sizes**[4]
Particle size keyframe values. The particle size will linearly interpolate between the size/time keys over the lifetime of the particle.

float ParticleData::**spinRandomMax**
Maximum allowed spin speed of this particle, between spinRandomMin and 1000.

float ParticleData::**spinRandomMin**
Minimum allowed spin speed of this particle, between -1000 and spinRandomMax.

float ParticleData::**spinSpeed**
Speed at which to spin the particle.

Point2F ParticleData::**textureCoords**[4]
4 element array defining the UV coords into textureName to use for this particle. Coords should be set for the first tile only when using animTexTiling; coordinates for other tiles will be calculated automatically. "0 0" is top left and "1 1" is bottom right.

string `ParticleData::textureName`
Texture file to use for this particle.

float `ParticleData::times`[4]
Time keys used with the colors and sizes keyframes. Values are from 0.0 (particle creation) to 1.0 (end of lifespace).

bool `ParticleData::useInvAlpha`
Controls how particles blend with the scene. If true, particles blend like ParticleBlendStyle NORMAL, if false, blend like ParticleBlendStyle ADDITIVE.

float `ParticleData::windCoefficient`
Strength of wind on the particles.

ParticleEmitter This object is responsible for spawning particles.

Inherit: GameBase

Description This object is responsible for spawning particles.

This class is the main interface for creating particles - though it is usually only accessed from within another object like ParticleEmitterNode or WheeledVehicle. If using this object class (via C++) directly, be aware that it does not track changes in source axis or velocity over the course of a single update, so emitParticles should be called at a fairly fine grain. The emitter will potentially track the last particle to be created into the next call to this function in order to create a uniformly random time distribution of the particles.

If the object to which the emitter is attached is in motion, it should try to ensure that for call (n+1) to this function, start is equal to the end from call (n). This will ensure a uniform spatial distribution.

ParticleEmitterData .

Inherit: GameBaseData

Description Defines particle emission properties such as ejection angle, period and velocity for a ParticleEmitter.

Example:

```
datablock ParticleEmitterData( GrenadeExpDustEmitter )
{
   ejectionPeriodMS = 1;
   periodVarianceMS = 0;
   ejectionVelocity = 15;
   velocityVariance = 0.0;
   ejectionOffset = 0.0;
   thetaMin = 85;
   thetaMax = 85;
   phiReferenceVel = 0;
   phiVariance = 360;
   overrideAdvance = false;
   lifetimeMS = 200;
   particles = "GrenadeExpDust";
};
```

Methods

void ParticleEmitterData::**reload**()
Reloads the ParticleData datablocks and other fields used by this emitter.

Example:

```
// Get the editors current particle emitter
%emitter = PE_EmitterEditor.currEmitter

// Change a field value
%emitter.setFieldValue( %propertyField, %value );

// Reload this emitter
%emitter.reload();
```

Fields

Point3F ParticleEmitterData::**alignDirection**
The direction aligned particles should face, only valid if alignParticles is true.

bool ParticleEmitterData::**alignParticles**
If true, particles always face along the axis defined by alignDirection.

float ParticleEmitterData::**ambientFactor**
Used to generate the final particle color by controlling interpolation between the particle color and the particle color multiplied by the ambient light color.

ParticleBlendStyle ParticleEmitterData::**blendStyle**
String value that controls how emitted particles blend with the scene.

float ParticleEmitterData::**ejectionOffset**
Distance along ejection Z axis from which to eject particles.

float ParticleEmitterData::**ejectionOffsetVariance**
Distance Padding along ejection Z axis from which to eject particles.

int ParticleEmitterData::**ejectionPeriodMS**
Time (in milliseconds) between each particle ejection.

float ParticleEmitterData::**ejectionVelocity**
Particle ejection velocity.

bool ParticleEmitterData::**highResOnly**
This particle system should not use the mixed-resolution renderer. If your particle system has large amounts of overdraw, consider disabling this option.

int ParticleEmitterData::**lifetimeMS**
Lifetime of emitted particles (in milliseconds).

int ParticleEmitterData::**lifetimeVarianceMS**
Variance in particle lifetime from 0 - lifetimeMS.

bool ParticleEmitterData::**orientOnVelocity**
If true, particles will be oriented to face in the direction they are moving.

bool ParticleEmitterData::**orientParticles**
If true, Particles will always face the camera.

bool ParticleEmitterData::**overrideAdvance**
If false, particles emitted in the same frame have their positions adjusted. If true, adjustment is skipped and particles will clump together.

string ParticleEmitterData::**particles**
List of space or TAB delimited ParticleData datablock names. A random one of these datablocks is selected each time a particle is emitted.

int ParticleEmitterData::**periodVarianceMS**
Variance in ejection period, from 1 - ejectionPeriodMS.

float ParticleEmitterData::**phiReferenceVel**
Reference angle, from the vertical plane, to eject particles from.

float ParticleEmitterData::**phiVariance**
Variance from the reference angle, from 0 - 360.

bool ParticleEmitterData::**renderReflection**
Controls whether particles are rendered onto reflective surfaces like water.

bool ParticleEmitterData::**reverseOrder**
If true, reverses the normal draw order of particles. Particles are normally drawn from newest to oldest, or in Z order (furthest first) if sortParticles is true. Setting this field to true will reverse that order: oldest first, or nearest first if sortParticles is true.

float ParticleEmitterData::**softnessDistance**
For soft particles, the distance (in meters) where particles will be faded based on the difference in depth between the particle and the scene geometry.

bool ParticleEmitterData::**sortParticles**
If true, particles are sorted furthest to nearest.

string ParticleEmitterData::**textureName**
Optional texture to override ParticleData::textureName .

float ParticleEmitterData::**thetaMax**
Maximum angle, from the horizontal plane, to eject particles from.

float ParticleEmitterData::**thetaMin**
Minimum angle, from the horizontal plane, to eject from.

bool ParticleEmitterData::**useEmitterColors**
If true, use emitter specified colors instead of datablock colors. Useful for ShapeBase dust and WheeledVehicle wheel particle emitters that use the current material to control particle color.

bool ParticleEmitterData::**useEmitterSizes**
If true, use emitter specified sizes instead of datablock sizes. Useful for Debris particle emitters that control the particle size.

float ParticleEmitterData::**velocityVariance**
Variance for ejection velocity, from 0 - ejectionVelocity.

ParticleEmitterNode A particle emitter object that can be positioned in the world and dynamically enabled or disabled.

Inherit: GameBase

Description A particle emitter object that can be positioned in the world and dynamically enabled or disabled.

Example:

```
datablock ParticleEmitterNodeData( SimpleEmitterNodeData )
{
   timeMultiple = 1.0;
};
```

```
%emitter = newParticleEmitterNode()
{
   datablock = SimpleEmitterNodeData;
   active = true;
   emitter = FireEmitterData;
   velocity = 3.5;
};

// Dynamically change emitter datablock
%emitter.setEmitterDataBlock( DustEmitterData );
```

Methods

void ParticleEmitterNode::**setActive**(bool *active*)

Turns the emitter on or off.

Parameters **active** – New emitter state

void ParticleEmitterNode::**setEmitterDataBlock**(ParticleEmitterData *emitterDatablock*)

Assigns the datablock for this emitter node.

Parameters **emitterDatablock** – ParticleEmitterData datablock to assign

Example:

```
// Assign a new emitter datablock
%emitter.setEmitterDatablock( %emitterDatablock );
```

Fields

bool ParticleEmitterNode::**active**

Controls whether particles are emitted from this node.

ParticleEmitterData ParticleEmitterNode::**emitter**

Datablock to use when emitting particles.

float ParticleEmitterNode::**velocity**

Velocity to use when emitting particles (in the direction of the ParticleEmitterNode object's up (Z) axis).

ParticleEmitterNodeData .

Inherit: GameBaseData

Description Contains additional data to be associated with a ParticleEmitterNode.

Fields

float ParticleEmitterNodeData::**timeMultiple**

Time multiplier for particle emitter nodes. Increasing timeMultiple is like running the emitter at a faster rate - single-shot emitters will complete in a shorter time, and continuous emitters will generate particles more quickly. Valid range is 0.01 - 100.

Precipitation Defines a precipitation based storm (rain, snow, etc).

Inherit: GameBase

Description Defines a precipitation based storm (rain, snow, etc).

The Precipitation effect works by creating many 'drops' within a fixed size box. This box can be configured to move around with the camera (to simulate level-wide precipitation), or to remain in a fixed position (to simulate localized precipitation). When followCam is true, the box containing the droplets can be thought of as centered on the camera then pushed slightly forward in the direction the camera is facing so most of the box is in front of the camera (allowing more drops to be visible on screen at once).

The effect can also be configured to create a small 'splash' whenever a drop hits another world object.

Example:

```
// The following is added to a level file (.mis) by the World EditornewPrecipitation( TheRain )
{
   dropSize = "0.5";
   splashSize = "0.5";
   splashMS = "250";
   animateSplashes = "1";
   dropAnimateMS = "0";
   fadeDist = "0";
   fadeDistEnd = "0";
   useTrueBillboards = "0";
   useLighting = "0";
   glowIntensity = "0 0 0 0";
   reflect = "0";
   rotateWithCamVel = "1";
   doCollision = "1";
   hitPlayers = "0";
   hitVehicles = "0";
   followCam = "1";
   useWind = "0";
   minSpeed = "1.5";
   maxSpeed = "2";
   minMass = "0.75";
   maxMass = "0.85";
   useTurbulence = "0";
   maxTurbulence = "0.1";
   turbulenceSpeed = "0.2";
   numDrops = "1024";
   boxWidth = "200";
   boxHeight = "100";
   dataBlock = "HeavyRain";
};
```

Methods

void `Precipitation::`**`modifyStorm`**(float *percentage*, float *seconds*)

Smoothly change the maximum number of drops in the effect (from current value to numDrops * percentage). This method can be used to simulate a storm building or fading in intensity as the number of drops in the Precipitation box changes.

Parameters

- **`percentage`** – New maximum number of drops value (as a percentage of numDrops). Valid range is 0-1.
- **`seconds`** – Length of time (in seconds) over which to increase the drops percentage value. Set to 0 to change instantly.

Example:

```
%percentage = 0.5;  // The percentage, from 0 to 1, of the maximum drops to display
%seconds = 5.0;     // The length of time over which to make the change.
%precipitation.modifyStorm( %percentage, %seconds );
```

void Precipitation::**setPercentage**(float *percentage*)
Sets the maximum number of drops in the effect, as a percentage of numDrops . The change occurs instantly (use modifyStorm() to change the number of drops over a period of time.

Parameters **percentage** – New maximum number of drops value (as a percentage of numDrops). Valid range is 0-1.

Example:

```
%percentage = 0.5;  // The percentage, from 0 to 1, of the maximum drops to display
%precipitation.setPercentage( %percentage );
```

void Precipitation::**setTurbulence**(float *max*, float *speed*, float *seconds*)
Smoothly change the turbulence parameters over a period of time.

Parameters

- **max** – New maxTurbulence value. Set to 0 to disable turbulence.
- **speed** – New turbulenceSpeed value.
- **seconds** – Length of time (in seconds) over which to interpolate the turbulence settings. Set to 0 to change instantly.

Example:

```
%turbulence = 0.5;     // Set the new turbulence value. Set to 0 to disable turbulence.
%speed = 5.0;          // The new speed of the turbulance effect.
%seconds = 5.0;        // The length of time over which to make the change.
%precipitation.setTurbulence( %turbulence, %speed, %seconds );
```

Fields

bool Precipitation::**animateSplashes**
Set to true to enable splash animations when drops collide with other surfaces.

float Precipitation::**boxHeight**
Height (vertical dimension) of the precipitation box.

float Precipitation::**boxWidth**
Width and depth (horizontal dimensions) of the precipitation box.

bool Precipitation::**doCollision**
Allow drops to collide with world objects. If animateSplashes is true, drops that collide with another object will produce a simple splash animation.

int Precipitation::**dropAnimateMS**
Length (in milliseconds) to display each drop frame. If dropAnimateMS lt = 0, drops select a single random frame at creation that does not change throughout the drop's lifetime. If dropAnimateMS gt 0, each drop cycles through the the available frames in the drop texture at the given rate.

float Precipitation::**dropSize**
Size of each drop of precipitation. This will scale the texture.

float Precipitation::**fadeDist**
The distance at which drops begin to fade out.

float Precipitation::**fadeDistEnd**
The distance at which drops are completely faded out.

bool Precipitation::**followCam**
Controls whether the Precipitation system follows the camera or remains where it is first placed in the scene. Set to true to make it seem like it is raining everywhere in the level (ie. the Player will always be in the rain). Set to false to have a single area affected by rain (ie. the Player can move in and out of the rainy area).

ColorF Precipitation::**glowIntensity**
Set to 0 to disable the glow or or use it to control the intensity of each channel.

bool Precipitation::**hitPlayers**
Allow drops to collide with Player objects; only valid if doCollision is true.

bool Precipitation::**hitVehicles**
Allow drops to collide with Vehicle objects; only valid if doCollision is true.

float Precipitation::**maxMass**
Maximum mass of a drop. Drop mass determines how strongly the drop is affected by wind and turbulence. On creation, the drop will be assigned a random speed between minMass and minMass .

float Precipitation::**maxSpeed**
Maximum speed at which a drop will fall. On creation, the drop will be assigned a random speed between minSpeed and maxSpeed .

float Precipitation::**maxTurbulence**
Radius at which precipitation drops spiral when turbulence is enabled.

float Precipitation::**minMass**
Minimum mass of a drop. Drop mass determines how strongly the drop is affected by wind and turbulence. On creation, the drop will be assigned a random speed between minMass and minMass .

float Precipitation::**minSpeed**
Minimum speed at which a drop will fall. On creation, the drop will be assigned a random speed between minSpeed and maxSpeed .

int Precipitation::**numDrops**
Maximum number of drops allowed to exist in the precipitation box at any one time. The actual number of drops in the effect depends on the current percentage, which can change over time using modifyStorm() .

bool Precipitation::**reflect**
This enables precipitation rendering during reflection passes.

bool Precipitation::**rotateWithCamVel**
Set to true to include the camera velocity when calculating drop rotation speed.

int Precipitation::**splashMS**
Lifetime of splashes in milliseconds.

float Precipitation::**splashSize**
Size of each splash animation when a drop collides with another surface.

float Precipitation::**turbulenceSpeed**
Speed at which precipitation drops spiral when turbulence is enabled.

bool Precipitation::**useLighting**
Set to true to enable shading of the drops and splashes by the sun color.

bool Precipitation::**useTrueBillboards**
Set to true to make drops true (non axis-aligned) billboards.

bool Precipitation::**useTurbulence**
Check to enable turbulence. This causes precipitation drops to spiral while falling.

bool Precipitation::**useWind**
Controls whether drops are affected by wind.

PrecipitationData Defines the droplets used in a storm (raindrops, snowflakes, etc).

Inherit: GameBaseData

Description Defines the droplets used in a storm (raindrops, snowflakes, etc).

Example:

```
datablock PrecipitationData( HeavyRain )
{
   soundProfile = "HeavyRainSound";
   dropTexture = "art/environment/precipitation/rain";
   splashTexture = "art/environment/precipitation/water_splash";
   dropsPerSide = 4;
   splashesPerSide = 2;
};
```

Fields

string `PrecipitationData::`**`dropShader`**
The name of the shader used for raindrops.

int `PrecipitationData::`**`dropsPerSide`**
How many rows and columns are in the raindrop texture. For example, if the texture has 16 raindrops arranged in a grid, this field should be set to 4.

filename `PrecipitationData::`**`dropTexture`**
Texture filename for drop particles. The drop texture can contain several different drop sub-textures arranged in a grid. There must be the same number of rows as columns. A random frame will be chosen for each drop.

SFXTrack `PrecipitationData::`**`soundProfile`**
Looping SFXProfile effect to play while Precipitation is active.

int `PrecipitationData::`**`splashesPerSide`**
How many rows and columns are in the splash texture. For example, if the texture has 9 splashes arranged in a grid, this field should be set to 3.

string `PrecipitationData::`**`splashShader`**
The name of the shader used for splashes.

filename `PrecipitationData::`**`splashTexture`**
Texture filename for splash particles. The splash texture can contain several different splash sub-textures arranged in a grid. There must be the same number of rows as columns. A random frame will be chosen for each splash.

Splash effect.

Inherit: GameBase

Description Manages the ring used for a Splash effect.

SplashData is created from.

Inherit: GameBaseData

Description Acts as the physical point in space in white a Splash is created from.

Fields

float SplashData::**acceleration**
Constant acceleration value to place upon the splash effect.

ColorF SplashData::**colors**[4]
Color values to set the splash effect, rgba. Up to 4 allowed. Will transition through colors based on values set in the times value. Example: colors[0] = "0.6 1.0 1.0 0.5".

int SplashData::**delayMS**
Time to delay, in milliseconds, before actually starting this effect.

int SplashData::**delayVariance**
Time variance for delayMS.

float SplashData::**ejectionAngle**
Rotational angle to create a splash ring.

float SplashData::**ejectionFreq**
Frequency in which to emit splash rings.

ParticleEmitterData SplashData::**emitter**[3]
List of particle emitters to create at the point of this Splash effect.

ExplosionData SplashData::**Explosion**
ExplosionData object to create at the creation position of this splash effect.

float SplashData::**height**
Height for the splash to reach.

int SplashData::**lifetimeMS**
Lifetime for this effect, in milliseconds.

int SplashData::**lifetimeVariance**
Time variance for lifetimeMS.

int SplashData::**numSegments**
Number of ejection points in the splash ring.

float SplashData::**ringLifetime**
Lifetime, in milliseconds, for a splash ring.

Point3F SplashData::**scale**
The scale of this splashing effect, defined as the F32 points X, Y, Z.

SFXProfile SplashData::**soundProfile**
SFXProfile effect to play.

float SplashData::**startRadius**
Starting radius size of a splash ring.

float SplashData::**texFactor**
Factor in which to apply the texture to the splash ring, 0.0f - 1.0f.

filename SplashData::**texture**[2]
Imagemap file to use as the texture for the splash effect.

float SplashData::**texWrap**
Amount to wrap the texture around the splash ring, 0.0f - 1.0f.

float SplashData::**times**[4]
Times to transition through the splash effect. Up to 4 allowed. Values are 0.0 - 1.0, and corrispond to the life of the particle where 0 is first created and 1 is end of lifespace.

float SplashData::**velocity**
Velocity for the splash effect to travel.

float SplashData::**width**
Width for the X and Y coordinates to create this effect within.

Enumeration

enum ParticleBlendStyle
The type of visual blending style to apply to the particles.

Parameters

- **NORMAL** – No blending style.
- **ADDITIVE** – Adds the color of the pixel to the frame buffer with full alpha for each pixel.
- **SUBTRACTIVE** – Subtractive Blending. Reverses the color model, causing dark colors to have a stronger visual effect.
- **PREMULTALPHA** – Color blends with the colors of the imagemap rather than the alpha.

Functions

float **calcExplosionCoverage**(Point3F *pos*, int *id*, int *covMask*)
Calculates how much an explosion effects a specific object. Use this to determine how much damage to apply to objects based on their distance from the explosion's center point, and whether the explosion is blocked by other objects.

Parameters

- **pos** – Center position of the explosion.
- **id** – Id of the object of which to check coverage.
- **covMask** – Mask of object types that may block the explosion.

Returns Coverage value from 0 (not affected by the explosion) to 1 (fully affected)

Example:

```
// Get the position of the explosion.
%position = %explosion.getPosition();

// Set a list of TypeMasks (defined in gameFunctioncs.cpp), seperated by the | character.
%TypeMasks = $TypeMasks::StaticObjectType | $TypeMasks::ItemObjectType

// Acquire the damage value from 0.0f - 1.0f.
%coverage = calcExplosionCoverage( %position, %sceneObject, %TypeMasks );

// Apply damage to object
%sceneObject.applyDamage( %coverage * 20 );
```

Decals

Decals are non-SimObject derived objects that are stored and loaded separately from the normal mission file.

The DecalManager handles all aspects of decal management including loading, creation, saving, and automatically deleting decals that have exceeded their lifeSpan.

The static decals associated with a mission are normally loaded immediately after the mission itself has loaded as shown below.

Example:

```
// Load the static mission decals.
decalManagerLoad( %missionName @ ".decals" );
```

Classes

Functions

int **decalManagerAddDecal**(Point3F *position*, Point3F *normal*, float *rot*, float *scale*, DecalData *decalData*, bool *isImmortal*)

Adds a new decal to the decal manager.

Parameters

- **position** – World position for the decal.
- **normal** – Decal normal vector (if the decal was a tire lying flat on a surface, this is the vector pointing in the direction of the axle).
- **rot** – Angle (in radians) to rotate this decal around its normal vector.
- **scale** – Scale factor applied to the decal.
- **decalData** – DecalData datablock to use for the new decal.
- **isImmortal** – Whether or not this decal is immortal. If immortal, it does not expire automatically and must be removed explicitly.

Returns Returns the ID of the new Decal object or -1 on failure.

Example:

```
// Specify the decal position
%position = "1.0 1.0 1.0";

// Specify the up vector
%normal = "0.0 0.0 1.0";

// Add the new decal.
%decalObj = decalManagerAddDecal( %position, %normal, 0.5, 0.35, ScorchBigDecal, false );
```

void **decalManagerClear**()

Removes all decals currently loaded in the decal manager.

Example:

```
// Tell the decal manager to remove all existing decals.decalManagerClear();
```

bool **decalManagerDirty**()

Returns whether the decal manager has unsaved modifications.

Returns True if the decal manager has unsaved modifications, false if everything has been saved.

Example:

```
// Ask the decal manager if it has unsaved modifications.
%hasUnsavedModifications = decalManagerDirty();
```

bool **decalManagerLoad**(string *fileName*)

Clears existing decals and replaces them with decals loaded from the specified file.

Parameters **fileName** – Filename to load the decals from.

Returns True if the decal manager was able to load the requested file, false if it could not.

Example:

```
// Set the filename to load the decals from.
%fileName = "./missionDecals.mis.decals";
// Inform the decal manager to load the decals from the entered filename.decalManagerLoad( %file
```

bool **decalManagerRemoveDecal**(int *decalID*)

Remove specified decal from the scene.

Parameters **decalID** – ID of the decal to remove.

Returns Returns true if successful, false if decal ID not found.

Example:

```
// Specify a decal ID to be removed
%decalID = 1;

// Tell the decal manager to remove the specified decal ID.
decalManagerRemoveDecal( %decalId )
```

void **decalManagerSave**(String *decalSaveFile*)

Saves the decals for the active mission in the entered filename.

Parameters **decalSaveFile** – Filename to save the decals to.

Example:

```
// Set the filename to save the decals in. If no filename is set, then the
// decals will default to <activeMissionName>.mis.decals
%fileName = "./missionDecals.mis.decals";
// Inform the decal manager to save the decals for the active mission.
decalManagerSave( %fileName );
```

Variables

bool $Decals::debugRender

If true, the decal spheres will be visualized when in the editor.

bool $pref::Decals::enabled

Controls whether decals are rendered.

float $pref::Decals::lifeTimeScale

Lifetime that decals will last after being created in the world. Deprecated. Use DecalData::lifeSpan instead.

bool $Decals::poolBuffers

If true, will merge all PrimitiveBuffers and VertexBuffers into a pair of pools before clearing them at the end of a frame. If false, will just clear them at the end of a frame.

float $Decals::sphereDistanceTolerance

The distance at which the decal system will start breaking up decal spheres when adding new decals.

float $Decals::sphereRadiusTolerance
The radius beyond which the decal system will start breaking up decal spheres when adding new decals.

AI

Classes and functions related to artificial intelligence for Torque 3D.

Classes

AIClient Simulated client driven by AI commands.

Inherit: AIConnection

Description This object is derived from the AIConnection class. It introduces its own Player object to solidify the purpose of this class: Simulated client connecting as a player

To get more specific, if you want a strong alternative to AIPlayer (and wish to make use of the AIConnection structure), consider AIClient. AIClient inherits from AIConnection, contains quite a bit of functionality you will find in AIPlayer, and has its own Player object.

Fields

string AIClient::**getAimLocation**
ai.getAimLocation();

string AIClient::**getLocation**
ai.getLocation();

string AIClient::**getMoveDestination**
ai.getMoveDestination();

int AIClient::**getTargetObject**
ai.getTargetObject();

void AIClient::**missionCycleCleanup**
ai.missionCycleCleanup();

void AIClient::**move**
ai.move();

void AIClient::**moveForward**
ai.moveForward();

void AIClient::**setAimLocation**
ai.setAimLocation(x y z);

void AIClient::**setMoveDestination**
ai.setMoveDestination(x y z);

void AIClient::**setMoveSpeed**
ai.setMoveSpeed(float);

void AIClient::**setTargetObject**
ai.setTargetObject(obj);

void AIClient::**stop**
ai.stop();

AIConnection Special client connection driven by an AI, rather than a human.

Inherit: GameConnection

Description Unlike other net connections, AIConnection is intended to run unmanned. Rather than gathering input from a human using a device, move events, triggers, and look events are driven through functions like AIConnection::setMove.

In addition to having its own set of functions for managing client move events, a member variable inherited by GameConnection is toggle: mAIControlled. This is useful for a server to determine if a connection is AI driven via the function GameConnection::isAIControlled

AIConnection is an alternative to manually creating an AI driven game object. When you want the server to manage AI, you will create a specific one from script using a class like AIPlayer. If you do not want the server managing the AI and wish to simulate a complete client connection, you will use AIConnection

.To get more specific, if you want a strong alternative to AIPlayer (and wish to make use of the AIConnection structure), consider AIClient. AIClient inherits from AIConnection, contains quite a bit of functionality you will find in AIPlayer, and has its own Player object.

Example:

```
// Create a new AI client connection
%botConnection = aiConnect("MasterBlaster" @ %i, -1, 0.5, false, "SDF", 1.0);

// In another area of the code, you can locate this and any other AIConnections
// using the isAIControlled function
for(%i = 0; %i < ClientGroup.getCount(); %i++)
{
   %client = ClientGroup.getObject(%i);
   if(%client.isAIControlled())
   {
      // React to this AI controlled client
   }
}
```

Methods

float `AIConnection::getMove` (string *field*)
Get the given field of a move.

Parameters **`field`** – One of {'x','y','z','yaw','pitch','roll'}

Returns The requested field on the current move.

bool `AIConnection::getTrigger` (int *trigger*)
Is the given trigger set?

void `AIConnection::setFreeLook` (bool *isFreeLook*)
Enable/disable freelook on the current move.

void `AIConnection::setMove` (string *field*, float *value*)
Set a field on the current move.

Parameters

- **`field`** – One of {'x','y','z','yaw','pitch','roll'}
- **`value`** – Value to set field to.

void `AIConnection::setTrigger` (int *trigger*, bool *set*)
Set a trigger.

Fields

string AIConnection::**getAddress**

bool AIConnection::**getFreeLook**

getFreeLook() Is freelook on for the current move?

NavMesh

Inherit: SceneObject

Description UNDOCUMENTED!

Methods

bool NavMesh::**build**(bool *background*, bool *save*)

Create a Recast nav mesh.

void NavMesh::**buildTiles**(Box3F *box*)

Rebuild the tiles overlapped by the input box.

void NavMesh::**cancelBuild**()

Cancel the current NavMesh build.

bool NavMesh::**load**()

Load this NavMesh from its file.

void NavMesh::**save**()

Save this NavMesh to its file.

Fields

float NavMesh::**actorClimb**

Maximum climbing height of an actor.

float NavMesh::**actorHeight**

Height of an actor.

float NavMesh::**actorRadius**

Radius of an actor.

bool NavMesh::**alwaysRender**

Display this NavMesh even outside the editor.

int NavMesh::**borderSize**

Size of the non-walkable border around the navigation mesh (in voxels).

float NavMesh::**cellHeight**

Height of a voxel.

float NavMesh::**cellSize**

Length/width of a voxel.

float NavMesh::**detailSampleDist**

Sets the sampling distance to use when generating the detail mesh.

float NavMesh::**detailSampleError**

The maximum distance the detail mesh surface should deviate from heightfield data.

string NavMesh::**fileName**

Name of the data file to store this navmesh in (relative to engine executable).

int NavMesh::**maxEdgeLen**

The maximum allowed length for contour edges along the border of the mesh.

int NavMesh::**maxPolysPerTile**
The maximum number of polygons allowed in a tile.

int NavMesh::**mergeRegionArea**
Any regions with a span count smaller than this value will, if possible, be merged with larger regions.

int NavMesh::**minRegionArea**
The minimum number of cells allowed to form isolated island areas.

float NavMesh::**simplificationError**
The maximum distance a simplfied contour's border edges should deviate from the original raw contour.

float NavMesh::**tileSize**
The horizontal size of tiles.

float NavMesh::**walkableSlope**
Maximum walkable slope in degrees.

NavPath

Inherit: SceneObject

Description UNDOCUMENTED!

Methods

int NavPath::**getCount**()
Return the number of nodes in this path.

float NavPath::**getLength**()
Get the length of this path in Torque units (i.e. the total distance it covers).

Point3F NavPath::**getNode**(int *idx*)
Get a specified node along the path.

bool NavPath::**replan**()
Find a path using the already-specified path properties.

Fields

bool NavPath::**alwaysRender**
Render this NavPath even when not selected.

Point3F NavPath::**from**
World location this path starts at.

bool NavPath::**isLooping**
Does this path loop?

NavMesh NavPath::**mesh**
NavMesh object this path travels within.

Point3F NavPath::**to**
World location this path should end at.

Path NavPath::**waypoints**
Path containing waypoints for this NavPath to visit.

bool NavPath::**xray**
Render this NavPath through other objects.

Functions

int **aiConnect** (...)
Creates a new AIConnection , and passes arguments to its onConnect script callback.

Returns AIConnection

Physics

Objects and functions related to Torque 3D's physics layer.

Classes

PhysicsDebris Represents one or more rigid bodies defined in a single mesh file with a limited lifetime.

Inherit: GameBase

Description A PhysicsDebris object can be viewed as a single system capable of generating multiple PhysicsBodies as debris when triggered. Vaguely similar to how a ParticleEmitter is capable of creating Particles, but isn't a particle in itself. After it's lifetime has elapsed, the object will be deleted.

PhysicsDebris loads a standard .DAE or .DTS file and creates a rigid body for each defined collision node.

For collision nodes to work correctly, they must be setup as follows:

- Visible mesh nodes are siblings of the collision node under a common parent dummy node.
- Collision node is a child of its visible mesh node.

Colmesh type nodes are NOT supported; physx and most standard rigid body simulations do not support arbitrary triangle meshs for dynamics do to the computational expense.

Therefore, collision nodes must be one of the following:

- Colbox
- Colsphere
- Colcapsule
- Col (convex)

PhysicsDebris should NOT be created on the server.

PhysicsDebrisData Defines the properties of a PhysicsDebris object.

Inherit: GameBaseData

Description Defines the properties of a PhysicsDebris object.

Fields

float PhysicsDebrisData::**angularDamping**
Value that reduces an object's rotational velocity over time. Larger values will cause velocity to decay quicker.

float PhysicsDebrisData::**angularSleepThreshold**
Minimum rotational velocity before the shape can be put to sleep. This should be a positive value. Shapes put to sleep will not be simulated in order to save system resources.

float PhysicsDebrisData::**buoyancyDensity**
The density of this shape for purposes of calculating buoyant forces. The result of the calculated buoyancy is relative to the density of the WaterObject the PhysicsDebris is within.

bool PhysicsDebrisData::**castShadows**
Determines if the shape's shadow should be cast onto the environment.

float PhysicsDebrisData::**friction**
Coefficient of kinetic friction to be applied to the shape. Kinetic friction reduces the velocity of a moving object while it is in contact with a surface. A larger coefficient will result in a larger reduction in velocity. A shape's friction should be smaller than it's staticFriction, but greater than 0.

float PhysicsDebrisData::**lifetime**
Base time, in seconds, that debris persists after time of creation.

float PhysicsDebrisData::**lifetimeVariance**
Range of variation randomly applied to lifetime when debris is created. Represents the maximum amount of seconds that will be added or subtracted to a shape's base lifetime. A value of 0 will apply the same lifetime to each shape created.

float PhysicsDebrisData::**linearDamping**
Value that reduces an object's linear velocity over time. Larger values will cause velocity to decay quicker.

float PhysicsDebrisData::**linearSleepThreshold**
Minimum linear velocity before the shape can be put to sleep. This should be a positive value. Shapes put to sleep will not be simulated in order to save system resources.

float PhysicsDebrisData::**mass**
Value representing the mass of the shape. A shape's mass influences the magnitude of any force applied to it.

void PhysicsDebrisData::**preload**
Loads some information to have readily available at simulation time. Forces generation of shaders, materials, and other data used by the PhysicsDebris object. This function should be used while a level is loading in order to shorten the amount of time to create a PhysicsDebris in game.

float PhysicsDebrisData::**restitution**
Bounce coeffecient applied to the shape in response to a collision. Restitution is a ratio of a shape's velocity before and after a collision. A value of 0 will zero out a shape's post-collision velocity, making it stop on contact. Larger values will remove less velocity after a collision, making it 'bounce' with greater force. Normal restitution values range between 0 and 1.0.

filename PhysicsDebrisData::**shapeFile**
Path to the .DAE or .DTS file to use for this shape. Compatable with Live-Asset Reloading.

float PhysicsDebrisData::**staticFriction**
Coefficient of static friction to be applied to the shape. Static friction determines the force needed to start moving an at-rest object in contact with a surface. If the force applied onto shape cannot overcome the force of static friction, the shape will remain at rest. A higher coefficient will require a larger force to start motion. This value should be both greater than 0 and the PhysicsDebrisData::friction .

float PhysicsDebrisData::**waterDampingScale**
Scale to apply to linear and angular dampening while underwater.

PhysicsForce Helper object for gameplay physical forces.

Inherit: SceneObject

Description PhysicsForces can be created and "attached" to other PhysicsBodies to attract them to the position of the PhysicsForce.

Methods

void PhysicsForce::**attach**(Point3F *start*, Point3F *direction*, float *maxDist*)
Attempts to associate the PhysicsForce with a PhysicsBody. Performs a physics ray cast of the provided length and direction. The PhysicsForce will attach itself to the first dynamic PhysicsBody the ray collides with. On every tick, the attached body will be attracted towards the position of the PhysicsForce. A PhysicsForce can only be attached to one body at a time.

void PhysicsForce::**detach**(Point3F *force*)
Disassociates the PhysicsForce from any attached PhysicsBody.

Parameters **force** – Optional force to apply to the attached PhysicsBody before detaching.

bool PhysicsForce::**isAttached**()
Returns true if the PhysicsForce is currently attached to an object.

PhysicsShape Represents a destructible physical object simulated through the plugin system.

Inherit: GameBase

Description Represents a destructible physical object simulated through the plugin system.

Methods

void PhysicsShape::**destroy**()
Disables rendering and physical simulation. Calling destroy() will also spawn any explosions, debris, and/or destroyedShape defined for it, as well as remove it from the scene graph. Destroyed objects are only created on the server. Ghosting will later update the client.

bool PhysicsShape::**isDestroyed**()
Returns if a PhysicsShape has been destroyed or not.

void PhysicsShape::**restore**()
Restores the shape to its state before being destroyed. Re-enables rendering and physical simulation on the object and adds it to the client's scene graph. Has no effect if the shape is not destroyed.

Fields

bool PhysicsShape::**playAmbient**
Enables or disables playing of an ambient animation upon loading the shape.

PhysicsShapeData Defines the properties of a PhysicsShape.

Inherit: GameBaseData

Description Defines the properties of a PhysicsShape.

Fields

float `PhysicsShapeData::`**`angularDamping`**
Value that reduces an object's rotational velocity over time. Larger values will cause velocity to decay quicker.

float `PhysicsShapeData::`**`angularSleepThreshold`**
Minimum rotational velocity before the shape can be put to sleep. This should be a positive value. Shapes put to sleep will not be simulated in order to save system resources.

float `PhysicsShapeData::`**`buoyancyDensity`**
The density of the shape for calculating buoyant forces. The result of the calculated buoyancy is relative to the density of the WaterObject the PhysicsShape is within.

PhysicsDebrisData `PhysicsShapeData::`**`Debris`**
Name of a PhysicsDebrisData to spawn when this shape is destroyed (optional).

PhysicsShapeData `PhysicsShapeData::`**`destroyedShape`**
Name of a PhysicsShapeData to spawn when this shape is destroyed (optional).

ExplosionData `PhysicsShapeData::`**`Explosion`**
Name of an ExplosionData to spawn when this shape is destroyed (optional).

float `PhysicsShapeData::`**`friction`**
Coefficient of kinetic friction to be applied to the shape. Kinetic friction reduces the velocity of a moving object while it is in contact with a surface. A higher coefficient will result in a larger velocity reduction. A shape's friction should be lower than it's staticFriction, but larger than 0.

float `PhysicsShapeData::`**`linearDamping`**
Value that reduces an object's linear velocity over time. Larger values will cause velocity to decay quicker.

float `PhysicsShapeData::`**`linearSleepThreshold`**
Minimum linear velocity before the shape can be put to sleep. This should be a positive value. Shapes put to sleep will not be simulated in order to save system resources.

float `PhysicsShapeData::`**`mass`**
Value representing the mass of the shape. A shape's mass influences the magnitude of any force exerted on it. For example, a PhysicsShape with a large mass requires a much larger force to move than the same shape with a smaller mass.

float `PhysicsShapeData::`**`restitution`**
Coeffecient of a bounce applied to the shape in response to a collision. Restitution is a ratio of a shape's velocity before and after a collision. A value of 0 will zero out a shape's post-collision velocity, making it stop on contact. Larger values will remove less velocity after a collision, making it 'bounce' with a greater force. Normal restitution values range between 0 and 1.0.

filename `PhysicsShapeData::`**`shapeName`**
Path to the .DAE or .DTS file to use for this shape. Compatable with Live-Asset Reloading.

PhysicsSimType `PhysicsShapeData::`**`simType`**
Controls whether this shape is simulated on the server, client, or both physics simulations.

float `PhysicsShapeData::`**`staticFriction`**
Coefficient of static friction to be applied to the shape. Static friction determines the force needed to start moving an at-rest object in contact with a surface. If the force applied onto shape cannot overcome the force of static friction, the shape will remain at rest. A larger coefficient will require a larger force to start motion. This value should be larger than zero and the physicsShape's friction.

float PhysicsShapeData::**waterDampingScale**
Scale to apply to linear and angular dampening while underwater. Used with the waterViscosity of the

PxCloth Rectangular patch of cloth simulated by PhysX.

Inherit: GameBase

Description PxCloth is affected by other objects in the simulation but does not itself affect others, it is essentially a visual effect. Eg, shooting at cloth will disturb it but will not explode the projectile.

Be careful with the cloth size and resolution because it can easily become performance intensive to simulate. A single piece of cloth that is very large or high resolution is also much more expensive than multiple pieces that add up to the same number of verts.

Note that most field docs have been copied from their PhysX counterpart.

Fields

PxClothAttachment PxCloth::**attachments**
Optional way to specify cloth verts that will be attached to the world position it is created at.

bool PxCloth::**bending**
Enables or disables bending resistance. Set the bending resistance through PxCloth::bendingStiffness .

float PxCloth::**bendingStiffness**
Bending stiffness of the cloth in the range 0 to 1.

bool PxCloth::**damping**
Enable/disable damping of internal velocities.

float PxCloth::**dampingCoefficient**
Spring damping of the cloth in the range 0 to 1.

float PxCloth::**density**
Density of the cloth (Mass per Area).

float PxCloth::**friction**
Friction coefficient in the range 0 to 1. Defines the damping of the velocities of cloth particles that are in contact.

string PxCloth::**Material**
Name of the material to render.

Point2I PxCloth::**samples**
The number of cloth vertices in width and length. At least two verts should be defined.

bool PxCloth::**selfCollision**
Enables or disables self-collision handling within a single piece of cloth.

Point2F PxCloth::**size**
The width and height of the cloth.

float PxCloth::**thickness**
Value representing how thick the cloth is. The thickness is usually a fraction of the overall extent of the cloth and should not be set to a value greater than that. A good value is the maximal distance between two adjacent cloth particles in their rest pose. Visual artifacts or collision problems may appear if the thickness is too small.

bool PxCloth::**triangleCollision**
Not supported in current release (according to PhysX docs). Enables or disables collision detection of cloth triangles against the scene. If not set, only collisions of cloth particles are detected. If set, collisions of cloth triangles are detected as well.

PxMaterial Defines a PhysX material assignable to a PxMaterial.

Inherit: SimDataBlock

Description When two actors collide, the collision behavior that results depends on the material properties of the actors' surfaces. For example, the surface properties determine if the actors will or will not bounce, or if they will slide or stick. Currently, the only special feature supported by materials is anisotropic friction, but according to Nvidia, other effects such as moving surfaces and more types of friction are slotted for future release.

For more information, refer to Nvidia's PhysX docs.

Fields

float PxMaterial::**dynamicFriction**
Coefficient of dynamic friction to be applied. Dynamic friction reduces the velocity of a moving object while it is in contact with a surface. A higher coefficient will result in a larger reduction in velocity. A shape's dynamicFriction should be equal to or larger than 0.

float PxMaterial::**restitution**
Coeffecient of a bounce applied to the shape in response to a collision. A value of 0 makes the object bounce as little as possible, while higher values up to 1.0 result in more bounce.

float PxMaterial::**staticFriction**
Coefficient of static friction to be applied. Static friction determines the force needed to start moving an at-rest object in contact with a surface. If the force applied onto shape cannot overcome the force of static friction, the shape will remain at rest. A higher coefficient will require a larger force to start motion.

PxMultiActor Represents a destructible physical object simulated using PhysX.

Inherit: GameBase

Description Usually it is prefered to use PhysicsShape and not PxMultiActor because it is not PhysX specific and much easier to setup.

Methods

void PxMultiActor::listMeshes(enum Hidden, enum Shown, enum All)
Lists all meshes of the provided type in the console window.

Parameters

- **All** – Lists all of the PxMultiActor's meshes.
- **Hidden** – Lists all of the PxMultiActor's hidden meshes.
- **Shown** – Lists all of the PxMultiActor's visible meshes.

void PxMultiActor::**setAllHidden**()
Hides or unhides all meshes contained in the PxMultiActor . Hidden meshes will not be rendered.

void PxMultiActor::**setBroken**()
Sets the PxMultiActor to a broken or unbroken state.

void PxMultiActor::**setMeshHidden**(string *meshName*, bool *isHidden*)
Prevents the provided mesh from being rendered.

Fields
bool PxMultiActor::**broken**
bool PxMultiActor::**debugRender**

PxMultiActorData Defines the properties of a type of PxMultiActor.

Inherit: GameBaseData

Description Usually it is prefered to use PhysicsShape rather than PxMultiActor because a PhysicsShape is not PhysX specific and can be much easier to setup.

For more information, refer to Nvidia's PhysX docs.

Fields
float PxMultiActorData::**angularDrag**
Value used to help calculate rotational drag force while submerged in water.
float PxMultiActorData::**breakForce**
Force required to break an actor. This value does not apply to joints. If an actor is associated with a joint it will break whenever the joint does. This allows an actor "not" associated with a joint to also be breakable.

float PxMultiActorData::**buoyancyDensity**
The density used to calculate buoyant forces. The result of the calculated buoyancy is relative to the density of the WaterObject the PxMultiActor is within.

bool PxMultiActorData::**clientOnly**

void PxMultiActorData::**dumpModel**
Dumps model hierarchy and details to a file. The file will be created as 'model.dump' in the game folder. If model.dump already exists, it will be overwritten.

float PxMultiActorData::**linearDrag**
Value used to help calculate linear drag force while submerged in water.

PxMaterial PxMultiActorData::**Material**
An optional PxMaterial to be used for the PxMultiActor . Defines properties such as friction and restitution. Unrelated to the material used for rendering. The physXStream will contain defined materials that can be customized in 3DS Max. To override the material for all physics shapes in the physXStream, specify a material here.

bool PxMultiActorData::**noCorrection**

filename PxMultiActorData::**physXStream**
.XML file containing data such as actors, shapes, and joints. These files can be created using a free PhysX plugin for 3DS Max.

void PxMultiActorData::**reload**
Reloads all data used for the PxMultiActorData . If the reload sucessfully completes, all PxMultiActor's will be notified.

filename PxMultiActorData::**shapeName**
Path to the .DAE or .DTS file to render.

bool PxMultiActorData::**singlePlayerOnly**

string PxMultiActorData::**string**

float PxMultiActorData::**waterDragScale**
Scale to apply to linear and angular dampening while submerged in water.

RadialImpulseEvent Creates a physics-based impulse effect from a defined central point and magnitude.

Description Creates a physics-based impulse effect from a defined central point and magnitude.

Methods

static void RadialImpulseEvent::**send**(string *inPosition*, float *radius*, float *magnitude*)
Applies a radial impulse to any SceneObjects within the area of effect. This event is performed both server and client-side.

Parameters

- **position** – Center point for this radial impulse.
- **radius** – Distance from the position for this radial impulse to affect.
- **magnitude** – The force applied to objects within the radius from the position of this radial impulse effect.

Example:

```
// Define the Position
%position = "10.0 15.0 10.0";

// Define the Radius
%radius = "25.0";

// Define the Magnitude
%magnitude = "30.0"
// Create a globalRadialImpulse physics effect.
RadialImpulseEvent::send(%position,%radius,%magnitude);
```

RigidShape Implements rigid-body physics for DTS objects in the world.

Inherit: ShapeBase

Description The RigidShape class implements rigid-body physics for DTS objects in the world.

"Rigid body physics" refers to a system whereby objects are assumed to have a finite size, equally distributed masses, and where deformations of the objects themselves are not accounted for. Uses the RigidShape class to control its physics.

Example:

```
datablock RigidShapeData( BouncingBoulder )
{
   category = "RigidShape";

   shapeFile = "~/data/shapes/boulder/boulder.dts";
   emap = true;

   // Rigid Body
```

```
   mass = 500;
   massCenter = "0 0 0";    // Center of mass for rigid body
   massBox = "0 0 0";         // Size of box used for moment of inertia,
                        // if zero it defaults to object bounding box
   drag = 0.2;                // Drag coefficient
   bodyFriction = 0.2;
   bodyRestitution = 0.1;
   minImpactSpeed = 5;        // Impacts over this invoke the script callback
   softImpactSpeed = 5;       // Play SoftImpact Sound
   hardImpactSpeed = 15;      // Play HardImpact Sound
   integration = 4;           // Physics integration: TickSec/Rate
   collisionTol = 0.1;        // Collision distance tolerance
   contactTol = 0.1;          // Contact velocity tolerance

   minRollSpeed = 10;

   maxDrag = 0.5;
   minDrag = 0.01;

   dustHeight = 10;

   dragForce = 0.05;
   vertFactor = 0.05;
};

     new RigidShape()
{
   dataBlock = "BouncingBoulder";
   parentGroup = EWCreatorWindow.objectGroup;
};
```

Methods

void RigidShape::**forceClientTransform**()

Forces the client to jump to the RigidShape's transform rather then warp to it.

void RigidShape::**freezeSim**(bool *isFrozen*)

Enables or disables the physics simulation on the RigidShape object.

Parameters **isFrozen** – Boolean frozen state to set the object.

Example:

```
// Define the frozen state.
%isFrozen = "true";

// Inform the object of the defined frozen state
%thisRigidShape.freezeSim(%isFrozen);
```

void RigidShape::**onEnterLiquid**(string *objId*, string *waterCoverage*, string *liquidType*)

Called whenever this RigidShape object enters liquid.

Parameters

- **objId** – The ID of the rigidShape object.
- **waterCoverage** – Amount of water coverage the RigidShape has.
- **liquidType** – Type of liquid that was entered.

Example:

```
// The RigidShape object falls in a body of liquid, causing the callback to occur.
RigidShape::onEnterLiquid(%this,%objId,%waterCoverage,%liquidType)
{
        // Code to run whenever this callback occurs.
}
```

void RigidShape::**onLeaveLiquid**(string *objId*, string *liquidType*)

Called whenever the RigidShape object exits liquid.

Parameters

- **objId** – The ID of the RigidShape object.
- **liquidType** – Type if liquid that was exited.

Example:

```
// The RigidShape object exits in a body of liquid, causing the callback to occur.
RigidShape::onLeaveLiquid(%this,%objId,%liquidType)
{
  // Code to run whenever this callback occurs.
}
```

void RigidShape::**reset**()

Clears physic forces from the shape and sets it at rest.

Example:

```
// Inform the RigidShape object to reset.
%thisRigidShape.reset();
```

RigidShapeData Physics object.

Inherit: ShapeBaseData

Description Defines the physics properties for an individual RigidShapeData physics object.

Example:

```
datablock RigidShapeData( BouncingBoulder )
{
   category = "RigidShape";

   shapeFile = "~/data/shapes/boulder/boulder.dts";
   emap = true;

   // Rigid Bodymass = 500;
   massCenter = "0 0 0";    // Center of mass for rigid bodymass
   Box = "0 0 0";         // Size of box used for moment of inertia,
                          // if zero it defaults to object bounding box
   drag = 0.2;                // Drag coefficientbodyFriction = 0.2;
   bodyRestitution = 0.1;
   minImpactSpeed = 5;        // Impacts over this invoke the script callback
   softImpactSpeed = 5;       // Play SoftImpact Sound
   hardImpactSpeed = 15;      // Play HardImpact Sound
   integration = 4;           // Physics integration: TickSec/Rate
   collisionTol = 0.1;        // Collision distance
   tolerancecontactTol = 0.1; // Contact velocity toleranceminRollSpeed = 10;
```

```
   maxDrag = 0.5;
   minDrag = 0.01;

   dustHeight = 10;

   dragForce = 0.05;
   vertFactor = 0.05;
};
```

Fields

float RigidShapeData::**bodyFriction**
How much friction this object has. Lower values will cause the object to appear to be more slippery.

float RigidShapeData::**bodyRestitution**
The percentage of kinetic energy kept by this object in a collision.

float RigidShapeData::**cameraDecay**
Scalar rate at which the third person camera offset decays, per tick.

float RigidShapeData::**cameraLag**
Scalar amount by which the third person camera lags the object, relative to the object's linear velocity.

float RigidShapeData::**cameraOffset**
The vertical offset of the object's camera.

bool RigidShapeData::**cameraRoll**
Specifies whether the camera's rotation matrix, and the render eye transform are multiplied during camera updates.

float RigidShapeData::**collisionTol**
Collision distance tolerance.

float RigidShapeData::**contactTol**
Contact velocity tolerance.

float RigidShapeData::**dragForce**
Used to simulate the constant drag acting on the object.

ParticleEmitterData RigidShapeData::**dustEmitter**
Array of pointers to ParticleEmitterData datablocks which will be used to emit particles at object/terrain contact point.

float RigidShapeData::**dustHeight**
Height of dust effects.

ParticleEmitterData RigidShapeData::**dustTrailEmitter**
Particle emitter used to create a dust trail for the moving object.

SFXTrack RigidShapeData::**exitingWater**
The AudioProfile will be used to produce sounds when emerging from water.

float RigidShapeData::**exitSplashSoundVelocity**
The minimum velocity at which the exit splash sound will be played when emerging from water.

SFXTrack RigidShapeData::**hardImpactSound**
Sound to play when body impacts with at least hardImpactSpeed.

float RigidShapeData::**hardImpactSpeed**
Minimum speed at which the object must be travelling for the hard impact sound to be played.

float RigidShapeData::**hardSplashSoundVelocity**
The minimum velocity at which the hard splash sound will be played when impacting water.

SFXTrack RigidShapeData::**impactWaterEasy**
The AudioProfile will be used to produce sounds when a soft impact with water occurs.

SFXTrack RigidShapeData::**impactWaterHard**
The AudioProfile will be used to produce sounds when a hard impact with water occurs.

SFXTrack RigidShapeData::**impactWaterMedium**
The AudioProfile will be used to produce sounds when a medium impact with water occurs.

int RigidShapeData::**integration**
Number of physics steps to process per tick.

Point3F RigidShapeData::**massBox**
Size of inertial box.

Point3F RigidShapeData::**massCenter**
Center of mass for rigid body.

float RigidShapeData::**maxDrag**
Maximum drag available to this object.

float RigidShapeData::**mediumSplashSoundVelocity**
The minimum velocity at which the medium splash sound will be played when impacting water.

float RigidShapeData::**minDrag**
Minimum drag available to this object.

float RigidShapeData::**minImpactSpeed**
Minimum collision speed to classify collision as impact (triggers onImpact on server object).

float RigidShapeData::**minRollSpeed**

SFXTrack RigidShapeData::**softImpactSound**
Sound to play when body impacts with at least softImageSpeed but less than hardImpactSpeed.

float RigidShapeData::**softImpactSpeed**
Minimum speed at which this object must be travelling for the soft impact sound to be played.

float RigidShapeData::**softSplashSoundVelocity**
The minimum velocity at which the soft splash sound will be played when impacting water.

ParticleEmitterData RigidShapeData::**splashEmitter**[2]
Array of pointers to ParticleEmitterData datablocks which will generate splash effects.

float RigidShapeData::**splashFreqMod**
The simulated frequency modulation of a splash generated by this object. Multiplied along with speed and time elapsed when determining splash emition rate.

float RigidShapeData::**splashVelEpsilon**
The threshold speed at which we consider the object's movement to have stopped when updating splash effects.

float RigidShapeData::**triggerDustHeight**
Maximum height from the ground at which the object will generate dust.

float RigidShapeData::**vertFactor**
The scalar applied to the vertical portion of the velocity drag acting on a object.

SFXTrack RigidShapeData::**waterWakeSound**
The AudioProfile will be used to produce sounds when a water wake is displayed.

Enumeration

enum PhysicsSimType
How to handle the physics simulation with the client's and server.

Parameters

- **ClientOnly** – Only handle physics on the client.
- **ServerOnly** – Only handle physics on the server.
- **ClientServer** – Handle physics on both the client and server.

Variables

bool $PhysXLogWarnings
Output PhysX warnings to the console.

bool $Physics::isSinglePlayer
Informs the physics simulation if only a single player exists. If true, optimizations will be implemented to better cater to a single player environmnent.

bool **physicsPluginPresent**
Returns true if a physics plugin exists and is initialized. physicsPluginPresent()

int $pref::Physics::threadCount
Number of threads to use in a single pass of the physics engine. Defaults to 2 if not set.

Vehicles

This section is dedicated to vehicle game objects, such as the base Vehicle class, WheeledVehicle, FlyingVehicle, and so on.

Classes

FlyingVehicle A flying vehicle.

Inherit: Vehicle

Description The model used for the FlyingVehicle should contain the elements shown below. Only the collision mesh is actually required for the object to be added to the simulation, but particle emitters will not work unless the relevant nodes are present.

The example below shows the datablock required for a simple FlyingVehicle. The script should be executed on the server, and the vehicle can then be added to the simulation programmatically from the level startup scripts, or by selecting the JetFighter datablock from the World Editor (Library->ScriptedObjects->Vehicles).

Example:

```
datablock FlyingVehicleData( JetFighter )
{
   category = "Vehicles";
   shapeFile = "art/shapes/fighterjet.dae";

   createHoverHeight       = 20;
```

```
   // 3rd person camera settings
   cameraRoll              = true;
   cameraMaxDist           = 16;
   cameraOffset            = 1.0;
   cameraLag               =  0.1;
   cameraDecay             = 1.25;

   // Rigid Body
   mass                    = 100;
   massCenter              = "0 -0.2 0";
   massBox                 = "0 0 0";
   integration             = 3;
   collisionTol            = 0.6;
   contactTol              = 0.4;

   bodyFriction            = 0;
   bodyRestitution         = 0.8;
   minRollSpeed            = 2000;
   minImpactSpeed          = 5;
   softImpactSpeed         = 3;
   hardImpactSpeed         = 15;

   drag                    = 0.25;
   minDrag                 = 40;
   rotationalDrag          = 20;

   // Autostabilizer
   maxAutoSpeed            = 6;
   autoAngularForce        = 400;
   autoLinearForce         = 300;
   autoInputDamping        = 0.55;

   // Maneuvering
   maxSteeringAngle        = 3;
   horizontalSurfaceForce  = 20;
   verticalSurfaceForce    = 20;
   maneuveringForce        = 6400;
   steeringForce           = 500;
   steeringRollForce       = 200;
   rollForce               = 10;
   hoverHeight             = 0.5;
   createHoverHeight       = 0.5;
   maxForwardSpeed         = 90;

   // Vertical jetting
   maxEnergy               = 100;
   jetForce                = 3000;
   minJetEnergy            = 28;
   jetEnergyDrain          = 2.8;
   vertThrustMultiple      = 3.0;

   // Emitters
   forwardJetEmitter       = FighterJettingEmitter;
   backwardJetEmitter      = FighterJettingEmitter;
   downJetEmitter          = FighterJettingEmitter;
   trailEmitter            = FighterContrailEmitter;
   minTrailSpeed           = 10;
```

```
   // Sounds
   engineSound             = FighterEngineSnd;
   jetSound                = FighterJettingSnd;
};

// This function is executed when the FlyingVehicle object is added to the simulation.
function JetFighter::onAdd( %this, %obj )
{
   Parent::onAdd( %this, %obj );

   // Allow jetting energy to recharge over time
   %obj.setRechargeRate( 2 );
}
```

Methods

void `FlyingVehicle::`**`useCreateHeight`**(bool *enabled*)

Set whether the vehicle should temporarily use the createHoverHeight specified in the datablock. This can help avoid problems with spawning.

Parameters **`enabled`** – true to use the datablock createHoverHeight, false otherwise

FlyingVehicleData Defines the properties of a FlyingVehicle.

Inherit: VehicleData

Description Defines the properties of a FlyingVehicle.

Fields

float `FlyingVehicleData::`**`autoAngularForce`**

Corrective torque applied to level out the vehicle when moving at less than maxAutoSpeed. The torque is inversely proportional to vehicle speed.

float `FlyingVehicleData::`**`autoInputDamping`**

Scale factor applied to steering input if speed is less than maxAutoSpeed to.improve handling at very low speeds. Smaller values make steering less sensitive.

float `FlyingVehicleData::`**`autoLinearForce`**

Corrective force applied to slow the vehicle when moving at less than maxAutoSpeed. The force is inversely proportional to vehicle speed.

ParticleEmitterData `FlyingVehicleData::`**`backwardJetEmitter`**

Emitter to generate particles for backward jet thrust. Backward jet thrust particles are emitted from model nodes JetNozzleX and JetNozzleY.

float `FlyingVehicleData::`**`createHoverHeight`**

The vehicle's height off the ground when useCreateHeight is active. This can help avoid problems with spawning the vehicle.

ParticleEmitterData `FlyingVehicleData::`**`downJetEmitter`**

Emitter to generate particles for downward jet thrust. Downward jet thrust particles are emitted from model nodes JetNozzle2 and JetNozzle3.

SFXProfile `FlyingVehicleData::`**`engineSound`**

Looping engine sound.

ParticleEmitterData `FlyingVehicleData::`**`forwardJetEmitter`**
Emitter to generate particles for forward jet thrust. Forward jet thrust particles are emitted from model nodes JetNozzle0 and JetNozzle1.

float `FlyingVehicleData::`**`horizontalSurfaceForce`**
Damping force in the opposite direction to sideways velocity. Provides "bite" into the wind for climbing/diving and turning).

float `FlyingVehicleData::`**`hoverHeight`**
The vehicle's height off the ground when at rest.

SFXProfile `FlyingVehicleData::`**`jetSound`**
Looping sound to play while the vehicle is jetting.

float `FlyingVehicleData::`**`maneuveringForce`**
Maximum X and Y (horizontal plane) maneuvering force. The actual force applied depends on the current thrust.

float `FlyingVehicleData::`**`maxAutoSpeed`**
Maximum speed for automatic vehicle control assistance - vehicles travelling at speeds above this value do not get control assitance.

float `FlyingVehicleData::`**`minTrailSpeed`**
Minimum speed at which to start generating contrail particles.

float `FlyingVehicleData::`**`rollForce`**
Damping torque against rolling maneuvers (rotation about the y-axis), proportional to linear velocity. Acts to adjust roll to a stable position over time as the vehicle moves.

float `FlyingVehicleData::`**`rotationalDrag`**
Rotational drag factor (slows vehicle rotation speed in all axes).

float `FlyingVehicleData::`**`steeringForce`**
Maximum X and Z (sideways and vertical) steering force. The actual force applied depends on the current steering input.

float `FlyingVehicleData::`**`steeringRollForce`**
Roll force induced by sideways steering input value (controls how much the vehicle rolls when turning).

ParticleEmitterData `FlyingVehicleData::`**`trailEmitter`**
Emitter to generate contrail particles from model nodes contrail0 - contrail3.

float `FlyingVehicleData::`**`verticalSurfaceForce`**
Damping force in the opposite direction to vertical velocity. Controls side slip; lower numbers give more slide.

float `FlyingVehicleData::`**`vertThrustMultiple`**
Multiplier applied to the jetForce (defined in VehicleData) when thrusting vertically.

HoverVehicle A hovering vehicle.

Inherit: Vehicle

Description A hover vehicle is a vehicle that maintains a specific distance between the vehicle and the ground at all times; unlike a flying vehicle which is free to ascend and descend at will.The model used for the HoverVehicle has the following requirements:

HoverVehicleData Defines the properties of a HoverVehicle.

Inherit: VehicleData

Description Defines the properties of a HoverVehicle.

Fields

float `HoverVehicleData::`**`brakingActivationSpeed`**
Maximum speed below which a braking force is applied.

float `HoverVehicleData::`**`brakingForce`**
Force generated by braking. The vehicle is considered to be braking if it is moving, but the throttle is off, and no left or right thrust is being applied. This force is only applied when the vehicle's velocity is less than brakingActivationSpeed.

float `HoverVehicleData::`**`dragForce`**
Drag force factor that acts opposite to the vehicle velocity. Also used to determnine the vehicle's maxThrustSpeed.

ParticleEmitterData `HoverVehicleData::`**`dustTrailEmitter`**
Emitter to generate particles for the vehicle's dust trail. The trail of dust particles is generated only while the vehicle is moving.

float `HoverVehicleData::`**`dustTrailFreqMod`**
Number of dust trail particles to generate based on vehicle speed. The vehicle's speed is divided by this value to determine how many particles to generate each frame. Lower values give a more dense trail, higher values a more sparse trail.

Point3F `HoverVehicleData::`**`dustTrailOffset`**
"X Y Z" offset from the vehicle's origin from which to generate dust trail particles. By default particles are emitted directly beneath the origin of the vehicle model.

SFXProfile `HoverVehicleData::`**`engineSound`**
Looping engine sound. The volume is dynamically adjusted based on the current thrust level.

float `HoverVehicleData::`**`floatingGravMag`**
Scale factor applied to the vehicle gravitational force when the vehicle is floating.

float `HoverVehicleData::`**`floatingThrustFactor`**
Scalar applied to the vehicle's thrust force when the vehicle is floating.

SFXProfile `HoverVehicleData::`**`floatSound`**
Looping sound played while the vehicle is floating.

ParticleEmitterData `HoverVehicleData::`**`forwardJetEmitter`**
Emitter to generate particles for forward jet thrust. Forward jet thrust particles are emitted from model nodes JetNozzle0 and JetNozzle1.

float `HoverVehicleData::`**`gyroDrag`**
Damping torque that acts against the vehicle's current angular momentum.

SFXProfile `HoverVehicleData::`**`jetSound`**
Looping sound played when the vehicle is jetting.

float `HoverVehicleData::`**`mainThrustForce`**
Force generated by thrusting the vehicle forward. Also used to determine the maxThrustSpeed:

Example:

```
maxThrustSpeed = (mainThrustForce + strafeThrustForce) / dragForce;
```

float `HoverVehicleData::`**`normalForce`**
Force generated in the ground normal direction when the vehicle is not floating (within stabalizer length from the ground).

float HoverVehicleData::**pitchForce**
Pitch (rotation about the X-axis) force applied when steering in the y-axis direction.

float HoverVehicleData::**restorativeForce**
Force generated to stabalize the vehicle (return it to neutral pitch/roll) when the vehicle is floating (more than stabalizer length from the ground.

float HoverVehicleData::**reverseThrustForce**
Force generated by thrusting the vehicle backward.

float HoverVehicleData::**rollForce**
Roll (rotation about the Y-axis) force applied when steering in the x-axis direction.

float HoverVehicleData::**stabDampingConstant**
Damping spring force acting against changes in the stabalizer length.

float HoverVehicleData::**stabLenMax**
Length of the base stabalizer when travelling at maximum speed (maxThrustSpeed).

float HoverVehicleData::**stabLenMin**
Length of the base stabalizer when travelling at minimum speed (0). Each tick, the vehicle performs 2 raycasts (from the center back and center front of the vehicle) to check for contact with the ground. The base stabalizer length determines the length of that raycast; if neither raycast hit the ground, the vehicle is floating, stabalizer spring and ground normal forces are not applied.

float HoverVehicleData::**stabSpringConstant**
Value used to generate stabalizer spring force. The force generated depends on stabilizer compression, that is how close the vehicle is to the ground proportional to current stabalizer length.

float HoverVehicleData::**steeringForce**
Yaw (rotation about the Z-axis) force applied when steering in the x-axis direction.about the vehicle's Z-axis).

float HoverVehicleData::**strafeThrustForce**
Force generated by thrusting the vehicle to one side. Also used to determine the vehicle's maxThrustSpeed.

float HoverVehicleData::**triggerTrailHeight**
Maximum height above surface to emit dust trail particles. If the vehicle is less than triggerTrailHeight above a static surface with a material that has 'showDust' set to true, the vehicle will emit particles from the dustTrailEmitter.

float HoverVehicleData::**turboFactor**
Scale factor applied to the vehicle's thrust force when jetting.

float HoverVehicleData::**vertFactor**
Scalar applied to the vertical portion of the velocity drag acting on the vehicle. For the horizontal (X and Y) components of velocity drag, a factor of 0.25 is applied when the vehicle is floating, and a factor of 1.0 is applied when the vehicle is not floating. This velocity drag is multiplied by the vehicle's dragForce, as defined above, and the result is subtracted from it's movement force.

Vehicle Base functionality shared by all Vehicles (FlyingVehicle, HoverVehicle, WheeledVehicle).

Inherit: ShapeBase

Description This object implements functionality shared by all Vehicle types, but should not be instantiated directly. Create a FlyingVehicle, HoverVehicle, or WheeledVehicle instead.

Fields
bool Vehicle::**disableMove**
When this flag is set, the vehicle will ignore throttle changes.

float Vehicle::**workingQueryBoxSizeMultiplier**[static]
How much larger the mWorkingQueryBox should be made when updating the working collision list. The larger this number the less often the working list will be updated due to motion, but any non-static shape that moves into the query box will not be noticed.

int Vehicle::**workingQueryBoxStaleThreshold**[static]
The maximum number of ticks that go by before the mWorkingQueryBox is considered stale and needs updating. Other factors can cause the collision working query box to become invalidated, such as the vehicle moving far enough outside of this cached box. The smaller this number, the more times the working list of triangles that are considered for collision is refreshed. This has the greatest impact with colliding with high triangle count meshes.

VehicleData Base properties shared by all Vehicles (FlyingVehicle, HoverVehicle, WheeledVehicle).

Inherit: ShapeBaseData

Description This datablock defines properties shared by all Vehicle types, but should not be instantiated directly. Instead, set the desired properties in the FlyingVehicleData, HoverVehicleData or WheeledVehicleData datablock.

Damage The VehicleData class extends the basic energy/damage functionality provided by ShapeBaseData to include damage from collisions, as well as particle emitters activated automatically when damage levels reach user specified thresholds.

The example below shows how to setup a Vehicle to:

Example:

```
// damage from collisionscollDamageMultiplier = 0.05;
collDamageThresholdVel = 15;

// damage levelsdamageLevelTolerance[0] = 0.5;
damageEmitter[0] = GraySmokeEmitter;     // emitter used when damage is >= 50%
damageLevelTolerance[1] = 0.85;
damageEmitter[1] = BlackSmokeEmitter;    // emitter used when damage is >= 85%
damageEmitter[2] = DamageBubbleEmitter;  // emitter used instead of damageEmitter[0:1]
// when offset point is underwater
// emit offsets (used for all active damage level emitters)
damageEmitterOffset[0] = "0.5 3 1";
damageEmitterOffset[1] = "-0.5 3 1";
numDmgEmitterAreas = 2;
```

Methods

void VehicleData::**onEnterLiquid**(Vehicle *obj*, float *coverage*, string *type*)
Called when the vehicle enters liquid.

Parameters

- **obj** – the Vehicle object
- **coverage** – percentage of the vehicle's bounding box covered by the liquid
- **type** – type of liquid the vehicle has entered

void VehicleData::**onLeaveLiquid**(Vehicle *obj*, string *type*)
Called when the vehicle leaves liquid.

Parameters

- **obj** – the Vehicle object
- **type** – type of liquid the vehicle has left

Fields

float VehicleData::**bodyFriction**
Collision friction coefficient. How well this object will slide against objects it collides with.

float VehicleData::**bodyRestitution**
Collision 'bounciness'. Normally in the range 0 (not bouncy at all) to 1 (100% bounciness).

float VehicleData::**cameraDecay**
How quickly the camera moves back towards the vehicle when stopped.

float VehicleData::**cameraLag**
How much the camera lags behind the vehicle depending on vehicle speed. Increasing this value will make the camera fall further behind the vehicle as it accelerates away.

float VehicleData::**cameraOffset**
Vertical (Z axis) height of the camera above the vehicle.

bool VehicleData::**cameraRoll**
If true, the camera will roll with the vehicle. If false, the camera will always have the positive Z axis as up.

float VehicleData::**collDamageMultiplier**
Damage to this vehicle after a collision (multiplied by collision velocity). Currently unused.

float VehicleData::**collDamageThresholdVel**
Minimum collision velocity to cause damage to this vehicle. Currently unused.

float VehicleData::**collisionTol**
Minimum distance between objects for them to be considered as colliding.

float VehicleData::**contactTol**
Maximum relative velocity between objects for collisions to be resolved as contacts. Velocities greater than this are handled as collisions.

ParticleEmitterData VehicleData::**damageEmitter**[3]
Array of particle emitters used to generate damage (dust, smoke etc) effects. Currently, the first two emitters (indices 0 and 1) are used when the damage level exceeds the associated damageLevelTolerance. The 3rd emitter is used when the emitter point is underwater.

Point3F VehicleData::**damageEmitterOffset**[2]
Object space "x y z" offsets used to emit particles for the active damageEmitter.

Example:

```
// damage levelsdamageLevelTolerance[0] = 0.5;
damageEmitter[0] = SmokeEmitter;
// emit offsets (used for all active damage level emitters)
damageEmitterOffset[0] = "0.5 3 1";
damageEmitterOffset[1] = "-0.5 3 1";
numDmgEmitterAreas = 2;
```

float VehicleData::**damageLevelTolerance**[2]
Damage levels (as a percentage of maxDamage) above which to begin emitting particles from the associated damageEmitter. Levels should be in order of increasing damage.

ParticleEmitterData VehicleData::**dustEmitter**
Dust particle emitter.

float VehicleData::**dustHeight**
Height above ground at which to emit particles from the dustEmitter.

SFXProfile VehicleData::**exitingWater**
Sound to play when exiting the water.

float VehicleData::**exitSplashSoundVelocity**
Minimum velocity when leaving the water for the exitingWater sound to play.

SFXProfile VehicleData::**hardImpactSound**
Sound to play on a 'hard' impact. This sound is played if the impact speed gt = hardImpactSpeed.

float VehicleData::**hardImpactSpeed**
Minimum collision speed for the hardImpactSound to be played.

float VehicleData::**hardSplashSoundVelocity**
Minimum velocity when entering the water for the imapactWaterHard sound to play.

SFXProfile VehicleData::**impactWaterEasy**
Sound to play when entering the water with speed gt = softSplashSoundVelocity and lt mediumSplashSoundVelocity.

SFXProfile VehicleData::**impactWaterHard**
Sound to play when entering the water with speed gt = hardSplashSoundVelocity.

SFXProfile VehicleData::**impactWaterMedium**
Sound to play when entering the water with speed gt = mediumSplashSoundVelocity and lt hardSplashSoundVelocity.

int VehicleData::**integration**
Number of integration steps per tick. Increase this to improve simulation stability (also increases simulation processing time).

float VehicleData::**jetEnergyDrain**
Energy amount to drain for each tick the vehicle is jetting. Once the vehicle's energy level reaches 0, it will no longer be able to jet.

float VehicleData::**jetForce**
Additional force applied to the vehicle when it is jetting. For WheeledVehicles, the force is applied in the forward direction. For FlyingVehicles, the force is applied in the thrust direction.

Point3F VehicleData::**massBox**
Define the box used to estimate the vehicle's moment of inertia. Currently only used by WheeledVehicle ; other vehicle types use a unit sphere to compute inertia.

Point3F VehicleData::**massCenter**
Defines the vehicle's center of mass (offset from the origin of the model).

float VehicleData::**maxDrag**
Maximum drag coefficient. Currently unused.

float VehicleData::**maxSteeringAngle**
Maximum yaw (horizontal) and pitch (vertical) steering angle in radians.

float VehicleData::**mediumSplashSoundVelocity**
Minimum velocity when entering the water for the imapactWaterMedium sound to play.

float VehicleData::**minDrag**
Minimum drag coefficient. Currently only used by FlyingVehicle .

float VehicleData::**minImpactSpeed**
Minimum collision speed for the onImpact callback to be invoked.

float VehicleData::**minJetEnergy**
Minimum vehicle energy level to begin jetting.

float VehicleData::**minRollSpeed**
Unused.

float VehicleData::**numDmgEmitterAreas**
Number of damageEmitterOffset values to use for each damageEmitter.

bool VehicleData::**powerSteering**
If true, steering does not auto-centre while the vehicle is being steered by its driver.

SFXProfile VehicleData::**softImpactSound**
Sound to play on a 'soft' impact. This sound is played if the impact speed is lt hardImpactSpeed and gt = softImpactSpeed.

float VehicleData::**softImpactSpeed**
Minimum collision speed for the softImpactSound to be played.

float VehicleData::**softSplashSoundVelocity**
Minimum velocity when entering the water for the imapactWaterEasy sound to play.

ParticleEmitterData VehicleData::**splashEmitter**[2]
Array of particle emitters used to generate splash effects.

float VehicleData::**splashFreqMod**
Number of splash particles to generate based on vehicle speed. This value is multiplied by the current speed to determine how many particles to generate each frame.

float VehicleData::**splashVelEpsilon**
Minimum speed when moving through water to generate splash particles.

float VehicleData::**steeringReturn**
Rate at which the vehicle's steering returns to forwards when it is moving.

float VehicleData::**steeringReturnSpeedScale**
Amount of effect the vehicle's speed has on its rate of steering return.

float VehicleData::**triggerDustHeight**
Maximum height above surface to emit dust particles. If the vehicle is less than triggerDustHeight above a static surface with a material that has 'showDust' set to true, the vehicle will emit particles from the dustEmitter.

SFXProfile VehicleData::**waterWakeSound**
Looping sound to play while moving through the water.

WheeledVehicle A wheeled vehicle.

Inherit: Vehicle

Description A multi-wheeled vehicle.

The model used for the WheeledVehicle should contain the elements shown below. Only the collision mesh and hub nodes are actually required for the object to be added to the simulation, but the suspension will look strange if the spring animations are not present.

Collision mesh A convex collision mesh at detail size -1.

Hub nodes The model must contain a node for each wheel called hubN, where N is a an integer value starting from 0. For example, a four wheeled vehicle would have nodes: hub0, hub1, hub2, and hub3. The wheel model (specified by WheeledVehicleTire) is positioned at the hub node, and automatically rotated to the right orientation (whether on the left or right side of the vehicle).

Spring animations To visualise the suspension action, the vehicle model should contain a non-cyclic animation sequence for each wheel that animates the appropriate hub node from t=0 (fully compressed to t=1 (fully extended). The sequences must be called springN, where N matches the wheel hub index.

Steering animation Optional non-cyclic animation called 'steering' that animates from t=0 (full right) to t=0.5 (center) to t=1 (full left)."

Brakelight animation Optional non-cyclic animation called 'brakeLight' that animates from t=0 (off) to t=1 (braking). This is usually a 2-frame animation controlling the visibility of a quad or mesh to represent each brake light.

The example below shows the datablocks required for a simple 4-wheeled vehicle. The script should be executed on the server, and the vehicle can then be added to the simulation programmatically from the level startup scripts, or by selecting the MyCar datablock from the World Editor (Library->ScriptedObjects->Vehicles).

Example:

```
datablock WheeledVehicleTire( MyCarTire )
{
   shapeFile = "art/shapes/wheel.dts";
   staticFriction = 4.2;
   kineticFriction = 1.0;

   lateralForce = 18000;
   lateralDamping = 6000;
   lateralRelaxation = 1;

   longitudinalForce = 18000;
   longitudinalDamping = 4000;
   longitudinalRelaxation = 1;
   radius = 0.61;
};

datablock WheeledVehicleSpring( MyCarSpring )
{
   length = 0.5;
   force = 2800;
   damping = 3600;
   antiSwayForce = 3;
};

datablock WheeledVehicleData( MyCar )
{
   category = "Vehicles";
   shapeFile = "art/shapes/car.dts";

   maxSteeringAngle = 0.585;

   // 3rd person camera settings
   cameraRoll = false;
   cameraMaxDist = 7.8;
   cameraOffset = 1.0;
   cameraLag = 0.3;
   cameraDecay = 1.25;

   useEyePoint = true;

   // Rigid Body
   mass = "400";
   massCenter = "0 -0.2 0";
```

```
   massBox = "0 0 0";

   drag = 0.6;
   bodyFriction = 0.6;
   bodyRestitution = 0.4;
   minImpactSpeed = 5;
   softImpactSpeed = 5;
   hardImpactSpeed = 15;
   integration = 8;
   collisionTol = 0.05;
   contactTol = 0.4;

   // Engine
   engineTorque = 4300;
   engineBrake = 5000;
   brakeTorque = 10000;
   maxWheelSpeed = 50;

   // Energy
   maxEnergy = 100;
   jetForce = 3000;
   minJetEnergy = 30;
   jetEnergyDrain = 2;

   // Sounds
   engineSound = CarEngineSnd;
   squealSound = CarSquealSnd;
   softImpactSound = SoftImpactSnd;
   hardImpactSound = HardImpactSnd;

   // Particles
   tireEmitter = "CarTireEmitter";
   dustEmitter = "CarTireEmitter";
   dustHeight = "1";
};

// This function is executed when the WheeledVehicle object is added to the// simulation.
function MyCar::onAdd( %this, %obj )
{
   Parent::onAdd( %this, %obj );

   // Setup the car with some tires & springsfor ( %i = %obj.getWheelCount() - 1; %i >= 0; %i-- )
   {
      %obj.setWheelTire( %i,MyCarTire );
      %obj.setWheelSpring( %i, MyCarSpring );
      %obj.setWheelPowered( %i, true );
   }

   // Steer with the front tires only
   %obj.setWheelSteering( 0, 1 );
   %obj.setWheelSteering( 1, 1 );
}
```

Methods

int `WheeledVehicle::`**`getWheelCount`**`()`

Get the number of wheels on this vehicle.

Returns the number of wheels (equal to the number of hub nodes defined in the model)

bool `WheeledVehicle::setWheelPowered`(int *wheel*, bool *powered*)

Set whether the wheel is powered (has torque applied from the engine). A rear wheel drive car for example would set the front wheels to false, and the rear wheels to true.

Parameters

- **wheel** – index of the wheel to set (hub node #)
- **powered** – flag indicating whether to power the wheel or not

Returns true if successful, false if failed

bool `WheeledVehicle::setWheelSpring`(int *wheel*, WheeledVehicleSpring *spring*)

Set the WheeledVehicleSpring datablock for this wheel.

Parameters

- **wheel** – index of the wheel to set (hub node #)
- **spring** – WheeledVehicleSpring datablock

Returns true if successful, false if failed

Example:

```
%obj.setWheelSpring( 0, FrontSpring );
```

bool `WheeledVehicle::setWheelSteering`(int *wheel*, float *steering*)

Set how much the wheel is affected by steering. The steering factor controls how much the wheel is rotated by the vehicle steering. For example, most cars would have their front wheels set to 1.0, and their rear wheels set to 0 since only the front wheels should turn. Negative values will turn the wheel in the opposite direction to the steering angle.

Parameters

- **wheel** – index of the wheel to set (hub node #)
- **steering** – steering factor from -1 (full inverse) to 1 (full)

Returns true if successful, false if failed

bool `WheeledVehicle::setWheelTire`(int *wheel*, WheeledVehicleTire *tire*)

Set the WheeledVehicleTire datablock for this wheel.

Parameters

- **wheel** – index of the wheel to set (hub node #)
- **tire** – WheeledVehicleTire datablock

Returns true if successful, false if failed

Example:

```
%obj.setWheelTire( 0, FrontTire );
```

WheeledVehicleData Defines the properties of a WheeledVehicle.

Inherit: VehicleData

Description Defines the properties of a WheeledVehicle.

Fields

float `WheeledVehicleData::brakeTorque`
Torque applied when braking. This controls how fast the vehicle will stop when the brakes are applied.

float `WheeledVehicleData::engineBrake`
Braking torque applied by the engine when the throttle and brake are both 0. This controls how quickly the vehicle will coast to a stop.

SFXTrack `WheeledVehicleData::engineSound`
Looping engine sound. The pitch is dynamically adjusted based on the current engine RPM

float `WheeledVehicleData::engineTorque`
Torque available from the engine at 100% throttle. This controls vehicle acceleration. ie. how fast it will reach maximum speed.

SFXTrack `WheeledVehicleData::jetSound`
Looping sound played when the vehicle is jetting.

float `WheeledVehicleData::maxWheelSpeed`
Maximum linear velocity of each wheel. This caps the maximum speed of the vehicle.

SFXTrack `WheeledVehicleData::squealSound`
Looping sound played while any of the wheels is slipping. The volume is dynamically adjusted based on how much the wheels are slipping.

ParticleEmitterData `WheeledVehicleData::tireEmitter`
ParticleEmitterData datablock used to generate particles from each wheel when the vehicle is moving and the wheel is in contact with the ground.

SFXTrack `WheeledVehicleData::WheelImpactSound`
Sound played when the wheels impact the ground. Currently unused.

WheeledVehicleSpring Defines the properties of a WheeledVehicle spring.

Inherit: SimDataBlock

Description Defines the properties of a WheeledVehicle spring.

Fields

float `WheeledVehicleSpring::antiSwayForce`
Force applied to equalize extension of the spring on the opposite wheel. This force helps to keep the suspension balanced when opposite wheels are at different heights.

float `WheeledVehicleSpring::damping`
Force applied to slow changes to the extension of this spring. Increasing this makes the suspension stiffer which can help stabilise bouncy vehicles.

float `WheeledVehicleSpring::force`
Maximum spring force (when compressed to minimum length, 0). Increasing this will make the vehicle suspension ride higher (for a given vehicle mass), and also make the vehicle more bouncy when landing jumps.

float `WheeledVehicleSpring::length`
Maximum spring length. ie. how far the wheel can extend from the root hub position. This should be set to the vertical (Z) distance the hub travels in the associated spring animation.

WheeledVehicleTire Defines the properties of a WheeledVehicle tire.

Inherit: SimDataBlock

Description Tires act as springs and generate lateral and longitudinal forces to move the vehicle. These distortion/spring forces are what convert wheel angular velocity into forces that act on the rigid body.

Fields

float `WheeledVehicleTire::kineticFriction`
Tire friction when the wheel is slipping (no traction).

float `WheeledVehicleTire::lateralDamping`
Damping force applied against lateral forces generated by the tire.

float `WheeledVehicleTire::lateralForce`
Tire force perpendicular to the direction of movement. Lateral force can in simple terms be considered left/right steering force. WheeledVehicles are acted upon by forces generated by their tires and the lateralForce measures the magnitude of the force exerted on the vehicle when the tires are deformed along the x-axis. With real wheeled vehicles, tires are constantly being deformed and it is the interplay of deformation forces which determines how a vehicle moves. In Torque's simulation of vehicle physics, tire deformation obviously can't be handled with absolute realism, but the interplay of a vehicle's velocity, its engine's torque and braking forces, and its wheels' friction, lateral deformation, lateralDamping, lateralRelaxation, longitudinal deformation, longitudinalDamping, and longitudinalRelaxation forces, along with its wheels' angular velocity are combined to create a robust real-time physical simulation. For this field, the larger the value supplied for the lateralForce, the larger the effect steering maneuvers can have. In Torque tire forces are applied at a vehicle's wheel hubs.

float `WheeledVehicleTire::lateralRelaxation`
Relaxing force applied against lateral forces generated by the tire. The lateralRelaxation force measures how strongly the tire effectively un-deforms.

float `WheeledVehicleTire::longitudinalDamping`
Damping force applied against longitudinal forces generated by the tire.

float `WheeledVehicleTire::longitudinalForce`
Tire force in the direction of movement. Longitudinal force can in simple terms be considered forward/backward movement force. WheeledVehicles are acted upon by forces generated by their tires and the longitudinalForce measures the magnitude of the force exerted on the vehicle when the tires are deformed along the y-axis. For this field, the larger the value, the larger the effect acceleration/deceleration inputs have.

float `WheeledVehicleTire::longitudinalRelaxation`
Relaxing force applied against longitudinal forces generated by the tire. The longitudinalRelaxation force measures how strongly the tire effectively un-deforms.

float `WheeledVehicleTire::mass`
The mass of the wheel. Currently unused.

float `WheeledVehicleTire::radius`
The radius of the wheel. The radius is determined from the bounding box of the shape provided in the shapefile field, and does not need to be specified in script. The tire should be built with its hub axis along the object's Y-axis.

float `WheeledVehicleTire::restitution`
Tire restitution. Currently unused.

filename `WheeledVehicleTire::shapeFile`
The path to the shape to use for the wheel.

float `WheeledVehicleTire::staticFriction`
Tire friction when the wheel is not slipping (has traction).

5.3.4 Environment

Objects that represent environmental features, such as, terrain, water, atmosphere, plants and trees.

Atmosphere

Objects that represent the atmosphere and weather, such as the sky, sun, clouds, and precipitation.

Classes

BasicClouds Renders up to three layers of scrolling cloud-cover textures overhead.

Inherit: SceneObject

Description BasicClouds always renders overhead, following the camera. It is intended as part of the background of your level, rendering in front of Sky/Sun type objects and behind everything else.

The parameters controlling the rendering of each texture are refered to and grouped as 'layers'. They are rendered in sequential order, so, layer 1 obscures layer 0, and so on.

BasicClouds is not affected by scene lighting and is therefore not appropriate for scenes in which lighting radically changes, such as day/night.

Fields

float `BasicClouds::height`[3]
: Abstract number which controls the curvature and height of the dome mesh.

bool `BasicClouds::layerEnabled`[3]
: Enable or disable rendering of this layer.

Point2F `BasicClouds::texDirection`[3]
: Texture scroll direction for this layer, relative to the world axis.

Point2F `BasicClouds::texOffset`[3]
: UV offset for this layer.

float `BasicClouds::texScale`[3]
: Texture repeat for this layer.

float `BasicClouds::texSpeed`[3]
: Texture scroll speed for this layer.

filename `BasicClouds::texture`[3]
: Texture for this layer.

CloudLayer A layer of clouds which change shape over time and are affected by scene lighting.

Inherit: SceneObject

Description CloudLayer always renders overhead, following the camera. It is intended as part of the background of your level, rendering in front of Sky/Sun type objects and behind everything else.

The illusion of clouds forming and changing over time is controlled by the normal/opacity texture and the three sets of texture animation parameters. The texture is sampled three times. The first sample defines overall cloud density, where clouds are likely to form and their general size and shape. The second two samples control how it changes over time; they are combined and used as modifiers to the first sample.

CloudLayer is affected by scene lighting and is designed to be used in scenes with dynamic lighting or time of day changes.

Fields

ColorF `CloudLayer::`**`baseColor`**
Base cloud color before lighting.

float `CloudLayer::`**`coverage`**
Fraction of sky covered by clouds 0-1.

float `CloudLayer::`**`exposure`**
Brightness scale so CloudLayer can be overblown if desired.

float `CloudLayer::`**`height`**
Abstract number which controls the curvature and height of the dome mesh.

Point2F `CloudLayer::`**`texDirection`**[3]
Controls the direction this slot scrolls.

float `CloudLayer::`**`texScale`**[3]
Controls the texture repeat of this slot.

float `CloudLayer::`**`texSpeed`**[3]
Controls the speed this slot scrolls.

filename `CloudLayer::`**`texture`**
An RGBA texture which should contain normals and opacity (density).

float `CloudLayer::`**`windSpeed`**
Overall scalar to texture scroll speed.

ScatterSky Represents both the sun and sky for scenes with a dynamic time of day.

Inherit: SceneObject

Description Represents both the sun and sky for scenes with a dynamic time of day.

ScatterSky renders as a dome shaped mesh which is camera relative and always overhead. It is intended to be part of the background of your scene and renders before all other objects types.

ScatterSky is designed for outdoor scenes which need to transition fluidly between radically different times of day. It will respond to time changes originating from a TimeOfDay object or the elevation field can be directly adjusted.

During day, ScatterSky uses atmosphereic sunlight scattering aproximations to generate a sky gradient and sun corona. It also calculates the fog color, ambient color, and sun color, which are used for scene lighting. This is user controlled by fields within the ScatterSky group.

During night, ScatterSky supports can transition to a night sky cubemap and moon sprite. The user can control this and night time colors used for scene lighting with fields within the Night group.

A scene with a ScatterSky should not have any other sky or sun objects as it already fulfills both roles.

ScatterSky is intended to be used with CloudLayer and TimeOfDay as part of a scene with dynamic lighting. Having a ScatterSky without a changing time of day would unnecessarily give up artistic control compared and fillrate compared to a SkyBox + Sun setup.

Methods

void `ScatterSky::`**`applyChanges`**`()`
Apply a full network update of all fields to all clients.

Fields

ColorF ScatterSky::**ambientScale**
Modulates the ambient color of sunlight.

Point3F ScatterSky::**attenuationRatio**
The proportions of constant, linear, and quadratic attenuation to use for the falloff for point and spot lights.

float ScatterSky::**azimuth**
The horizontal angle of the sun measured clockwise from the positive Y world axis. This field is networked.

float ScatterSky::**brightness**
The brightness of the ScatterSky's light object.

bool ScatterSky::**castShadows**
Enables/disables shadows cast by objects due to ScatterSky light.

ColorF ScatterSky::**colorize**
Tints the sky the color specified, the alpha controls the brigthness. The brightness is multipled by the value of colorizeAmt.

float ScatterSky::**colorizeAmount**
Controls how much the the alpha component of colorize brigthens the sky. Setting to 0 returns default behavior.

filename ScatterSky::**cookie**
A custom pattern texture which is projected from the light.

float ScatterSky::**elevation**
The elevation angle of the sun above or below the horizon. This field is networked.

float ScatterSky::**exposure**
Controls the contrast of the sky and sun during daytime.

float ScatterSky::**fadeStartDistance**
Start fading shadows out at this distance. 0 = auto calculate this distance.

float ScatterSky::**flareScale**
Changes the size and intensity of the flare.

LightFlareData ScatterSky::**flareType**
Datablock for the flare produced by the ScatterSky .

ColorF ScatterSky::**fogScale**
Modulates the fog color. Note that this overrides the LevelInfo.fogColor property, so you should not use LevelInfo.fogColor if the level contains a ScatterSky object.

bool ScatterSky::**includeLightmappedGeometryInShadow**
This light should render lightmapped geometry during its shadow-map update (ignored if 'representedInLightmap' is false).

bool ScatterSky::**lastSplitTerrainOnly**
This toggles only terrain being rendered to the last split of a PSSM shadow map.

float ScatterSky::**logWeight**
The logrithmic PSSM split distance factor.

float ScatterSky::**moonAzimuth**
The horizontal angle of the moon measured clockwise from the positive Y world axis. This is not animated by time or networked.

float ScatterSky::**moonElevation**
The elevation angle of the moon above or below the horizon. This is not animated by time or networked.

bool ScatterSky::**moonEnabled**
Enable or disable rendering of the moon sprite during night.

ColorF ScatterSky::**moonLightColor**
Color of light cast by the directional light during night.

string ScatterSky::**moonMat**
Material for the moon sprite.

float ScatterSky::**moonScale**
Controls size the moon sprite renders, specified as a fractional amount of the screen height.

ColorF ScatterSky::**nightColor**
The ambient color during night. Also used for the sky color if useNightCubemap is false.

string ScatterSky::**nightCubemap**
Cubemap visible during night.

ColorF ScatterSky::**nightFogColor**
The fog color during night.

int ScatterSky::**numSplits**
The logrithmic PSSM split distance factor.

Point4F ScatterSky::**overDarkFactor**
The ESM shadow darkening factor.

float ScatterSky::**rayleighScattering**
Controls how blue the atmosphere is during the day.

bool ScatterSky::**representedInLightmap**
This light is represented in lightmaps (static light, default: false).

ColorF ScatterSky::**shadowDarkenColor**
The color that should be used to multiply-blend dynamic shadows onto lightmapped geometry (ignored if 'representedInLightmap' is false).

float ScatterSky::**shadowDistance**
The distance from the camera to extend the PSSM shadow.

float ScatterSky::**shadowSoftness**

ShadowType ScatterSky::**shadowType**
The type of shadow to use on this light.

float ScatterSky::**skyBrightness**
Global brightness and intensity applied to the sky and objects in the level.

ColorF ScatterSky::**sunScale**
Modulates the directional color of sunlight.

float ScatterSky::**sunSize**
Affects the size of the sun's disk.

int ScatterSky::**texSize**
The texture size of the shadow map.

bool ScatterSky::**useNightCubemap**
Transition to the nightCubemap during night. If false we use nightColor.

SkyBox Represents the sky with an artist-created cubemap.

Inherit: SceneObject

Description Represents the sky with an artist-created cubemap.

SkyBox is not a directional light and should be used in conjunction with a Sun object.

Fields

bool SkyBox::**drawBottom**
If false the bottom of the skybox is not rendered.

float SkyBox::**fogBandHeight**
The height (0-1) of the fog band from the horizon to the top of the SkyBox .

string SkyBox::**Material**
The name of a cubemap material for the sky box.

void SkyBox::**postApply**

Sun A global light affecting your entire scene and optionally renders a corona effect.

Inherit: SceneObject

Description A global light affecting your entire scene and optionally renders a corona effect.

Sun is both the directional and ambient light for your entire scene.

Fields

ColorF Sun::**ambient**
Color shading applied to surfaces not in direct contact with light source, such as in the shadows or interiors.

void Sun::**animate**
animate(F32 duration, F32 startAzimuth, F32 endAzimuth, F32 startElevation, F32 endElevation)

void Sun::**apply**

Point3F Sun::**attenuationRatio**
The proportions of constant, linear, and quadratic attenuation to use for the falloff for point and spot lights.

float Sun::**azimuth**
The horizontal angle of the sun measured clockwise from the positive Y world axis.

float Sun::**brightness**
Adjust the Sun's global contrast/intensity.

bool Sun::**castShadows**
Enables/disables shadows cast by objects due to Sun light.

ColorF Sun::**color**
Color shading applied to surfaces in direct contact with light source.

filename Sun::**cookie**
A custom pattern texture which is projected from the light.

bool Sun::**coronaEnabled**
Enable or disable rendering of the corona sprite.

string Sun::**coronaMaterial**
Texture for the corona sprite.

float Sun::**coronaScale**
Controls size the corona sprite renders, specified as a fractional amount of the screen height.

ColorF Sun::**coronaTint**
Modulates the corona sprite color (if coronaUseLightColor is false).

bool Sun::**coronaUseLightColor**
Modulate the corona sprite color by the color of the light (overrides coronaTint).

float Sun::**elevation**
The elevation angle of the sun above or below the horizon.

float Sun::**fadeStartDistance**
Start fading shadows out at this distance. 0 = auto calculate this distance.

float Sun::**flareScale**
Changes the size and intensity of the flare.

LightFlareData Sun::**flareType**
Datablock for the flare produced by the Sun .

bool Sun::**includeLightmappedGeometryInShadow**
This light should render lightmapped geometry during its shadow-map update (ignored if 'representedInLightmap' is false).

bool Sun::**lastSplitTerrainOnly**
This toggles only terrain being rendered to the last split of a PSSM shadow map.

float Sun::**logWeight**
The logrithmic PSSM split distance factor.

int Sun::**numSplits**
The logrithmic PSSM split distance factor.

Point4F Sun::**overDarkFactor**
The ESM shadow darkening factor.

bool Sun::**representedInLightmap**
This light is represented in lightmaps (static light, default: false).

ColorF Sun::**shadowDarkenColor**
The color that should be used to multiply-blend dynamic shadows onto lightmapped geometry (ignored if 'representedInLightmap' is false).

float Sun::**shadowDistance**
The distance from the camera to extend the PSSM shadow.

float Sun::**shadowSoftness**

ShadowType Sun::**shadowType**
The type of shadow to use on this light.

int Sun::**texSize**
The texture size of the shadow map.

Water

Objects that represent water features, from puddles to rivers and oceans.

Classes

River A water volume defined by a 3D spline.

Inherit: WaterObject

Description A water volume defined by a 3D spline.

User may control width and depth per node and overall spline shape in three dimensions.

River supports dynamic planar reflections (fullReflect) like all WaterObject classes, but keep in mind it is not necessarily a planar surface. For best visual quality a River should be less reflective the more it twists and bends. This caution only applies to Rivers with fullReflect on.

Methods

void River::**regenerate**()
Intended as a helper to developers and editor scripts. Force River to recreate its geometry.

void River::**setBatchSize**(float *meters*)
Intended as a helper to developers and editor scripts. BatchSize is not currently used.

void River::**setMaxDivisionSize**(float *meters*)
Intended as a helper to developers and editor scripts.

void River::**setMetersPerSegment**(float *meters*)
Intended as a helper to developers and editor scripts.

void River::**setNodeDepth**(int *idx*, float *meters*)
Intended as a helper to developers and editor scripts. Sets the depth in meters of a particular node.

Fields

bool River::**EditorOpen**[static]
For editor use.

float River::**FlowMagnitude**
Magnitude of the force vector applied to dynamic objects within the River .

float River::**LowLODDistance**
Segments of the river at this distance in meters or greater will render as a single unsubdivided without undulation effects.

string River::**Node**
For internal use, do not modify.

float River::**SegmentLength**
Divide the River lengthwise into segments of this length in meters. These geometric volumes are used for spacial queries like determining containment.

bool River::**showNodes**[static]
For editor use.

bool River::**showRiver**[static]
For editor use.

bool River::**showSpline**[static]
For editor use.

bool River::**showWalls**[static]
For editor use.

bool River::**showWireframe**[static]
For editor use.

float River::**SubdivideLength**
For purposes of generating the renderable geometry River segments are further subdivided such that no quad is of greater width or length than this distance in meters.

WaterBlock A block shaped water volume defined by a 3D scale and orientation.

Inherit: WaterObject

Description A block shaped water volume defined by a 3D scale and orientation.

Fields

float `WaterBlock::gridElementSize`
Spacing between vertices in the WaterBlock mesh.

float `WaterBlock::gridSize`
Duplicate of gridElementSize for backwards compatility.

WaterObject Abstract base class for representing a body of water.

Inherit: SceneObject

Description Abstract base class for representing a body of water.

WaterObject is abstract and may not be created. It defines functionality shared by its derived classes.

WaterObject exposes many fields for controlling it visual quality.

WaterObject surface rendering has the following general features:

It will, however, look significantly different depending on the LightingManager that is active. With Basic Lighting, we do not have a prepass texture to lookup per-pixel depth and therefore cannot use our rendering techniques that depend on it.

In particular, the following field groups are not used under Basic Lighting:

WaterObject also defines several fields for gameplay use and objects that support buoyancy.

Fields

ColorI `WaterObject::baseColor`
Changes color of water fog.

float `WaterObject::clarity`
Relative opacity or transparency of the water surface.

string `WaterObject::cubemap`
Cubemap used instead of reflection texture if fullReflect is off.

float `WaterObject::density`
Affects buoyancy of an object, thus affecting the Z velocity of a player (jumping, falling, etc.

float `WaterObject::depthGradientMax`
Depth in world units, the max range of the color gradient texture.

filename `WaterObject::depthGradientTex`
1D texture defining the base water color by depth

float `WaterObject::distortEndDist`
Max distance that distortion algorithm is performed. The lower, the more distorted the effect.

float `WaterObject::distortFullDepth`
Determines the scaling down of distortion in shallow water.

float `WaterObject::distortStartDist`
Determines start of distortion effect where water surface intersects the camera near plane.

bool WaterObject::**emissive**
When true the water colors don't react to changes to environment lighting.

float WaterObject::**foamAmbientLerp**

Point2F WaterObject::**foamDir**[2]

float WaterObject::**foamMaxDepth**

float WaterObject::**foamOpacity**[2]

float WaterObject::**foamRippleInfluence**

float WaterObject::**foamSpeed**[2]

filename WaterObject::**foamTex**
Diffuse texture for foam in shallow water (advanced lighting only).

Point2F WaterObject::**foamTexScale**[2]
applied to the surface.

float WaterObject::**fresnelBias**
Extent of fresnel affecting reflection fogging.

float WaterObject::**fresnelPower**
Measures intensity of affect on reflection based on fogging.

bool WaterObject::**fullReflect**
Enables dynamic reflection rendering.

string WaterObject::**liquidType**
Liquid type of WaterBlock , such as water, ocean, lava Currently only Water is defined and used.

float WaterObject::**overallFoamOpacity**

float WaterObject::**overallRippleMagnitude**
Master variable affecting entire surface.

float WaterObject::**overallWaveMagnitude**
Master variable affecting entire body of water's undulation.

float WaterObject::**reflectDetailAdjust**
scale up or down the detail level for objects rendered in a reflection

float WaterObject::**reflectivity**
Overall scalar to the reflectivity of the water surface.

int WaterObject::**reflectMaxRateMs**
Affects the sort time of reflected objects.

bool WaterObject::**reflectNormalUp**
always use z up as the reflection normal

float WaterObject::**reflectPriority**
Affects the sort order of reflected objects.

int WaterObject::**reflectTexSize**
The texture size used for reflections (square).

Point2F WaterObject::**rippleDir**[3]
Modifies the direction of ripples on the surface.

float WaterObject::**rippleMagnitude**[3]
Intensifies the vertext modification of the surface.

float WaterObject::**rippleSpeed**[3]
Modifies speed of surface ripples.

filename WaterObject::**rippleTex**
Normal map used to simulate small surface ripples.

Point2F WaterObject::**rippleTexScale**[3]
Intensifies the affect of the normal map applied to the surface.

SFXAmbience WaterObject::**soundAmbience**
Ambient sound environment when listener is submerged.

ColorF WaterObject::**specularColor**
Color used for specularity on the water surface (sun only).

float WaterObject::**specularPower**
Power used for specularity on the water surface (sun only).

ColorI WaterObject::**underwaterColor**
Changes the color shading of objects beneath the water surface.

bool WaterObject::**useOcclusionQuery**
turn off reflection rendering when occluded (delayed).

float WaterObject::**viscosity**
Affects drag force applied to an object submerged in this container.

float WaterObject::**waterFogDensity**
Intensity of underwater fogging.

float WaterObject::**waterFogDensityOffset**
Delta, or limit, applied to waterFogDensity.

Point2F WaterObject::**waveDir**[3]
Direction waves flow toward shores.

float WaterObject::**waveMagnitude**[3]
Height of water undulation.

float WaterObject::**waveSpeed**[3]
Speed of water undulation.

float WaterObject::**wetDarkening**
The refract color intensity scaled at wetDepth.

float WaterObject::**wetDepth**
The depth in world units at which full darkening will be received, giving a wet look to objects underwater.

WaterPlane Represents a large body of water stretching to the horizon in all directions.

Inherit: WaterObject

Description Represents a large body of water stretching to the horizon in all directions.

WaterPlane's position is defined only height, the z element of position, it is infinite in xy and depth. WaterPlane is designed to represent the ocean on an island scene and viewed from ground level; other uses may not be appropriate and a WaterBlock may be used.

Limitations:

Because WaterPlane cannot be projected exactly to the far-clip distance, other objects nearing this distance can have noticible artifacts as they clip through first the WaterPlane and then the far plane.

To avoid this large objects should be positioned such that they will not line up with the far-clip from vantage points the player is expected to be. In particular, your TerrainBlock should be completely contained by the far-clip distance.

Viewing WaterPlane from a high altitude with a tight far-clip distance will accentuate this limitation. WaterPlane is primarily designed to be viewed from ground level.

Fields

float `WaterPlane::gridElementSize`
Duplicate of gridElementSize for backwards compatility.

int `WaterPlane::gridSize`
Spacing between vertices in the WaterBlock mesh.

Variables

`bool $pref::Water::disableTrueReflections`
Force all water objects to use static cubemap reflections.

bool `WaterObject::wireframe`[static, inherited]
If true, will render the wireframe of the WaterObject .

Terrain

Objects that specialize in representing terrain and other collidable/walkable surfaces.

Classes

DecalRoad A strip shaped decal defined by spine nodes which clips against Terrain objects.

Inherit: SceneObject

Description A strip shaped decal defined by spine nodes which clips against Terrain objects.

DecalRoad is for representing a road or path (or other inventive things) across a TerrainBlock. It renders as a decal and is therefore only for features that do not need geometric depth.

The Material assigned to DecalRoad should tile vertically.

Methods

void `DecalRoad::postApply()`
Intended as a helper to developers and editor scripts. Force trigger an inspectPostApply. This will transmit the material and other fields (not including nodes) to client objects.

void `DecalRoad::regenerate()`
Intended as a helper to developers and editor scripts. Force DecalRoad to update it's spline and reclip geometry.

Fields

float `DecalRoad::breakAngle`
Angle in degrees - DecalRoad will subdivided the spline if its curve is greater than this threshold.

bool `DecalRoad::discardAll`[static]
For use by the Decal Editor.

bool `DecalRoad::EditorOpen`[static]
For use by the Decal Editor.

string DecalRoad::**Material**
Material used for rendering.

string DecalRoad::**Node**
Do not modify, for internal use.

int DecalRoad::**renderPriority**
DecalRoad(s) are rendered in descending renderPriority order.

bool DecalRoad::**showBatches**[static]
For use by the Decal Editor.

bool DecalRoad::**showRoad**[static]
For use by the Decal Editor.

bool DecalRoad::**showSpline**[static]
For use by the Decal Editor.

float DecalRoad::**textureLength**
The length in meters of textures mapped to the DecalRoad .

int DecalRoad::**updateDelay**[static]
For use by the Decal Editor.

bool DecalRoad::**wireframe**[static]
For use by the Decal Editor.

GroundPlane An infinite plane extending in all direction.

Inherit: SceneObject

Description An infinite plane extending in all direction.

GroundPlane is useful for setting up simple testing scenes, or it can be placed under an existing scene to keep objects from falling into 'nothing'.

GroundPlane may not be moved or rotated, it is always at the world origin.

Methods

void GroundPlane::**postApply**()
Intended as a helper to developers and editor scripts. Force trigger an inspectPostApply. This will transmit material and other fields to client objects.

Fields

string GroundPlane::**Material**
Name of Material used to render GroundPlane's surface.

float GroundPlane::**scaleU**
Scale of texture repeat in the U direction.

float GroundPlane::**scaleV**
Scale of texture repeat in the V direction.

float GroundPlane::**squareSize**
Square size in meters to which GroundPlane subdivides its geometry.

MeshRoad A strip of rectangular mesh segments defined by a 3D spline for prototyping road-shaped objects in your scene.

Inherit: SceneObject

Description A strip of rectangular mesh segments defined by a 3D spline for prototyping road-shaped objects in your scene.

User may control width and depth per node, overall spline shape in three dimensions, and seperate Materials for rendering the top, bottom, and side surfaces.

MeshRoad is not capable of handling intersections, branches, curbs, or other desirable features in a final 'road' asset and is therefore intended for prototyping and experimentation.

Materials assigned to MeshRoad should tile vertically.

Methods

void `MeshRoad::postApply()`
Intended as a helper to developers and editor scripts. Force trigger an inspectPostApply. This will transmit material and other fields (not including nodes) to client objects.

void `MeshRoad::regenerate()`
Intended as a helper to developers and editor scripts. Force MeshRoad to recreate its geometry.

void `MeshRoad::setNodeDepth` (int *idx*, float *meters*)
Intended as a helper to developers and editor scripts. Sets the depth in meters of a particular node.

Fields

string `MeshRoad::bottomMaterial`
Material for the bottom surface of the road.

float `MeshRoad::breakAngle`
Angle in degrees - MeshRoad will subdivide the spline if its curve is greater than this threshold.

bool `MeshRoad::EditorOpen`[static]
True if the MeshRoad editor is open, otherwise false.

string `MeshRoad::Node`
Do not modify, for internal use.

bool `MeshRoad::showBatches`[static]
Determines if the debug rendering of the batches cubes is displayed or not.

bool `MeshRoad::showRoad`[static]
If true, the road will be rendered. When in the editor, roads are always rendered regardless of this flag.

bool `MeshRoad::showSpline`[static]
If true, the spline on which the curvature of this road is based will be rendered.

string `MeshRoad::sideMaterial`
Material for the left, right, front, and back surfaces of the road.

float `MeshRoad::textureLength`
The length in meters of textures mapped to the MeshRoad .

string `MeshRoad::topMaterial`
Material for the upper surface of the road.

int `MeshRoad::widthSubdivisions`
Subdivide segments widthwise this many times when generating vertices.

bool MeshRoad::**wireframe**[static]
If true, will render the wireframe of the road.

TerrainBlock Represent a terrain object in a Torque 3D level.

Inherit: SceneObject

Description Represent a terrain object in a Torque 3D level.

Example:

```
newTerrainBlock(theTerrain)
{
   terrainFile = "art/terrains/Deathball Desert_0.ter";
   squareSize = "2";
   tile = "0";
   baseTexSize = "1024";
   screenError = "16";
   position = "-1024 -1024 179.978";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   isRenderEnabled = "true";
   canSaveDynamicFields = "1";
};
```

Methods

bool TerrainBlock::**exportHeightMap**(string *filename*)
export the terrain block's heightmap to a bitmap file (default: png)

bool TerrainBlock::**exportLayerMaps**(string *filePrefix*)
export the terrain block's layer maps to bitmap files (default: png)

int TerrainBlock::**import**(String *terrainName*, String *heightMap*, F32 *metersPerPixel*, F32 *heightScale*, String *materials*, String *opacityLayers*)

bool TerrainBlock::**save**(string *fileName*)
Saves the terrain block's terrain file to the specified file name.

Parameters **fileName** – Name and path of file to save terrain data to.

Returns True if file save was successful, false otherwise

Fields

int TerrainBlock::**baseTexSize**
Size of base texture size per meter.

bool TerrainBlock::**castShadows**
Allows the terrain to cast shadows onto itself and other objects.

int TerrainBlock::**createNew**
TerrainBlock.create(String terrainName, U32 resolution, String materialName, bool genNoise).

int TerrainBlock::**lightMapSize**
Light map dimensions in pixels.

int TerrainBlock::**screenError**
Not yet implemented.

float TerrainBlock::**squareSize**
Indicates the spacing between points on the XY plane on the terrain.

filename TerrainBlock::**terrainFile**
The source terrain data file.

Functions

bool **getTerrainHeight**(Point2I *position*)
Gets the terrain height at the specified position.

Parameters **position** – The world space point, minus the z (height) value. Formatted as ("x y")

Returns Returns the terrain height at the given point as an F32 value.

bool **getTerrainHeight**(F32 *x*, F32 *y*)
Gets the terrain height at the specified position.

Parameters

- **x** – The X coordinate in world space
- **y** – The Y coordinate in world space

Returns Returns the terrain height at the given point as an F32 value.

bool **getTerrainHeightBelowPosition**(Point2I *position*)
Takes a world point and find the "highest" terrain underneath it.

Parameters **position** – The world space point, minus the z (height) value. Formatted as ("x y")

Returns Returns the closest terrain height below the given point as an F32 value.

bool **getTerrainHeightBelowPosition**(F32 *x*, F32 *y*)
Takes a world point and find the "highest" terrain underneath it.

Parameters

- **x** – The X coordinate in world space
- **y** – The Y coordinate in world space

Returns Returns the closest terrain height below the given point as an F32 value.

bool **getTerrainUnderWorldPoint**(Point3F *position*)
Gets the terrain block that is located under the given world point.

Parameters **position** – The world space coordinate you wish to query at. Formatted as ("x y z")

Returns Returns the ID of the requested terrain block (0 if not found).

bool **getTerrainUnderWorldPoint**(F32 *x*, F32 *y*, F32 *z*)
Takes a world point and find the "highest" terrain underneath it.

Parameters

- **x** – The X coordinate in world space
- **y** – The Y coordinate in world space
- **z** – The Z coordinate in world space

Returns Returns the ID of the requested terrain block (0 if not found).

Variables

bool TerrainBlock::**debugRender**[static, inherited]
Triggers debug rendering of terrain cells.

float $pref::Terrain::detailScale
A global detail scale used to tweak the material detail distances.

float $pref::Terrain::lodScale
A global LOD scale used to tweak the default terrain screen error value.

Forest

Objects for efficiently placing and rendering trees, rocks, foliage, or any such feature needed in large number.

Classes

Forest Forest is a global-bounds scene object provides collision and rendering for a (.forest) data file.

Inherit: SceneObject

Description Forest is a global-bounds scene object provides collision and rendering for a (.forest) data file.

Forest is designed to efficiently render a large number of static meshes: trees, rocks plants, etc. These cannot be moved at game-time or play animations but do support wind effects using vertex shader transformations guided by vertex color in the asset and user placed wind emitters (or weapon explosions).

Script level manipulation of forest data is not possible through Forest, it is only the rendering/collision. All editing is done through the world editor.

Methods
void Forest::**clear**()
bool Forest::**isDirty**()

void Forest::**regenCells**()

Fields
filename Forest::**dataFile**
The source forest data file.
float Forest::**lodReflectScalar**
Scalar applied to the farclip distance when Forest renders into a reflection.

bool Forest::**saveDataFile**
saveDataFile([path])

ForestBrushElement Represents a type of ForestItem and parameters for how it is placed when painting with a ForestBrush that contains it.

Inherit: SimObject

Description Represents a type of ForestItem and parameters for how it is placed when painting with a ForestBrush that contains it.

Fields

float `ForestBrushElement::`**`elevationMax`**
The max world space elevation this item will be placed.

float `ForestBrushElement::`**`elevationMin`**
The min world space elevation this item will be placed.

ForestItemData `ForestBrushElement::`**`ForestItemData`**
The type of ForestItem this element holds placement parameters for.

float `ForestBrushElement::`**`probability`**
The probability that this element will be created during an editor brush stroke is the sum of all element probabilities in the brush divided by the probability of this element.

float `ForestBrushElement::`**`rotationRange`**
The max rotation in degrees that items will be placed.

float `ForestBrushElement::`**`scaleExponent`**
An exponent used to bias between the minimum and maximum random sizes.

float `ForestBrushElement::`**`scaleMax`**
The maximum random size of each item.

float `ForestBrushElement::`**`scaleMin`**
The minimum random size for each item.

float `ForestBrushElement::`**`sinkMax`**
Max variation in the sink radius.

float `ForestBrushElement::`**`sinkMin`**
Min variation in the sink radius.

float `ForestBrushElement::`**`sinkRadius`**
This is the radius used to calculate how much to sink the trunk at its base and is used to sink the tree into the ground when its on a slope.

float `ForestBrushElement::`**`slopeMax`**
The max surface slope in degrees this item will be placed on.

float `ForestBrushElement::`**`slopeMin`**
The min surface slope in degrees this item will be placed on.

ForestItemData Base class for defining a type of ForestItem. It does not implement loading or rendering of the shapeFile.

Inherit: SimDataBlock

Description Base class for defining a type of ForestItem. It does not implement loading or rendering of the shapeFile.

Fields

float `ForestItemData::`**`branchAmp`**
Amplitude of the effect on larger branches.

bool `ForestItemData::`**`collidable`**
Can other objects or spacial queries hit items of this type.

float `ForestItemData::`**`dampingCoefficient`**
Coefficient used in calculating spring forces on the trunk. Causes oscillation and forces to decay faster over time.

float ForestItemData::**detailAmp**
Amplitude of the winds effect on leafs/fronds.

float ForestItemData::**detailFreq**
Frequency (speed) of the effect on leafs/fronds.

float ForestItemData::**mass**
Mass used in calculating spring forces on the trunk. Generally how springy a plant is.

float ForestItemData::**radius**
Radius used during placement to ensure items are not crowded.

float ForestItemData::**rigidity**
Rigidity used in calculating spring forces on the trunk. How much the plant resists the wind force.

filename ForestItemData::**shapeFile**
Shape file for this item type.

float ForestItemData::**tightnessCoefficient**
Coefficient used in calculating spring forces on the trunk. How much the plant resists bending.

float ForestItemData::**trunkBendScale**
Overall bend amount of the tree trunk by wind and impacts.

float ForestItemData::**windScale**
Overall scale to the effect of wind.

TSForestItemData Concrete implementation of ForestItemData which loads and renders dts format shapeFiles.

Inherit: ForestItemData

Variables

bool Forest::**disableImposters**[static, inherited]
A debugging aid which will disable rendering of all imposters in the forest.

bool Forest::**drawBounds**[static, inherited]
A debugging aid which renders the forest bounds.

bool Forest::**drawCells**[static, inherited]
A debugging aid which renders the forest cell bounds.

bool Forest::**forceImposters**[static, inherited]
A debugging aid which will force all forest items to be rendered as imposters.

Foliage

Objects used for mass replication of foliage, such as grass, rocks, and bushes.

Classes

fxFoliageReplicator An emitter to replicate fxFoliageItem objects across an area.

Inherit: SceneObject

Description An emitter to replicate fxFoliageItem objects across an area.

Fields

int fxFoliageReplicator::**AllowedTerrainSlope**
Maximum surface angle allowed for foliage instances.

bool fxFoliageReplicator::**AllowOnStatics**
Foliage will be placed on Static shapes when set.

bool fxFoliageReplicator::**AllowOnTerrain**
Foliage will be placed on terrain when set.

bool fxFoliageReplicator::**AllowOnWater**
Foliage will be placed on/under water when set.

bool fxFoliageReplicator::**AllowWaterSurface**
Foliage will be placed on water when set. Requires AllowOnWater.

float fxFoliageReplicator::**AlphaCutoff**
Minimum alpha value allowed on foliage instances.

int fxFoliageReplicator::**CullResolution**
Minimum size of culling bins. Must be gt = 8 and lt = OuterRadius.

float fxFoliageReplicator::**DebugBoxHeight**
Height multiplier for drawn culling bins.

float fxFoliageReplicator::**FadeInRegion**
Region beyond ViewDistance where foliage fades in/out.

float fxFoliageReplicator::**FadeOutRegion**
Region before ViewClosest where foliage fades in/out.

bool fxFoliageReplicator::**FixAspectRatio**
Maintain aspect ratio of image if true. This option ignores MaxWidth.

bool fxFoliageReplicator::**FixSizeToMax**
Use only MaxWidth and MaxHeight for billboard size. Ignores MinWidth and MinHeight.

int fxFoliageReplicator::**FoliageCount**
Maximum foliage instance count.

filename fxFoliageReplicator::**FoliageFile**
Image file for the foliage texture.

int fxFoliageReplicator::**FoliageRetries**
Number of times to try placing a foliage instance before giving up.

float fxFoliageReplicator::**GroundAlpha**
Alpha of the foliage at ground level. 0 = transparent, 1 = opaque.

bool fxFoliageReplicator::**HideFoliage**
Foliage is hidden when set to true.

int fxFoliageReplicator::**InnerRadiusX**
Placement area inner radius on the X axis.

int fxFoliageReplicator::**InnerRadiusY**
Placement area inner radius on the Y axis.

bool fxFoliageReplicator::**LightOn**
Foliage should be illuminated with changing lights when true.

bool fxFoliageReplicator::**LightSync**
Foliage instances have the same lighting when set and LightOn is set.

float fxFoliageReplicator::**lightTime**
Time before foliage illumination cycle repeats.

float fxFoliageReplicator::**MaxHeight**
Maximum height of foliage billboards.

float fxFoliageReplicator::**MaxLuminance**
Maximum luminance for foliage instances.

float fxFoliageReplicator::**MaxSwayTime**
Maximum sway cycle time in seconds.

float fxFoliageReplicator::**MaxWidth**
Maximum width of foliage billboards.

float fxFoliageReplicator::**MinHeight**
Minimum height of foliage billboards.

float fxFoliageReplicator::**MinLuminance**
Minimum luminance for foliage instances.

float fxFoliageReplicator::**MinSwayTime**
Minumum sway cycle time in seconds.

float fxFoliageReplicator::**MinWidth**
Minimum width of foliage billboards.

float fxFoliageReplicator::**OffsetZ**
Offset billboards by this amount vertically.

int fxFoliageReplicator::**OuterRadiusX**
Placement area outer radius on the X axis.

int fxFoliageReplicator::**OuterRadiusY**
Placement area outer radius on the Y axis.

int fxFoliageReplicator::**PlacementAreaHeight**
Height of the placement ring in world units.

ColorF fxFoliageReplicator::**PlacementColour**
Color of the placement ring.

bool fxFoliageReplicator::**RandomFlip**
Randomly flip billboards left-to-right.

int fxFoliageReplicator::**seed**
Random seed for foliage placement.

bool fxFoliageReplicator::**ShowPlacementArea**
Draw placement rings when set to true.

float fxFoliageReplicator::**SwayMagFront**
Front-to-back sway magnitude.

float fxFoliageReplicator::**SwayMagSide**
Left-to-right sway magnitude.

bool fxFoliageReplicator::**SwayOn**
Foliage should sway randomly when true.

bool fxFoliageReplicator::**SwaySync**
Foliage instances should sway together when true and SwayOn is enabled.

bool fxFoliageReplicator::**UseCulling**
Use culling bins when enabled.

bool fxFoliageReplicator::**UseDebugInfo**
Culling bins are drawn when set to true.

bool fxFoliageReplicator::**useTrueBillboards**
Use camera facing billboards (including the z axis).

float fxFoliageReplicator::**ViewClosest**
Minimum distance from camera where foliage appears.

float fxFoliageReplicator::**ViewDistance**
Maximum distance from camera where foliage appears.

fxShapeReplicatedStatic The object definition for shapes that will be replicated across an area using an fxShapeReplicator.

Inherit: SceneObject

Description The object definition for shapes that will be replicated across an area using an fxShapeReplicator.

Fields

bool fxShapeReplicatedStatic::**allowPlayerStep**
Allow a Player to walk up sloping polygons in the TSStatic (based on the collisionType). When set to false, the slightest bump will stop the player from walking on top of the object.

TSMeshType fxShapeReplicatedStatic::**collisionType**
The type of mesh data to use for collision queries.

TSMeshType fxShapeReplicatedStatic::**decalType**
The type of mesh data used to clip decal polygons against.

int fxShapeReplicatedStatic::**forceDetail**
Forces rendering to a particular detail level.

bool fxShapeReplicatedStatic::**meshCulling**
Enables detailed culling of meshes within the TSStatic . Should only be used with large complex shapes like buildings which contain many submeshes.

bool fxShapeReplicatedStatic::**originSort**
Enables translucent sorting of the TSStatic by its origin instead of the bounds.

bool fxShapeReplicatedStatic::**playAmbient**
Enables automatic playing of the animation sequence named “ambient” (if it exists) when the TSStatic is loaded.

float fxShapeReplicatedStatic::**renderNormals**
Debug rendering mode shows the normals for each point in the TSStatic’s mesh.

filename fxShapeReplicatedStatic::**shapeName**
Path and filename of the model file (.DTS, .DAE) to use for this TSStatic .

string fxShapeReplicatedStatic::**skin**
The skin applied to the shape. ‘Skinning’ the shape effectively renames the material targets, allowing different materials to be used on different instances of the same model. Any material targets that start with the old skin name have that part of the name replaced with the new skin name. The initial old skin name is “base”. For example, if a new skin of “blue” was applied to a model that had material targets base_body and face , the new targets would be blue_body and face . Note that face was not renamed since it did not start with the old skin name of “base”. To support models that do not use the default “base” naming convention, you can also specify

the part of the name to replace in the skin field itself. For example, if a model had a material target called shapemat , we could apply a new skin "shape=blue", and the material target would be renamed to bluemat (note "shape" has been replaced with "blue"). Multiple skin updates can also be applied at the same time by separating them with a semicolon. For example: "base=blue;face=happy_face". Material targets are only renamed if an existing Material maps to that name, or if there is a diffuse texture in the model folder with the same name as the new target.

fxShapeReplicator An emitter for objects to replicate across an area.

Inherit: SceneObject

Description An emitter for objects to replicate across an area.

Fields

bool `fxShapeReplicator::`**`AlignToTerrain`**
Align shapes to surface normal when set.

int `fxShapeReplicator::`**`AllowedTerrainSlope`**
Maximum surface angle allowed for shape instances.

bool `fxShapeReplicator::`**`AllowOnStatics`**
Shapes will be placed on Static shapes when set.

bool `fxShapeReplicator::`**`AllowOnTerrain`**
Shapes will be placed on terrain when set.

bool `fxShapeReplicator::`**`AllowOnWater`**
Shapes will be placed on/under water when set.

bool `fxShapeReplicator::`**`AllowWaterSurface`**
Shapes will be placed on water when set. Requires AllowOnWater.

bool `fxShapeReplicator::`**`HideReplications`**
Replicated shapes are hidden when set to true.

int `fxShapeReplicator::`**`InnerRadiusX`**
Placement area inner radius on the X axis.

int `fxShapeReplicator::`**`InnerRadiusY`**
Placement area inner radius on the Y axis.

bool `fxShapeReplicator::`**`Interactions`**
Allow physics interactions with shapes.

int `fxShapeReplicator::`**`OffsetZ`**
Offset shapes by this amount vertically.

int `fxShapeReplicator::`**`OuterRadiusX`**
Placement area outer radius on the X axis.

int `fxShapeReplicator::`**`OuterRadiusY`**
Placement area outer radius on the Y axis.

int `fxShapeReplicator::`**`PlacementAreaHeight`**
Height of the placement ring in world units.

ColorF `fxShapeReplicator::`**`PlacementColour`**
Color of the placement ring.

int fxShapeReplicator::**seed**
Random seed for shape placement.

int fxShapeReplicator::**ShapeCount**
Maximum shape instance count.

filename fxShapeReplicator::**shapeFile**
Filename of shape to replicate.

int fxShapeReplicator::**ShapeRetries**
Number of times to try placing a shape instance before giving up.

Point3F fxShapeReplicator::**ShapeRotateMax**
Maximum shape rotation angles.

Point3F fxShapeReplicator::**ShapeRotateMin**
Minimum shape rotation angles.

Point3F fxShapeReplicator::**ShapeScaleMax**
Maximum shape scale.

Point3F fxShapeReplicator::**ShapeScaleMin**
Minimum shape scale.

bool fxShapeReplicator::**ShowPlacementArea**
Draw placement rings when set to true.

Point3F fxShapeReplicator::**TerrainAlignment**
Surface normals will be multiplied by these values when AlignToTerrain is enabled.

GroundCover Covers the ground in a field of objects (IE: Grass, Flowers, etc).

Inherit: SceneObject

Description Covers the ground in a field of objects (IE: Grass, Flowers, etc).

Fields

RectF GroundCover::**billboardUVs**[8]
Subset material UV coordinates for this cover billboard.

float GroundCover::**clumpExponent**[8]
An exponent used to bias between the minimum and maximum clump counts for a particular clump.

float GroundCover::**clumpRadius**[8]
The maximum clump radius.

float GroundCover::**dissolveRadius**
This is less than or equal to radius and defines when fading of cover elements begins.

int GroundCover::**gridSize**
The number of cells per axis in the grid.

bool GroundCover::**invertLayer**[8]
Indicates that the terrain material index given in 'layer' is an exclusion mask.

string GroundCover::**layer**[8]
Terrain material name to limit coverage to, or blank to not limit.

bool GroundCover::**lockFrustum**
Debug parameter for locking the culling frustum which will freeze the cover generation.

string GroundCover::**Material**
Material used by all GroundCover segments.

float GroundCover::**maxBillboardTiltAngle**
The maximum amout of degrees the billboard will tilt down to match the camera.

int GroundCover::**maxClumpCount**[8]
The maximum amount of elements in a clump.

int GroundCover::**maxElements**
The maximum amount of cover elements to include in the grid at any one time.

float GroundCover::**maxElevation**[8]
The maximum world space elevation for placement.

float GroundCover::**maxSlope**[8]
The maximum slope angle in degrees for placement.

int GroundCover::**minClumpCount**[8]
The minimum amount of elements in a clump.

float GroundCover::**minElevation**[8]
The minimum world space elevation for placement.

bool GroundCover::**noBillboards**
Debug parameter for turning off billboard rendering.

bool GroundCover::**noShapes**
Debug parameter for turning off shape rendering.

float GroundCover::**probability**[8]
The probability of one cover type verses another (relative to all cover types).

float GroundCover::**radius**
Outer generation radius from the current camera position.

float GroundCover::**reflectScale**
Scales the various culling radii when rendering a reflection. Typically for water.

bool GroundCover::**renderCells**
Debug parameter for displaying the grid cells.

int GroundCover::**seed**
This RNG seed is saved and sent to clients for generating the same cover.

float GroundCover::**shapeCullRadius**
This is the distance at which DTS elements are completely culled out.

filename GroundCover::**shapeFilename**[8]
The cover shape filename. [Optional].

bool GroundCover::**shapesCastShadows**
Whether DTS elements should cast shadows or not.

float GroundCover::**sizeExponent**[8]
An exponent used to bias between the minimum and maximum random sizes.

float GroundCover::**sizeMax**[8]
The maximum random size of this cover type.

float GroundCover::**sizeMin**[8]
The minimum random size for each cover type.

Point2F GroundCover::**windDirection**
The direction of the wind.

float GroundCover::**windGustFrequency**
Controls how often the wind gust peaks per second.

float GroundCover::**windGustLength**
The length in meters between peaks in the wind gust.

float GroundCover::**windGustStrength**
The maximum distance in meters that the peak wind gust will displace an element.

float GroundCover::**windScale**[8]
The wind effect scale.

float GroundCover::**windTurbulenceFrequency**
Controls the overall rapidity of the wind turbulence.

float GroundCover::**windTurbulenceStrength**
The maximum distance in meters that the turbulence can displace a ground cover element.

float GroundCover::**zOffset**
Offset along the Z axis to render the ground cover.

Functions

void **StartClientReplication**()
Activates the shape replicator.

Example:

```
// Call the function
StartClientReplication()
```

void **StartFoliageReplication**()
Activates the foliage replicator.

Example:

```
// Call the function
StartFoliageReplication();
```

Variables

float $pref::GroundCover::densityScale
A global LOD scalar which can reduce the overall density of placed GroundCover .

int GroundCover::**renderedBatches**[static, inherited]
Stat for number of rendered billboard batches.

int GroundCover::**renderedBillboards**[static, inherited]
Stat for number of rendered billboards.

int GroundCover::**renderedCells**[static, inherited]
Stat for number of rendered cells.

int GroundCover::**renderedShapes**[static, inherited]
Stat for number of rendered shapes.

Miscellaneous

Miscellaneous environmental and level objects.

Classes

ConvexShape A renderable, collidable convex shape defined by a collection of surface planes.

Inherit: SceneObject

Description ConvexShape is intended to be used as a temporary asset for quickly blocking out a scene or filling in approximate shapes to be later replaced with final assets. This is most easily done by using the WorldEditor's Sketch Tool.

Fields

string `ConvexShape::Material`
Material used to render the ConvexShape surface.

string `ConvexShape::surface`
Do not modify, for internal use.

LevelInfo Stores and controls the rendering and status information for a game level.

Inherit: NetObject

Description Stores and controls the rendering and status information for a game level.

Example:

```
newLevelInfo(theLevelInfo)
{
  visibleDistance = "1000";
  fogColor = "0.6 0.6 0.7 1";
  fogDensity = "0";
  fogDensityOffset = "700";
  fogAtmosphereHeight = "0";
  canvasClearColor = "0 0 0 255";
  canSaveDynamicFields = "1";
  levelName = "Blank Room";
  desc0 = "A blank room ready to be populated with Torque objects.";
  Enabled = "1";
};
```

Fields

bool `LevelInfo::advancedLightmapSupport`
Enable expanded support for mixing static and dynamic lighting (more costly).

EaseF `LevelInfo::ambientLightBlendCurve`
Interpolation curve to use for blending from one ambient light color to a different one.

float `LevelInfo::ambientLightBlendPhase`
Number of seconds it takes to blend from one ambient light color to a different one.

ColorI `LevelInfo::canvasClearColor`
The color used to clear the background before the scene or any GUIs are rendered.

float LevelInfo::**decalBias**
NearPlane bias used when rendering Decal and DecalRoad . This should be tuned to the visibleDistance in your level.

float LevelInfo::**fogAtmosphereHeight**
A height in meters for altitude fog falloff.

ColorF LevelInfo::**fogColor**
The default color for the scene fog.

float LevelInfo::**fogDensity**
The 0 to 1 density value for the exponential fog falloff.

float LevelInfo::**fogDensityOffset**
An offset from the camera in meters for moving the start of the fog effect.

float LevelInfo::**nearClip**
Closest distance from the camera's position to render the world.

SFXAmbience LevelInfo::**soundAmbience**
The global ambient sound environment.

SFXDistanceModel LevelInfo::**soundDistanceModel**
The distance attenuation model to use.

float LevelInfo::**visibleDistance**
Furthest distance fromt he camera's position to render the world.

Marker A single joint, or knot, along a path.

Inherit: SceneObject

Description A single joint, or knot, along a path. Should be stored inside a Path container object. A path markers can be one of three primary movement types: "normal", "Position Only", or "Kink".

Example:

```
new path()
   {
     isLooping = "1";

     newMarker()
      {
         seqNum = "0";
         type = "Normal";
         msToNext = "1000";
         smoothingType = "Spline";
         position = "-0.054708 -35.0612 234.802";
         rotation = "1 0 0 0";
      };

   };
```

Fields

int Marker::**msToNext**
Milliseconds to next marker in sequence.

int Marker::**seqNum**
Marker position in sequence of markers on this path.

MarkerSmoothingType `Marker::smoothingType`
Path smoothing at this marker/knot. "Linear" means no smoothing, while "Spline" means to smooth.

MarkerKnotType `Marker::type`
Type of this marker/knot. A "normal" knot will have a smooth camera translation/rotation effect. "Position Only" will do the same for translations, leaving rotation un-touched. Lastly, a "Kink" means the rotation will take effect immediately for an abrupt rotation change.

MissionArea Level object which defines the boundaries of the level.

Inherit: NetObject

Description This is a simple box with starting points, width, depth, and height. It does not have any default functionality. Instead, when objects hit the boundaries certain script callbacks will be made allowing you to control the reaction.

Example:

```
newMissionArea(GlobalMissionArea)
{
     Area = "-152 -352 1008 864";
     flightCeiling = "300";
     flightCeilingRange = "20";
     canSaveDynamicFields = "1";
       enabled = "1";
       TypeBool locked = "false";
};
```

Methods

string `MissionArea::getArea`()
Returns 4 fields: starting x, starting y, extents x, extents y.

void `MissionArea::postApply`()
Intended as a helper to developers and editor scripts. Force trigger an inspectPostApply. This will transmit material and other fields (not including nodes) to client objects.

void `MissionArea::setArea`(int *x*, int *y*, int *width*, int *height*)
Defines the size of the MissionArea param x Starting X coordinate position for MissionArea param y Starting Y coordinate position for MissionArea param width New width of the MissionArea param height New height of the MissionArea

Fields

RectI `MissionArea::area`
Four corners (X1, X2, Y1, Y2) that makes up the level's boundaries.

float `MissionArea::flightCeiling`
Represents the top of the mission area, used by FlyingVehicle .

float `MissionArea::flightCeilingRange`
Distance from ceiling before FlyingVehicle thrust is cut off.

MissionMarker This is a base class for all "marker" related objets. It is a 3D representation of a point in the level.

Inherit: ShapeBase

Description The main use of a MissionMarker is to represent a point in 3D space with a mesh and basic ShapeBase information. If you simply need to mark a spot in your level, with no overhead from additional fields, this is a useful object.

Example:

```
newMissionMarker()
{
   dataBlock = "WayPointMarker";
   position = "295.699 -171.817 280.124";
   rotation = "0 0 -1 13.8204";
   scale = "1 1 1";
   isRenderEnabled = "true";
   canSaveDynamicFields = "1";
   enabled = "1";
};
```

MissionMarkerData A very basic class containing information used by MissionMarker objects for rendering.

Inherit: ShapeBaseData

Description MissionMarkerData, is an extremely barebones class derived from ShapeBaseData. It is solely used by MissionMarker classes (such as SpawnSphere), so that you can see the object while editing a level.

Example:

```
datablock MissionMarkerData(SpawnSphereMarker)
{
   category = "Misc";
   shapeFile = "core/art/shapes/octahedron.dts";
};
```

OcclusionVolume An invisible shape that causes objects hidden from view behind it to not be rendered.

Inherit: SceneObject

Description OcclusionVolume is a class for scene optimization. It's main use is for outdoor spaces where zones and portals do not help in optimizing scene culling as they almost only make sense for modeling visibility in indoor scenarios (and for connecting indoor spaces to outdoor spaces).

During rendering, every object that is fully behind an occluder

Be aware that occluders add overhead to scene culling. Only if this overhead is outweighed by the time saved by not rendering hidden objects, is the occluder actually effective. Because of this, chose only those spots for placing occluders where a significant number of objects will be culled from points that the player will actually be at during the game.

Like zones and portals, OcclusionVolumes may have a default box shape or a more complex.

Fields

string `OcclusionVolume::edge`
: For internal use only.

string `OcclusionVolume::plane`
: For internal use only.

string OcclusionVolume::**point**
For internal use only.

Path A spline along which various objects can move along.

Inherit: SimGroup

Description A spline along which various objects can move along. The spline object acts like a container for Marker objects, which make up the joints, or knots, along the path. Paths can be assigned a speed, can be looping or non-looping. Each of a path's markers can be one of three primary movement types: "normal", "Position Only", or "Kink".

Example:

```
new path()
   {
     isLooping = "1";

     newMarker()
      {
         seqNum = "0";
         type = "Normal";
         msToNext = "1000";
         smoothingType = "Spline";
         position = "-0.054708 -35.0612 234.802";
         rotation = "1 0 0 0";
      };

   };
```

Methods

int Path::**getPathId**()
Returns the PathID (not the object ID) of this path.

Returns PathID (not the object ID) of this path.

Example:

```
// Acquire the PathID of this path object.
%pathID = %thisPath.getPathId();
```

Fields

bool Path::**isLooping**
If this is true, the loop is closed, otherwise it is open.

PhysicalZone Physical Zones are areas that modify the player's gravity and/or velocity and/or applied force.

Inherit: SceneObject

Description The datablock properties determine how the physics, velocity and applied forces affect a player who enters this zone.

Example:

```
newPhysicalZone(Team1JumpPad) {
velocityMod = "1";gravityMod = "0";
appliedForce = "0 0 20000";
polyhedron = "0.0000000 0.0000000 0.0000000 1.0000000 0.0000000 0.0000000 0.0000000 -1.0000000 0.0000
position = "273.559 -166.371 249.856";
rotation = "0 0 1 13.0216";
scale = "8 4.95 28.31";
isRenderEnabled = "true";
canSaveDynamicFields = "1";
enabled = "1";
};
```

Methods

void PhysicalZone::**activate**()

Activate the physical zone's effects.

Example:

```
// Activate effects for a specific physical zone.
%thisPhysicalZone.activate();
```

void PhysicalZone::**deactivate**()

Deactivate the physical zone's effects.

Example:

```
// Deactivate effects for a specific physical zone.
%thisPhysicalZone.deactivate();
```

Fields

Point3F PhysicalZone::**appliedForce**

Three-element floating point value representing forces in three axes to apply to objects entering PhysicalZone .

float PhysicalZone::**gravityMod**

Gravity in PhysicalZone . Multiplies against standard gravity.

floatList PhysicalZone::**polyhedron**

The polyhedron type is really a quadrilateral and consists of a cornerpoint followed by three vectors representing the edges extending from the corner.

bool PhysicalZone::**renderZones**[static]

If true, a box will render around the location of all PhysicalZones.

float PhysicalZone::**velocityMod**

Multiply velocity of objects entering zone by this value every tick.

Portal An object that provides a "window" into a zone, allowing a viewer to see what's rendered in the zone.

Inherit: Zone

Description A portal is an object that connects zones such that the content of one zone becomes visible in the other when looking through the portal.

Each portal is a full zone which is divided into two sides by the portal plane that intersects it. This intersection polygon is shown in red in the editor. Either of the sides of a portal can be connected to one or more zones.

A connection from a specific portal side to a zone is made in either of two ways:

1. By moving a Zone object to intersect with the portal at the respective side. While usually it makes sense for this overlap to be small, the connection is established correctly as long as the center of the Zone object that should connect is on the correct side of the portal plane.
2. By the respective side of the portal free of Zone objects that would connect to it. In this case, given that the other side is connected to one or more Zones, the portal will automatically connect itself to the outdoor "zone" which implicitly is present in any level.

From this, it follows that there are two types of portals:

Exterior Portals An exterior portal is one that is connected to one or more Zone objects on one side and to the outdoor zone at the other side. This kind of portal is most useful for covering transitions from outdoor spaces to indoor spaces.

Interior Portals An interior portal is one that is connected to one or more Zone objects on both sides. This kind of portal is most useful for covering transitions between indoor spaces

Strictly speaking, there is a third type of portal called an "invalid portal". This is a portal that is not connected to a Zone object on either side in which case the portal serves no use.

Portals in Torque are bidirectional meaning that they connect zones both ways and you can look through the portal's front side as well as through its back-side.

Like Zones, Portals can either be box-shaped or use custom convex polyhedral shapes.

Portals will usually be created in the editor but can, of course, also be created in script code as such:

Example:

```
// Example declaration of a Portal.
// This will create a box-shaped portal.
newPortal( PortalToTestZone )
{
   position = "12.8467 -4.02246 14.8017";
    rotation = "0 0 -1 97.5085";
    scale = "1 0.25 1";
    canSave = "1";
    canSaveDynamicFields = "1";
};
```

Note: Keep in mind that zones and portals are more or less strictly a scene optimization mechanism meant to improve render times.

Methods

bool `Portal::isExteriorPortal()`
Test whether the portal connects interior zones to the outdoor zone.

Returns True if the portal is an exterior portal.

bool `Portal::isInteriorPortal()`
Test whether the portal connects interior zones only.

Returns True if the portal is an interior portal.

Fields

bool `Portal::backSidePassable`
Whether one can view through the back-side of the portal.

bool `Portal::frontSidePassable`
Whether one can view through the front-side of the portal.

Prefab A collection of arbitrary objects which can be allocated and manipulated as a group.

Inherit: SceneObject

Description Prefab always points to a (.prefab) file which defines its objects. In fact more than one Prefab can reference this file and both will update if the file is modified.

Prefab is a very simple object and only exists on the server. When it is created it allocates children objects by reading the (.prefab) file like a list of instructions. It then sets their transform relative to the Prefab and Torque networking handles the rest by ghosting the new objects to clients. Prefab itself is not ghosted.

Methods

void `Prefab::onLoad`(SimGroup *children*)

Called when the prefab file is loaded and children objects are created.

Parameters **`children`** – SimGroup containing all children objects.

Fields

filename `Prefab::fileName`

(.prefab) File describing objects within this prefab.

ReflectorDesc A datablock which defines performance and quality properties for dynamic reflections.

Inherit: SimDataBlock

Description ReflectorDesc is not itself a reflection and does not render reflections. It is a dummy class for holding and exposing to the user a set of reflection related properties. Objects which support dynamic reflections may then reference a ReflectorDesc.

Example:

```
datablock ReflectorDesc( ExampleReflectorDesc )
{
   texSize = 256;
   nearDist = 0.1;
   farDist = 500;
   objectTypeMask = 0xFFFFFFFF;
   detailAdjust = 1.0;
   priority = 1.0;
   maxRateMs = 0;
   useOcclusionQuery = true;
};
```

Fields

float `ReflectorDesc::detailAdjust`

Scale applied to lod calculation of objects rendering into this reflection (modulates $pref::TS::detailAdjust).

float `ReflectorDesc::farDist`

Far plane distance to use when rendering reflections.

int `ReflectorDesc::maxRateMs`

If less than maxRateMs has elapsed since this relfection was last updated, then do not update it again. This 'skip' can be disabled by setting maxRateMs to zero.

float `ReflectorDesc::nearDist`

Near plane distance to use when rendering this reflection. Adjust this to limit self-occlusion artifacts.

int ReflectorDesc::**objectTypeMask**
Object types which render into this reflection.

float ReflectorDesc::**priority**
Priority for updating this reflection, relative to others.

int ReflectorDesc::**texSize**
Size in pixels of the (square) reflection texture. For a cubemap this value is interpreted as size of each face.

bool ReflectorDesc::**useOcclusionQuery**
If available on the device use HOQs to determine if the reflective object is visible before updating its reflection.

TerrainMaterial The TerrainMaterial class orginizes the material settings for a single terrain material layer.

Inherit: SimObject

Description The TerrainMaterial class orginizes the material settings for a single terrain material layer.

Example:

```
// Created by the Terrain Painter tool in the World EditornewTerrainMaterial()
{
   internalName = "grass1";
   diffuseMap = "art/terrains/Test/grass1";
   detailMap = "art/terrains/Test/grass1_d";
   detailSize = "10";
   isManaged = "1";
   detailBrightness = "1";
   Enabled = "1";
   diffuseSize = "200";
};
```

Fields

float TerrainMaterial::**detailDistance**
Changes how far camera can see the detail map rendering on the material.

filename TerrainMaterial::**detailMap**
Detail map for the material.

float TerrainMaterial::**detailSize**
Used to scale the detail map to the material square.

float TerrainMaterial::**detailStrength**
Exponentially sharpens or lightens the detail map rendering on the material.

filename TerrainMaterial::**diffuseMap**
Base texture for the material.

float TerrainMaterial::**diffuseSize**
Used to scale the diffuse map to the material square.

float TerrainMaterial::**macroDistance**
Changes how far camera can see the Macro map rendering on the material.

filename TerrainMaterial::**macroMap**
Macro map for the material.

float TerrainMaterial::**macroSize**
Used to scale the Macro map to the material square.

float TerrainMaterial::**macroStrength**
Exponentially sharpens or lightens the Macro map rendering on the material.

filename TerrainMaterial::**normalMap**
Bump map for the material.

float TerrainMaterial::**parallaxScale**
Used to scale the height from the normal map to give some self occlusion effect (aka parallax) to the terrain material.

bool TerrainMaterial::**useSideProjection**
Makes that terrain material project along the sides of steep slopes instead of projected downwards.

TimeOfDay Environmental object that triggers a day/night cycle in level.

Inherit: SceneObject

Description Environmental object that triggers a day/night cycle in level.

Example:

```
newTimeOfDay(tod)
{
   axisTilt = "23.44";
   dayLength = "120";
   startTime = "0.15";
   time = "0.15";
   play = "0";
   azimuthOverride = "572.958";
   dayScale = "1";
   nightScale = "1.5";
   position = "598.399 550.652 196.297";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   canSave = "1";
   canSaveDynamicFields = "1";
};
```

Methods
void TimeOfDay::**addTimeOfDayEvent**(float *elevation*, string *identifier*)
void TimeOfDay::**animate**(float *elevation*, float *degreesPerSecond*)

void TimeOfDay::**setDayLength**(float *seconds*)

void TimeOfDay::**setPlay**(bool *enabled*)

void TimeOfDay::**setTimeOfDay**(float *time*)

Fields
float TimeOfDay::**axisTilt**
The angle in degrees between global equator and tropic.
float TimeOfDay::**azimuthOverride**

float TimeOfDay::**dayLength**
The length of a virtual day in real world seconds.

float TimeOfDay::**dayScale**
Scalar applied to time that elapses while the sun is up.

float TimeOfDay::**nightScale**
Scalar applied to time that elapses while the sun is down.

bool TimeOfDay::**play**
True when the TimeOfDay object is operating.

float TimeOfDay::**startTime**

float TimeOfDay::**time**
Current time of day.

WayPoint Special type of marker, distinguished by a name and team ID number.

Inherit: MissionMarker

Description The original Torque engines were built from a multi-player game called Tribes. The Tribes series featured various team based game modes, such as capture the flag. The WayPoint class survived the conversion from game (Tribes) to game engine (Torque).

Essentially, this is a MissionMarker with the addition of two variables: markerName and team. Whenever a WayPoint is created, it is added to a unique global list called WayPointSet. You can iterate through this set, seeking out specific markers determined by their markerName and team ID. This avoids the overhead of constantly calling commandToClient and commandToServer to determine a WayPoint object's name, unique ID, etc.

Example:

```
newWayPoint()
{
   team = "1";
   dataBlock = "WayPointMarker";
   position = "-0.0224786 1.53471 2.93219";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   canSave = "1";
   canSaveDynamicFields = "1";
};
```

Fields
caseString WayPoint::**markerName**
Unique name representing this waypoint.
WayPointTeam WayPoint::**team**
Unique numerical ID assigned to this waypoint, or set of waypoints.

Zone An object that represents an interior space.

Inherit: SceneObject

Description A zone is an invisible volume that encloses an interior space. All objects that have their world space axis-aligned bounding boxes (AABBs) intersect the zone's volume are assigned to the zone. This assignment happens automatically as objects are placed and transformed. Also, assignment is not exclusive meaning that an object can be assigned to many zones at the same time if it intersects all of them.

In itself, the volume of a zone is fully sealed off from the outside. This means that while viewing the scene from inside the volume, only objects assigned to the zone are rendered while when viewing the scene from outside the volume, objects exclusively only assigned the zone are not rendered.

Usually, you will want to connect zones to each other by means of portals. A portal overlapping with a zone

Example:

```
// Example declaration of a Zone.  This creates a box-shaped zone.newZone( TestZone )
{
   position = "3.61793 -1.01945 14.7442";
   rotation = "1 0 0 0";
   scale = "10 10 10";
};
```

Zone Groups Normally, Zones will not connect to each other when they overlap. This means that if viewing the scene from one zone, the contents of the other zone will not be visible except when there is a portal connecting the zones. However, sometimes it is convenient to represent a single interior space through a combination of Zones so that when any of these zones is visible, all other zones that are part of the same interior space are visible. This is possible by employing "zone groups".

Methods

void Zone::**dumpZoneState**(bool *updateFirst*)

Dump a list of all objects assigned to the zone to the console as well as a list of all connected zone spaces.

Parameters **updateFirst** – Whether to update the contents of the zone before dumping. Since zoning states of objects are updated on demand, the zone contents can be outdated.

int Zone::**getZoneId**()

Get the unique numeric ID of the zone in its scene.

Returns The ID of the zone.

Fields

ColorF Zone::**ambientLightColor**

Color of ambient lighting in this zone. Only used if useAmbientLightColor is true.

string Zone::**edge**

For internal use only.

string Zone::**plane**

For internal use only.

string Zone::**point**

For internal use only.

SFXAmbience Zone::**soundAmbience**

Ambient sound environment for the space.

bool Zone::**useAmbientLightColor**

Whether to use ambientLightColor for ambient lighting in this zone or the global ambient color.

int Zone::**zoneGroup**

ID of group the zone is part of.

Enumeration

enum MarkerKnotType

The type of knot that this marker will be.

Parameters

- **Normal** – Knot will have a smooth camera translation/rotation effect.

- **Only** – Will do the same for translations, leaving rotation un-touched.
- **Kink** – The rotation will take effect immediately for an abrupt rotation change.

enum MarkerSmoothingType
The type of smoothing this marker will have for pathed objects.

Parameters

- **Spline** – Marker will cause the movements of the pathed object to be smooth.
- **Linear** – Marker will have no smoothing effect.

Functions

MissionArea **getMissionAreaServerObject**()
Get the MissionArea object, if any.

5.3.5 Miscellaneous

Camera, sound, input and networking.

Camera

This section is dedicated to the various camera objects in Torque 3D. The base Camera object that is typically manipulated by a GameConnection's input. A Path Camera moves along a path.

Classes

Camera Represents a position, direction and field of view to render a scene from.

Inherit: ShapeBase

Description A camera is typically manipulated by a GameConnection. When set as the connection's control object, the camera handles all movement actions ($mvForwardAction, $mvPitch, etc.) just like a Player.

Example:

```
// Set an already created camera as the GameConnections control object
%connection.setControlObject(%camera);
```

Methods of Operation The camera has two general methods of operation. The first is the standard mode where the camera starts and stops its motion and rotation instantly. This is the default operation of the camera and is used by most games. It may be specifically set with Camera::setFlyMode() for 6 DoF motion. It is also typically the method used with Camera::setOrbitMode() or one of its helper methods to orbit about a specific object (such as the Player's dead body) or a specific point.

The second method goes under the name of Newton as it follows Newton's 2nd law of motion: F=ma. This provides the camera with an ease-in and ease-out feel for both movement and rotation. To activate this method for movement, either use Camera::setNewtonFlyMode() or set the Camera::newtonMode field to true. To activate this method for rotation, set the Camera::newtonRotation to true. This method of operation is not typically used in games, and was developed to allow for a smooth fly through of a game level while recording a demo video. But with the right force and drag settings, it may give a more organic feel to the camera to games that use an overhead view, such as a RTS.

There is a third, minor method of operation but it is not generally used for games. This is when the camera is used with Torque's World Editor in Edit Orbit Mode. When set, this allows the camera to rotate about a specific point in the world, and move towards and away from this point. See Camera::setEditOrbitMode() and Camera::setEditOrbitPoint(). While in this mode, Camera::autoFitRadius() may also be used.

Example:

```
// Create a camera in the level and set its position to a given spawn point.
// Note: The camera starts in the standard fly mode.
%cam = newCamera() {
   datablock = "Observer";
};
MissionCleanup.add( %cam );
%cam.setTransform( %spawnPoint.getTransform() );
```

Example:

```
// Create a camera at the given spawn point for the specified
// GameConnection i.e. the client.  Uses the standard
// Sim::spawnObject() function to create the camera using the
// defined default settings.
// Note: The camera starts in the standard fly mode.
function GameConnection::spawnCamera(%this, %spawnPoint)
{
   // Set the control object to the default camera
   if (!isObject(%this.camera))
   {
      if (isDefined("$Game::DefaultCameraClass"))
         %this.camera = spawnObject($Game::DefaultCameraClass, $Game::DefaultCameraDataBlock);
   }

   // If we have a camera then set up some properties
   if (isObject(%this.camera))
   {
      // Make sure were cleaned up when the mission ends
      MissionCleanup.add( %this.camera );

      // Make sure the camera is always in scope for the connection
      %this.camera.scopeToClient(%this);

      // Send all user input from the connection to the camera
      %this.setControlObject(%this.camera);

      if (isDefined("%spawnPoint"))
      {
         // Attempt to treat %spawnPoint as an object, such as a
         // SpawnSphere class.
         if (getWordCount(%spawnPoint) == 1 &&isObject(%spawnPoint))
         {
            %this.camera.setTransform(%spawnPoint.getTransform());
         }
         else
         {
            // Treat %spawnPoint as an AngleAxis transform
            %this.camera.setTransform(%spawnPoint);
         }
      }
   }
}
```

Motion Modes Beyond the different operation methods, the Camera may be set to one of a number of motion modes. These motion modes determine how the camera will respond to input and may be used to constrain how the Camera moves. The CameraMotionMode enumeration defines the possible set of modes and the Camera's current may be obtained by using getMode().

Some of the motion modes may be set using specific script methods. These often provide additional parameters to set up the mode in one go. Otherwise, it is always possible to set a Camera's motion mode using the controlMode property. Just pass in the name of the mode enum. The following table lists the motion modes, how to set them up, and what they offer:

Mode	Set From Script	Input Move	Input Rotate	Can Use Newton Mode?
Stationary	controlMode property	No	No	No
FreeRotate	controlMode property	No	Yes	Rotate Only
Fly	setFlyMode()	Yes	Yes	Yes
OrbitObject	setOrbitMode()	Orbits object	Points to object	Move only
OrbitPoint	setOrbitPoint()	Orbits point	Points to location	Move only
TrackObject	setTrackObject()	No	Points to object	Yes
Overhead	controlMode property	Yes	No	Yes
EditOrbit (object selected)	setEditOrbitMode()	Orbits object	Points to object	Move only
EditOrbit (no object)	setEditOrbitMode()	Yes	Yes	Yes

Trigger Input Passing a move trigger ($mvTriggerCount0, $mvTriggerCount1, etc.) on to a Camera performs different actions depending on which mode the camera is in. While in Fly, Overhead or EditOrbit mode, either trigger0 or trigger1 will cause a camera to move twice its normal movement speed. You can see this in action within the World Editor, where holding down the left mouse button while in mouse look mode (right mouse button is also down) causes the Camera to move faster.

Passing along trigger2 will put the camera into strafe mode. While in this mode a Fly, FreeRotate or Overhead Camera will not rotate from the move input. Instead the yaw motion will be applied to the Camera's x motion, and the pitch motion will be applied to the Camera's z motion. You can see this in action within the World Editor where holding down the middle mouse button allows the user to move the camera up, down and side-to-side.

While the camera is operating in Newton Mode, trigger0 and trigger1 behave slightly differently. Here trigger0 activates a multiplier to the applied acceleration force as defined by speedMultiplier. This has the affect of making the camera move up to speed faster. trigger1 has the opposite affect by acting as a brake. When trigger1 is active a multiplier is added to the Camera's drag as defined by brakeMultiplier.

Methods

void `Camera::autoFitRadius` (float *radius*)
Move the camera to fully view the given radius.

Parameters **`radius`** – The radius to view.

VectorF `Camera::getAngularVelocity` ()
Get the angular velocity for a Newton mode camera.

Returns The angular velocity in the form of "x y z".

Camera::CameraMotionMode `Camera::getMode` ()
Returns the current camera control mode.

Point3F Camera::**getOffset**()
Get the camera's offset from its orbit or tracking point. The offset is added to the camera's position when set to CameraMode::OrbitObject.

Returns The offset in the form of "x y z".

Point3F Camera::**getPosition**()
Get the camera's position in the world. Reimplemented from SceneObject .

Returns The position in the form of "x y z".

Point3F Camera::**getRotation**()
Get the camera's Euler rotation in radians.

Returns The rotation in radians in the form of "x y z".

VectorF Camera::**getVelocity**()
Get the velocity for the camera. Reimplemented from ShapeBase .

Returns The camera's velocity in the form of "x y z".

bool Camera::**isEditOrbitMode**()
Is the camera in edit orbit mode?

Returns true if the camera is in edit orbit mode.

bool Camera::**isRotationDamped**()
Is this a Newton Fly mode camera with damped rotation?

Returns is set to true.

void Camera::**lookAt**(Point3F *point*)
Point the camera at the specified position. Does not work in Orbit or Track modes.

Parameters **point** – The position to point the camera at.

void Camera::**setAngularDrag**(float *drag*)
Set the angular drag for a Newton mode camera.

Parameters **drag** – The angular drag applied while the camera is rotating.

void Camera::**setAngularForce**(float *force*)
Set the angular force for a Newton mode camera.

Parameters **force** – The angular force applied when attempting to rotate the camera.

void Camera::**setAngularVelocity**(VectorF *velocity*)
Set the angular velocity for a Newton mode camera.

Parameters **velocity** – The angular velocity infor form of "x y z".

void Camera::**setBrakeMultiplier**(float *multiplier*)
Set the Newton mode camera brake multiplier when trigger[1] is active.

Parameters **multiplier** – The brake multiplier to apply.

void Camera::**setDrag**(float *drag*)
Set the drag for a Newton mode camera.

Parameters **drag** – The drag applied to the camera while moving.

void Camera::**setEditOrbitMode**()
Set the editor camera to orbit around a point set with Camera::setEditOrbitPoint() .

void Camera::**setEditOrbitPoint**(Point3F *point*)
Set the editor camera's orbit point.

Parameters **point** – The point the camera will orbit in the form of "x y z".

void Camera::**setFlyForce**(float *force*)
Set the force applied to a Newton mode camera while moving.

Parameters **force** – The force applied to the camera while attempting to move.

void Camera::**setFlyMode**()
Set the camera to fly freely. Allows the camera to have 6 degrees of freedom. Provides for instantaneous motion and rotation unless one of the Newton fields has been set to true. See Camera::newtonMode and Camera::newtonRotation .

void Camera::**setMass**(float *mass*)
Set the mass for a Newton mode camera.

Parameters **mass** – The mass used during ease-in and ease-out calculations.

void Camera::**setNewtonFlyMode**()
Set the camera to fly freely, but with ease-in and ease-out. This method allows for the same 6 degrees of freedom as Camera::setFlyMode() but activates the ease-in and ease-out on the camera's movement. To also activate Newton mode for the camera's rotation, set Camera::newtonRotation to true.

void Camera::**setOffset**(Point3F *offset*)
Set the camera's offset. The offset is added to the camera's position when set to CameraMode::OrbitObject.

Parameters **offset** – The distance to offset the camera by in the form of "x y z".

void Camera::**setOrbitMode**(GameBase *orbitObject*, TransformF *orbitPoint*, float *minDistance*, float *maxDistance*, float *initDistance*, bool *ownClientObj*, Point3F *offset*, bool *locked*)
Set the camera to orbit around the given object, or if none is given, around the given point.

Parameters

- **orbitObject** – The object to orbit around. If no object is given (0 or blank string is passed in) use the orbitPoint instead
- **orbitPoint** – The point to orbit around when no object is given. In the form of "x y z ax ay az aa" such as returned by SceneObject::getTransform().
- **minDistance** – The minimum distance allowed to the orbit object or point.
- **maxDistance** – The maximum distance allowed from the orbit object or point.
- **initDistance** – The initial distance from the orbit object or point.
- **ownClientObj** – [optional] Are we orbiting an object that is owned by us? Default is false.
- **offset** – [optional] An offset added to the camera's position. Default is no offset.
- **locked** – [optional] Indicates the camera does not receive input from the player. Default is false.

bool Camera::**setOrbitObject**(GameBase *orbitObject*, VectorF *rotation*, float *minDistance*, float *maxDistance*, float *initDistance*, bool *ownClientObject*, Point3F *offset*, bool *locked*)
Set the camera to orbit around a given object.

Parameters

- **orbitObject** – The object to orbit around.
- **rotation** – The initial camera rotation about the object in radians in the form of "x y z".
- **minDistance** – The minimum distance allowed to the orbit object or point.

- **maxDistance** – The maximum distance allowed from the orbit object or point.
- **initDistance** – The initial distance from the orbit object or point.
- **ownClientObject** – [optional] Are we orbiting an object that is owned by us? Default is false.
- **offset** – [optional] An offset added to the camera's position. Default is no offset.
- **locked** – [optional] Indicates the camera does not receive input from the player. Default is false.

Returns false if the given object could not be found.

void Camera::**setOrbitPoint**(TransformF *orbitPoint*, float *minDistance*, float *maxDistance*, float *initDistance*, Point3F *offset*, bool *locked*)

Set the camera to orbit around a given point.

Parameters

- **orbitPoint** – The point to orbit around. In the form of "x y z ax ay az aa" such as returned by SceneObject::getTransform().
- **minDistance** – The minimum distance allowed to the orbit object or point.
- **maxDistance** – The maximum distance allowed from the orbit object or point.
- **initDistance** – The initial distance from the orbit object or point.
- **offset** – [optional] An offset added to the camera's position. Default is no offset.
- **locked** – [optional] Indicates the camera does not receive input from the player. Default is false.

void Camera::**setRotation**(Point3F *rot*)

Set the camera's Euler rotation in radians.

Parameters **rot** – The rotation in radians in the form of "x y z".

void Camera::**setSpeedMultiplier**(float *multiplier*)

Set the Newton mode camera speed multiplier when trigger[0] is active.

Parameters **multiplier** – The speed multiplier to apply.

bool Camera::**setTrackObject**(GameBase *trackObject*, Point3F *offset*)

Set the camera to track a given object.

Parameters

- **trackObject** – The object to track.
- **offset** – [optional] An offset added to the camera's position. Default is no offset.

Returns false if the given object could not be found.

void Camera::**setValidEditOrbitPoint**(bool *validPoint*)

Set if there is a valid editor camera orbit point. When validPoint is set to false the Camera operates as if it is in Fly mode rather than an Orbit mode.

Parameters **validPoint** – Indicates the validity of the orbit point.

void Camera::**setVelocity**(VectorF *velocity*)

Set the velocity for the camera.

Parameters **velocity** – The camera's velocity in the form of "x y z".

Fields

float Camera::**angularDrag**
Drag on camera when rotating (Newton mode only). Default value is 2.

float Camera::**angularForce**
Force applied on camera when asked to rotate (Newton mode only). Default value is 100.

float Camera::**brakeMultiplier**
Speed multiplier when triggering the brake (Newton mode only). Default value is 2.

CameraMotionMode Camera::**controlMode**
The current camera control mode.

float Camera::**drag**
Drag on camera when moving (Newton mode only). Default value is 2.

float Camera::**force**
Force applied on camera when asked to move (Newton mode only). Default value is 500.

float Camera::**mass**
The camera's mass (Newton mode only). Default value is 10.

bool Camera::**newtonMode**
Apply smoothing (acceleration and damping) to camera movements.

bool Camera::**newtonRotation**
Apply smoothing (acceleration and damping) to camera rotations.

float Camera::**speedMultiplier**
Speed multiplier when triggering the accelerator (Newton mode only). Default value is 2.

CameraData A datablock that describes a camera.

Inherit: ShapeBaseData

Description A datablock that describes a camera.

Example:

```
datablock CameraData(Observer)
{
   mode = "Observer";
};
```

Datablocks and Networking

PathCamera A camera that moves along a path. The camera can then be made to travel along this path forwards or backwards.

Inherit: ShapeBase

Description A camera's path is made up of knots, which define a position, rotation and speed for the camera. Traversal from one knot to another may be either linear or using a Catmull-Rom spline. If the knot is part of a spline, then it may be a normal knot or defined as a kink. Kinked knots are a hard transition on the spline rather than a smooth one. A knot may also be defined as a position only. In this case the knot is treated as a normal knot but is ignored when determining how to smoothly rotate the camera while it is travelling along the path (the algorithm moves on to the next knot in the path for determining rotation).

The datablock field for a PathCamera is a previously created PathCameraData, which acts as the interface between the script and the engine for this PathCamera object.

Example:

```
%newPathCamera = newPathCamera()
{
  dataBlock = LoopingCam;
  position = "0 0 300 1 0 0 0";
};
```

Methods

void PathCamera::**onNode**(string *node*)

A script callback that indicates the path camera has arrived at a specific node in its path. Server side only.

Parameters **Node** – Unique ID assigned to this node.

void PathCamera::**popFront**()

Removes the knot at the front of the camera's path.

Example:

```
// Remove the first knot in the cameras path.
%pathCamera.popFront();
```

void PathCamera::**pushBack**(TransformF *transform*, float *speed*, string *type*, string *path*)

Adds a new knot to the back of a path camera's path.

Parameters

- **transform** – Transform for the new knot. In the form of "x y z ax ay az aa" such as returned by SceneObject::getTransform()
- **speed** – Speed setting for this knot.
- **type** – Knot type (Normal, Position Only, Kink).
- **path** – Path type (Linear, Spline).

Example:

```
// Transform vector for new knot.
//  (Pos_X Pos_Y Pos_Z Rot_X Rot_Y Rot_Z Angle)
%transform = "15.0 5.0 5.0 1.4 1.0 0.2 1.0"
// Speed setting for knot.
%speed = "1.0"
// Knot type.
//  (Normal, Position Only, Kink)
%type = "Normal";

// Path Type. (Linear, Spline)
%path = "Linear";

// Inform the path camera to add a new knot to the back of its path
%pathCamera.pushBack(%transform,%speed,%type,%path);
```

void PathCamera::**pushFront**(TransformF *transform*, float *speed*, string *type*, string *path*)

Adds a new knot to the front of a path camera's path.

Parameters

- **transform** – Transform for the new knot. In the form of "x y z ax ay az aa" such as returned by SceneObject::getTransform()

- **speed** – Speed setting for this knot.
- **type** – Knot type (Normal, Position Only, Kink).
- **path** – Path type (Linear, Spline).

Example:

```
// Transform vector for new knot.
//  (Pos_X,Pos_Y,Pos_Z,Rot_X,Rot_Y,Rot_Z,Angle)
%transform = "15.0 5.0 5.0 1.4 1.0 0.2 1.0"
// Speed setting for knot.
%speed = "1.0";

// Knot type.
//  (Normal, Position Only, Kink)
%type = "Normal";

// Path Type. (Linear, Spline)
%path = "Linear";

// Inform the path camera to add a new knot to the front of its path
%pathCamera.pushFront(%transform, %speed, %type, %path);
```

void PathCamera::**reset**(float *speed*)

Clear the camera's path and set the camera's current transform as the start of the new path. What specifically occurs is a new knot is created from the camera's current transform. Then the current path is cleared and the new knot is pushed onto the path. Any previous target is cleared and the camera's movement state is set to Forward. The camera is now ready for a new path to be defined.

Parameters **speed** – Speed for the camera to move along its path after being reset.

Example:

```
//Determine the new movement speed of this camera. If not set, the speed will default to 1.0.
%speed = "0.50";

// Inform the path camera to start a new path at
// the cameras current position, and set the new
// paths speed value.
%pathCamera.reset(%speed);
```

void PathCamera::**setPosition**(float *position*)

Set the current position of the camera along the path.

Parameters **position** – Position along the path, from 0.0 (path start) - 1.0 (path end), to place the camera.

Example:

```
// Set the camera on a position along its path from 0.0 - 1.0.
%position = "0.35";

// Force the pathCamera to its new position along the path.
%pathCamera.setPosition(%position);
```

void PathCamera::**setState**(string *newState*)

Set the movement state for this path camera.

Parameters **newState** – New movement state type for this camera. Forward, Backward or Stop.

Example:

```
    // Set the state type (forward, backward, stop).
    // In this example, the camera will travel from the first node
    // to the last node (or target if given with setTarget())
    %state = "forward";

    // Inform the pathCamera to change its movement state to the defined value.
    %pathCamera.setState(%state);
```

void PathCamera::**setTarget**(float *position*)

Set the movement target for this camera along its path. The camera will attempt to move along the path to the given target in the direction provided by setState() (the default is forwards). Once the camera moves past this target it will come to a stop, and the target state will be cleared.

Parameters **position** – Target position, between 0.0 (path start) and 1.0 (path end), for the camera to move to along its path.

Example:

```
    // Set the position target, between 0.0 (path start)
    // and 1.0 (path end), for this camera to move to.
    %position = "0.50";

    // Inform the pathCamera of the new target position it will move to.
    %pathCamera.setTarget(%position);
```

PathCameraData General interface to control a PathCamera object from the script level.

Inherit: ShapeBaseData

Description General interface to control a PathCamera object from the script level.

Example:

```
datablock PathCameraData(LoopingCam)
   {
      mode = "";
   };
```

Functions

void **setDefaultFov**(float *defaultFOV*)

Set the default FOV for a camera.

Parameters **defaultFOV** – The default field of view in degrees

void **setFov**(float *FOV*)

Set the FOV of the camera.

Parameters **FOV** – The camera's new FOV in degrees

void **setZoomSpeed**(int *speed*)

Set the zoom speed of the camera. This affects how quickly the camera changes from one field of view to another.

Parameters **speed** – The camera's zoom speed in ms per 90deg FOV change

Enumerations

enum CameraMotionMode
Movement behavior type for Camera.

Parameters

- **Stationary** – Camera does not rotate or move.
- **FreeRotate** – Camera may rotate but does not move.
- **Fly** – Camera may rotate and move freely.
- **OrbitObject** – Camera orbits about a given object. Damage flash and white out is determined by the object being orbited. See Camera::setOrbitMode() to set the orbit object and other parameters.
- **OrbitPoint** – Camera orbits about a given point. See Camera::setOrbitMode() to set the orbit point and other parameters.
- **TrackObject** – Camera always faces a given object. See Camera::setTrackObject() to set the object to track and a distance to remain from the object.
- **Overhead** – Camera moves in the XY plane.
- **EditOrbit** – Used by the World Editor to orbit about a point. When first activated, the camera is rotated to face the orbit point rather than move to it.

Variables

int Camera::**extendedMovePosRotIndex**
The ExtendedMove position/rotation index used for camera movements.

float Camera::**movementSpeed**
Global camera movement speed in units/s (typically m/s), with a base value of 40.

Used in the following camera modes:

•Edit Orbit Mode

•Fly Mode

•Overhead Mode

Input

Functions and classes relating to to user input.

Classes

ActionMap ActionMaps assign platform input events to console commands.

Inherit: SimObject

Description Any platform input event can be bound in a single, generic way. In theory, the game doesn't need to know if the event came from the keyboard, mouse, joystick or some other input device. This allows users of the game to map keys and actions according to their own preferences. Game action maps are arranged in a stack for processing so individual parts of the game can define specific actions. For example, when the player jumps into a vehicle it could push a vehicle action map and pop the default player action map.

Creating an ActionMap The input system allows for the creation of multiple ActionMaps, so long as they have unique names and do not already exist. It's a simple three step process.

1. Check to see if the ActionMap exists
2. Delete it if it exists
3. Instantiate the ActionMap

The following is an example of how to create a new ActionMap:

Example:

```
if ( isObject( moveMap ) )
   moveMap.delete();
newActionMap(moveMap);
```

Binding Functions Once you have created an ActionMap, you can start binding functionality to events. Currently, Torque 3D supports the following devices out of the box

- Mouse
- Keyboard
- Joystick/Gamepad
- Xbox 360 Controller

The two most commonly used binding methods are bind() and bindCmd(). Both are similar in that they will bind functionality to a device and event, but different in how the event is interpreted. With bind(), you specify a device, action to bind, then a function to be called when the event happens.

Example:

```
// Simple function that prints to console// %val - Sent by the device letting the user know// if an i
function testInput(%val)
{
   if(%val)
     echo("Key is down");
   elseecho("Key was released");
}

// Bind the K key to the testInput function
moveMap.bind(keyboard, "k", testInput);
```

bindCmd is an alternative method for binding commands. This function is similar to bind(), except two functions are set to be called when the event is processed.

One will be called when the event is activated (input down), while the other is activated when the event is broken (input release). When using bindCmd(), pass the functions as strings rather than the function names.

Example:

```
// Print to the console when the spacebar is pressed
function onSpaceDown()
{
   echo("Space bar down!");
}

// Print to the console when the spacebar is released
function onSpaceUp()
{
   echo("Space bar up!");
}

// Bind the commands onSpaceDown and onSpaceUp to spacebar events
moveMap.bindCmd(keyboard, "space", "onSpaceDown();", "onSpaceUp();");
```

Switching ActionMaps Let's say you want to have different ActionMaps activated based on game play situations. A classic example would be first person shooter controls and racing controls in the same game. On foot, spacebar may cause your player to jump. In a vehicle, it may cause some kind of "turbo charge". You simply need to push/pop the ActionMaps appropriately:

First, create two separate ActionMaps:

Example:

```
// Create the two ActionMaps
if ( isObject( moveMap ) )
   moveMap.delete();
newActionMap(moveMap);

if ( isObject( carMap ) )
   carMap.delete();
newActionMap(carMap);
```

Next, create the two separate functions. Both will be bound to spacebar, but not the same ActionMap:

Example:

```
// Print to the console the player is jumping
function playerJump(%val)
{
   if(%val)
     echo("Player jumping!");
}

// Print to the console the vehicle is charging
function turboCharge()
{
   if(%val)
     echo("Vehicle turbo charging!");
}
```

You are now ready to bind functions to your ActionMaps' devices:

Example:

```
// Bind the spacebar to the playerJump function
// when moveMap is the active ActionMap
moveMap.bind(keyboard, "space", playerJump);
```

```
// Bind the spacebar to the turboCharge function
// when carMap is the active ActionMap
carMap.bind(keyboard, "space", turboCharge);
```

Finally, you can use the push() and pop() commands on each ActionMap to toggle activation. To activate an ActionMap, use push():

Example:

```
// Make moveMap the active action map
// You should now be able to activate playerJump with spacebar
moveMap.push();
```

To switch ActionMaps, first pop() the old one. Then you can push() the new one:

Example:

```
// Deactivate moveMap
moveMap.pop();

// Activate carMap
carMap.push();
```

Methods

bool ActionMap**::bind**(string *device*, string *action*, string *command*)

Associates a function to an input event. When the input event is raised, the specified function will be called.

Parameters

- **device** – The input device, such as mouse or keyboard.
- **action** – The input event, such as space, button0, etc.
- **command** – The function to bind to the action. Function must have a single boolean argument.

Returns True if the binding was successful, false if the device was unknown or description failed.

Example:

```
// Simple function that prints to console
// %val - Sent by the device letting the user know
// if an input was pressed (true) or released (false)
function testInput(%val)
{
   if(%val)
     echo("Key is down");
   elseecho("Key was released");
}

// Bind the K key to the testInput function
moveMap.bind(keyboard, k, testInput);
```

bool ActionMap**::bind**(string *device*, string *action*, string *flag*, string *deadZone*, string *scale*, string *command*)

Associates a function and input parameters to an input event. When the input event is raised, the specified function will be called. Modifier flags may be specified to process dead zones, input inversion, and more. Valid modifier flags:

•R - Input is Ranged.

•S - Input is Scaled.

•I - Input is inverted.

•D - Dead zone is present.

•N - Input should be re-fit to a non-linear scale.

Parameters

- **device** – The input device, such as mouse or keyboard.
- **action** – The input event, such as space, button0, etc.
- **flag** – Modifier flag assigned during binding, letting event know there are additional parameters to consider.
- **deadZone** – Restricted region in which device motion will not be acknowledged.
- **scale** – Modifies the deadZone region.
- **command** – The function bound to the action. Must take in a single argument.

Returns True if the binding was successful, false if the device was unknown or description failed.

Example:

```
// Simple function that adjusts the pitch of the camera
// based on the mouses movement along the X axis.
function testPitch(%val)
{
   %pitchAdj = getMouseAdjustAmount(%val);
    $mvPitch += %pitchAdj;
}

// Bind the mouses X axis to the testPitch function
// DI is flagged, meaning input is inverted and has a deadzone
%this.bind( mouse, "xaxis", "DI", "-0.23 0.23", testPitch );
```

bool `ActionMap::bindCmd`(string *device*, string *action*, string *makeCmd*, string *breakCmd*)

Associates a make command and optional break command to a specified input device action. Must include parenthesis and semicolon in the make and break command strings.

Parameters

- **device** – The device to bind to. Can be a keyboard, mouse, joystick or gamepad.
- **action** – The device action to bind to. The action is dependant upon the device. Specify a key for keyboards.
- **makeCmd** – The command to execute when the device/action is made.
- **breakCmd** – [optional] The command to execute when the device or action is unmade.

Returns True the bind was successful, false if the device was unknown or description failed.

Example:

```
// Print to the console when the spacebar is pressed
function onSpaceDown()
{
   echo("Space bar down!");
}

// Print to the console when the spacebar is released
function onSpaceUp()
```

```
    {
       echo("Space bar up!");
    }

    // Bind the commands onSpaceDown() and onSpaceUp() to spacebar events

    moveMap.bindCmd(keyboard, "space", "onSpaceDown();", "onSpaceUp();");
```

bool ActionMap**::bindObj**(string *device*, string *action*, string *command*, SimObjectID *object*)

Associates a function to an input event for a specified class or object. You must specify a device, the action to bind, a function, and an object to be called when the event happens. The function specified must be set to receive a single boolean value passed.

Parameters

- **device** – The input device, such as mouse or keyboard.
- **action** – The input event, such as space, button0, etc.
- **command** – The function bound to the action.
- **object** – The object or class bound to the action.

Returns True if the binding was successful, false if the device was unknown or description failed.

Example:

```
    moveMap.bindObj(keyboard, "numpad1", "rangeChange", %player);
```

bool ActionMap**::bindObj**(string *device*, string *action*, string *flag*, string *deadZone*, string *scale*, string *command*, SimObjectID *object*)

Associates a function to an input event for a specified class or object. You must specify a device, the action to bind, a function, and an object to be called when the event happens. The function specified must be set to receive a single boolean value passed. Modifier flags may be specified to process dead zones, input inversion, and more. Valid modifier flags:

•R - Input is Ranged.

•S - Input is Scaled.

•I - Input is inverted.

•D - Dead zone is present.

•N - Input should be re-fit to a non-linear scale.

Parameters

- **device** – The input device, such as mouse or keyboard.
- **action** – The input event, such as space, button0, etc.
- **flag** – Modifier flag assigned during binding, letting event know there are additional parameters to consider.
- **deadZone** – [Required only when flag is set] Restricted region in which device motion will not be acknowledged.
- **scale** – [Required only when flag is set] Modifies the deadZone region.
- **command** – The function bound to the action.
- **object** – The object or class bound to the action.

Returns True if the binding was successful, false if the device was unknown or description failed.

Example:

```
// Bind the mouses movement along the x-axis to
// the testInput function of the Player class
// DSI is flagged, meaning input is inverted,
// has scale and has a deadzone
%this.bindObj( mouse, "xaxis", "DSI", %deadZone, %scale, "testInput", %player );
```

string ActionMap::**getBinding**(string *command*)
Gets the ActionMap binding for the specified command. Use getField() on the return value to get the device and action of the binding.

Parameters **command** – The function to search bindings for.

Returns The binding against the specified command. Returns an empty string("") if a binding wasn't found.

Example:

```
// Find what the function "jump()" is bound to in moveMap
%bind = moveMap.getBinding( "jump" );

if ( %bind !$= "" )
{
   // Find out what device is used in the binding
   %device = getField( %bind, 0 );

   // Find out what action (such as a key) is used in the binding
   %action = getField( %bind, 1 );
}
```

string ActionMap::**getCommand**(string *device*, string *action*)
Gets ActionMap command for the device and action.

Parameters

- **device** – The device that was bound. Can be a keyboard, mouse, joystick or a gamepad.
- **action** – The device action that was bound. The action is dependant upon the device. Specify a key for keyboards.

Returns The command against the specified device and action.

Example:

```
// Find what function is bound to a devices action
// In this example, "jump()" was assigned to the space key in another script
%command = moveMap.getCommand("keyboard", "space");

// Should print "jump" in the console
echo(%command)
```

string ActionMap::**getDeadZone**(string *device*, string *action*)
Gets the Dead zone for the specified device and action.

Parameters

- **device** – The device that was bound. Can be a keyboard, mouse, joystick or a gamepad.
- **action** – The device action that was bound. The action is dependant upon the device. Specify a key for keyboards.

Returns The dead zone for the specified device and action. Returns "0 0" if there is no dead zone or an empty string("") if the mapping was not found.

Example:

```
%deadZone = moveMap.getDeadZone( "gamepad", "thumbrx");
```

float ActionMap::**getScale**(string *device*, string *action*)
Get any scaling on the specified device and action.

Parameters

- **device** – The device that was bound. Can be keyboard, mouse, joystick or gamepad.
- **action** – The device action that was bound. The action is dependant upon the device. Specify a key for keyboards.

Returns Any scaling applied to the specified device and action.

Example:

```
%scale = %moveMap.getScale( "gamepad", "thumbrx");
```

bool ActionMap::**isInverted**(string *device*, string *action*)
Determines if the specified device and action is inverted. Should only be used for scrolling devices or gamepad/joystick axes.

Parameters

- **device** – The device that was bound. Can be a keyboard, mouse, joystick or a gamepad.
- **action** – The device action that was bound. The action is dependant upon the device. Specify a key for keyboards.

Returns True if the specified device and action is inverted.

Example:

```
%if ( moveMap.isInverted( "mouse", "xaxis"))
   echo("Mouses xAxis is inverted");
```

void ActionMap::**pop**()
Pop the ActionMap off the ActionMap stack. Deactivates an ActionMap and removes it from the stack.

Example:

```
// Deactivate moveMap
moveMap.pop();
```

void ActionMap::**push**()
Push the ActionMap onto the ActionMap stack. Activates an ActionMap and placees it at the top of the ActionMap stack.

Example:

```
// Make moveMap the active action map
moveMap.push();
```

void ActionMap::**save**(string *fileName*, bool *append*)
Saves the ActionMap to a file or dumps it to the console.

Parameters

- **fileName** – The file path to save the ActionMap to. If a filename is not specified the ActionMap will be dumped to the console.

- **append** – Whether to write the ActionMap at the end of the file or overwrite it.

Example:

```
// Write out the actionmap into the config.cs file
moveMap.save( "scripts/client/config.cs" );
```

bool ActionMap::**unbind**(string *device*, string *action*)
Removes the binding on an input device and action.

Parameters

- **device** – The device to unbind from. Can be a keyboard, mouse, joystick or a gamepad.
- **action** – The device action to unbind from. The action is dependant upon the device. Specify a key for keyboards.

Returns True if the unbind was successful, false if the device was unknown or description failed.

Example:

```
moveMap.unbind("keyboard", "space");
```

bool ActionMap::**unbindObj**(string *device*, string *action*, string *obj*)
Remove any object-binding on an input device and action.

Parameters

- **device** – The device to bind to. Can be keyboard, mouse, joystick or gamepad.
- **action** – The device action to unbind from. The action is dependant upon the device. Specify a key for keyboards.
- **obj** – The object to perform unbind against.

Returns True if the unbind was successful, false if the device was unknown or description failed.

Example:

```
moveMap.unbindObj("keyboard", "numpad1", "rangeChange", %player);
```

LeapMotionFrame

Inherit: SimObject

Description UNDOCUMENTED!

Methods

int LeapMotionFrame::**getFrameInternalId**()
Provides the internal ID for this frame.

Returns Internal ID of this frame.

int LeapMotionFrame::**getFrameRealTime**()
Get the real time that this frame was generated.

Returns Real time of this frame in milliseconds.

int LeapMotionFrame::**getFrameSimTime**()
Get the sim time that this frame was generated.

Returns Sim time of this frame in milliseconds.

int LeapMotionFrame::**getHandCount**()
Get the number of hands defined in this frame.

Returns The number of defined hands.

int LeapMotionFrame::**getHandId**(int *index*)
Get the ID of the requested hand.

Parameters **index** – The hand index to check.

Returns ID of the requested hand.

int LeapMotionFrame::**getHandPointablesCount**(int *index*)
Get the number of pointables associated with this hand.

Parameters **index** – The hand index to check.

Returns Number of pointables that belong with this hand.

Point3I LeapMotionFrame::**getHandPos**(int *index*)
Get the position of the requested hand. The position is the hand's integer position converted to Torque 3D coordinates (in millimeters).

Parameters **index** – The hand index to check.

Returns Integer position of the requested hand (in millimeters).

Point3F LeapMotionFrame::**getHandRawPos**(int *index*)
Get the raw position of the requested hand. The raw position is the hand's floating point position converted to Torque 3D coordinates (in millimeters).

Parameters **index** – The hand index to check.

Returns Raw position of the requested hand.

TransformF LeapMotionFrame::**getHandRawTransform**(int *index*)
Get the raw transform of the requested hand.

Parameters **index** – The hand index to check.

Returns The raw position and rotation of the requested hand (in Torque 3D coordinates).

AngAxisF LeapMotionFrame::**getHandRot**(int *index*)
Get the rotation of the requested hand. The Leap Motion hand rotation as converted into the Torque 3Dcoordinate system.

Parameters **index** – The hand index to check.

Returns Rotation of the requested hand.

Point2F LeapMotionFrame::**getHandRotAxis**(int *index*)
Get the axis rotation of the requested hand. This is the axis rotation of the hand as if the hand were a gamepad thumb stick. Imagine a stick coming out the top of the hand and tilting the hand front, back, left and right controls that stick. The values returned along the x and y stick axis are normalized from -1.0 to 1.0 with the maximum hand tilt angle for these values as defined by $LeapMotion::MaximumHandAxisAngle .

Parameters **index** – The hand index to check.

Returns Axis rotation of the requested hand.

TransformF LeapMotionFrame::**getHandTransform**(int *index*)
Get the transform of the requested hand.

Parameters **index** – The hand index to check.

Returns The position and rotation of the requested hand (in Torque 3D coordinates).

bool LeapMotionFrame::**getHandValid**(int *index*)
Check if the requested hand is valid.

Parameters **index** – The hand index to check.

Returns True if the hand is valid.

int LeapMotionFrame::**getPointableHandIndex**(int *index*)
Get the index of the hand that this pointable belongs to, if any.

Parameters **index** – The pointable index to check.

Returns Index of the hand this pointable belongs to, or -1 if there is no associated hand.

int LeapMotionFrame::**getPointableId**(int *index*)
Get the ID of the requested pointable.

Parameters **index** – The pointable index to check.

Returns ID of the requested pointable.

float LeapMotionFrame::**getPointableLength**(int *index*)
Get the length of the requested pointable.

Parameters **index** – The pointable index to check.

Returns Length of the requested pointable (in millimeters).

Point3I LeapMotionFrame::**getPointablePos**(int *index*)
Get the position of the requested pointable. The position is the pointable's integer position converted to Torque 3D coordinates (in millimeters).

Parameters **index** – The pointable index to check.

Returns Integer position of the requested pointable (in millimeters).

Point3F LeapMotionFrame::**getPointableRawPos**(int *index*)
Get the raw position of the requested pointable. The raw position is the pointable's floating point position converted to Torque 3D coordinates (in millimeters).

Parameters **index** – The pointable index to check.

Returns Raw position of the requested pointable.

TransformF LeapMotionFrame::**getPointableRawTransform**(int *index*)
Get the raw transform of the requested pointable.

Parameters **index** – The pointable index to check.

Returns The raw position and rotation of the requested pointable (in Torque 3D coordinates).

AngAxisF LeapMotionFrame::**getPointableRot**(int *index*)
Get the rotation of the requested pointable. The Leap Motion pointable rotation as converted into the Torque 3Dcoordinate system.

Parameters **index** – The pointable index to check.

Returns Rotation of the requested pointable.

int LeapMotionFrame::**getPointablesCount**()
Get the number of pointables defined in this frame.

Returns The number of defined pointables.

TransformF LeapMotionFrame::**getPointableTransform**(int *index*)
Get the transform of the requested pointable.

Parameters **index** – The pointable index to check.

Returns The position and rotation of the requested pointable (in Torque 3D coordinates).

LeapMotionFramePointableType LeapMotionFrame::**getPointableType**(int *index*)
Get the type of the requested pointable.

Parameters **index** – The pointable index to check.

Returns Type of the requested pointable.

bool LeapMotionFrame::**getPointableValid**(int *index*)
Check if the requested pointable is valid.

Parameters **index** – The pointable index to check.

Returns True if the pointable is valid.

float LeapMotionFrame::**getPointableWidth**(int *index*)
Get the width of the requested pointable.

Parameters **index** – The pointable index to check.

Returns Width of the requested pointable (in millimeters).

bool LeapMotionFrame::**isFrameValid**()
Checks if this frame is valid.

Returns True if the frame is valid.

RazerHydraFrame

Inherit: SimObject

Description UNDOCUMENTED!

Methods

bool RazerHydraFrame::**getControllerButton1**(int *index*)
Get the button 1 state for the requested controller.

Parameters **index** – The controller index to check.

Returns Button 1 state requested controller as true or false.

bool RazerHydraFrame::**getControllerButton2**(int *index*)
Get the button 2 state for the requested controller.

Parameters **index** – The controller index to check.

Returns Button 2 state requested controller as true or false.

bool RazerHydraFrame::**getControllerButton3**(int *index*)
Get the button 3 state for the requested controller.

Parameters **index** – The controller index to check.

Returns Button 3 state requested controller as true or false.

bool RazerHydraFrame::**getControllerButton4**(int *index*)
Get the button 4 state for the requested controller.

Parameters **index** – The controller index to check.

Returns Button 4 state requested controller as true or false.

int RazerHydraFrame::**getControllerCount**()
Get the number of controllers defined in this frame.

Returns The number of defined controllers.

bool RazerHydraFrame::**getControllerDocked**(int *index*)
Get the docked state of the controller.

Parameters **index** – The controller index to check.

Returns True if the requested controller is docked.

bool RazerHydraFrame::**getControllerEnabled**(int *index*)
Get the enabled state of the controller.

Parameters **index** – The controller index to check.

Returns True if the requested controller is enabled.

Point3I RazerHydraFrame::**getControllerPos**(int *index*)
Get the position of the requested controller. The position is the controller's integer position converted to Torque 3D coordinates (in millimeters).

Parameters **index** – The controller index to check.

Returns Integer position of the requested controller (in millimeters).

Point3F RazerHydraFrame::**getControllerRawPos**(int *index*)
Get the raw position of the requested controller. The raw position is the controller's floating point position converted to Torque 3D coordinates (in millimeters).

Parameters **index** – The controller index to check.

Returns Raw position of the requested controller (in millimeters).

TransformF RazerHydraFrame::**getControllerRawTransform**(int *index*)
Get the raw transform of the requested controller.

Parameters **index** – The controller index to check.

Returns The raw position and rotation of the requested controller (in Torque 3D coordinates).

AngAxisF RazerHydraFrame::**getControllerRot**(int *index*)
Get the rotation of the requested controller. The Razer Hydra controller rotation as converted into the Torque 3Dcoordinate system.

Parameters **index** – The controller index to check.

Returns Rotation of the requested controller.

Point2F RazerHydraFrame::**getControllerRotAxis**(int *index*)
Get the axis rotation of the requested controller. This is the axis rotation of the controller as if the controller were a gamepad thumb stick. Imagine a stick coming out the top of the controller and tilting the controller front, back, left and right controls that stick. The values returned along the x and y stick axis are normalized from -1.0 to 1.0 with the maximum controller tilt angle for these values as defined by $RazerHydra::MaximumAxisAngle .

Parameters **index** – The controller index to check.

Returns Axis rotation of the requested controller.

int RazerHydraFrame::**getControllerSequenceNum**(int *index*)
Get the controller sequence number.

Parameters **index** – The controller index to check.

Returns The sequence number of the requested controller.

bool RazerHydraFrame::**getControllerShoulderButton**(int *index*)
Get the shoulder button state for the requested controller.

Parameters **index** – The controller index to check.

Returns Shoulder button state requested controller as true or false.

bool RazerHydraFrame::**getControllerStartButton**(int *index*)
Get the start button state for the requested controller.

Parameters **index** – The controller index to check.

Returns Start button state requested controller as true or false.

bool RazerHydraFrame::**getControllerThumbButton**(int *index*)
Get the thumb button state for the requested controller.

Parameters **index** – The controller index to check.

Returns Thumb button state requested controller as true or false.

Point2F RazerHydraFrame::**getControllerThumbStick**(int *index*)
Get the thumb stick values of the requested controller. The thumb stick values are in the range of -1.0..1.0

Parameters **index** – The controller index to check.

Returns Thumb stick values of the requested controller.

TransformF RazerHydraFrame::**getControllerTransform**(int *index*)
Get the transform of the requested controller.

Parameters **index** – The controller index to check.

Returns The position and rotation of the requested controller (in Torque 3D coordinates).

float RazerHydraFrame::**getControllerTrigger**(int *index*)
Get the trigger value for the requested controller. The trigger value is in the range of -1.0..1.0

Parameters **index** – The controller index to check.

Returns value of the requested controller.

int RazerHydraFrame::**getFrameInternalId**()
Provides the internal ID for this frame.

Returns Internal ID of this frame.

int RazerHydraFrame::**getFrameRealTime**()
Get the real time that this frame was generated.

Returns Real time of this frame in milliseconds.

int RazerHydraFrame::**getFrameSimTime**()
Get the sim time that this frame was generated.

Returns Sim time of this frame in milliseconds.

bool RazerHydraFrame::**isFrameValid**()
Checks if this frame is valid.

Returns True if the frame is valid.

Description

Input events come from the OS, are translated in the platform layer and then posted to the game. By default the game then checks the input event against a global ActionMap (which supercedes all other action handlers). If there is no action specified for the event, it is passed on to the GUI system. If the GUI does not handle the input event it is passed to the currently active (non-global) ActionMap stack.

Example: the user presses the ~ (tilde) key, which is bound in the global ActionMap to toggleConsole.

This causes the console function associated with the bind to be executed, which in this case is toggleConsole, resulting in the console output window being shown. If the key had not been bound in the global map, it would have passed to the first gui that could have handled it, and if none did, it would pass to any game actions that were bound to that key.

Input Events The following table represents all keyboard, mouse, and joystick input events available to stock Torque 3D. It should be noted that letter and number keys directly correlate to their mapping. For example "a" is literally the letter a. The button0, button1, and button2 are the most commonly used input mappings for left mouse button, right mouse button, and middle mouse button (respectively).

Keyboard General Events:

backspace	end	win_apps	tilde
tab	home	cmd	minus
return	left	equals	enter
up	lopt	lbracket	opt
shift	right	ropt	rbracket
ctrl	down	numlock	backslash
alt	print	scrolllock	semicolon
pause	insert	rshift	apostrophe
capslock	delete	lcontrol	comma
escape	help	rcontrol	period
space	win_lwindow	lalt	slash
pagedown	win_rwindow	ralt	lessthan
pageup			

Note: All general keys can be bound by simply using the key... ex. "u" will trigger the u key response.

Keyboard Numpad Events:

numpad0	numpad5	numpad9	numpadminus
numpad1	numpad6	numpadadmult	numpaddecimal
numpad2	numpad7	numpadadd	numpaddivide
numpad3	numpad8	numpadsep	numpadenter
numpad4			

Keyboard Function Key Events:

f1	f7	f13	f19
f2	f8	f14	f20
f3	f9	f15	f21
f4	f10	f16	f22
f5	f11	f17	f23
f6	f12	f18	f24

Joystick/Mouse Events:

button0	button8	button16	button24
button1	button9	button17	button25
button2	button10	button18	button26
button3	button11	button19	button27
button4	button12	button20	button28
button5	button13	button21	button29
button6	button14	button22	button30
button7	button15	button23	button31

Joystick/Mouse Axes:

xaxis	zaxis	ryaxis	slider
yaxis	rxaxis	rzaxis	

Joystick POV:

xpov	dpov	xpov2	dpov2
ypov	lpov	ypov2	lpov2
upov	rpov	upov2	rpov2

Miscellaneous Events:

anykey	nomatch

Functions

void **activateDirectInput**()
Activates DirectInput. Also activates any connected joysticks.

void **deactivateDirectInput**()
Disables DirectInput. Also deactivates any connected joysticks.

void **disableJoystick**()
Disables use of the joystick.

Note: DirectInput must be enabled and active to use this function.

void **disableXInput**()
Disables XInput for Xbox 360 controllers.

void **echoInputState**()
Prints information to the console stating if DirectInput and a Joystick are enabled and active.

bool **enableJoystick**()
Enables use of the joystick.

Note: DirectInput must be enabled and active to use this function.

---

bool **enableXInput**()
Enables XInput for Xbox 360 controllers.

Note: XInput is enabled by default. Disable to use an Xbox 360 Controller as a joystick device.

ActionMap **getCurrentActionMap**()
Returns the current ActionMap.

See also:

ActionMap

int **getXInputState**(int *controllerID*, string *property*, bool *current*)
Queries the current state of a connected Xbox 360 controller.

XInput Properties:

•XI_THUMBLX, XI_THUMBLY - X and Y axes of the left thumbstick.

•XI_THUMBRX, XI_THUMBRY - X and Y axes of the right thumbstick.

•XI_LEFT_TRIGGER, XI_RIGHT_TRIGGER - Left and Right triggers.

•SI_UPOV, SI_DPOV, SI_LPOV, SI_RPOV - Up, Down, Left, and Right on the directional pad.

•XI_START, XI_BACK - The Start and Back buttons.

•XI_LEFT_THUMB, XI_RIGHT_THUMB - Clicking in the left and right thumbstick.

•XI_LEFT_SHOULDER, XI_RIGHT_SHOULDER - Left and Right bumpers.

•XI_A, XI_B , XI_X, XI_Y - The A, B, X, and Y buttons.

Parameters

- **controllerID** – Zero-based index of the controller to return information about.
- **property** – Name of input action being queried, such as "XI_THUMBLX".
- **current** – True checks current device in action.

Returns Button queried - 1 if the button is pressed, 0 if it's not. Thumbstick queried - Int representing displacement from rest position. Trigger queried - Int from 0 to 255 representing how far the trigger is displaced.

bool **isJoystickEnabled**()
Queries input manager to see if a joystick is enabled.

Returns 1 if a joystick exists and is enabled, 0 if it's not.

bool **isXInputConnected**(int *controllerID*)
Checks to see if an Xbox 360 controller is connected.

Parameters **controllerID** – Zero-based index of the controller to check.

Returns 1 if the controller is connected, 0 if it isn't, and 205 if XInput hasn't been initialized.

void **lockMouse**(bool *isLocked*)
Lock or unlock the mouse to the window.

When true, prevents the mouse from leaving the bounds of the game window.

void **resetXInput**()
Rebuilds the XInput section of the InputManager.

Requests a full refresh of events for all controllers. Useful when called at the beginning of game code after actionMaps are set up to hook up all appropriate events.

void **rumble**(string *device*, float *xRumble*, float *yRumble*)
Activates the vibration motors in the specified controller.

The controller will constantly at it's xRumble and yRumble intensities until changed or told to stop.Valid inputs for xRumble/yRumble are [0 - 1].

Parameters

- **device** – Name of the device to rumble.

- **xRumble** – Intensity to apply to the left motor.
- **yRumble** – Intensity to apply to the right motor.

Note: In an Xbox 360 controller, the left motor is low-frequency, while the right motor is high-frequency.

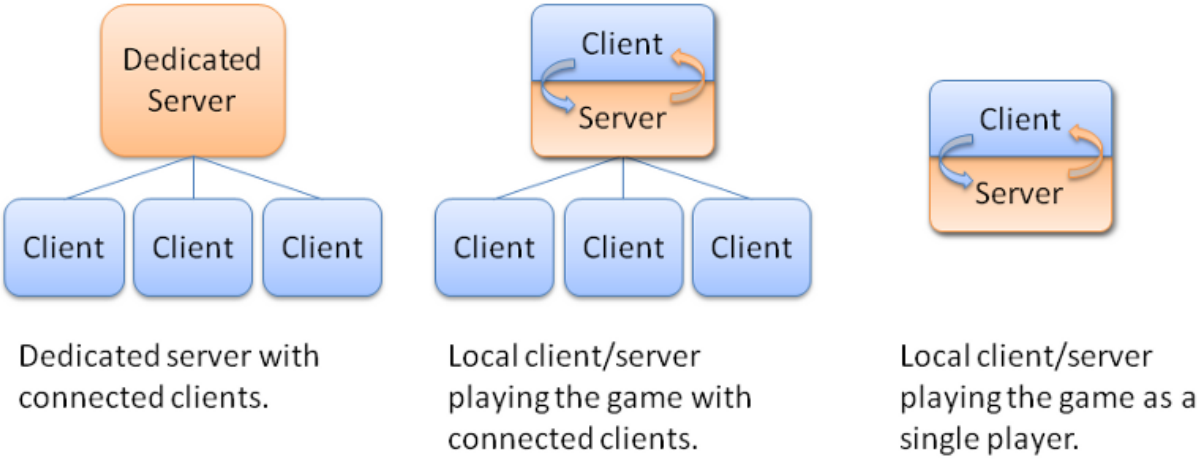
Network

Classes and functions related Torque 3D networking.

Description

Torque was designed from the foundation up to offer robust client/server networked simulations. Performance over the internet drove the design for the networking model. Torque attempts to deal with three fundamental problems of network simulation programming: limited bandwidth, packet loss and latency.

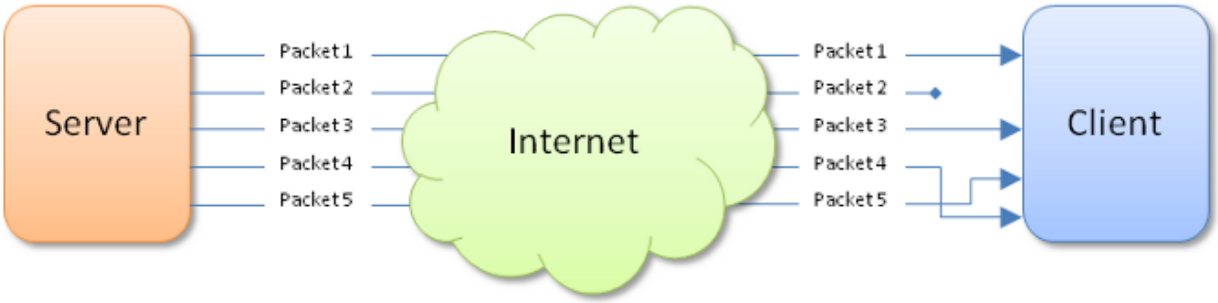
An instance of Torque can be set up as a dedicated server, a client, or both a client and a server. If the game is a client and a server, it still behaves as a client connected to a server - instead of using the network, however, the NetConnection object has a short-circuit link to another NetConnection object in the same application instance. This is known as a local connection.



Handling Limited Bandwidth Bandwidth is a problem because in the large, open environments that Torque allows, and with the large number of clients that the engine supports (depending on amount of data sent per client, game world complexity, and available bandwidth), potentially many different objects can be moving and updating at once.

Torque uses three main strategies to maximize available bandwidth. First, it prioritizes data, sending updates to what is most "important" to a client at a greater frequency than it updates data that is less important. Second, it sends only data that is necessary. Using the BitStream class, only the absolute minimum number of bits needed for a given piece of data will be sent. Also, when object state changes, Torque only sends the part of the object state that changed. Last, Torque caches common strings (NetStringTable) and data (SimDataBlock) so that they only need to be transmitted once.

Handling Packet Loss Packet loss is a problem because the information in lost data packets must somehow be retransmitted, yet in many cases the data in the dropped packet, if resent directly, will be stale by the time it gets to the client.



In this diagram:
-Packet 1 makes it to the client as expected
-Packet 2 never makes it to the client
-Packet 3 makes it to the client as expected
-Packet 5 arrives at the client prior to packet 4

For example, suppose that packet 2 contains a position update for a player and packet 3 contains a more recent position update for that same player. If packet 2 is dropped but packet 3 makes it across to the client, the engine shouldn't resend the data that was in packet 2. It is older than the version that was received by the client. In order to minimize data that gets resent unnecessarily, the engine classifies data into five groups:

- Unguaranteed Data (NetEvent) - If this data is lost, don't re-transmit it. An example of this type of data could be real-time voice traffic. By the time it is resent subsequent voice segments will already have played.
- Guaranteed Data (NetEvent) - If this data is lost, resend it. This is good for important, one-time information, like which team the player is on, or mission end messages are all examples of guaranteed data.
- Guaranteed Ordered Data (NetEvent) - If this data is lost, not only resend it, but make sure it arrives in the correct order. Chat messages, and messages for players joining and leaving the game all examples of guaranteed, ordered data. In the diagram above, packet 5 arrives before packet 4. If these consist of guaranteed ordered data, the client will not process packet 5 until packet 4 is first handled.
- Most-Recent State Data (NetObject) - Only the most current version of the data is important. If an update is lost, send the current state, unless it has been sent already. Most scene objects transmit their information in this manner.
- Guaranteed Quickest Data (Move) - Critical data that must get through as soon as possible. An example of this is movement information from the client to the server, which is transmitted with every packet by the Move Manager.

Handling Latency Latency is a problem in the simulation because the network delay in information transfer (which, for modems, can be up to a quarter of a second or more) makes the client's view of the world perpetually out-of-sync with the server.

Twitch FPS games, for which Torque was initially designed, require instant control response in order to feel anything but sluggish. Also, fast moving objects can be difficult for highly latent players to hit. In order to solve these problems Torque employs several strategies:

- Interpolation is used to smoothly move an object from where the client thinks it is to where the server says it is.
- Extrapolation is used to guess where the object is going based on its state and rules of movement.
- Prediction is used to form an educated guess about where an object is going based on rules of movement and client input.

The network architecture is layered: at the bottom is the platform layer, above that the notify protocol layer, followed by the NetConnection object and event management layer.

On Ghosting and Scoping One of the most powerful aspects of Torque's networking code is its support for ghosting and prioritized, most-recent-state network updates. The way this works is a bit complex, but it is immensely efficient. Let's run through the steps that the server goes through for each client in this part of Torque's networking:

- First, the server determines what objects are in-scope for the client. This is done by calling onCameraScopeQuery() on the object which is considered the "scope" object. This is usually the player object, but it can be something else. (For instance, the current vehicle, or an object we're remote controlling.)
- Second, it ghosts them to the client. A ghost is the client's representation of the server's object, and only maintains data that the client requires for the simulation. Ghosts come and go on the client according to the scope rules in the first step above.
- Finally, the server sends updates as needed, by checking the dirty list and packing updates. By only sending dirty data and using bit packing, no excess bandwidth is wasted. The order of ghost updates and their frequency is prioritized by the results of the object's getUpdatePriority() method.

Each object ghosted is assigned a ghost ID; the client is only aware of the ghost ID. This acts to enhance game security, as it becomes difficult to map objects from one connection to another, or to reliably identify objects from ID alone. IDs are also reassigned based on need, making it hard to track objects that have fallen out of scope (as any object which the player shouldn't see would).

NetConnection::resolveGhostID() is used on the client side, and NetConnection::resolveObjectFromGhostIndex() on the server side, to turn ghost IDs into object references. NetConnection::getGhostID() is used in the other direction to determine an object's ghost ID based on its SimObject ID. There is a cap on the maximum number of ghosts per client. Ghost IDs are currently sent via a 12-bit field, ergo, there is a cap of 4096 objects ghosted per client. This can be easily raised; see the GhostConstants enum in the source code.

See also:

NetObject for a further description of ghosting and individual objects.

NetConnection Group The NetConnection is a SimGroup. On the client side, it contains all the objects which have been ghosted to that client. On the server side, it is empty. It can be used (typically in script) to hold objects related to the connection. For instance, you might place an observation camera in the NetConnnection, or the current Player obejct. In both cases, when the connection is destroyed, so are the contained objects.

See also:

NetConnection, the basis for implementing a multiplayer game protocol. Also see NetObject, which is the superclass for ghostable objects, and ShapeBase, which is the base for player and vehicle classes.

Local Connections It is possible to run both the server and client within the same process. This is typically done while developing your multiplayer game, and is often required when using Torque's built-in world creation tools. This is also how a single player game is run. Having both a server and client together is known as a local connection.

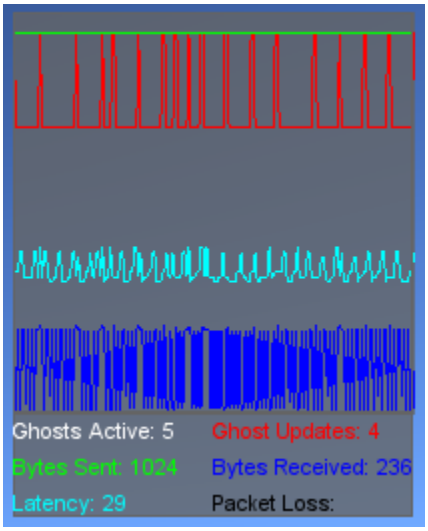
Any time a player launches the game and chooses to host a mission, they are also making use of a local connection. All other players joining the game use a regular, networked connection, and are considered clients.

Internally, a local connection short-circuits the networking layer and allows for data to pass immediately between the internal server and client. However, it should be noted that there is still the additional overhead of having seperate server and client branches within the code, even when creating a single player game. When developing your single player game, you need to be mindful that a client and server still exist within the engine.

See also:

NetConnection, the basis for implementing a multiplayer game protocol.

Monitoring the Network If you are interested in seeing Torque's various network statistics, use the Net Graph.



The Net Graph is from a client, or ServerConnection, point of view. To activate the Net Graph, either press the 'n' key, or open the console and type 'toggleNetGraph();'. The Net Graph presents a number of networking statistics, as described below:

- Ghosts Active - The number of active ghosts on the connection.
- Ghost Updates - The total number of ghosts added, removed or updated since the last update.
- Bytes Sent - The total number of bytes sent to the server since the last update.
- Bytes Received - The total number of bytes received from the server since the last update.
- Latency - The average round trip time (in ms) for the connection. Also known as ping.
- Packet Loss - The percentage of packets lost since the last update.

Classes

GameConnection The game-specific subclass of NetConnection.

Inherit: NetConnection

Description The GameConnection introduces the concept of the control object. The control object is simply the object that the client is associated with that network connection controls. By default the control object is an instance of the Player class, but can also be an instance of Camera (when editing the mission, for example), or any other ShapeBase derived class as appropriate for the game.

Torque uses a model in which the server is the authoritative master of the simulation. To prevent clients from cheating, the server simulates all player moves and then tells the client where his player is in the world. This model, while secure, can have problems. If the network latency is high, this round-trip time can give the player a very noticeable sense of movement lag. To correct this problem, the game uses a form of prediction - it simulates the movement of the control object on the client and on the server both. This way the client doesn't need to wait for round-trip verification of his moves. Only in the case of a force acting on the control object on the server that doesn't exist on the client does the client's position need to be forcefully changed.

To support this, all control objects (derivative of ShapeBase) must supply a writePacketData() and readPacketData() function that send enough data to accurately simulate the object on the client. These functions are only called for the current control object, and only when the server can determine that the client's simulation is somehow out of sync

with the server. This occurs usually if the client is affected by a force not present on the server (like an interpolating object) or if the server object is affected by a server only force (such as the impulse from an explosion).

The Move structure is a 32 millisecond snapshot of player input, containing x, y, and z positional and rotational changes as well as trigger state changes. When time passes in the simulation moves are collected (depending on how much time passes), and applied to the current control object on the client. The same moves are then packed over to the server in GameConnection::writePacket(), for processing on the server's version of the control object.

Methods

void GameConnection::**activateGhosting**()

Called by the server during phase 2 of the mission download to start sending ghosts to the client. Ghosts represent objects on the server that are in scope for the client. These need to be synchronized with the client in order for the client to see and interact with them. This is typically done during the standard mission start phase 2 when following Torque's example mission startup sequence.

Example:

```
function serverCmdMissionStartPhase2Ack(%client, %seq, %playerDB)
{
   // Make sure to ignore calls from a previous mission load
   if (%seq != $missionSequence || !$MissionRunning)
      return;
   if (%client.currentPhase != 1.5)
      return;
   %client.currentPhase = 2;

   // Set the player datablock choice
   %client.playerDB = %playerDB;

   // Update mod paths, this needs to get there before the objects.
   %client.transmitPaths();

   // Start ghosting objects to the client
   %client.activateGhosting();
}
```

bool GameConnection::**chaseCam**(int *size*)

Sets the size of the chase camera's matrix queue.

void GameConnection::**clearCameraObject**()

Clear the connection's camera object reference.

void GameConnection::**clearDisplayDevice**()

Clear any display device. A display device may define a number of properties that are used during rendering.

void GameConnection::delete(string reason)

On the server, disconnect a client and pass along an optional reason why. This method performs two operations: it disconnects a client connection from the server, and it deletes the connection object. The optional reason is sent in the disconnect packet and is often displayed to the user so they know why they've been disconnected.

Parameters **reason** – [optional] The reason why the user has been disconnected from the server.

Example:

```
function kick(%client)
{
   messageAll( MsgAdminForce, \c2The Admin has kicked %1., %client.playerName);

   if (!%client.isAIControlled())
      BanList::add(%client.guid, %client.getAddress(), $Pref::Server::KickBanTime);
```

```
        %client.delete("You have been kicked from this server");
    }
```

SimObject GameConnection::**getCameraObject**()
Returns the connection's camera object used when not viewing through the control object.

float GameConnection::**getControlCameraDefaultFov**()
Returns the default field of view as used by the control object's camera.

float GameConnection::**getControlCameraFov**()
Returns the field of view as used by the control object's camera.

GameBase GameConnection::**getControlObject**()
On the server, returns the object that the client is controlling.By default the control object is an instance of the Player class, but can also be an instance of Camera (when editing the mission, for example), or any other ShapeBase derived class as appropriate for the game.

bool GameConnection::**getControlSchemeAbsoluteRotation**()
Get the connection's control scheme absolute rotation property.

Returns True if the connection's control object should use an absolute rotation control scheme.

float GameConnection::**getDamageFlash**()
On the client, get the control object's damage flash level.

Returns flash level

static int GameConnection::**getServerConnection**()
On the client, this static mehtod will return the connection to the server, if any.

Returns ID of the server connection, or -1 if none is found.

float GameConnection::**getWhiteOut**()
On the client, get the control object's white-out level.

Returns white-out level

void GameConnection::**initialControlSet**()
Called on the client when the first control object has been set by the server and we are now ready to go. A common action to perform when this callback is called is to switch the GUI canvas from the loading screen and over to the 3D game GUI.

bool GameConnection::**isAIControlled**()
Returns true if this connection is AI controlled.

bool GameConnection::**isControlObjectRotDampedCamera**()
Returns true if the object being controlled by the client is making use of a rotation damped camera.

bool GameConnection::**isDemoPlaying**()
Returns true if a previously recorded demo file is now playing.

bool GameConnection::**isDemoRecording**()
Returns true if a demo file is now being recorded.

bool GameConnection::**isFirstPerson**()
Returns true if this connection is in first person mode.

void GameConnection::**listClassIDs**()
List all of the classes that this connection knows about, and what their IDs are. Useful for debugging network problems.

void GameConnection::**onConnectionAccepted**()
Called on the client when the connection to the server has been established.

void GameConnection::**onConnectionDropped**(string *reason*)
Called on the client when the connection to the server has been dropped.

Parameters **reason** – The reason why the connection was dropped.

void GameConnection::**onConnectionError**(string *errorString*)
Called on the client when there is an error with the connection to the server.

Parameters **errorString** – The connection error text.

void GameConnection::**onConnectionTimedOut**()
Called on the client when the connection to the server times out.

void GameConnection::**onConnectRequestRejected**(string *reason*)
Called on the client when the connection to the server has been rejected.

Parameters **reason** – The reason why the connection request was rejected.

void GameConnection::**onConnectRequestTimedOut**()
Called when connection attempts have timed out.

void GameConnection::**onControlObjectChange**()
Called on the client when the control object has been changed by the server.

void GameConnection::**onDataBlocksDone**(int *sequence*)
Called on the server when all datablocks has been sent to the client. During phase 1 of the mission download, all datablocks are sent from the server to the client. Once all datablocks have been sent, this callback is called and the mission download procedure may move on to the next phase.

Parameters **sequence** – The sequence is common between the server and client and ensures that the client is acting on the most recent mission start process. If an errant network packet (one that was lost but has now been found) is received by the client with an incorrect sequence, it is just ignored. This sequence number is updated on the server every time a mission is loaded.

void GameConnection::**onDrop**(string *disconnectReason*)
Called on the server when the client's connection has been dropped.

Parameters **disconnectReason** – The reason why the connection was dropped.

void GameConnection::**onFlash**(bool *state*)
Called on the client when the damage flash or white out states change. When the server changes the damage flash or white out values, this callback is called either is on or both are off. Typically this is used to enable the flash postFx.

Parameters **state** – Set to true if either the damage flash or white out conditions are active.

bool GameConnection::**play2D**(SFXProfile *profile*)
Used on the server to play a 2D sound that is not attached to any object.

Parameters **profile** – The SFXProfile that defines the sound to play.

Example:

```
function ServerPlay2D(%profile)
{
   // Play the given sound profile on every client.
   // The sounds will be transmitted as an event, not attached to any object.
   for(%idx = 0; %idx < ClientGroup.getCount(); %idx++)
      ClientGroup.getObject(%idx).play2D(%profile);
}
```

bool GameConnection::**play3D**(SFXProfile *profile*, TransformF *location*)
Used on the server to play a 3D sound that is not attached to any object.

Parameters

- **profile** – The SFXProfile that defines the sound to play.
- **location** – The position and orientation of the 3D sound given in the form of "x y z ax ay az aa".

Example:

```
function ServerPlay3D(%profile,%transform)
{
   // Play the given sound profile at the given position on every client
   // The sound will be transmitted as an event, not attached to any object.
   for(%idx = 0; %idx < ClientGroup.getCount(); %idx++)
      ClientGroup.getObject(%idx).play3D(%profile,%transform);
}
```

bool GameConnection::**playDemo**(string *demoFileName*)

On the client, play back a previously recorded game session. It is often useful to play back a game session. This could be for producing a demo of the game that will be shown at a later time, or for debugging a game. By recording the entire network stream it is possible to later play game the game exactly as it unfolded during the actual play session. This is because all user control and server results pass through the connection.

Returns True if the playback was successful. False if there was an issue, such as not being able to open the demo file for playback.

void GameConnection::**resetGhosting**()

On the server, resets the connection to indicate that ghosting has been disabled. Typically when a mission has ended on the server, all connected clients are informed of this change and their connections are reset back to a starting state. This method resets a connection on the server to indicate that ghosts are no longer being transmitted. On the client end, all ghost information will be deleted.

Example:

```
// Inform the clients
for (%clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++)
   {
      // clear ghosts and paths from all clients
      %cl = ClientGroup.getObject(%clientIndex);
      %cl.endMission();
      %cl.resetGhosting();
      %cl.clearPaths();
   }
```

void GameConnection::**setBlackOut**(bool *doFade*, int *timeMS*)

On the server, sets the client's 3D display to fade to black.

Parameters

- **doFade** – Set to true to fade to black, and false to fade from black.
- **timeMS** – Time it takes to perform the fade as measured in ms.

bool GameConnection::**setCameraObject**(GameBase *camera*)

On the server, set the connection's camera object used when not viewing through the control object.

void GameConnection::**setConnectArgs**(**const** char **args*)

On the client, pass along a variable set of parameters to the server. Once the connection is established with the server, the server calls its onConnect() method with the client's passed in parameters as aruments.

void GameConnection::**setControlCameraFov**(float *newFOV*)

On the server, sets the control object's camera's field of view.

Parameters **newFOV** – New field of view (in degrees) to force the control object's camera to use. This value is clamped to be within the range of 1 to 179 degrees.

bool GameConnection::**setControlObject**(GameBase *ctrlObj*)

On the server, sets the object that the client will control. By default the control object is an instance of the Player class, but can also be an instance of Camera (when editing the mission, for example), or any other ShapeBase derived class as appropriate for the game.

Parameters **ctrlObj** – The GameBase object on the server to control.

void GameConnection::**setControlSchemeParameters**(bool *absoluteRotation*, bool *addYawToAbsRot*, bool *addPitchToAbsRot*)

Set the control scheme that may be used by a connection's control object.

Parameters

- **absoluteRotation** – Use absolute rotation values from client, likely through ExtendedMove.
- **addYawToAbsRot** – Add relative yaw control to the absolute rotation calculation. Only useful when absoluteRotation is true.

void GameConnection::**setFirstPerson**(bool *firstPerson*)

On the server, sets this connection into or out of first person mode.

Parameters **firstPerson** – Set to true to put the connection into first person mode.

void GameConnection::**setJoinPassword**(string *password*)

On the client, set the password that will be passed to the server. On the server, this password is compared with what is stored in pref::Server::Password is empty then the client's sent password is ignored. Otherwise, if the passed in client password and the server password do not match, the CHR_PASSWORD error string is sent back to the client and the connection is immediately terminated. This password checking is performed quite early on in the connection request process so as to minimize the impact of multiple failed attempts – also known as hacking.

void GameConnection::**setLagIcon**(bool *state*)

Called on the client to display the lag icon. When the connection with the server is lagging, this callback is called to allow the game GUI to display some indicator to the player.

Parameters **state** – Set to true if the lag icon should be displayed.

void GameConnection::**setMissionCRC**(int *CRC*)

On the server, transmits the mission file's CRC value to the client. Typically, during the standard mission start phase 1, the mission file's CRC value on the server is send to the client. This allows the client to determine if the mission has changed since the last time it downloaded this mission and act appropriately, such as rebuilt cached lightmaps.

Parameters **CRC** – The mission file's CRC value on the server.

Example:

```
function serverCmdMissionStartPhase1Ack(%client, %seq)
{
   // Make sure to ignore calls from a previous mission loadif (%seq != $missionSequence || !$Mi
      return;
   if (%client.currentPhase != 0)
      return;
   %client.currentPhase = 1;

   // Start with the CRC
   %client.setMissionCRC( $missionCRC );
```

```
      // Send over the datablocks...
      // OnDataBlocksDone will get called when have confirmation
      // that theyve all been received.
      %client.transmitDataBlocks($missionSequence);
   }
```

void GameConnection::**startRecording**(string *fileName*)

On the client, starts recording the network connection's traffic to a demo file. It is often useful to play back a game session. This could be for producing a demo of the game that will be shown at a later time, or for debugging a game. By recording the entire network stream it is possible to later play game the game exactly as it unfolded during the actual play session. This is because all user control and server results pass through the connection.

Parameters **fileName** – The file name to use for the demo recording.

void GameConnection::**stopRecording**()

On the client, stops the recording of a connection's network traffic to a file.

void GameConnection::**transmitDataBlocks**(int *sequence*)

Sent by the server during phase 1 of the mission download to send the datablocks to the client. SimDataBlocks, also known as just datablocks, need to be transmitted to the client prior to the client entering the game world. These represent the static data that most objects in the world reference. This is typically done during the standard mission start phase 1 when following Torque's example mission startup sequence. When the datablocks have all been transmitted, onDataBlocksDone() is called to move the mission start process to the next phase.

Parameters **sequence** – The sequence is common between the server and client and ensures that the client is acting on the most recent mission start process. If an errant network packet (one that was lost but has now been found) is received by the client with an incorrect sequence, it is just ignored. This sequence number is updated on the server every time a mission is loaded.

Example:

```
   function serverCmdMissionStartPhase1Ack(%client, %seq)
   {
      // Make sure to ignore calls from a previous mission load
      if (%seq != $missionSequence || !$MissionRunning)
         return;
      if (%client.currentPhase != 0)
         return;
      %client.currentPhase = 1;

      // Start with the CRC
      %client.setMissionCRC( $missionCRC );

      // Send over the datablocks...
      // OnDataBlocksDone will get called when have confirmation
      // that theyve all been received.
      %client.transmitDataBlocks($missionSequence);
   }
```

HTTPObject Allows communications between the game and a server using HTTP.

Inherit: TCPObject

Description HTTPObject is derrived from TCPObject and makes use of the same callbacks for dealing with connections and received data. However, the way in which you use HTTPObject to connect with a server is different than

TCPObject. Rather than opening a connection, sending data, waiting to receive data, and then closing the connection, you issue a get() or post() and handle the response. The connection is automatically created and destroyed for you.

Example:

```
// In this example well retrieve the weather in Las Vegas using
// Googles API.  The response is in XML which could be processed
// and used by the game using SimXMLDocument, but well just output
// the results to the console in this example.

// Define callbacks for our specific HTTPObject using our instances
// name (WeatherFeed) as the namespace.

// Handle an issue with resolving the servers name
function WeatherFeed::onDNSFailed(%this)
{
   // Store this state
   %this.lastState = "DNSFailed";

   // Handle DNS failure
}

function WeatherFeed::onConnectFailed(%this)
{
   // Store this state
   %this.lastState = "ConnectFailed";

   // Handle connection failure
}

function WeatherFeed::onDNSResolved(%this)
{
   // Store this state
   %this.lastState = "DNSResolved";

}

function WeatherFeed::onConnected(%this)
{
   // Store this state
   %this.lastState = "Connected";

   // Clear our buffer
   %this.buffer = "";
}

function WeatherFeed::onDisconnect(%this)
{
   // Store this state
   %this.lastState = "Disconnected";

   // Output the buffer to the consoleecho("Google Weather Results:");
   echo(%this.buffer);
}

// Handle a line from the server
function WeatherFeed::onLine(%this, %line)
{
   // Store this line in out buffer
```

```
   %this.buffer = %this.buffer @ %line;
}

// Create the HTTPObject
%feed = newHTTPObject(WeatherFeed);

// Define a dynamic field to store the last connection state
%feed.lastState = "None";

// Send the GET command
%feed.get("www.google.com:80", "/ig/api", "weather=Las-Vegas,US");
```

Methods

void HTTPObject**::get**(string *Address*, string *requirstURI*, string *query*)

Send a GET command to a server to send or retrieve data.

Parameters

- **Address** – HTTP web address to send this get call to. Be sure to include the port at the end (IE: "www.garagegames.com:80").
- **requirstURI** – Specific location on the server to access (IE: "index.php".)
- **query** – Optional. Actual data to transmit to the server. Can be anything required providing it sticks with limitations of the HTTP protocol. If you were building the URL manually, this is the text that follows the question mark. For example: http://www.google.com/ig/api?weather=Las-Vegas,US

Example:

```
// Create an HTTP object for communications
%httpObj = newHTTPObject();

// Specify a URL to transmit to
%url = "www.garagegames.com:80";

// Specify a URI to communicate with
%URI = "/index.php";

// Specify a query to send.
%query = "";

// Send the GET command to the server
%httpObj.get(%url,%URI,%query);
```

void HTTPObject**::post**(string *Address*, string *requirstURI*, string *query*, string *post*)

Send POST command to a server to send or retrieve data.

Parameters

- **Address** – HTTP web address to send this get call to. Be sure to include the port at the end (IE: "www.garagegames.com:80").
- **requirstURI** – Specific location on the server to access (IE: "index.php".)
- **query** – Actual data to transmit to the server. Can be anything required providing it sticks with limitations of the HTTP protocol.
- **post** – Submission data to be processed.

Example:

```
// Create an HTTP object for communications
%httpObj = newHTTPObject();

// Specify a URL to transmit to
%url = "www.garagegames.com:80";

// Specify a URI to communicate with
%URI = "/index.php";

// Specify a query to send.
%query = "";

// Specify the submission data.
%post = "";

// Send the POST command to the server
%httpObj.POST(%url,%URI,%query,%post);
```

NetConnection Provides the basis for implementing a multiplayer game protocol.

Inherit: SimGroup

Description NetConnection combines a low-level notify protocol implemented in ConnectionProtocol with a SimGroup, and implements several distinct subsystems:

Event Manager This is responsible for transmitting NetEvents over the wire. It deals with ensuring that the various types of NetEvents are delivered appropriately, and with notifying the event of its delivery status.

Move Manager This is responsible for transferring a Move to the server 32 times a second (on the client) and applying it to the control object (on the server).

Ghost Manager This is responsible for doing scoping calculations (on the server side) and transmitting most-recent ghost information to the client.

File Transfer It is often the case that clients will lack important files when connecting to a server which is running a mod or new map. This subsystem allows the server to transfer such files to the client.

Networked String Table String data can easily soak up network bandwidth, so for efficiency, we implement a networked string table. We can then notify the connection of strings we will reference often, such as player names, and transmit only a tag, instead of the whole string.

Demo Recording A demo in Torque is a log of the network traffic between client and server; when a NetConnection records a demo, it simply logs this data to a file. When it plays a demo back, it replays the logged data.

Connection Database This is used to keep track of all the NetConnections; it can be iterated over (for instance, to send an event to all active connections), or queried by address.

The NetConnection is a SimGroup. On the client side, it contains all the objects which have been ghosted to that client. On the server side, it is empty; it can be used (typically in script) to hold objects related to the connection. For instance, you might place an observation camera in the NetConnnection. In both cases, when the connection is destroyed, so are the contained objects.

The NetConnection also has the concept of local connections. These are used when the client and server reside in the same process. A local connection is typically required to use the standard Torque world building tools. A local connection is also required when building a single player game.

Methods

void NetConnection::**checkMaxRate**()

Ensures that all configured packet rates and sizes meet minimum requirements. This method is normally only called when a NetConnection class is first constructed. It need only be manually called if the global variables that set the packet rate or size have changed.

void NetConnection::**clearPaths**()

On the server, resets the connection to indicate that motion spline paths have not been transmitted. Typically when a mission has ended on the server, all connected clients are informed of this change and their connections are reset back to a starting state. This method resets a connection on the server to indicate that motion spline paths have not been transmitted.

Example:

```
// Inform the clients
for (%clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++)
   {
      // clear ghosts and paths from all clients
      %cl = ClientGroup.getObject(%clientIndex);
      %cl.endMission();
      %cl.resetGhosting();
      %cl.clearPaths();
   }
```

void NetConnection::**connect**(string *remoteAddress*)

Connects to the remote address. Attempts to connect with another NetConnection on the given address. Typically once connected, a game's information is passed along from the server to the client, followed by the player entering the game world. The actual procedure is dependent on the NetConnection subclass that is used. i.e. GameConnection .

Parameters **remoteAddress** – The address to connect to in the form of IP:<address>:<port although the IP: portion is optional. The address portion may be in the form of w.x.y.z or as a host name, in which case a DNS lookup will be performed. You may also substitue the word broadcast for the address to broadcast the connect request over the local subnet.

string NetConnection::**connectLocal**()

Connects with the server that is running within the same process as the client.

Returns An error text message upon failure, or an empty string when successful.

string NetConnection::**getAddress**()

Returns the far end network address for the connection. The address will be in one of the following forms:

•IP:Broadcast:<port> for broadcast type addresses

•IP:<address>:<port> for IP addresses

•local when connected locally (server and client running in same process

int NetConnection::**getGhostID**(int *realID*)

On server or client, convert a real id to the ghost id for this connection. Torque's network ghosting system only exchanges ghost ID's between the server and client. Use this method on the server or client to discover an object's ghost ID based on its real SimObject ID.

Parameters **realID** – The real SimObject ID of the object.

Returns The ghost ID of the object for this connection, or -1 if it could not be resolved.

int NetConnection::**getGhostsActive**()

Provides the number of active ghosts on the connection.

Returns The number of active ghosts.

int NetConnection::**getPacketLoss**()
Returns the percentage of packets lost per tick.

int NetConnection::**getPing**()
Returns the average round trip time (in ms) for the connection. The round trip time is recalculated every time a notify packet is received. Notify packets are used to information the connection that the far end successfully received the sent packet.

int NetConnection::**resolveGhostID**(int *ghostID*)
On the client, convert a ghost ID from this connection to a real SimObject ID. Torque's network ghosting system only exchanges ghost ID's between the server and client. Use this method on the client to discover an object's local SimObject ID when you only have a ghost ID.

Parameters **ghostID** – The ghost ID of the object as sent by the server.

Returns ID of the object, or 0 if it could not be resolved.

Example:

```
%object = ServerConnection.resolveGhostID( %ghostId );
```

int NetConnection::**resolveObjectFromGhostIndex**(int *ghostID*)
On the server, convert a ghost ID from this connection to a real SimObject ID. Torque's network ghosting system only exchanges ghost ID's between the server and client. Use this method on the server to discover an object's local SimObject ID when you only have a ghost ID.

Parameters **ghostID** – The ghost ID of the object as sent by the server.

Returns ID of the object, or 0 if it could not be resolved.

Example:

```
%object = %client.resolveObjectFromGhostIndex( %ghostId );
```

void NetConnection::**setSimulatedNetParams**(float *packetLoss*, int *delay*)
Simulate network issues on the connection for testing.

Parameters

- **packetLoss** – The fraction of packets that will be lost. Ranges from 0.0 (no loss) to 1.0 (complete loss)
- **delay** – Delays packets being transmitted by simulating a particular ping. This is an absolute integer, measured in ms.

void NetConnection::**transmitPaths**()
Sent by the server during phase 2 of the mission download to update motion spline paths. The server transmits all spline motion paths that are within the mission (Path) separate from other objects. This is due to the potentially large number of nodes within each path, which may saturate a packet sent to the client. By managing this step separately, Torque has finer control over how packets are organised vs. doing it during the ghosting stage. Internally a PathManager is used to track all paths defined within a mission on the server, and each one is transmitted using a PathManagerEvent. The client side collects these events and builds the given paths within its own PathManager. This is typically done during the standard mission start phase 2 when following Torque's example mission startup sequence. When a mission is ended, all paths need to be cleared from their respective path managers.

Example:

```
function serverCmdMissionStartPhase2Ack(%client, %seq, %playerDB)
{
   // Make sure to ignore calls from a previous mission load
   if (%seq != $missionSequence || !$MissionRunning)
```

```
         return;
      if (%client.currentPhase != 1.5)
         return;
      %client.currentPhase = 2;

      // Set the player datablock choice
      %client.playerDB = %playerDB;

      // Update mission paths (SimPath), this needs to get there before the objects.
      %client.transmitPaths();

      // Start ghosting objects to the client
      %client.activateGhosting();
   }
```

NetObject Superclass for all ghostable networked objects.

Inherit: SimObject

Description

Introduction To NetObject And Ghosting This class is the basis of the ghost implementation in Torque 3D. Every 3D object is a NetObject. One of the most powerful aspects of Torque's networking code is its support for ghosting and prioritized, most-recent-state network updates. The way this works is a bit complex, but it is immensely efficient. Let's run through the steps that the server goes through for each client in this part of Torque's networking:

- First, the server determines what objects are in-scope for the client. This is done by calling onCameraScopeQuery() on the object which is considered the "scope" object. This is usually the player object, but it can be something else. (For instance, the current vehicle, or a object we're remote controlling.)
- Second, it ghosts them to the client; this is implemented in netGhost.cc.
- Finally, it sends updates as needed, by checking the dirty list and packing updates.

There several significant advantages to using this networking system:

- Efficient network usage, since we only send data that has changed. In addition, since we only care about most-recent data, if a packet is dropped, we don't waste effort trying to deliver stale data.
- Cheating protection; since we don't deliver information about game objects which aren't in scope, we dramatically reduce the ability of clients to hack the game and gain a meaningful advantage. (For instance, they can't find out about things behind them, since objects behind them don't fall in scope.) In addition, since ghost IDs are assigned per-client, it's difficult for any sort of co-ordination between cheaters to occur.

NetConnection contains the Ghost Manager implementation, which deals with transferring data to the appropriate clients and keeping state in synch.

An Example Implementation The basis of the ghost implementation in Torque is NetObject. It tracks the dirty flags for the various states that the object wants to network, and does some other book-keeping to allow more efficient operation of the networking layer.

Using a NetObject is very simple; let's go through a simple example implementation:

```
class SimpleNetObject : public NetObject
{
public:
```

```
  typedef NetObject Parent;
  DECLARE_CONOBJECT(SimpleNetObject);
```

Above is the standard boilerplate code for a Torque class. You can find out more about this in SimObject:

```
char message1[256];
char message2[256];
enum States {
   Message1Mask = BIT(0),
   Message2Mask = BIT(1),
};
```

For our example, we're having two "states" that we keep track of, message1 and message2. In a real object, we might map our states to health and position, or some other set of fields. You have 32 bits to work with, so it's possible to be very specific when defining states. In general, you should try to use as few states as possible (you never know when you'll need to expand your object's functionality!), and in fact, most of your fields will end up changing all at once, so it's not worth it to be too fine-grained. (As an example, position and velocity on Player are controlled by the same bit, as one rarely changes without the other changing, too.):

```
SimpleNetObject()
{
   // In order for an object to be considered by the network system,
   // the Ghostable net flag must be set.
   // The ScopeAlways flag indicates that the object is always scoped
   // on all active connections.
   mNetFlags.set(ScopeAlways | Ghostable);
   dStrcpy(message1, "Hello World 1!");
   dStrcpy(message2, "Hello World 2!");
}
```

Here is the constructor. Here, you see that we initialize our net flags to show that we should always be scoped, and that we're to be taken into consideration for ghosting. We also provide some initial values for the message fields:

```
U32 packUpdate(NetConnection *, U32 mask, BitStream *stream)
{
   // check which states need to be updated, and update them
   if(stream->writeFlag(mask & Message1Mask))
      stream->writeString(message1);
   if(stream->writeFlag(mask & Message2Mask))
      stream->writeString(message2);


   // the return value from packUpdate can set which states still
   // need to be updated for this object.
   return 0;
}
```

Here's half of the meat of the networking code, the packUpdate() function. (The other half, unpackUpdate(), we'll get to in a second.) The comments in the code pretty much explain everything, however, notice that the code follows a pattern of if(writeFlag(mask & StateMask)) { ... write data ... }. The packUpdate()/unpackUpdate() functions are responsible for reading and writing the dirty bits to the bitstream by themselves:

```
void unpackUpdate(NetConnection *, BitStream *stream)
{
   // the unpackUpdate function must be symmetrical to packUpdate
   if(stream->readFlag())
   {
      stream->readString(message1);
      Con::printf("Got message1: %s", message1);
```

```
   }
   if(stream->readFlag())
   {
      stream->readString(message2);
      Con::printf("Got message2: %s", message2);
   }
}
```

The other half of the networking code in any NetObject, unpackUpdate(). In our simple example, all that the code does is print the new messages to the console; however, in a more advanced object, you might trigger animations, update complex object properties, or even spawn new objects, based on what packet data you unpack:

```
void setMessage1(const char *msg)
{
   setMaskBits(Message1Mask);
   dStrcpy(message1, msg);
}
void setMessage2(const char *msg)
{
   setMaskBits(Message2Mask);
   dStrcpy(message2, msg);
}
```

Here are the accessors for the two properties. It is good to encapsulate your state variables, so that you don't have to remember to make a call to setMaskBits every time you change anything; the accessors can do it for you. In a more complex object, you might need to set multiple mask bits when you change something; this can be done using the | operator, for instance, setMaskBits(Message1Mask | Message2Mask); if you changed both messages:

```
IMPLEMENT_CO_NETOBJECT_V1(SimpleNetObject);


ConsoleMethod(SimpleNetObject, setMessage1, void, 3, 3, "(string msg) Set message 1.")
{
   object->setMessage1(argv[2]);
}


ConsoleMethod(SimpleNetObject, setMessage2, void, 3, 3, "(string msg) Set message 2.")
{
   object->setMessage2(argv[2]);
}
```

Finally, we use the NetObject implementation macro, IMPLEMENT_CO_NETOBJECT_V1(), to implement our NetObject. It is important that we use this, as it makes Torque perform certain initialization tasks that allow us to send the object over the network. IMPLEMENT_CONOBJECT() doesn't perform these tasks, see the documentation on AbstractClassRep for more details.

Methods

void `NetObject::clearScopeToClient` (NetConnection *client*)

Undo the effects of a scopeToClient() call.

Parameters **`client`** – The connection to remove this object's scoping from

int `NetObject::getClientObject` ()

Returns a pointer to the client object when on a local connection. Short-Circuit-Networking: this is only valid for a local-client / singleplayer situation.

Returns ID of the client object.

Example:

```
// Psuedo-code, some values left out for this example
%node = newParticleEmitterNode(){};
%clientObject = %node.getClientObject();
if(isObject(%clientObject)
   %clientObject.setTransform("0 0 0");
```

int NetObject::**getGhostID**()
Get the ghost index of this object from the server.

Returns on the server

Example:

```
%ghostID = LocalClientConnection.getGhostId( %serverObject );
```

int NetObject::**getServerObject**()
Returns a pointer to the client object when on a local connection. Short-Circuit-Netorking: this is only valid for a local-client / singleplayer situation.

Returns ID of the server object.

Example:

```
// Psuedo-code, some values left out for this example
%node = newParticleEmitterNode(){};
%serverObject = %node.getServerObject();
if(isObject(%serverObject)
   %serverObject.setTransform("0 0 0");
```

bool NetObject::**isClientObject**()
Called to check if an object resides on the clientside.

Returns True if the object resides on the client, false otherwise.

bool NetObject::**isServerObject**()
Checks if an object resides on the server.

Returns True if the object resides on the server, false otherwise.

void NetObject::**scopeToClient**(NetConnection *client*)
Cause the NetObject to be forced as scoped on the specified NetConnection .

Parameters **client** – The connection this object will always be scoped to

Example:

```
// Called to create new cameras in TorqueScript
// %this - The active GameConnection
// %spawnPoint - The spawn point location where we creat the camera
function GameConnection::spawnCamera(%this, %spawnPoint)
{
   // If this connections camera existsif(isObject(%this.camera))
   {
      // Add it to the mission group to be cleaned up later
      MissionCleanup.add( %this.camera );

      // Force it to scope to the client side
      %this.camera.scopeToClient(%this);
   }
}
```

void NetObject::**setScopeAlways**()
Always scope this object on all connections. The object is marked as ScopeAlways and is immediately ghosted to all active connections. This function has no effect if the object is not marked as Ghostable.

SimpleMessageEvent A very simple example of a network event.

Description This object exists purely for instructional purposes. It is primarily geared toward developers that wish to understand the inner-working of Torque 3D's networking system. This is not intended for actual game development.

Methods
static void SimpleMessageEvent::**msg**(NetConnection *con*, string *message*)
Send a SimpleMessageEvent message to the specified connection. The far end that receives the message will print the message out to the console.

Parameters

- **con** – The unique ID of the connection to transmit to
- **message** – The string containing the message to transmit

Example:

```
// Send a message to the other end of the given
NetConnectionSimpleMessageEvent::msg( %conn, "A message from me!");

// The far end will see the following in the console
// (Note: The number may be something other than 1796 as it is the SimObjectID
// of the received event)
//
// RMSG 1796  A message from me!
```

SimpleNetObject A very simple example of a class derived from NetObject.

Inherit: NetObject

Description This object exists purely for instructional purposes. It is primarily geared toward developers that wish to understand the inner-working of Torque 3D's networking system. This is not intended for actual game development.

Example:

```
// On the server, create a new SimpleNetObject.  This is a ghost always
// object so it will be immediately ghosted to all connected clients.
$s = newSimpleNetObject();

// All connected clients will see the following in their console:
//
// Got message: Hello World!
```

Methods
void SimpleNetObject::**setMessage**(string *msg*)
Sets the internal message variable. SimpleNetObject is set up to automatically transmit this new message to all connected clients. It will appear in the clients' console.

Parameters **msg** – The new message to send

Example:

```
// On the server, create a new SimpleNetObject.  This is a ghost always
// object so it will be immediately ghosted to all connected clients.
$s = newSimpleNetObject();

// All connected clients will see the following in their console:
//
// Got message: Hello World!
// Now again on the server, change the message.  This will cause it to
// be sent to all connected clients.
$s.setMessage("A new message from me!");

// All connected clients will now see in their console:
//
// Go message: A new message from me!
```

TCPObject Allows communications between the game and a server using TCP/IP protocols.

Inherit: SimObject

Description To use TCPObject you set up a connection to a server, send data to the server, and handle each line of the server's response using a callback. Once you are done communicating with the server, you disconnect.

TCPObject is intended to be used with text based protocols which means you'll need to delineate the server's response with an end-of-line character. i.e. the newline character n. You may optionally include the carriage return character r prior to the newline and TCPObject will strip it out before sending the line to the callback. If a newline character is not included in the server's output, the received data will not be processed until you disconnect from the server (which flushes the internal buffer).

TCPObject may also be set up to listen to a specific port, making Torque into a TCP server. When used in this manner, a callback is received when a client connection is made. Following the outside connection, text may be sent and lines are processed in the usual manner.

If you want to work with HTTP you may wish to use HTTPObject instead as it handles all of the HTTP header setup and parsing.

Example:

```
// In this example well retrieve the new forum threads RSS
// feed from garagegames.com.  As were using TCPObject, the
// raw text response will be received from the server, including
// the HTTP header.

// Define callbacks for our specific TCPObject using our instances
// name (RSSFeed) as the namespace.

// Handle an issue with resolving the servers name
function RSSFeed::onDNSFailed(%this)
{
   // Store this state
   %this.lastState = "DNSFailed";

   // Handle DNS failure
}

function RSSFeed::onConnectFailed(%this)
{
```

```
   // Store this state
   %this.lastState = "ConnectFailed";

   // Handle connection failure
}

function RSSFeed::onDNSResolved(%this)
{
   // Store this state
   %this.lastState = "DNSResolved";

}

function RSSFeed::onConnected(%this)
{
   // Store this state
   %this.lastState = "Connected";

}

function RSSFeed::onDisconnect(%this)
{
   // Store this state
   %this.lastState = "Disconnected";

}

// Handle a line from the server
function RSSFeed::onLine(%this, %line)
{
   // Print the line to the consoleecho( %line );
}

// Create the TCPObject
%rss = newTCPObject(RSSFeed);

// Define a dynamic field to store the last connection state
%rss.lastState = "None";

// Connect to the server
%rss.connect("www.garagegames.com:80");

// Send the RSS feed request to the server.  Response will be// handled in onLine() callback above
%rss.send("GET /feeds/rss/threads HTTP/1.1\r\nHost: www.garagegames.com\r\n\r\n");
```

Methods

void TCPObject::**connect**(string *address*)

Connect to the given address.

Parameters **address** – Server address (including port) to connect to.

Example:

```
    // Set the address.
    %address = "www.garagegames.com:80";

    // Inform this TCPObject to connect to the specified address.
    %thisTCPObj.connect(%address);
```

void TCPObject::**disconnect**()
Disconnect from whatever this TCPObject is currently connected to, if anything.

Example:

```
// Inform this TCPObject to disconnect from anything it is currently connected to.
%thisTCPObj.disconnect();
```

void TCPObject::**listen**(int *port*)
Start listening on the specified port for connections. This method starts a listener which looks for incoming TCP connections to a port. You must overload the onConnectionRequest callback to create a new TCPObject to read, write, or reject the new connection.

Parameters **port** – Port for this TCPObject to start listening for connections on.

Example:

```
// Create a listener on port 8080.
newTCPObject( TCPListener );
TCPListener.listen( 8080 );

function TCPListener::onConnectionRequest( %this, %address, %id )
{
   // Create a new object to manage the connection.newTCPObject( TCPClient, %id );
}

function TCPClient::onLine( %this, %line )
{
   // Print the line of text from client.echo( %line );
}
```

void TCPObject::**onConnected**()
Called whenever a connection is established with a server.

void TCPObject::**onConnectFailed**()
Called whenever a connection has failed to be established with a server.

void TCPObject::**onConnectionRequest**(string *address*, string *ID*)
Called whenever a connection request is made. This callback is used when the TCPObject is listening to a port and a client is attempting to connect.

Parameters

- **address** – Server address connecting from.
- **ID** – Connection ID

void TCPObject::**onDisconnect**()
Called whenever the TCPObject disconnects from whatever it is currently connected to.

void TCPObject::**onDNSFailed**()
Called whenever the DNS has failed to resolve.

void TCPObject::**onDNSResolved**()
Called whenever the DNS has been resolved.

void TCPObject::**onLine**(string *line*)
Called whenever a line of data is sent to this TCPObject . This callback is called when the received data contains a newline n character, or the connection has been disconnected and the TCPObject's buffer is flushed.

Parameters **line** – Data sent from the server.

void TCPObject::**send**(string *data*)

Transmits the data string to the connected computer. This method is used to send text data to the connected computer regardless if we initiated the connection using connect() , or listening to a port using listen() .

Parameters **data** – The data string to send.

Example:

```
// Set the command data
%data = "GET " @ $RSSFeed::serverURL @ " HTTP/1.0\r\n";
%data = %data @ "Host: " @ $RSSFeed::serverName @ "\r\n";
%data = %data @ "User-Agent: " @ $RSSFeed::userAgent @ "\r\n\r\n"
// Send the command to the connected server.
%thisTCPObj.send(%data);
```

BanList Used for kicking and banning players from a server.

Description There is only a single instance of BanList. It is very important to note that you do not ever create this object in script like you would other game play objects. You simply reference it via namespace.

For this to be used effectively, make sure you are hooking up other functions to BanList. For example, functions like GameConnection::onConnectRequestRejected(this, msg) and function GameConnection::onConnectRequest are excellent places to make use of the BanList. Other systems can be used in conjunction for strict control over a server

Methods

static void BanList::**add**(int *uniqueId*, string *transportAddress*, int *banLength*)

Ban a user for banLength seconds.

Parameters

- **uniqueId** – Unique ID of the player.
- **transportAddress** – Address from which the player connected.
- **banLength** – Time period over which to ban the player.

Example:

```
// Kick someone off the server
// %client - This is the connection to the person we are kicking
function kick(%client)
{
      // Let the server know what happened
      messageAll( MsgAdminForce, \c2The Admin has kicked %1., %client.playerName);

      // If it is not an AI Player, execute the ban.
      if (!%client.isAIControlled())
         BanList::add(%client.guid, %client.getAddress(), $pref::Server::KickBanTime);

      // Let the player know they messed up
      %client.delete("You have been kicked from this server");
}
```

static void BanList::**addAbsolute**(int *uniqueId*, string *transportAddress*, int *banTime*)

Ban a user until a given time.

Parameters

- **uniqueId** – Unique ID of the player.

- **transportAddress** – Address from which the player connected.
- **banTime** – Time at which they will be allowed back in.

Example:

```
// Kick someone off the server
// %client - This is the connection to the person we are kicking
function kick(%client)
{
      // Let the server know what happened
      messageAll( MsgAdminForce, \c2The Admin has kicked %1., %client.playerName);

      // If it is not an AI Player, execute the ban.
      if (!%client.isAIControlled())
         BanList::addAbsolute(%client.guid, %client.getAddress(), $pref::Server::KickBanTime);

      // Let the player know they messed up
      %client.delete("You have been kicked from this server");
}
```

static void BanList::export(string filename)

Dump the banlist to a file.

Parameters **filename** – Path of the file to write the list to.

Example:

```
BanList::Export("./server/banlist.cs");
```

static bool BanList::**isBanned**(int *uniqueId*, string *transportAddress*)

Is someone banned?

Parameters

- **uniqueId** – Unique ID of the player.
- **transportAddress** – Address from which the player connected.

Example:

```
// This script function is called before a client connection
// is accepted.  Returning  will accept the connection,
// anything else will be sent back as an error to the client.
// All the connect args are passed also to onConnectRequest
function GameConnection::onConnectRequest( %client, %netAddress, %name )
{
     // Find out who is trying to connect
     echo("Connect request from: " @ %netAddress);

     // Are they allowed in?
     if(BanList::isBanned(%client.guid, %netAddress))
        return"CR_YOUAREBANNED";

     // Is there room for an unbanned player?
     if($Server::PlayerCount >= $pref::Server::MaxPlayers)
        return"CR_SERVERFULL";
     return ;
}
```

static void BanList::**removeBan**(int *uniqueId*, string *transportAddress*)

Unban someone.

Parameters

- **uniqueId** – Unique ID of the player.
- **transportAddress** – Address from which the player connected.

Example:

```
BanList::removeBan(%userID, %ipAddress);
```

Functions

string **addTaggedString** (string *str*)
Use the addTaggedString function to tag a new string and add it to the NetStringTable.

Parameters **str** – The string to be tagged and placed in the NetStringTable. Tagging ignores case, so tagging the same string (excluding case differences) will be ignored as a duplicated tag.

Returns Returns a string(containing a numeric value) equivalent to the string ID for the newly tagged string

string **buildTaggedString** (string *format*, ...)
Build a string using the specified tagged string format. This function takes an already tagged string (passed in as a tagged string ID) and one or more additional strings. If the tagged string contains argument tags that range from %1 through %9, then each additional string will be substituted into the tagged string. The final (non-tagged) combined string will be returned. The maximum length of the tagged string plus any inserted additional strings is 511 characters.

Parameters

- **format** – A tagged string ID that contains zero or more argument tags, in the form of %1 through %9.
- **...** – A variable number of arguments that are insterted into the tagged string based on the argument tags within the format string.

Returns An ordinary string that is a combination of the original tagged string with any additional strings passed in inserted in place of each argument tag.

Example:

```
// Create a tagged string with argument tags
%taggedStringID = addTaggedString("Welcome %1 to the game!");

// Some point later, combine the tagged string with some other string
%string = buildTaggedString(%taggedStringID, %playerName);
echo(%string);
```

void **closeNetPort** ()
Closes the current network port.

void **commandToClient** (NetConnection *client*, string *func*, ...)
Send a command from the server to the client.

Parameters

- **client** – The numeric ID of a client GameConnection
- **func** – Name of the client function being called
- **...** – Various parameters being passed to client command

Example:

```
// Set up the client command.  Needs to be executed on the client, such as
// within scripts/client/client.cs
// Update the Ammo Counter with current ammo, if not any then hide the counter.
function clientCmdSetAmmoAmountHud(%amount)
{
   if (!%amount)
     AmmoAmount.setVisible(false);
   else
   {
     AmmoAmount.setVisible(true);
     AmmoAmount.setText("Ammo: "@%amount);
   }
}

// Call it from a server function.
// Needs to be executed on the server,
// such as within scripts/server/game.cs
function GameConnection::setAmmoAmountHud(%client, %amount)
{
   commandToClient(%client, SetAmmoAmountHud, %amount);
}
```

void **commandToServer** (string *func*, ...)

Send a command to the server.

Parameters

- **func** – Name of the server command being called
- **...** – Various parameters being passed to server command

Example:

```
// Create a standard function.
// Needs to be executed on the client, such
// as within scripts/client/default.bind.cs
function toggleCamera(%val)
{
   // If key was down, call a server command named ToggleCamera
   if (%val)
      commandToServer(ToggleCamera);
}

// Server command being called from above.  Needs to be executed on the
// server, such as within scripts/server/commands.cs
function serverCmdToggleCamera(%client)
{
   if (%client.getControlObject() == %client.player)
   {
     %client.camera.setVelocity("0 0 0");
     %control = %client.camera;
   }
   else
   {
     %client.player.setVelocity("0 0 0");
     %control = %client.player;
  }
   %client.setControlObject(%control);
```

```
      clientCmdSyncEditorGui();
   }
```

string **detag** (string *str*)
Returns the string from a tag string. Should only be used within the context of a function that receives a tagged string, and is not meant to be used outside of this context. Use getTaggedString() to convert a tagged string ID back into a regular string at any time.

Example:

```
// From scripts/client/message.cs
function clientCmdChatMessage(%sender, %voice, %pitch, %msgString, %a1, %a2, %a3, %a4, %a5, %a6,
{
   onChatMessage(detag(%msgString), %voice, %pitch);
}
```

void **DNetSetLogging** (bool *enabled*)
Enables logging of the connection protocols. When enabled a lot of network debugging information is sent to the console.

Parameters **enabled** – True to enable, false to disable

void **dumpNetStats** ()
Dumps network statistics for each class to the console. The returned avg , min and max values are in bits sent per update. The num value is the total number of events collected.

void **dumpNetStringTable** ()
Dump the current contents of the networked string table to the console. The results are returned in three columns. The first column is the network string ID. The second column is the string itself. The third column is the reference count to the network string.

string **getTag** (string *textTagString*)
Extracts the tag from a tagged string. Should only be used within the context of a function that receives a tagged string, and is not meant to be used outside of this context.

Parameters **textTagString** – The tagged string to extract.

Returns The tag ID of the string.

string **getTaggedString** (string *tag*)
Use the getTaggedString function to convert a tag to a string. This is not the same as detag() which can only be used within the context of a function that receives a tag. This function can be used any time and anywhere to convert a tag to a string.

Parameters **tag** – A numeric tag ID.

Returns The string as found in the Net String table.

void **onDataBlockObjectReceived** (int *index*, int *total*)
Called on the client each time a datablock has been received. This callback is typically used to notify the player of how far along in the datablock download process they are.

Parameters

- **index** – The index of the datablock just received.
- **total** – The total number of datablocks to be received.

void **onLightManagerActivate** (string *name*)
A callback called by the engine when a light manager is activated.

Parameters **name** – The name of the light manager being activated.

void **onLightManagerDeactivate**(string *name*)
A callback called by the engine when a light manager is deactivated.

Parameters **name** – The name of the light manager being deactivated.

void **pathOnMissionLoadDone**()
Load all Path information from the mission. This function is usually called from the loadMissionStage2() server-side function after the mission file has loaded. Internally it places all Paths into the server's PathManager. From this point the Paths are ready for transmission to the clients.

Example:

```
// Inform the engine to load all Path information from the mission.
pathOnMissionLoadDone();
```

void **removeTaggedString**(int *tag*)
Remove a tagged string from the Net String Table.

Parameters **tag** – The tag associated with the string

bool **setNetPort**(int *port*, bool *bind*)
Set the network port for the game to use.

Parameters

- **port** – The port to use.
- **bind** – True if bind() should be called on the port.

Returns True if the port was successfully opened. This will trigger a windows firewall prompt. If you don't have firewall tunneling tech you can set this to false to avoid the prompt.

Variables

void **allowConnections**
Sets whether or not the global NetInterface allows connections from remote hosts. allowConnections(bool allow);

Parameters **allow** – Set to true to allow remote connections.

int $pref::Net::LagThreshold
How long between received packets before the client is considered as lagging (in ms). This is used by GameConnection to determine if the client is lagging. If the client is indeed lagging, setLagIcon() is called to inform the user in some way. i.e. display an icon on screen.

int $Stats::netBitsReceived
The number of bytes received during the last packet process operation.

int $Stats::netBitsSent
The number of bytes sent during the last packet send operation.

int $Stats::netGhostUpdates
The total number of ghosts added, removed, and/or updated on the client during the last packet process operation.

int $pref::Net::PacketRateToClient
Sets how often packets are sent from the server to a client. It is possible to control how often packets may be sent to the clients. This may be used to throttle the amount of bandwidth being used, but should be adjusted with caution. The actual formula used to calculate the delay between sending packets to a client is:

int $pref::Net::PacketRateToServer
Sets how often packets are sent from the client to the server. It is possible to control how often packets may be

sent to the server. This may be used to throttle the amount of bandwidth being used, but should be adjusted with caution. The actual formula used to calculate the delay between sending packets to the server is:

int $pref::Net::PacketSize
Sets the maximum size in bytes an individual network packet may be. It is possible to control how large each individual network packet may be. Increasing its size from the default allows for more data to be sent on each network send. However, this value should be changed with caution as too large a value will cause packets to be split up by the networking platform or hardware, which is something Torque cannot handle. A minimum packet size of 100 bytes is enforced in the source code. There is no enforced maximum. The default value is 200 bytes.

Platform

Enumeration

enum MBButtons
Which buttons to display on a message box.

Parameters

- **Ok** –
- **OkCancel** –
- **RetryCancel** –
- **SaveDontSave** –
- **SaveDontSaveCancel** –

enum MBIcons
What icon to show on a message box.

Parameters

- **Information** –
- **Warning** –
- **Stop** –
- **Question** –

enum MBReturnVal
Return value for messageBox() indicating which button was pressed by the user.

Parameters

- **OK** –
- **Cancelled** –
- **Retry** –
- **DontSave** –

Functions

bool **displaySplashWindow**(string *path* = "art/gui/splash.bmp")
Display a startup splash window suitable for showing while the engine still starts up.

Note: This is currently only implemented on Windows.

Returns True if the splash window could be successfully initialized.

int **getRealTime**()

Returns Return the current real time in milliseconds. Real time is platform defined; typically time since the computer booted.

int **getSimTime**()

Sim time is time since the game started.

Returns Return the current sim time in milliseconds.

bool **getWebDeployment**()

Test whether Torque is running in web-deployment mode.

In this mode, Torque will usually run within a browser and certain restrictions apply (e.g. Torque will not be able to enter fullscreen exclusive mode).

Returns True if Torque is running in web-deployment mode.

void **gotoWebPage**(string *address*)

Open the given URL or file in the user's web browser.

Parameters **address** – The address to open. If this is not prefixed by a protocol specifier ("...://"), then the function checks whether the address refers to a file or directory and if so, prepends "file://" to adress; if the file check fails, "http://" is prepended to address.

Example:

```
gotoWebPage( "http://www.garagegames.com" );
```

bool **isDebugBuild**()

Test whether the engine has been compiled with TORQUE_DEBUG, i.e. if it includes debugging functionality.

Returns True if this is a debug build; false otherwise.

bool **isShippingBuild**()

Test whether the engine has been compiled with TORQUE_SHIPPING, i.e. in a form meant for final release.

Returns True if this is a shipping build; false otherwise.

bool **isToolBuild**()

Test whether the engine has been compiled with TORQUE_TOOLS, i.e. if it includes tool-related functionality.

Returns True if this is a tool build; false otherwise.

int **messageBox**(string *title*, string *message*, MBButtons *buttons* = MBOkCancel, MBIcons *icons* = MIInformation)

Display a modal message box using the platform's native message box implementation.

Parameters

- **title** – The title to display on the message box window.
- **message** – The text message to display in the box.
- **buttons** – Which buttons to put on the message box.
- **icons** – Which icon to show next to the message.

Returns One of $MROK, $MRCancel, $MRRetry, and $MRDontSave identifying the button that the user pressed.

Example:

```
    messageBox( "Error", "" );
```

void **playJournal**(string *filename*)
Begin playback of a journal from a specified field.

Parameters **filename** – Name and path of file journal file

void **quit**()
Shut down the engine and exit its process. This function cleanly uninitializes the engine and then exits back to the system with a process exit status indicating a clean exit.

void **quitWithErrorMessage**(string *message*)
Display an error message box showing the given message and then shut down the engine and exit its process. This function cleanly uninitialized the engine and then exits back to the system with a process exit status indicating an error.

Parameters **message** – The message to log to the console and show in an error message box.

void **saveJournal**(string *filename*)
Save the journal to the specified file.

bool **shellExecute**(string *executable*, string *args*, string *directory*)
Launches an outside executable or batch file.

Parameters

- **executable** – Name of the executable or batch file
- **args** – Optional list of arguments, in string format, to pass to the executable
- **directory** – Optional string containing path to output or shell

Variables

int $MRCancel
Determines the cancel button press state in a message box.

int $MRDontSave
Determines the don't save button press state in a message box.

int $MROk
Determines the ok button press state in a message box.

int $MRRetry
Determines the retry button press state in a message box.

int $platform::backgroundSleepTime
Controls processor time usage when the game window is out of focus.

int $platform::timeManagerProcessInterval
Controls processor time usage when the game window is in focus.

Localization

Localization of games to multiple languages.

Classes

LangTable Provides the code necessary to handle the low level management of the string tables for localization.

Inherit: SimObject

Description One LangTable is created for each mod, as well as one for the C++ code. LangTable is responsible for obtaining the correct strings from each and relaying it to the appropriate controls.

Methods

int `LangTable::addLanguage`(string *filename*)
Adds a language to the table.

Parameters

- **`filename`** – Name and path to the language file
- **`languageName`** – Optional name to assign to the new language entry

Returns True If file was successfully found and language created

int `LangTable::getCurrentLanguage`()
Get the ID of the current language table.

Returns Numerical ID of the current language table

string `LangTable::getLangName`(int *language*)
Return the readable name of the language table.

Parameters **`language`** – Numerical ID of the language table to access

Returns String containing the name of the table, NULL if ID was invalid or name was never specified

int `LangTable::getNumLang`()
Used to find out how many languages are in the table.

Returns Size of the vector containing the languages, numerical

string `LangTable::getString`(string *filename*)
Grabs a string from the specified table. If an invalid is passed, the function will attempt to to grab from the default table

Parameters **`filename`** – Name of the language table to access

Returns Text from the specified language table, "" if ID was invalid and default table is not set

void `LangTable::setCurrentLanguage`(int *language*)
Sets the current language table for grabbing text.

Parameters **`language`** – ID of the table

void `LangTable::setDefaultLanguage`(int *language*)
Sets the default language table.

Parameters **`language`** – ID of the table

Description

Localizing large applications can be an extremely time consuming and aggravating process. This manual is intended to guide you through the changes made to Torque to support localization and help you reduce the ongoing headaches

in day-to-day development in a localized codebase. This manual assumes you are at least familiar with Torque Script. C++ and localization experience is not necessary.

Languages and String Tables Lets say you want to print some text to the console. From script, you'd usually just write something like:

echo("Hello, World!"); This is all well and good, if all you care about is English. What do you do if the user wants your game to talk French? You could do something like the following:

```
switch($Pref::Language)
{
   case $ENGLISH:
      echo("Hello, World!");
   case $FRENCH:
      echo("Salut, Monde!");
... other languages here ...
}
```

Sure, this works. It's also a complete pain in the backside to keep up to date, let alone that you'd have to repeat the switch for every string you use. In the C++ code alone there is upwards of 4000 strings that are candidates for localization, so this method is clearly not even close to an option.

String tables are the answer to this problem. A string table is an array of strings that you can reference based on an ID instead of specifying the string directly in the source code. Changing which language the string is displayed in is simply a matter of using a string table that contains the strings in the language you wish to display. The above example can be reduced down to the following:

```
echo(L($STR_HELLO_WORLD));
```

The code is not much different to how you wrote it before, but the string will now be displayed in the correct language based on the user's selection. This does come with a price, however. You now have to create and maintain string tables for all the languages you support, which can be a big headache. Torque tries to make this as simple as possible, and some tips for creating and maintaining these string tables are given later.

Script Interface Torque contains a lot of strings, not just in the C++ code but in the script code too. For example, all your GUIs are created in script and must also be localized. To further complicate matters the scripts are contained in mod directories, each of which is a separate entity.

Strings in Script It would be cumbersome and difficult to maintain one string table for all the strings, so Torque doesn't try. Each mod has it's own string table, as well as the core C++ code. You would be forgiven for thinking that this leads to having to write code as follows:

```
echo(L($StarterFPSTable, $STR_HELLO_WORLD));
```

I don't know about you, but I'm a lazy programmer and that's just too much typing for me to put up with every time I want to display a string. Instead, you just use:

```
echo(L($STR_HELLO_WORLD));
```

Torque takes care of the rest: the correct table will be used based on which mod the code is executed from.

Loading strings and specifying languages The new LangTable class, accessible in script as well as in C++, provides the code necessary to handle the low level management of the string tables and obtaining the correct strings from them. One LangTable is created for each mod, as well as one for the C++ code.

An important portion of the localization system is implemented in script. This support code is responsible for managing the language tables and provides you with a simpler API than would be the case using LangTable directly. To provide this API, a new mod called "lang" has been added. The lang mod works a little like the common mod in that it just provides code needed by all mods. As the localization code is needed very early in the initialisation code, adding it to the common mod was not possible.

To ease loading of language tables, language maps are provided. These map the displayed name of the language to a string table on disk and are used to provide automatic loading across all mods. You specify the language map in the lang/languages.cs script, for example:

```
addLanguageMap("English", "english.lso");
addLanguageMap("Pony", "pony.lso");
```

The first argument to addLanguageMap() is the name of the language as it will be displayed to the user, and the second is the filename of the language. The filename will be concatenated onto the mod path to find the actual filename name of the table to load.

To load the language tables for the current mod, in the onStart() function for the mod, before any other code, use the following:

```
table = loadModLangTables("starter.fps/lang/tables");
setModLangTable(table);
loadModLangTables() loads the tables from the specified directory based on the information in the lar

 // setup localization:
   $I18N::starter.fps = new LangTable();
   exec("starter.fps/lang/tables/German.cs");
   exec("starter.fps/lang/tables/English.cs");
   $I18N::starter.fps.addLanguage("starter.fps/lang/tables/German.lso", "German");
   $I18N::starter.fps.addLanguage("starter.fps/lang/tables/English.lso", "English");
   $I18N::starter.fps.setDefaultLanguage(0); // German is default here
   $I18N::starter.fps.setCurrentLanguage(0); // German is current
$I18N::starter.fps.setCurrentLanguage(1); // this would set the current language to english
```

Handling user language preferences In the Torque demo, the user is provided two ways to change their language settings. If they have not already chosen a language, for example the first time they run the game, a dialog will pop up after the splash screens asking them to choose a language. The language can also be changed at any time from the options dialog. Note that some strings will not be in the new language until the game is restarted. There will be further discussion on this topic in later chapters.

When the language is changed, the onChangeLanguage() function is called. This should be overriden for each mod in the same way as onStart(). The following code can be used for all mods:

```
function onChangeLanguage()
{
   setCurrentLanguage(getModLangTable(), $pref::I18N::language);
   Parent::onChangeLanguage();
}
```

To change the language for all mods, you can simply use the setLanguage() function. setLanguage() takes one argument, the display name of the language to switch to.

Localizing the GUI The GUI provides some interesting challenges for localization. Since GUI files are just script files that create the objects for the GUI, text is specified as a member field of the object. For example:

```
new GuiButtonCtrl() {
      text = "Quit!";
```

```
... other fields ...
};
```

As this is just a script, you can localize it easily by using the L() function in the same way you would with other scripts, for example:

```
new GuiButtonCtrl() {
      text = L($STR_QUIT);
... other fields ...
};
```

Unfortunately, this is not a viable solution. If you save that GUI from the GUI editor, the call to L() will be replaced with the text it returned when creating the GUI, reverting it to the first example. The only way around that is to edit all your GUIs by hand, which is not a particularly pleasant option when you have an editor to do it for you.

In order to sensibly support editing of GUIs, most of the GUI controls that support the text field now also support a textID field. This is the name of the ID variable for the string, for example:

```
new GuiButtonCtrl() {
      textID = "STR_QUIT";
... other fields ...
};
```

In addition, a setTextID() method was added to all controls that expose a setText() method to allow you to set the text at run time. It functions the same as the setText() method, except it works with IDs:

```
obj.setTextID("STR_ABOUT");
```

The only caveat is you must specify the name of the variable as a string, and not the variable itself. If you used the variable, for example obj.setText($STR_ABOUT), then the textID field would get set to the numeric ID, and the GUI control would be looking for a variable called, for example, 45.

If you don't specify an ID, or the ID you do specify is invalid for whatever reason, then the control will simply use the text field as before.

Specifying the language table Unfortunately, there is not enough context available in the GUI system to determine which mod the control was defined in, so the tricks used to make the L() function work cannot be used in the GUI.

To work around the problem, a field was added to GuiControl to specify the table to use, called langTableMod, which is simply the name of the mod to get the language table from. Usually, this will be the same as the mod that the .gui file resides in.

To avoid having to specify the table for every control that needs it, a crude form of inheritance was implemented. If langTableMod is not specified for a control, the control's parent will be checked. This means you only have to specify the table in the root control for the GUI. However, you could also specify a different table on a per-control basis if needed, for example if you wanted to use a string from the language table for common. The following example GUI illustrates this better:

```
// The value that you set langTableMod to is the same as name after "$I18N::".
// If you use "$I18N::bob" then langTableMod = "bob".
new GuiChunkedBitmapCtrl(MainMenuGui)
{
   profile = "GuiContentProfile";
   horizSizing = "width";
   vertSizing = "height";
   position = "0 0";
   extent = "640 480";
   minExtent = "8 8";
   visible = "1";
```

```
   langTableMod = "starter.fps";
   bitmap = "./background";
   useVariable = "0";
   tile = "0";
   helpTag = "0";
   new GuiButtonCtrl() {
      profile = "GuiButtonProfile";
      horizSizing = "right";
      vertSizing = "top";
      position = "36 413";
      extent = "110 20";
      minExtent = "8 8";
      visible = "1";
      command = "quit();";
      textID = "STR_QUIT";
      groupNum = "-1";
      buttonType = "PushButton";
         helpTag = "0";
   };
};
```

Refreshing the GUI When the user has chosen a new language, it would be nice to be able to get the GUI to refresh without forcing them to restart the game. One nice side effect of the updates to the GUI system is that we can do that.

The way the textID field was implemented is it checks for a valid ID, then if it has one it passes the string directly to setText(). The initial update of the text field is done on onWake(), so to get the control to update you only have to cause it to get an onWake() event.

It turns out that you can trick Torque into resending onWake() events quite simply, without having to write a ton of code. For the main canvas, you can simply do:

```
if(isObject(Canvas))
Canvas.setContent(Canvas.getContent());
```

When implementing the options dialog, I found that this wont refresh the dialogs. To work around that, when you click apply the script will pop the dialog then re-push it, as follows:

```
Canvas.popDialog(optionsDlg);
Canvas.pushDialog(optionsDlg);
```

Of course, some things will not be possible to update when you change the language. It may be worth considering requiring users to restart the game for language changes to take effect for the sake of consistency.

C++ Interface The C++ interface is largely in flux at the moment pending fixing the string extractor to cope with multi-line strings. In terms of how you reference strings, it will be largely the same as the script interface.

Caveats This will have caveats for the localization thingy.

The Localization Tools The current toolset consists of the command line language compiler, called langc, which is used to compile a language file into the various formats needed by the engine.

Language Files Language files are simple text files containing the identifiers and strings. An example follows:

```
TEST_STR_1=This is a test
TEST_STR_2=and so is this!
```

The string definitions are similar to variable definitions in C++ and Torque Script, except you do not need quotes or a semicolon. The "variable" becomes the ID as it will be referred to in the game code.

If you need the string to span multiple lines, you can use the usual as you would in C++. Note that currently you can't use a trailing on the line to continue to the next line. (Note: this should be fixed)

You can use any amount of white space on either side of the equals sign. Two options control how the compiler handles whitespace. By default, any whitespace after the equals (leading spaces) will be stripped and any space on the end of the string (trailing spaces) will be left alone. Two options control this behaviour:

S Strip leading space. When this option is specified, any space after the = sign will become part of the string.

T Strip trailing space. When specified, space on the end of the string will be stripped.

Comments may be inserted in the language file at the beginning of a line only. You can use #, ; or // to delimit comments. For example:

```
# This is a comment
; So is this
And this
```

Compiling Language Files Before a language file can be used in the engine, it has to be compiled into a .lso file. As the process is slightly different depending on whether you are compiling the default english language file or a translation, it is not possible to do this automatically, as is done with scripts.

The default language is usually English. There is no hard and fast restriction on which language is the default, but for the purposes of documentation it will be assumed that English is your default language. It is worth pointing out that because strings that are missing from a particular language will use the default language, and some strings (particularly in the C++ code) are not localized, not using English for the default language may cause inconsistencies.

Language files are compiled with the langc tool, found in the tools directory. Like other Torque tools, langc must be compiled before you can use it.

langc can also produce some other related files, for example header files for the IDs and templates for translations.

In its simplest usage, langc requires the name of the language file to compile and an output basename, for example:

```
> langc english.lang english
```

The first argument is the name of the language file and the second is the basename for output files. langc can create a number of different files, so rather then force you to specify the filename of each one explicitly, you only have to specify the first part of the filename. The relevant extension will be added automatically.

If you were to run the above example, english.lang would be compiled, but no output files will be generated. This can be useful for checking if there are any errors in the file, but most of the time you'll want to produce some output files.

To compile the language to an lso, you use the -l option, as follows:

```
> langc -l english.lang english
```

This will compile english.lang and, assuming there were no errors, will produce english.lso, ready to load into the engine.

Identifiers ~~~~~~~~~~-~

Earlier in this manual, global variables were used as identifiers for the strings, but there was no mention of where they came from. Because these are a pain in the backside to maintain, langc does the job for you. When you compile the

english language file, you can also generate a script file containing global variables for each string ID in the language file.

To generate the script, simply use the -s option. For example:

```
> langc -s english.lang english
```

This will generate a file called english.cs. Using the earlier example .lang file, the generated script would look something like this:

```
// Automatically generated. DO NOT EDIT
This is a test
$TEST_STR_1 = 0;
and so is this!
$TEST_STR_2 = 1;
```

If you'd like to save some time, you can specify multiple options at once, as in the following example:

```
> langc -ls english.lang english
```

This will create both english.lso and english.cs.

Compiling Translations Although translations compile to the same lso files as the default language, the process is slightly more involved. This is because the identifiers must come from the default english file to ensure they are correct.

You can create a template for a new translation, based on the english file, as follows:

```
> langc -r english.lang french
```

This will generate a file called french.tran, the format of which is identical to the .lang files described earlier:

```
# Automatically generated.
# [TEST_STR_1:0] This is a test
TEST_STR_1=
# [TEST_STR_2:1] and so is this!
TEST_STR_2=
```

The comments preceding each string definition are intended to tell the translator what the original english string was. The actual string definitions are left blank so that langc will issue a warning for empty, and thus untranslated, strings.

Compiling a translation requires a few more options then compiling the english file:

```
> langc -tl -e english.lang french.tran french
```

Here, -t tells langc you want to compile a translation, -e english.lang specifies the filename of the english translation and, as before, -l causes the .lso file to be compiled.

When the translation is loaded into the engine, any strings that are left empty will automatically fall back to the english version of them. By default, langc will issue a warning when compiling the file if a string is empty. This is useful while compiling translations to determine which strings have not been translated. However, it's less useful when compiling the english file, so these warnings may be turned off with the -W option. It is not recommended to use -W when compiling translations.

Other langc options Aside from the above, langc provides a few additional options that you may find useful. These were purposefully not mentioned before as they are not particularly useful at this time.

A complete list of current langc options may be obtained by running langc with no arguments, as follows:

```
> langc
Usage: langc [options] <filename> <outbasename>
Where options is one or more of:
 -l    Write Language File              -h    Write C++ Header
 -s    Write Script                     -d    Write C++ Defaults
 -t    Compile a translation            -r    Write translation file
 -e <filename>   Specify english file when compiling translations
 -S    Don't strip leading spaces       -T    Strip trailing spaces
 -I    Don't warn for invalid chars     -W    Don't warn for empty identifiers
 -q    Quiet mode, no warnings at all
```

Of particular note are the options -h and -d, used when building files for the C++ localization. The -h option generates a C++ header file in the same way as the -s option does for scripts. The -d option writes a C++ source file that provides a big array of default strings. This is intended to be used as an extra fallback when a string can't be found in the language table (or when there is no language table), which allows the same executable to be used whether localization is active or not.

Identifiers must be valid variable names, as they are used directly both as script variables and C++ defines. By default, langc will warn you if an invalid character is used in an identifier. The -I option prevents this.

The -q option is simply a shortcut to disable all warnings at once. If any errors occur, they will still be displayed.

Functions

int **getCoreLangTable**()
Gets the primary LangTable used by the game.

Returns ID of the core LangTable

void **setCoreLangTable**(string *LangTable*)
Sets the primary LangTable used by the game.

Parameters **LangTable** – ID of the core LangTable

Sound

A broad range of functionality for creating rich game audio.

Classes

SFXAmbience A datablock that describes an ambient sound space.

Inherit: SimDataBlock

Description Each ambience datablock captures the properties of a unique ambient sound space. A sound space is comprised of:

- an ambient audio track that is played when the listener is inside the space,
- a reverb environment that is active inside the space, and
- a number of SFXStates that are activated when entering the space and deactivated when exiting it.

Each of these properties is optional.

An important characteristic of ambient audio spaces is that their unique nature is not determined by their location in space but rather by their SFXAmbience datablock. This means that the same SFXAmbience datablock assigned to multiple locations in a level represents the same unique audio space to the sound system.

This is an important distinction for the ambient sound mixer which will activate a given ambient audio space only once at any one time regardless of how many intersecting audio spaces with the same SFXAmbience datablock assigned the listener may currently be in.

All SFXAmbience instances are automatically added to the global SFXAmbienceSet.

At the moment, transitions between reverb environments are not blended and different reverb environments from multiple active SFXAmbiences will not be blended together. This will be added in a future version.

Example:

```
singleton SFXAmbience( Underwater )
{
   environment = AudioEnvUnderwater;
   soundTrack = ScubaSoundList;
   states[ 0 ] = AudioLocationUnderwater;
};
```

Fields

float SFXAmbience::**dopplerFactor**
: The factor to apply to the doppler affect in this space. Defaults to 0.5. Doppler Effect

SFXEnvironment SFXAmbience::**environment**
: Reverb environment active in the ambience zone. Audio Reverb

float SFXAmbience::**rolloffFactor**
: The rolloff factor to apply to distance-based volume attenuation in this space. Defaults to 1.0. Volume Attenuation

SFXTrack SFXAmbience::**soundTrack**
: Sound track to play in the ambience zone.

SFXState SFXAmbience::**states**[4]
: States to activate when the ambient zone is entered. When the ambient sound state is entered, all states associated with the state will be activated (given that they are not disabled) and deactivated when the space is exited again.

SFXController A sound source that drives multi-source playback.

Inherit: SFXSource

Description This class acts as an interpreter for SFXPlayLists. It goes through the slots of the playlist it is attached to and performs the actions described by each of the slots in turn. As SFXControllers are created implicitly by the SFX system when instantiating a source for a play list it is in most cases not necessary to directly deal with the class. The following example demonstrates how a controller would commonly be created.

Example:

```
// Create a play list from two SFXProfiles.
%playList = newSFXPlayList()
{
   // Use a looped description so the list playback will loop.description = AudioMusicLoop2D;

   track[ 0 ] = Profile1;
   track[ 1 ] = Profile2;
};
```

```
// Play the list.  This will implicitly create a controller.sfxPlayOnce( %playList );
```

Methods

int SFXController::**getCurrentSlot**()

Get the index of the playlist slot currently processed by the controller.

Returns The slot index currently being played.

void SFXController::**setCurrentSlot**(int *index*)

Set the index of the playlist slot to play by the controller. This can be used to seek in the playlist.

Parameters **index** – Index of the playlist slot.

Fields

bool SFXController::**trace**

If true, the controller logs its operation to the console. This is a non-networked field that will work locally only.

SFXDescription A description for how a sound should be played.

Inherit: SimDataBlock

Description SFXDescriptions are used by the sound system to collect all parameters needed to set up a given sound for playback. This includes information like its volume level, its pitch shift, etc. as well as more complex information like its fade behavior, 3D properties, and per-sound reverb properties.

Any sound playback will require a valid SFXDescription.

As datablocks, SFXDescriptions can be set up as either networked datablocks or non-networked datablocks, though it generally makes sense to keep all descriptions non-networked since they will be used exclusively by clients.

Example:

```
// A description for a 3D sound with a reasonable default range setting.
// The description is set up to assign sounds to the AudioChannelEffects source group
// (defined in the core scripts).  An alternative means to achieve this is to use the
// AudioEffects description as a copy source (": AudioEffects").

singleton SFXDescription( Audio3DSound )
{
  sourceGroup       = AudioChannelEffects;
  is3D              = true;
  referenceDistance = 20.0;
  maxDistance       = 100.0;
};
```

Fields

int SFXDescription::**coneInsideAngle**

Inner sound cone angle in degrees. This value determines the angle of the inner volume cone that protrudes out in the direction of a sound. Within this cone, the sound source retains full volume that is unaffected by sound cone settings (though still affected by distance attenuation.) Valid values are from 0 to 360. Must be less than coneOutsideAngle. Default is 360. Only for 3D sounds. Sound Cones

int SFXDescription::**coneOutsideAngle**

Outer sound cone angle in degrees. This value determines the angle of the outer volume cone that protrudes out in the direction of a sound and surrounds the inner volume cone. Within this cone, volume will linearly interpolate from the outer cone hull inwards to the inner coner hull starting with the base volume scaled by

coneOutsideVolume and ramping up/down to the full base volume. Valid values are from 0 to 360. Must be gt = coneInsideAngle. Default is 360. Only for 3D sounds. Sound Cones

float SFXDescription::**coneOutsideVolume**
Determines the volume scale factor applied the a source's base volume level outside of the outer cone. In the outer cone, starting from outside the inner cone, the scale factor smoothly interpolates from 1.0 (within the inner cone) to this value. At the moment, the allowed range is 0.0 (silence) to 1.0 (no attenuation) as amplification is only supported on XAudio2 but not on the other devices. Only for 3D sound. Sound Cones

EaseF SFXDescription::**fadeInEase**
Easing curve for fade-in transition. Volume fade-ins will interpolate volume along this curve. Volume Fades

float SFXDescription::**fadeInTime**
Number of seconds to gradually fade in volume from zero when playback starts. Must be gt = 0. Volume Fades

bool SFXDescription::**fadeLoops**
Fade each cycle of a loop in and/or out; otherwise only fade-in first cycle. By default, volume fading is applied to the beginning and end of the playback range, i.e. a fade-in segment is placed at the beginning of the sound and a fade-out segment is paced at the end of a sound. However, when looping playback, this may be undesirable as each iteration of the sound will then have a fade-in and fade-out effect. To set up looping sounds such that a fade-in is applied only when the sound is first started (or playback resumed) and a fade-out is only applied when the sound is explicitly paused or stopped, set this field to true. Default is false. Volume Fades

EaseF SFXDescription::**fadeOutEase**
Easing curve for fade-out transition. Volume fade-outs will interpolate volume along this curve. Volume Fades

float SFXDescription::**fadeOutTime**
Number of seconds to gradually fade out volume down to zero when playback is stopped or paused. Must be gt =0. Volume Fades

bool SFXDescription::**is3D**
If true, sounds played with this description will have a position and orientation in space. Unlike a non-positional sound, a 3D sound will have its volume attenuated depending on the distance to the listener in space. The farther the sound moves away from the listener, the less audible it will be. Non-positional sounds, in contrast, will remain at their original volume regardless of where the listener is. 3D Audio Volume Attenuation

bool SFXDescription::**isLooping**
If true, the sound will be played in an endless loop. Default is false.

bool SFXDescription::**isStreaming**
If true, incrementally stream sounds; otherwise sounds are loaded in full. Streaming vs. Buffered Audio

float SFXDescription::**maxDistance**
The distance at which attenuation stops. In the linear distance model, the attenuated volume will be zero at this distance. In the logarithmic model, attenuation will simply stop at this distance and the sound will keep its attenuated volume from there on out. As such, it primarily functions as a cutoff factor to exponential distance attentuation to limit the number of voices relevant to updates. Only applies to 3D sounds. 3D Audio Volume Attenuation

string SFXDescription::**parameters**[8]
Names of the parameters to which sources using this description will automatically be linked. Individual parameters are identified by their internalName . Interactive Audio

float SFXDescription::**pitch**
Pitch shift to apply to playback. The pitch assigned to a sound determines the speed at which it is played back. A pitch shift of 1 plays the sound at its default speed. A greater shift factor speeds up playback and a smaller shift factor slows it down. Must be gt 0. Default is 1.

float SFXDescription::**priority**
Priority level for virtualization of sounds (1 = base level). When there are more concurrently active sounds than

supported by the audio mixer, some of the sounds need to be culled. Which sounds are culled first depends primarily on total audibility of individual sounds. However, the priority of invidual sounds may be decreased or decreased through this field. Sounds and Voices

float SFXDescription::**referenceDistance**
Distance at which volume attenuation begins. Up to this distance, the sound retains its base volume. In the linear distance model, the volume will linearly from this distance onwards up to maxDistance where it reaches zero. In the logarithmic distance model, the reference distance determine how fast the sound volume decreases with distance. Each referenceDistance steps (scaled by the rolloff factor), the volume halves. A rule of thumb is that for sounds that require you to be close to hear them in the real world, set the reference distance to small values whereas for sounds that are widely audible set it to larger values. Only applies to 3D sounds. 3D Audio Volume Attenuation

const int SFXDescription::**REVERB_DIRECTHFAUTO**[static]
Automatic setting of SFXDescription::reverbDirect due to distance to listener.

const int SFXDescription::**REVERB_INSTANCE0**[static]
EAX4/SFX/GameCube/Wii: Specify channel to target reverb instance 0. Default target.

const int SFXDescription::**REVERB_INSTANCE1**[static]
EAX4/SFX/GameCube/Wii: Specify channel to target reverb instance 1.

const int SFXDescription::**REVERB_INSTANCE2**[static]
EAX4/SFX/GameCube/Wii: Specify channel to target reverb instance 2.

const int SFXDescription::**REVERB_INSTANCE3**[static]
EAX4/SFX/GameCube/Wii: Specify channel to target reverb instance 3.

const int SFXDescription::**REVERB_ROOMAUTO**[static]
Automatic setting of SFXDescription::reverbRoom due to distance to listener.

const int SFXDescription::**REVERB_ROOMHFAUTO**[static]
Automatic setting of SFXDescription::reverbRoomHF due to distance to listener.

float SFXDescription::**reverbAirAbsorptionFactor**
Multiplies SFXEnvironment::airAbsorptionHR.

int SFXDescription::**reverbDirect**
Direct path level (at low and mid frequencies).

int SFXDescription::**reverbDirectHF**
Relative direct path level at high frequencies.

float SFXDescription::**reverbDopplerFactor**
Per-source doppler factor.

int SFXDescription::**reverbExclusion**
Main exclusion control (attenuation at high frequencies).

float SFXDescription::**reverbExclusionLFRatio**
Exclusion low-frequency level re. main control.

int SFXDescription::**reverbFlags**
Bitfield combination of per-sound reverb flags.

int SFXDescription::**reverbObstruction**
Main obstruction control (attenuation at high frequencies).

float SFXDescription::**reverbObstructionLFRatio**
Obstruction low-frequency level re. main control.

int SFXDescription::**reverbOcclusion**
Main occlusion control (attenuation at high frequencies).

float SFXDescription::**reverbOcclusionDirectRatio**
Relative occlusion control for direct path.

float SFXDescription::**reverbOcclusionLFRatio**
Occlusion low-frequency level re. main control.

float SFXDescription::**reverbOcclusionRoomRatio**
Relative occlusion control for room effect.

int SFXDescription::**reverbOutsideVolumeHF**
Outside sound cone level at high frequencies.

float SFXDescription::**reverbReverbRolloffFactor**
Per-source logarithmic falloff factor.

int SFXDescription::**reverbRoom**
Room effect level (at low and mid frequencies).

int SFXDescription::**reverbRoomHF**
Relative room effect level at high frequencies.

float SFXDescription::**reverbRoomRolloffFactor**
Room effect falloff factor.

float SFXDescription::**rolloffFactor**
Scale factor to apply to logarithmic distance attenuation curve. If -1, the global rolloff setting is used.

Point3F SFXDescription::**scatterDistance**
Bounds on random displacement of 3D sound positions. When a 3D sound is created and given its initial position in space, this field is used to determine the amount of randomization applied to the actual position given to the sound system. The randomization uses the following scheme:

SFXSource SFXDescription::**sourceGroup**
Group that sources playing with this description should be put into. When a sound source is allocated, it will be made a child of the source group that is listed in its description. This group will then modulate several properties of the sound as it is played. For example, one use of groups is to segregate sounds so that volume levels of different sound groups such as interface audio and game audio can be controlled independently. Source Hierarchies

int SFXDescription::**streamPacketSize**
Number of seconds of sample data per single streaming packet. This field allows to fine-tune streaming for individual sounds. The streaming system processes streamed sounds in batches called packets. Each packet will contain a set amount of sample data determined by this field. The greater its value, the more sample data each packet contains, the more work is done per packet. Streaming vs. Buffered Audio

int SFXDescription::**streamReadAhead**
Number of sample packets to read and buffer in advance. This field determines the number of packets that the streaming system will try to keep buffered in advance. As such it determines the number of packets that can be consumed by the sound device before the playback queue is running dry. Greater values thus allow for more lag in the streaming pipeline. Streaming vs. Buffered Audio

bool SFXDescription::**useCustomReverb**
If true, use the reverb properties defined here on sounds. By default, sounds will be assigned a generic reverb profile. By setting this flag to true, a custom reverb setup can be defined using the "Reverb" properties that will then be assigned to sounds playing with the description. Audio Reverb

bool SFXDescription::**useHardware**
Whether the sound is allowed to be mixed in hardware. If true, the sound system will try to allocate the voice for the sound directly on the sound hardware for mixing by the hardware mixer. Be aware that a hardware mixer may not provide all features available to sounds mixed in software.

float SFXDescription::**volume**
Base volume level for the sound. This will be the starting point for volume attenuation on the sound. The final effective volume of a sound will be dependent on a number of parameters. Must be between 0 (mute) and 1 (full volume). Default is 1. Volume Attenuation

SFXEmitter An invisible 3D object that emits sound.

Inherit: SceneObject

Description Sound emitters are used to place sounds in the level. They are full 3D objects with their own position and orientation and when assigned 3D sounds, the transform and velocity of the sound emitter object will be applied to the 3D sound.

Sound emitters can be set up of in either of two ways:

- By assigning an existing SFXTrack to the emitter's track property. In this case the general sound setup (3D, streaming, looping, etc.) will be taken from track. However, the emitter's own properties will still override their corresponding properties in the track's SFXDescription.
- By directly assigning a sound file to the emitter's fileName property. In this case, the sound file will be set up for playback according to the properties defined on the emitter.

Using playOnAdd emitters can be configured to start playing immediately when they are added to the system (e.g. when the level objects are loaded from the mission file).

Note: A sound emitter need not necessarily emit a 3D sound. Instead, sound emitters may also be used to play non-positional sounds. For placing background audio to a level, however, it is usually easier to use LevelInfo::soundAmbience.

Sound Emitters and Networking It is important to be aware of the fact that sounds will only play client-side whereas SFXEmitter objects are server-side entities. This means that a server-side object has no connection to the actual sound playing on the client. It is thus not possible for the server-object to perform queries about playback status and other source-related properties as these may in fact differ from client to client.

Methods

SFXSource SFXEmitter::**getSource**()
Get the sound source object from the emitter.

Returns The sound source used by the emitter or null.

void SFXEmitter::**play**()
Manually start playback of the emitter's sound. If this is called on the server-side object, the play command will be related to all client-side ghosts.

void SFXEmitter::**stop**()
Manually stop playback of the emitter's sound. If this is called on the server-side object, the stop command will be related to all client-side ghosts.

Fields

int SFXEmitter::**coneInsideAngle**
Angle of inner volume cone of 3D sound in degrees.

int SFXEmitter::**coneOutsideAngle**
Angle of outer volume cone of 3D sound in degrees.

float SFXEmitter::**coneOutsideVolume**
Volume scale factor of outside of outer volume 3D sound cone.

float SFXEmitter::**fadeInTime**
Number of seconds to gradually fade in volume from zero when playback starts.

float SFXEmitter::**fadeOutTime**
Number of seconds to gradually fade out volume down to zero when playback is stopped or paused.

filename SFXEmitter::**fileName**
The sound file to play. Use either this property or track . If both are assigned, track takes precendence. The primary purpose of this field is to avoid the need for the user to define SFXTrack datablocks for all sounds used in a level.

bool SFXEmitter::**is3D**
Whether to play fileName as a positional (3D) sound or not. If a track is assigned, the value of this field is ignored.

bool SFXEmitter::**isLooping**
Whether to play fileName in an infinite loop. If a track is assigned, the value of this field is ignored.

bool SFXEmitter::**isStreaming**
Whether to use streamed playback for fileName . If a track is assigned, the value of this field is ignored. Streaming vs. Buffered Audio

float SFXEmitter::**maxDistance**
Distance at which to stop volume attenuation of the 3D sound.

float SFXEmitter::**pitch**
Pitch shift to apply to the sound. Default is 1 = play at normal speed.

bool SFXEmitter::**playOnAdd**
Whether playback of the emitter's sound should start as soon as the emitter object is added to the level. If this is true, the emitter will immediately start to play when the level is loaded.

float SFXEmitter::**referenceDistance**
Distance at which to start volume attenuation of the 3D sound.

Point3F SFXEmitter::**scatterDistance**
Bounds on random offset to apply to initial 3D sound position.

SFXSource SFXEmitter::**sourceGroup**
The SFXSource to which to assign the sound of this emitter as a child.

SFXTrack SFXEmitter::**track**
The track which the emitter should play.

bool SFXEmitter::**useTrackDescriptionOnly**
If this is true, all fields except for playOnAdd and track are ignored on the emitter object. This is useful to prevent fields in the track 's description from being overridden by emitter fields.

float SFXEmitter::**volume**
Volume level to apply to the sound.

SFXEnvironment Description of a reverb environment.

Inherit: SimDataBlock

Description A reverb environment specifies how the audio mixer should render advanced environmental audio effects.

To use reverb environments in your level, set up one or more ambient audio spaces, assign reverb environments appropriately, and then attach the SFXAmbiences to your LevelInfo (taking effect globally) or Zone objects (taking effect locally).

To define your own custom reverb environments, it is usually easiest to adapt one of the pre-existing reverb definitions:

```
singleton SFXEnvironment( AudioEnvCustomUnderwater : AudioEnvUnderwater )
{
   // Override select properties from AudioEnvUnderwater here.
};
```

In the Datablock Editor, this can be done by selecting an existing environment to copy from when creating the SFX-Environment datablock.

For a precise description of reverb audio and the properties of this class, please consult the EAX documentation.

All SFXEnvironment instances are automatically added to the global SFXEnvironmentSet.

Fields

float SFXEnvironment::**airAbsorptionHF**
Change in level per meter at high frequencies.

float SFXEnvironment::**decayHFRatio**
High-frequency to mid-frequency decay time ratio.

float SFXEnvironment::**decayLFRatio**
Low-frequency to mid-frequency decay time ratio.

float SFXEnvironment::**decayTime**
Reverberation decay time at mid frequencies.

float SFXEnvironment::**density**
Value that controls the modal density in the late reverberation decay.

float SFXEnvironment::**diffusion**
Value that controls the echo density in the late reverberation decay.

float SFXEnvironment::**echoDepth**
Echo depth.

float SFXEnvironment::**echoTime**
Echo time.

float SFXEnvironment::**envDiffusion**
Environment diffusion.

float SFXEnvironment::**envSize**
Environment size in meters.

int SFXEnvironment::**flags**
A bitfield of reverb flags.

float SFXEnvironment::**HFReference**
Reference high frequency in Hertz.

float SFXEnvironment::**LFReference**
Reference low frequency in Hertz.

float SFXEnvironment::**modulationDepth**
Modulation depth.

float SFXEnvironment::**modulationTime**
Modulation time.

int SFXEnvironment::**reflections**
Early reflections level relative to room effect.

float SFXEnvironment::**reflectionsDelay**
Initial reflection delay time.

float SFXEnvironment::**reflectionsPan**[3]
Early reflections panning vector.

int SFXEnvironment::**reverb**
Late reverberation level relative to room effect.

const int SFXEnvironment::**REVERB_CORE0**[static]
PS2 Only - Reverb is applied to CORE0 (hw voices 0-23).

const int SFXEnvironment::**REVERB_CORE1**[static]
PS2 Only - Reverb is applied to CORE1 (hw voices 24-47).

const int SFXEnvironment::**REVERB_DECAYHFLIMIT**[static]
SFXEnvironment::airAbsorptionHF affects SFXEnvironment::decayHFRatio .

const int SFXEnvironment::**REVERB_DECAYTIMESCALE**[static]
SFXEnvironment::envSize affects reverberation decay time.

const int SFXEnvironment::**REVERB_ECHOTIMESCALE**[static]
SFXEnvironment::envSize affects echo time.

const int SFXEnvironment::**REVERB_HIGHQUALITYDPL2REVERB**[static]
GameCube/Wii Only - Use high-quality DPL2 reverb.

const int SFXEnvironment::**REVERB_HIGHQUALITYREVERB**[static]
GameCube/Wii Only - Use high-quality reverb.

const int SFXEnvironment::**REVERB_MODULATIONTIMESCALE**[static]
SFXEnvironment::envSize affects modulation time.

const int SFXEnvironment::**REVERB_REFLECTIONSDELAYSCALE**[static]
SFXEnvironment::envSize affects initial reflection delay time.

const int SFXEnvironment::**REVERB_REFLECTIONSSCALE**[static]
SFXEnvironment::envSize affects reflection level.

const int SFXEnvironment::**REVERB_REVERBDELAYSCALE**[static]
SFXEnvironment::envSize affects late reverberation delay time.

const int SFXEnvironment::**REVERB_REVERBSCALE**[static]
SFXEnvironment::envSize affects reflections level.

float SFXEnvironment::**reverbDelay**
Late reverberation delay time relative to initial reflection.

float SFXEnvironment::**reverbPan**[3]
Late reverberation panning vector.

int SFXEnvironment::**room**
Room effect level at mid-frequencies.

int SFXEnvironment::**roomHF**
Relative room effect level at high frequencies.

int SFXEnvironment::**roomLF**
Relative room effect level at low frequencies.

float SFXEnvironment::**roomRolloffFactor**
Logarithmic distance attenuation rolloff scale factor for reverb room size effect.

SFXParameter A sound channel value that can be bound to multiple sound sources.

Inherit: SimObject

Description Parameters are special objects that isolate a specific property that sound sources can have and allows to bind this isolated instance to multiple sound sources such that when the value of the parameter changes, all sources bound to the parameter will have their respective property changed.

Parameters are identified and referenced by their internalName. This means that the SFXDescription::parameters and SFXTrack::parameters fields should contain the internalNames of the SFXParameter objects which should be attached to the SFXSources when they are created. No two SFXParameters should have the same internalName.

All SFXParameter instances are automatically made children of the SFXParameterGroup.

Parameter Updates Parameters are periodically allowed to update their own values. This makes it possible to attach custom logic to a parameter and have individual parameters synchronize their values autonomously. Use the onUpdate() callback to attach script code to a parameter update.

Example:

```
newSFXParameter( EngineRPMLevel )
{
   // Set the name by which this parameter is identified.
   internalName = "EngineRPMLevel";

   // Let this parameter control the pitch of attached sources to simulate engine RPM ramping up and
   channel = "Pitch";

   // Start out with unmodified pitch.
   defaultValue = 1;

   // Add a texture description of what this parameter does.
   description = "Engine RPM Level";
};

// Create a description that automatically attaches the engine RPM parameter.
singleton SFXDescription( EngineRPMSound : AudioLoop2D )
{
   parameters[ 0 ] = "EngineRPMLevel";
};

// Create sound sources for the engine.
sfxCreateSource( EngineRPMSound, "art/sound/engine/enginePrimary" );
sfxCreateSource( EngineRPMSound, "art/sound/engine/engineSecondary" );

// Setting the parameter value will now affect the pitch level of both sound sources.
EngineRPMLevel.value = 0.5;
EngineRPMLevel.value = 1.5;
```

Methods

String SFXParameter::**getParameterName**()
Get the name of the parameter.

Returns The paramete name.

void SFXParameter::**onUpdate**()
Called when the sound system triggers an update on the parameter. This occurs periodically during system operation.

void SFXParameter::**reset**()
Reset the parameter's value to its default.

Fields

SFXChannel SFXParameter::**channel**
Channel that the parameter controls. This controls which property of the sources it is attached to the parameter controls.

float SFXParameter::**defaultValue**
Value to which the parameter is initially set. When the parameter is first added to the system, value will be set to defaultValue .

string SFXParameter::**description**
Textual description of the parameter. Primarily for use in the Audio Parameters dialog of the editor to allow for easier identification of parameters.

Point2F SFXParameter::**range**
Permitted range for value . Minimum and maximum allowed value for the parameter. Both inclusive. For all but the User0-3 channels, this property is automatically set up by SFXParameter .

float SFXParameter::**value**
Current value of the audio parameter. All attached sources are notified when this value changes.

SFXPlayList A datablock describing a playback pattern of sounds.

Inherit: SFXTrack

Description Playlists allow to define intricate playback patterns of invidual tracks and thus allow the sound system to be easily used for playing multiple sounds in single operations.

As playlists are SFXTracks, they can thus be used anywhere in the engine where sound data can be assigned.

Each playlist can hold a maximum of 16 tracks. Longer playlists may be constructed by cascading lists, i.e. by creating a playlist that references other playlists.

Processing of a single playlist slot progresses in a fixed set of steps that are invariably iterated through for each slot (except the slot is assigned a state and its state is deactivated; in this case, the controller will exit out of the slot directly):

1. delayIn: Waits a set amount of time before processing the slot. This is 0 by default and is determined by the delayTimeIn (seconds to wait) and delayTimeInVariance (bounds on randomization) properties.
2. transitionIn: Decides what to do before playing the slot. Defaults to None which makes this stage a no-operation. Alternatively, the slot can be configured to wait for playback of other slots to finish (Wait and WaitAll) or to stop playback of other slots (Stop and StopAll). Note that Wait and Stop always refer to the source that was last started by the list.
3. play: Finally, the track attached to the slot is played. However, this will only start playback of the track and then immediately move on to the next stage. It will not wait for the track to finish playing. Note also that depending on the replay setting for the slot, this stage may pick up a source that is already playing on the slot rather than starting a new one. Several slot properties (fade times, min/max distance, and volume/pitch scale) are used in this stage.

4. delayOut: Waits a set amount of time before transitioning out of the slot. This works the same as delayIn and is set to 0 by default (i.e. no delay).
5. transitionOut: Decides what to do after playing the slot. This works like transitionIn.

This is a key difference to playlists in normal music players where upon reaching a certain slot, the slot will immediately play and the player then wait for playback to finish before moving on to the next slot.

Note: Be aware that time limits set on slot delays are soft limits. The sound system updates sound sources in discrete (and equally system update frequency dependent) intervals which thus determines the granularity at which time-outs can be handled.

Value Randomization For greater variety, many of the values for individual slots may be given a randomization limit that will trigger a dynamic variance of the specified base value.

Any given field xyz that may be randomized has a corresponding field xyzVariance which is a two-dimensional vector. The first number specifies the greatest value that may be subtracted from the given base value (i.e. the xyz field) whereas the second number specifies the greatest value that may be added to the base value. Between these two limits, a random number is generated.

The default variance settings of "0 0" will thus not allow to add or subtract anything from the base value and effectively disable randomization.

Randomization is re-evaluated on each cycle through a list.

Playlists and States A unique aspect of playlists is that they allow their playback to be tied to the changing set of active sound states. This feature enables playlists to basically connect to an extensible state machine that can be leveraged by the game code to signal a multitude of different gameplay states with the audio system then automatically reacting to state transitions.

Playlists react to states in three ways:

- Before a controller starts processing a slot it checks whether the slot is assigned a state. If this is the case, the controller checks whether the particular state is active. If it is not, the entire slot is skipped. If it is, the controller goes on to process the slot.
- If a controller is in one of the delay stages for a slot that has a state assigned and the state is deactivated, the controller will stop the delay and skip any of the remaining processing stages for the slot.
- Once the play stage has been processed for a slot that has a state assigned, the slot's stateMode will determine what happens with the playing sound source if the slot's state is deactivated while the sound is still playing.

A simple example of how to make use of states in combination with playlists would be to set up a playlist for background music that reacts to the mood of the current gameplay situation. For example, during combat, tenser music could play than during normal exploration. To set this up, different SFXStates would represent different moods in the game and the background music playlist would have one slot set up for each such mood. By making use of volume fades and the PauseWhenDeactivatedstateMode, smooth transitions between the various audio tracks can be produced.

Example:

```
// Create a play list from two SFXProfiles.
%playList = newSFXPlayList()
{
   // Use a looped description so the list playback will loop.description = AudioMusicLoop2D;

   track[ 0 ] = Profile1;
   track[ 1 ] = Profile2;
```

```
};

// Play the list.sfxPlayOnce( %playList );
```

Fields

float SFXPlayList::**delayTimeIn**[16]
Seconds to wait after moving into slot before transitionIn .

Point2F SFXPlayList::**delayTimeInVariance**[16]
Bounds on randomization of delayTimeIn . Value Randomization

float SFXPlayList::**delayTimeOut**[16]
Seconds to wait before moving out of slot after transitionOut .

Point2F SFXPlayList::**delayTimeOutVariance**[16]
Bounds on randomization of delayTimeOut . Value Randomization

float SFXPlayList::**fadeTimeIn**[16]
Seconds to fade sound in (-1 to use the track's own fadeInTime.).

Point2F SFXPlayList::**fadeTimeInVariance**[16]
Bounds on randomization of fadeInTime. Value Randomization

float SFXPlayList::**fadeTimeOut**[16]
Seconds to fade sound out (-1 to use the track's own fadeOutTime.).

Point2F SFXPlayList::**fadeTimeOutVariance**[16]
Bounds on randomization of fadeOutTime. Value Randomization

SFXPlayListLoopMode SFXPlayList::**loopMode**
Behavior when description has looping enabled. The loop mode determines whether the list will loop over a single slot or loop over all the entire list of slots being played.

float SFXPlayList::**maxDistance**[16]
maxDistance to apply to 3D sounds in this slot (lt 1 to use maxDistance of track's own description).

Point2F SFXPlayList::**maxDistanceVariance**[16]
Bounds on randomization of maxDistance . Value Randomization

int SFXPlayList::**numSlotsToPlay**
Number of slots to play. Up to a maximum of 16, this field determines the number of slots that are taken from the list for playback. Only slots that have a valid track assigned will be considered for this.

float SFXPlayList::**pitchScale**[16]
Scale factor to apply to pitch of sounds played on this list slot. This value will scale the actual pitch set on the track assigned to the slot, i.e. a value of 0.5 will cause the track to play at half its assigned speed.

Point2F SFXPlayList::**pitchScaleVariance**[16]
Bounds on randomization of pitchScale . Value Randomization

SFXPlayListRandomMode SFXPlayList::**random**
Slot playback order randomization pattern. By setting this field to something other than "NotRandom" to order in which slots of the playlist are processed can be changed from sequential to a random pattern. This allows to to create more varied playback patterns. Defaults to "NotRandom".

float SFXPlayList::**referenceDistance**[16]
referenceDistance to set for 3D sounds in this slot (lt 1 to use referenceDistance of track's own description).

Point2F SFXPlayList::**referenceDistanceVariance**[16]
Bounds on randomization of referenceDistance . Value Randomization

int SFXPlayList::**repeatCount**[16]
Number of times to loop this slot.

SFXPlayListReplayMode SFXPlayList::**replay**[16]
Behavior when an already playing sound is encountered on this slot from a previous cycle. Each slot can have an arbitrary number of sounds playing on it from previous cycles. This field determines how SFXController will handle these sources.

SFXState SFXPlayList::**state**[16]
State that must be active for this slot to play. Playlists and States

SFXPlayListStateMode SFXPlayList::**stateMode**[16]
Behavior when assigned state is deactivated while slot is playing. Playlists and States

bool SFXPlayList::**trace**
Enable/disable execution tracing for this playlist (local only). If this is true, SFXControllers attached to the list will automatically run in trace mode.

SFXTrack SFXPlayList::**track**[16]
Track to play in this slot. This must be set for the slot to be considered for playback. Other settings for a slot will not take effect except this field is set.

SFXPlayListTransitionMode SFXPlayList::**transitionIn**[16]
Behavior when moving into this slot. After the delayIn time has expired (if any), this slot determines what the controller will do before actually playing the slot.

SFXPlayListTransitionMode SFXPlayList::**transitionOut**[16]
Behavior when moving out of this slot. After the detailTimeOut has expired (if any), this slot determines what the controller will do before moving on to the next slot.

float SFXPlayList::**volumeScale**[16]
Scale factor to apply to volume of sounds played on this list slot. This value will scale the actual volume level set on the track assigned to the slot, i.e. a value of 0.5 will cause the track to play at half-volume.

Point2F SFXPlayList::**volumeScaleVariance**[16]
Bounds on randomization of volumeScale . Value Randomization

SFXProfile Encapsulates a single sound file for playback by the sound system.

Inherit: SFXTrack

Description SFXProfile combines a sound description (SFXDescription) with a sound file such that it can be played by the sound system. To be able to play a sound file, the sound system will always require a profile for it to be created. However, several of the SFX functions (sfxPlayOnce(), sfxCreateSource()) perform this creation internally for convenience using temporary profile objects.

Sound files can be in either OGG or WAV format. However, extended format support is available when using FMOD. See Supported Sound File Formats.

Profile Loading By default, the sound data referenced by a profile will be loaded when the profile is first played and the data then kept until either the profile is deleted or until the sound device on which the sound data is held is deleted.

This initial loading my incur a small delay when the sound is first played. To avoid this, a profile may be expicitly set to load its sound data immediately when the profile is added to the system. This is done by setting the preload property to true.

Example:

```
datablock SFXProfile( Shore01Snd )
{
   fileName     = "art/sound/Lakeshore_mono_01";
   description  = Shore01Looping3d;
   preload      = true;
};
```

Methods

float SFXProfile::**getSoundDuration**()

Return the length of the sound data in seconds.

Returns The length of the sound data in seconds or 0 if the sound referenced by the profile could not be found.

Fields

filename SFXProfile::**fileName**

Path to the sound file. If the extension is left out, it will be inferred by the sound system. This allows to easily switch the sound format without having to go through the profiles and change the filenames there, too.

bool SFXProfile::**preload**

Whether to preload sound data when the profile is added to system. Profile Loading

SFXSound A sound controller that directly plays a single sound file.

Inherit: SFXSource

Description When playing individual audio files, SFXSounds are implicitly created by the sound system.

Each sound source has an associated play cursor that can be queried and explicitly positioned by the user. The cursor is a floating-point value measured in seconds.

For streamed sources, playback may not be continuous in case the streaming queue is interrupted.

Sounds and Voices To actually emit an audible signal, a sound must allocate a resource on the sound device through which the sound data is being played back. This resource is called 'voice'.

As with other types of resources, the availability of these resources may be restricted, i.e. a given sound device will usually only support a fixed number of voices that are playing at the same time. Since, however, there may be arbitrary many SFXSounds instantiated and playing at the same time, this needs to be solved.

Methods

float SFXSound::**getDuration**()

Get the total play time (in seconds) of the sound data attached to the sound.

Returns Returns:

float SFXSound::**getPosition**()

Get the current playback position in seconds.

Returns The current play cursor offset.

bool SFXSound::**isReady**()

Test whether the sound data associated with the sound has been fully loaded and is ready for playback. For streamed sounds, this will be false during playback when the stream queue for the sound is starved and waiting for data. For buffered sounds, only an initial loading phase will potentially cause isReady to return false.

Returns True if the sound is ready for playback.

void SFXSound::**setPosition**(float *position*)

Set the current playback position in seconds. If the source is currently playing, playback will jump to the new position. If playback is stopped or paused, playback will resume at the given position when play() is called.

Parameters **position** – The new position of the play cursor (in seconds).

SFXSource Playback controller for a sound source.

Inherit: SimGroup

Description All sound playback is driven by SFXSources. Each such source represents an independent playback controller that directly or indirectly affects sound output.

While this class itself is instantiable, such an instance will not by itself emit any sound. This is the responsibility of its subclasses. Note, however, that none of these subclasses must be instantiated directly but must instead be instantiated indirectly through the SFX interface.

Play-Once Sources Often, a sound source need only exist for the duration of the sound it is playing. In this case so-called "play-once" sources simplify the bookkeeping involved by leaving the deletion of sources that have expired their playtime to the sound system.

Play-once sources can be created in either of two ways:

- sfxPlayOnce(): Directly create a play-once source from a SFXTrack or file.
- sfxDeleteWhenStopped(): Retroactively turn any source into a play-once source that will automatically be deleted when moving into stopped state.

Source Hierarchies Source are arranged into playback hierarchies where a parent source will scale some of the properties of its children and also hand on any play(), pause(), and stop() commands to them. This allows to easily group sounds into logical units that can then be operated on as a whole.

An example of this is the segregation of sounds according to their use in the game. Volume levels of background music, in-game sound effects, and character voices will usually be controlled independently and putting their sounds into different hierarchies allows to achieve that easily.

The source properties that are scaled by parent values are:

- volume,
- pitch, and
- priority

This means that if a parent has a volume of 0.5, the child will play at half the effective volume it would otherwise have.

Additionally, parents affect the playback state of their children:

- A parent that is in stopped state will force all its direct and indirect children into stopped state.
- A parent that is in paused state will force all its direct and indirect children that are playing into paused state. However, children that are in stopped state will not be affected.
- A parent that is in playing state will not affect the playback state of its children.

Each source maintains a state that is wants to be in which may differ from the state that is enforced on it by its parent. If a parent changes its states in a way that allows a child to move into its desired state, the child will do so.

For logically grouping sources, instantiate the SFXSource class directly and make other sources children to it. A source thus instantiated will not effect any real sound output on its own but will influence the sound output of its direct and indirect children.

Note: Be aware that the property values used to scale child property values are the effective values. For example, the value used to scale the volume of a child is the effective volume of the parent, i.e. the volume after fades, distance attenuation, etc. has been applied.

Volume Attenuation During its lifetime, the volume of a source will be continually updated. This update process always progresses in a fixed set of steps to compute the final effective volume of the source based on the base volume level that was either assigned from the SFXDescription associated with the source (SFXDescription::volume) or manually set by the user. The process of finding a source's final effective volume is called "volume attenuation". The steps involved in attenuating a source's volume are (in order):

Fading If the source currently has a fade-effect applied, the volume is interpolated along the currently active fade curve.

Modulation If the source is part of a hierarchy, it's volume is scaled according to the effective volume of its parent.

Distance Attenuation If the source is a 3D sound source, then the volume is interpolated according to the distance model in effect and current listener position and orientation (see 3D Audio).

Volume Fades To ease-in and ease-out playback of a sound, fade effects may be applied to sources. A fade will either go from zero volume to full effective volume (fade-in) or from full effective volume to zero volume (fade-out).

Fading is coupled to the play(), pause(), and stop() methods as well as to loop iterations when SFXDescription::fadeLoops is true for the source. play() and the start of a loop iteration will trigger a fade-in whereas pause(), stop() and the end of loop iterations will trigger fade-outs.

For looping sources, if SFXDescription::fadeLoops is false, only the initial play() will trigger a fade-in and no further fading will be applied to loop iterations.

By default, the fade durations will be governed by the SFXDescription::fadeInTime and SFXDescription::fadeOutTime properties of the SFXDescription attached to the source. However, these may be overridden on a per-source basis by setting fade times explicitly with setFadeTimes(). Additionally, the set values may be overridden for individual play(), pause(), and stop() calls by supplying appropriate fadeInTime/fadeOutTime parameters.

By default, volume will interpolate linearly during fades. However, custom interpolation curves can be assigned through the SFXDescription::fadeInEase and SFXDescription::fadeOutTime properties.

Sound Cones

Doppler Effect

Playback Markers Playback markers allow to attach notification triggers to specific playback positions. Once the play cursor crosses a position for which a marker is defined, the onMarkerPassed callback will be triggered on the SFXSource thus allowing to couple script logic to .

Be aware that the precision with which marker callbacks are triggered are bound by global source update frequency. Thus there may be a delay between the play cursor actually passing a marker position and the callback being triggered.

Methods

void SFXSource::**addMarker**(String *name*, float *pos*)

Add a notification marker called name at pos seconds of playback.

Parameters

- **name** – Symbolic name for the marker that will be passed to the onMarkerPassed() callback.
- **pos** – Playback position in seconds when the notification should trigger. Note that this is a soft limit and there may be a delay between the play cursor actually passing the position and the callback being triggered.

Example:

```
// Create a new source.
$source = sfxCreateSource( AudioMusicLoop2D, "art/sound/backgroundMusic" );

// Assign a class to the source.
$source.class = "BackgroundMusic";

// Add a playback marker at one minute into playback.
$source.addMarker( "first", 60 );

// Define the callback function.  This function will be called when the playback position passes
function BackgroundMusic::onMarkerPassed( %this, %markerName )
{
   if( %markerName $= "first" )
      echo( "Playback has passed the 60 seconds mark." );
}

// Play the sound.
$source.play();
```

void SFXSource::**addParameter**(SFXParameter *parameter*)

Attach parameter to the source,. Once attached, the source will react to value changes of the given parameter . Attaching a parameter will also trigger an initial read-out of the parameter's current value.

Parameters **parameter** – The parameter to attach to the source.

float SFXSource::**getAttenuatedVolume**()

Get the final effective volume level of the source. This method returns the volume level as it is after source group volume modulation, fades, and distance-based volume attenuation have been applied to the base volume level. Volume Attenuation

Returns The effective volume of the source.

float SFXSource::**getFadeInTime**()

Get the fade-in time set on the source. This will initially be SFXDescription::fadeInTime . Volume Fades

Returns The fade-in time set on the source in seconds.

float SFXSource::**getFadeOutTime**()

Get the fade-out time set on the source. This will initially be SFXDescription::fadeOutTime . Volume Fades

Returns The fade-out time set on the source in seconds.

SFXParameter SFXSource::**getParameter**(int *index*)

Get the parameter at the given index.

Parameters **index** – Index of the parameter to fetch. Must be 0<=index<=getParameterCount().

Returns is out of range.

Example:

```
// Print the name ofo each parameter attached to %source.
%numParams = %source.getParameterCount();
for( %i = 0; %i < %numParams; %i ++ )
   echo( %source.getParameter( %i ).getParameterName() );
```

int SFXSource::**getParameterCount**()
Get the number of SFXParameters that are attached to the source.

Returns The number of parameters attached to the source.

Example:

```
// Print the name ofo each parameter attached to %source.
%numParams = %source.getParameterCount();
for( %i = 0; %i < %numParams; %i ++ )
   echo( %source.getParameter( %i ).getParameterName() );
```

float SFXSource::**getPitch**()
Get the pitch scale of the source. Pitch determines the playback speed of the source (default: 1).

Returns The current pitch scale factor of the source.

SFXStatus SFXSource::**getStatus**()
Get the current playback status.

Returns Te current playback status

float SFXSource::**getVolume**()
Get the current base volume level of the source. This is not the final effective volume that the source is playing at but rather the starting volume level before source group modulation, fades, or distance-based volume attenuation are applied. Volume Attenuation

Returns The current base volume level.

bool SFXSource::**isPaused**()
Test whether the source is currently paused.

Returns True if the source is in paused state, false otherwise.

bool SFXSource::**isPlaying**()
Test whether the source is currently playing.

Returns True if the source is in playing state, false otherwise.

bool SFXSource::**isStopped**()
Test whether the source is currently stopped.

Returns True if the source is in stopped state, false otherwise.

void SFXSource::**onParameterValueChange**(SFXParameter *parameter*)
Called when a parameter attached to the source changes value. This callback will be triggered before the value change has actually been applied to the source.

Parameters **parameter** – The parameter that has changed value.

void SFXSource::**onStatusChange**(SFXStatus *newStatus*)
Called when the playback status of the source changes.

Parameters **newStatus** – The new playback status.

void SFXSource::**pause**(float *fadeOutTime*)
Pause playback of the source.

Parameters **fadeOutTime** – Seconds for the sound to fade down to zero volume. If -1, the SFXDescription::fadeOutTime set in the source's associated description is used. Pass 0 to disable a fade-out effect that may be configured on the description. Be aware that if a fade-out effect is used, the source will not immediately to paused state but will rather remain in playing state until the fade-out time has expired..

void SFXSource::**removeParameter**(SFXParameter *parameter*)
Detach parameter from the source. Once detached, the source will no longer react to value changes of the given parameter . If the parameter is not attached to the source, the method will do nothing.

Parameters **parameter** – The parameter to detach from the source.

void SFXSource::**setCone**(float *innerAngle*, float *outerAngle*, float *outsideVolume*)
Set up the 3D volume cone for the source.

Parameters

- **innerAngle** – Angle of the inner sound cone in degrees (SFXDescription::coneInsideAngle). Must be 0<=innerAngle<=360.
- **outerAngle** – Angle of the outer sound cone in degrees (SFXDescription::coneOutsideAngle). Must be 0<=outerAngle<=360.
- **outsideVolume** – Volume scale factor outside of outer cone (SFXDescription::coneOutsideVolume). Must be 0<=outsideVolume<=1.

void SFXSource::**setFadeTimes**(float *fadeInTime*, float *fadeOutTime*)
Set the fade time parameters of the source. Volume Fades

Parameters

- **fadeInTime** – The new fade-in time in seconds.
- **fadeOutTime** – The new fade-out time in seconds.

void SFXSource::**setPitch**(float *pitch*)
Set the pitch scale of the source. Pitch determines the playback speed of the source (default: 1).

Parameters **pitch** – The new pitch scale factor.

void SFXSource::**setTransform**(Point3F *position*, Point3F *direction*)
Start playback of the source. Set the position and orientation of the source's 3D sound. Set the position of the source's 3D sound. If the sound data for the source has not yet been fully loaded, there will be a delay after calling play and playback will start after the data has become available.

Parameters

- **position** – The new position in world space.
- **direction** – The forward vector.
- **position** – The new position in world space.
- **fadeInTime** – Seconds for the sound to reach full volume. If -1, the SFXDescription::fadeInTime set in the source's associated description is used. Pass 0 to disable a fade-in effect that may be configured on the description.

void SFXSource::**setVolume**(float *volume*)
Set the base volume level for the source. This volume will be the starting point for source group volume modulation, fades, and distance-based volume attenuation. Volume Attenuation

Parameters **volume** – The new base volume level for the source. Must be 0>=volume<=1.

void SFXSource::**stop**(float *fadeOutTime*)
Stop playback of the source.

Parameters **fadeOutTime** – Seconds for the sound to fade down to zero volume. If -1, the SFXDescription::fadeOutTime set in the source's associated description is used. Pass 0 to disable a fade-out effect that may be configured on the description. Be aware that if a fade-out effect is used, the source will not immediately transtion to stopped state but will rather remain in playing state until the fade-out time has expired.

Fields

SFXDescription `SFXSource::description`
The playback configuration that determines the initial sound properties and setup. Any SFXSource must have an associated SFXDescription .

string `SFXSource::statusCallback`
Name of function to call when the status of the source changes. The source that had its status changed is passed as the first argument to the function and the new status of the source is passed as the second argument.

SFXSpace A volume in space that defines an ambient sound zone.

Inherit: SceneObject

Description A volume in space that defines an ambient sound zone.

Fields

string `SFXSpace::edge`
For internal use only.

string `SFXSpace::plane`
For internal use only.

string `SFXSpace::point`
For internal use only.

SFXAmbience `SFXSpace::soundAmbience`
Ambient sound environment for the space.

SFXState A boolean switch used to modify playlist behavior.

Inherit: SimDataBlock

Description Sound system states are used to allow playlist controllers to make decisions based on global state. This is useful, for example, to couple audio playback to gameplay state. Certain states may, for example, represent different locations that the listener can be in, like underwater, in open space, or indoors. Other states could represent moods of the current gameplay situation, like, for example, an aggressive mood during combat.

By activating and deactivating sound states according to gameplay state, a set of concurrently running playlists may react and adapt to changes in the game.

Activation and Deactivation At any time, a given state can be either active or inactive. Calling activate() on a state increases an internal counter and calling deactivate() decreases the counter. Only when the count reaches zero will the state be deactivated.

In addition to the activation count, states also maintain a disabling count. Calling disable() increases this count and calling enable() decreases it. As long as this count is greater than zero, a given state will not be activated even if its activation count is non-zero. Calling disable() on an active state will not only increase the disabling count but also deactivate the state. Calling enable() on a state with a positive activation count will re-activate the state when the disabling count reaches zero.

State Dependencies By listing other states in in its includedStates and excludedStates fields, a state may automatically trigger the activation or disabling of other states in the sytem when it is activated. This allows to form dependency chains between individual states.

Example:

```
// State indicating that the listener is submerged.
singleton SFXState( AudioLocationUnderwater )
{
   parentGroup = AudioLocation;
   // AudioStateExclusive is a class defined in the core scripts that will automatically// ensure fo
};

// State suitable e.g. for combat.
singleton SFXState( AudioMoodAggressive )
{
   parentGroup = AudioMood;
   className = "AudioStateExclusive";
};
```

Methods

void SFXState::**activate**()
Increase the activation count on the state. If the state isn't already active and it is not disabled, the state will be activated.

void SFXState::**deactivate**()
Decrease the activation count on the state. If the count reaches zero and the state was not disabled, the state will be deactivated.

void SFXState::**disable**()
Increase the disabling count of the state. If the state is currently active, it will be deactivated.

void SFXState::**enable**()
Decrease the disabling count of the state. If the disabling count reaches zero while the activation count is still non-zero, the state will be reactivated again.

bool SFXState::**isActive**()
Test whether the state is currently active. This is true when the activation count is gt 0 and the disabling count is =0.

Returns True if the state is currently active.

bool SFXState::**isDisabled**()
Test whether the state is currently disabled. This is true when the disabling count of the state is non-zero.

Returns True if the state is disabled.

void SFXState::**onActivate**()
Called when the state goes from inactive to active.

void SFXState::**onDeactivate**()
called when the state goes from active to deactive.

Fields

SFXState SFXState::**excludedStates**[4]
States that will automatically be disabled when this state is activated. Activation and Deactivation

SFXState SFXState::**includedStates**[4]
States that will automatically be activated when this state is activated. Activation and Deactivation

SFXTrack Abstract base class for sound data that can be played back by the sound system.

Inherit: SimDataBlock

Description The term "track" is used in the sound system to refer to any entity that can be played back as a sound source. These can be individual files (SFXProfile), patterns of other tracks (SFXPlayList), or special sound data defined by a device layer (SFXFMODEvent).

Any track must be paired with a SFXDescription that tells the sound system how to set up playback for the track.

All objects that are of type SFXTrack will automatically be added to SFXTrackSet.

Fields

SFXDescription `SFXTrack::description`
Playback setup description for this track. If unassigned, the description named "AudioEffects" will automatically be assigned to the track. If this description is not defined, track creation will fail.

string `SFXTrack::parameters`[8]
Parameters to automatically attach to SFXSources created from this track. Individual parameters are identified by their internalName .

Enumeration

enum SFXChannel
Channels are individual properties of sound sources that may be animated over time.

Parameters

- **Volume** – Channel controls volume level of attached sound sources. See also:SFXDescription::volume
- **Pitch** – Channel controls pitch of attached sound sources. See also:SFXDescription::pitch
- **Priority** – Channel controls virtualizaton priority level of attached sound sources. See also:SFXDescription::priority
- **PositionX** – Channel controls X coordinate of 3D sound position of attached sources.
- **PositionY** – Channel controls Y coordinate of 3D sound position of attached sources.
- **PositionZ** – Channel controls Z coordinate of 3D sound position of attached sources.
- **RotationX** – Channel controls X rotation (in degrees) of 3D sound orientation of attached sources.
- **RotationY** – Channel controls Y rotation (in degrees) of 3D sound orientation of attached sources.
- **RotationZ** – Channel controls Z rotation (in degrees) of 3D sound orientation of attached sources.
- **VelocityX** – Channel controls X coordinate of 3D sound velocity vector of attached sources.
- **VelocityY** – Channel controls Y coordinate of 3D sound velocity vector of attached sources.
- **VelocityZ** – Channel controls Z coordinate of 3D sound velocity vector of attached sources.

- **ReferenceDistance** – Channel controls reference distance of 3D sound of attached sources. See also:SFXDescription::referenceDistance
- **MaxDistance** – Channel controls max volume attenuation distance of 3D sound of attached sources. See also:SFXDescription::maxDistance
- **ConeInsideAngle** – Channel controls angle (in degrees) of 3D sound inner volume cone of attached sources. See also:SFXDescription::coneInsideAngle
- **ConeOutsideAngle** – Channel controls angle (in degrees) of 3D sound outer volume cone of attached sources. See also:SFXDescription::coneOutsideAngle
- **ConeOutsideVolume** – Channel controls volume outside of 3D sound outer cone of attached sources. See also:SFXDescription::coneOutsideVolume
- **Cursor** – Channel controls playback cursor of attached sound sources. Note:Be aware that different types of sound sources interpret play cursor positions differently or do not actually have play cursors (these sources will ignore the channel).
- **Status** – Channel controls playback status of attached sound sources. The channel's value is rounded down to the nearest integer and interpreted in the following way:1: Play2: Stop3: Pause
- **User0** – Channel available for custom use. By default ignored by sources. Note:For FMOD Designer event sources (SFXFMODEventSource), this channel is used for event parameters defined in FMOD Designer and should not be used otherwise.See also:SFXSource::onParameterValueChange
- **User1** – Channel available for custom use. By default ignored by sources. See also:SFXSource::onParameterValueChange
- **User2** – Channel available for custom use. By default ignored by sources. See also:SFXSource::onParameterValueChange
- **User3** – Channel available for custom use. By default ignored by sources. See also:SFXSource::onParameterValueChange

enum SFXDistanceModel

Type of volume distance attenuation curve. The distance model determines the falloff curve applied to the volume of 3D sounds over distance.

Parameters

- **Linear** – Volume attenuates linearly from the references distance onwards to max distance where it reaches zero.
- **Logarithmic** – Volume attenuates logarithmically starting from the reference distance and halving every reference distance step from there on. Attenuation stops at max distance but volume won't reach zero.

enum SFXPlayListLoopMode

Playlist behavior when description is set to loop.

Parameters

- **All** – Loop over all slots, i.e. jump from last to first slot after all slots have played.
- **Single** – Loop infinitely over the current slot. Only useful in combination with either states or manual playlist control.

enum SFXPlayListRandomMode

Randomization pattern to apply to playlist slot playback order.

Parameters

- **NotRandom** – Play slots in sequential order. No randomization.
- **StrictRandom** – Play a strictly random selection of slots. In this mode, a set of numSlotsToPlay random numbers between 0 and numSlotsToPlay-1 (inclusive), i.e. in the range of valid slot indices, is generated and playlist slots are played back in the order of this list. This allows the same slot to occur multiple times in a list and, consequentially, allows for other slots to not appear at all in a given slot ordering.
- **OrderedRandom** – Play all slots in the list in a random order. In this mode, the numSlotsToPlay slots from the list with valid tracks assigned are put into a random order and played. This guarantees that each slots is played exactly once albeit at a random position in the total ordering.

enum SFXPlayListReplayMode

Behavior when hitting the play stage of a slot that is still playing from a previous cycle.

Parameters

- **IgnorePlaying** – Ignore any sources that may already be playing on the slot and just create a new source.
- **RestartPlaying** – Restart all sources that was last created for the slot.
- **KeepPlaying** – Keep playing the current source(s) as if the source started last on the slot was created in this cycle. For this, the sources associated with the slot are brought to the top of the play stack.
- **StartNew** – Stop all sources currently playing on the slot and then create a new source.
- **SkipIfPlaying** – If there are sources already playing on the slot, skip the play stage.

enum SFXPlayListStateMode

Reaction behavior when a state is changed incompatibly on a slot that has already started playing.

Parameters

- **StopWhenDeactivated** – Stop the sources playing on the slot when a state changes to a setting that is incompatible with the slot's state setting.
- **PauseWhenDeactivated** – Pause all sources playing on the slot when a state changes to a setting that is incompatible with the slot's state setting. When the slot's state is reactivated again, the sources will resume playback.
- **IgnoreWhenDeactivated** – Ignore when a state changes to a setting incompatible with the slot's state setting and just keep playing sources attached to the slot.

enum SFXPlayListTransitionMode

Playlist behavior when transitioning in and out of invididual slots. Transition behaviors apply when the playback controller starts processing a playlist slot and when it ends processing a slot. Using transition behaviors, playback can be synchronized.

Parameters

- **None** – No transition. Immediately move on to processing the slot or immediately move on to the next slot.
- **Wait** – Wait for the sound source spawned last by this playlist to finish playing. Then proceed.
- **WaitAll** – Wait for all sound sources currently spawned by the playlist to finish playing. Then proceed.
- **Stop** – Stop the sound source spawned last by this playlist. Then proceed.

- **StopAll** – Stop all sound sources spawned by the playlist. Then proceed.

enum SFXStatus

Playback status of sound source.

Parameters

- **Playing** – The source is currently playing.
- **Stopped** – Playback of the source is stopped. When transitioning to Playing state, playback will start at the beginning of the source.
- **Paused** – Playback of the source is paused. Resuming playback will play from the current playback position.

Functions

bool **sfxCreateDevice**(string *provider*, string *device*, bool *useHardware*, int *maxBuffers*)

Try to create a new sound device using the given properties. If a sound device is currently initialized, it will be uninitialized first. However, be aware that in this case, if this function fails, it will not restore the previously active device but rather leave the sound system in an uninitialized state. Sounds that are already playing while the new device is created will be temporarily transitioned to virtualized playback and then resume normal playback once the device has been created. In the core scripts, sound is automatically set up during startup in the sfxStartup() function. Providers and Devices

Parameters

- **provider** – The name of the device provider as returned by sfxGetAvailableDevices().
- **device** – The name of the device as returned by sfxGetAvailableDevices().
- **useHardware** – Whether to enabled hardware mixing on the device or not. Only relevant if supported by the given device.
- **maxBuffers** – The maximum number of concurrent voices for this device to use or -1 for the device to pick its own reasonable default.

Returns True if the initialization was successful, false if not.

SFXSource **sfxCreateSource**(SFXTrack *track*)

Create a new source that plays the given track. The source will be returned in stopped state. Call SFXSource::play() to start playback. In contrast to play-once sources, the source object will not be automatically deleted once playback stops. Call delete() to release the source object. This function will automatically create the right SFXSource type for the given SFXTrack .

Parameters **track** – The track the source should play.

Returns for playback of the given track or 0 if no source could be created from the given track.

Example:

```
// Create and play a source from a pre-existing profile:
%source = sfxCreateSource( SoundFileProfile );
%source.play();
```

SFXSource **sfxCreateSource**(SFXTrack *track*, float *x*, float *y*, float *z*)

Create a new source that plays the given track and position its 3D sounds source at the given coordinates (if it is a 3D sound). The source will be returned in stopped state. Call SFXSource::play() to start playback. In contrast to play-once sources, the source object will not be automatically deleted once playback stops. Call delete() to release the source object. This function will automatically create the right SFXSource type for the given SFXTrack .

Parameters

- **track** – The track the source should play.
- **x** – The X coordinate of the 3D sound position.
- **y** – The Y coordinate of the 3D sound position.
- **z** – The Z coordinate of the 3D sound position.

Returns for playback of the given track or 0 if no source could be created from the given track.

Example:

```
// Create and play a source from a pre-existing profile and position it at (100, 200, 300):
%source = sfxCreateSource( SoundFileProfile, 100, 200, 300 );
%source.play();
```

SFXSound **sfxCreateSource**(SFXDescription *description*, string *filename*)

Create a temporary SFXProfile from the given description and filename and then create and return a new source that plays the profile. The source will be returned in stopped state. Call SFXSource::play() to start playback. In contrast to play-once sources, the source object will not be automatically deleted once playback stops. Call delete() to release the source object.

Parameters

- **description** – The description to use for setting up the temporary SFXProfile.
- **filename** – Path to the sound file to play.

Returns for playback of the given track or 0 if no source or no temporary profile could be created.

Example:

```
// Create a source for a music track:
%source = sfxCreateSource( AudioMusicLoop2D, "art/sound/backgroundMusic" );
%source.play();
```

SFXSound **sfxCreateSource**(SFXDescription *description*, string *filename*, float *x*, float *y*, float *z*)

Create a temporary SFXProfile from the given description and filename and then create and return a new source that plays the profile. Position the sound source at the given coordinates (if it is a 3D sound). The source will be returned in stopped state. Call SFXSource::play() to start playback. In contrast to play-once sources, the source object will not be automatically deleted once playback stops. Call delete() to release the source object.

Parameters

- **description** – The description to use for setting up the temporary SFXProfile.
- **filename** – Path to the sound file to play.
- **x** – The X coordinate of the 3D sound position.
- **y** – The Y coordinate of the 3D sound position.
- **z** – The Z coordinate of the 3D sound position.

Returns for playback of the given track or 0 if no source or no temporary profile could be created.

Example:

```
// Create a source for a music track and position it at (100, 200, 300):
%source = sfxCreateSource( AudioMusicLoop3D, "art/sound/backgroundMusic", 100, 200, 300 );
%source.play();
```

void **sfxDeleteDevice**()
Delete the currently active sound device and release all its resources. SFXSources that are still playing will be transitioned to virtualized playback mode. When creating a new device, they will automatically transition back to normal playback. In the core scripts, this is done automatically for you during shutdown in the sfxShutdown() function. Providers and Devices

void **sfxDeleteWhenStopped**(SFXSource *source*)
Mark the given source for deletion as soon as it moves into stopped state. This function will retroactively turn the given source into a play-once source (see Play-Once Sources).

Parameters **source** – A sound source.

void **sfxDumpSources**(bool *includeGroups*)
Dump information about all current SFXSource instances to the console. The dump includes information about the playback status for each source, volume levels, virtualization, etc.

Parameters **includeGroups** – If true, direct instances of SFXSources (which represent logical sound groups) will be included. Otherwise only instances of subclasses of SFXSources are included in the dump.

string **sfxDumpSourcesToString**(bool *includeGroups*)
Dump information about all current SFXSource instances to a string. The dump includes information about the playback status for each source, volume levels, virtualization, etc.

Parameters **includeGroups** – If true, direct instances of SFXSources (which represent logical sound groups) will be included. Otherwise only instances of subclasses of SFXSources are included in the dump.

Returns A string containing a dump of information about all currently instantiated SFXSources.

string **sfxGetActiveStates**()
Return a newline-separated list of all active states.

Returns where each element is the name of an active state object.

Example:

```
// Disable all active states.
foreach$( %state in sfxGetActiveStates() )
   %state.disable();
```

string **sfxGetAvailableDevices**()
Get a list of all available sound devices. The return value will be a newline-separated list of entries where each line describes one available sound device. Each such line will have the following format:

Returns A newline-separated list of information about all available sound devices.

string **sfxGetDeviceInfo**()
Return information about the currently active sound device. The return value is a tab-delimited string of the following format:

Returns A tab-separated list of properties of the currently active sound device or the empty string if no sound device has been initialized.

SFXDistanceModel **sfxGetDistanceModel**()
Get the falloff curve type currently being applied to 3D sounds. Volume Attenuation 3D Audio

Returns The current distance model type.

float **sfxGetDopplerFactor**()
Get the current global doppler effect setting. Doppler Effect

Returns =0).

float **sfxGetRolloffFactor**()

Get the current global scale factor applied to volume attenuation of 3D sounds in the logarithmic model. Volume Attenuation 3D Audio

Returns The current scale factor for logarithmic 3D sound falloff curves.

SFXSource **sfxPlay**(SFXSource *source*)

Start playback of the given source. This is the same as calling SFXSource::play() directly.

Parameters **source** – The source to start playing.

Returns

.

Example:

```
// Create and play a source from a pre-existing profile:
%source = sfxCreateSource( SoundFileProfile );
%source.play();
```

void **sfxPlay**(SFXTrack *track*)

Create a new play-once source for the given track and start playback of the source. This is equivalent to calling sfxCreateSource() on and SFXSource::play() on the resulting source. Play-Once Sources

Parameters **track** – The sound datablock to play.

Returns The newly created play-once source or 0 if the creation failed.

void **sfxPlay**(SFXTrack *track*, float *x*, float *y*, float *z*)

Create a new play-once source for the given track , position its 3D sound at the given coordinates (if the track's description is set up for 3D sound) and start playback of the source. This is equivalent to calling sfxCreateSource() on and SFXSource::play() on the resulting source. Play-Once Sources

Parameters

- **track** – The sound datablock to play.
- **x** – The X coordinate of the 3D sound position.
- **y** – The Y coordinate of the 3D sound position.
- **z** – The Z coordinate of the 3D sound position.

Returns The newly created play-once source or 0 if the creation failed.

SFXSource **sfxPlayOnce**(SFXTrack *track*)

Create a play-once source for the given track . Once playback has finished, the source will be automatically deleted in the next sound system update. Play-Once Sources

Parameters **track** – The sound datablock.

Returns A newly created temporary source in "Playing" state or 0 if the operation failed.

SFXSource **sfxPlayOnce**(SFXTrack *track*, float *x*, float *y*, float *z*, float *fadeInTime*)

Create a play-once source for the given given track and position the source's 3D sound at the given coordinates only if the track's description is set up for 3D sound). Once playback has finished, the source will be automatically deleted in the next sound system update. Play-Once Sources

Parameters

- **track** – The sound datablock.
- **x** – The X coordinate of the 3D sound position.
- **y** – The Y coordinate of the 3D sound position.

- **z** – The Z coordinate of the 3D sound position.
- **fadeInTime** – If >=0, this overrides the SFXDescription::fadeInTime value on the track's description.

Returns A newly created temporary source in "Playing" state or 0 if the operation failed.

Example:

```
// Immediately start playing the given track.  Fade it in to full volume over 5 seconds.
sfxPlayOnce( MusicTrack, 0, 0, 0, 5.f );
```

SFXSource **sfxPlayOnce**(SFXDescription *description*, string *filename*)

Create a new temporary SFXProfile from the given description and filename , then create a play-once source for it and start playback. Once playback has finished, the source will be automatically deleted in the next sound system update. If not referenced otherwise by then, the temporary SFXProfile will also be deleted. Play-Once Sources

Parameters

- **description** – The description to use for playback.
- **filename** – Path to the sound file to play.

Returns A newly created temporary source in "Playing" state or 0 if the operation failed.

Example:

```
// Play a sound effect file once.
sfxPlayOnce( AudioEffects, "art/sound/weapons/Weapon_pickup" );
```

SFXSource **sfxPlayOnce**(SFXDescription *description*, string *filename*, float *x*, float *y*, float *z*, float *fadeInTime*)

Create a new temporary SFXProfile from the given description and filename , then create a play-once source for it and start playback. Position the source's 3D sound at the given coordinates (only if the description is set up for 3D sound). Once playback has finished, the source will be automatically deleted in the next sound system update. If not referenced otherwise by then, the temporary SFXProfile will also be deleted. Play-Once Sources

Parameters

- **description** – The description to use for playback.
- **filename** – Path to the sound file to play.
- **x** – The X coordinate of the 3D sound position.
- **y** – The Y coordinate of the 3D sound position.
- **z** – The Z coordinate of the 3D sound position.
- **fadeInTime** – If >=0, this overrides the SFXDescription::fadeInTime value on the track's description.

Returns A newly created temporary source in "Playing" state or 0 if the operation failed.

Example:

```
// Play a sound effect file once using a 3D sound with a default falloff placed at the origin.
sfxPlayOnce( AudioDefault3D, "art/sound/weapons/Weapon_pickup", 0, 0, 0 );
```

void **sfxSetDistanceModel**(SFXDistanceModel *model*)

Set the falloff curve type to use for distance-based volume attenuation of 3D sounds.

Parameters **model** – The distance model to use for 3D sound.

void **sfxSetDopplerFactor**(float *value*)
Set the global doppler effect scale factor. Doppler Effect

Parameters **value** – The new doppler shift scale factor.

void **sfxSetRolloffFactor**(float *value*)
Set the global scale factor to apply to volume attenuation of 3D sounds in the logarithmic model. Volume Attenuation 3D Audio

Parameters **value** – The new scale factor for logarithmic 3D sound falloff curves.

void **sfxStop**(SFXSource *source*)
Stop playback of the given source . This is equivalent to calling SFXSource::stop() .

Parameters **source** – The source to put into stopped state.

void **sfxStopAndDelete**(SFXSource *source*)
Stop playback of the given source (if it is not already stopped) and delete the source . The advantage of this function over directly calling delete() is that it will correctly handle volume fades that may be configured on the source. Whereas calling delete() would immediately stop playback and delete the source, this functionality will wait for the fade-out to play and only then stop the source and delete it. Volume Fades

Parameters **source** – A sound source.

Variables

int $SFX::ambientUpdateTime
Milliseconds spent on the last ambient audio update. Sound System Updates Ambient Audio

int $SFX::DEVICE_CAPS_DSPEFFECTS
Sound device capability flag indicating that the sound device supports adding DSP effect chains to sounds.

int $SFX::DEVICE_CAPS_FMODDESIGNER
Sound device capability flag indicating that the sound device supports FMOD Designer audio projects. FMOD Designer Audio

int $SFX::DEVICE_CAPS_MULTILISTENER
Sound device capability flag indicating that the sound device supports multiple concurrent listeners.

int $SFX::DEVICE_CAPS_OCCLUSION
Sound device capability flag indicating that the sound device implements sound occlusion.

int $SFX::DEVICE_CAPS_REVERB
Sound device capability flag indicating that the sound device supports reverb. Audio Reverb

int $SFX::DEVICE_CAPS_VOICEMANAGEMENT
Sound device capability flag indicating that the sound device implements its own voice virtualization. For these devices, the sound system will deactivate its own voice management and leave voice virtualization entirely to the device. Sounds and Voices

int $SFX::DEVICE_INFO_CAPS
Index of device capability flags in device info string.

int $SFX::DEVICE_INFO_MAXBUFFERS
Index of buffer limit number in device info string.

int $SFX::DEVICE_INFO_NAME
Index of device name field in device info string.

int $SFX::DEVICE_INFO_PROVIDER
Index of sound provider field in device info string.

int $SFX::DEVICE_INFO_USEHARDWARE
Index of use hardware flag in device info string.

int $SFX::numCulled
Number of SFXSounds that are currently in virtualized playback mode. Sounds and Voices

int $SFX::numPlaying
Number of SFXSources that are currently in playing state.

int $SFX::numSounds
Number of SFXSound type objects (i.e. actual single-file sounds) that are currently instantiated.

int $SFX::numSources
Number of SFXSource type objects that are currently instantiated.

int $SFX::numVoices
Number of voices that are currently allocated on the sound device.

int $SFX::parameterUpdateTime
Milliseconds spent on the last SFXParameter update loop. Sound System Updates Interactive Audio

ColorI SFXEmitter::**renderColorInnerCone**[static, inherited]
The color with which to render dots in the inner sound cone (Editor only).

ColorI SFXEmitter::**renderColorOuterCone**[static, inherited]
The color with which to render dots in the outer sound cone (Editor only).

ColorI SFXEmitter::**renderColorOutsideVolume**[static, inherited]
The color with which to render dots outside of the outer sound cone (Editor only).

ColorI SFXEmitter::**renderColorPlayingInRange**[static, inherited]
The color with which to render a sound emitter's marker cube in the editor when the emitter's sound is playing and in range of the listener.

ColorI SFXEmitter::**renderColorPlayingOutOfRange**[static, inherited]
The color with which to render a sound emitter's marker cube in the editor when the emitter's sound is playing but out of the range of the listener.

ColorI SFXEmitter::**renderColorRangeSphere**[static, inherited]
The color of the range sphere with which to render sound emitters in the editor.

ColorI SFXEmitter::**renderColorStoppedInRange**[static, inherited]
The color with which to render a sound emitter's marker cube in the editor when the emitter's sound is not playing but the emitter is in range of the listener.

ColorI SFXEmitter::**renderColorStoppedOutOfRange**[static, inherited]
The color with which to render a sound emitter's marker cube in the editor when the emitter's sound is not playing and the emitter is out of range of the listener.

bool SFXEmitter::**renderEmitters**[static, inherited]
Whether to render enhanced range feedback in the editor on all emitters regardless of selection state.

float SFXEmitter::**renderPointDistance**[static, inherited]
The distance between individual points in the sound emitter rendering in the editor as the points move from the emitter's center away to maxDistance.

float SFXEmitter::**renderRadialIncrements**[static, inherited]
The stepping (in degrees) for the radial sweep along the axis of the XY plane sweep for sound emitter rendering in the editor.

float SFXEmitter::**renderSweepIncrements**[static, inherited]
The stepping (in degrees) for the radial sweep on the XY plane for sound emitter rendering in the editor.

int $SFX::sourceUpdateTime
Milliseconds spent on the last SFXSource update loop. Sound System Updates

FMOD

Functionality specific to the FMOD SFX implementation.

Classes

SFXFMODEvent A playable sound event in an FMOD Designer audio project.

Inherit: SFXTrack

Description A playable sound event in an FMOD Designer audio project.

Fields
SFXFMODEventGroup SFXFMODEvent::**fmodGroup**
DO NOT MODIFY!!
string SFXFMODEvent::**fmodName**
DO NOT MODIFY!!

Point2F SFXFMODEvent::**fmodParameterRanges**[8]
DO NOT MODIFY!!

float SFXFMODEvent::**fmodParameterValues**[8]
DO NOT MODIFY!!

SFXFMODEventGroup A group of events in an imported FMOD Designer project.

Inherit: SimDataBlock

Description A group of events in an imported FMOD Designer project.

Methods
void SFXFMODEventGroup::**freeData**()
Release the resource data for this group and its subgroups.
bool SFXFMODEventGroup::**isDataLoaded**()
Test whether the resource data for this group has been loaded.

Returns True if the resource data for this group is currently loaded.

bool SFXFMODEventGroup::**loadData**(bool *loadStreams*, bool *loadSamples*)
Load the resource data for this group, if it has not already been loaded (either directly or indirectly through a parent group). This method works recursively and thus data for direct and indirect child groups to this group will be loaded as well.

Parameters

- **loadStreams** – Whether to open streams.
- **loadSamples** – Whether to load sample banks.

Returns True if the data has been successfully loaded; false otherwise.

Fields

SFXFMODEventGroup `SFXFMODEventGroup::`**`fmodGroup`**
DO NOT MODIFY!!

string `SFXFMODEventGroup::`**`fmodName`**
DO NOT MODIFY!!

SFXFMODProject `SFXFMODEventGroup::`**`fmodProject`**
DO NOT MODIFY!!

SFXFMODEventSource A sound source controller playing an FMOD Designer event (SFXFMODEvent).

Inherit: SFXSource

Description A sound source controller playing an FMOD Designer event (SFXFMODEvent).

FMOD event sources are internally created by the sound system to play events from imported FMOD Designer projects.

SFXFMODProject An FMOD Designer project loaded into Torque.

Inherit: SimDataBlock

Description An FMOD Designer project loaded into Torque.

Resource Loading

Fields

filename `SFXFMODProject::`**`fileName`**
The compiled .fev file from FMOD Designer.

filename `SFXFMODProject::`**`mediaPath`**
Path to the media files; if unset, defaults to project directory.

Description When using FMOD for audio output in combination with Torque's sound system, an extended set of features is available to the user. This includes:

- Reverb support
- Enhanced voice virtualization
- Support for multiple listeners
- Enhanced sound format support: .aiff, .asf, .asx, .dls, .flac .fsb, .it, .m3u, .mid, .mod, .mp2, .mp3, .ogg, .pls, .s3m, .vag, .wav, .wax, .wma, .xm, .xma (on Xbox only)
- FMOD Designer enhanced audio design support

Functions

void **`fmodDumpDSPInfo`**`()`
Dump information about the standard DSP effects.

void **`fmodDumpMemoryStats`**`()`

Returns Prints the current memory consumption of the FMOD module

Variables

bool $pref::SFX::FMOD::disableSoftware
Whether to disable the FMOD software mixer to conserve memory. All sounds not created with SFXDescription::useHardware or using DSP effects will fail to load.

string $pref::SFX::FMOD::DSoundHRTF
The type of HRTF to use for hardware-mixed 3D sounds when FMOD is using DirectSound for sound output and hardware-acceleration is not available. Options are

- "none": simple stereo panning/doppler/attenuation
- "light": slightly higher quality than "none"
- "full": full quality 3D playback

bool $pref::SFX::FMOD::enableProfile
Whether to enable support for FMOD's profiler. Using the FMOD Profiler with Torque

int $SFX::Device::fmodCoreMem
Current number of bytes allocated by the core FMOD sound system.

int $SFX::Device::fmodEventMem
Current number of bytes allocated by the FMOD Designer event system.

int $SFX::Device::fmodNumEventSources
The current number of SFXFMODEventSource instances in the system. This tells the number of sounds in the system that are currently playing FMOD Designer events.

string $pref::SFX::FMOD::pluginPath
Path to additional FMOD plugins.

bool $pref::SFX::FMOD::useSoftwareHRTF
Whether to enable HRTF in FMOD's software mixer. This will add a lowpass filter effect to the DSP effect chain of all sounds mixed in software.

5.3.6 Rendering

All rendering related functionality.

Rendering

All rendering related functionality.

Classes

BarrelDistortionPostEffect A fullscreen shader effect used with the Oculus Rift.

Inherit: PostEffect

Description A fullscreen shader effect used with the Oculus Rift.

PFXTextureIdentifiers

Fields

int BarrelDistortionPostEffect::**hmdIndex**
Oculus VR HMD index to reference.

float BarrelDistortionPostEffect::**scaleOutput**
Used to increase the size of the window into the world at the expense of apparent resolution.

int BarrelDistortionPostEffect::**sensorIndex**
Oculus VR sensor index to reference.

PostEffect A fullscreen shader effect.

Inherit: SimGroup

Description A fullscreen shader effect.

PFXTextureIdentifiers

Methods

void PostEffect::**clearShaderMacros**()
Remove all shader macros.

void PostEffect::**disable**()
Disables the effect.

String PostEffect::**dumpShaderDisassembly**()
Dumps this PostEffect shader's disassembly to a temporary text file.

Returns Full path to the dumped file or an empty string if failed.

void PostEffect::**enable**()
Enables the effect.

float PostEffect::**getAspectRatio**()

Returns Width over height of the backbuffer.

bool PostEffect::**isEnabled**()

Returns True if the effect is enabled.

void PostEffect::**onAdd**()
Called when this object is first created and registered.

void PostEffect::**onDisabled**()
Called when this effect becomes disabled.

bool PostEffect::**onEnabled**()
Called when this effect becomes enabled. If the user returns false from this callback the effect will not be enabled.

Returns True to allow this effect to be enabled.

void PostEffect::**preProcess**()
Called when an effect is processed but before textures are bound. This allows the user to change texture related paramaters or macros at runtime.

Example:

```
function SSAOPostFx::preProcess( %this )
{
   if ( $SSAOPostFx::quality !$= %this.quality )
   {
      %this.quality = mClamp( mRound( $SSAOPostFx::quality ), 0, 2 );

      %this.setShaderMacro( "QUALITY", %this.quality );
   }
   %this.targetScale = $SSAOPostFx::targetScale;
}
```

void PostEffect::**reload**()
Reloads the effect shader and textures.

void PostEffect::**removeShaderMacro**(string *key*)
Remove a shader macro. This will usually be called within the preProcess callback.

Parameters **key** – Macro to remove.

void PostEffect::**setShaderConst**(string *name*, string *value*)
Sets the value of a uniform defined in the shader. This will usually be called within the setShaderConsts callback. Array type constants are not supported.

Parameters

- **name** – Name of the constanst, prefixed with '$'.
- **value** – Value to set, space seperate values with more than one element.

Example:

```
function MyPfx::setShaderConsts( %this )
{
   // example float4 uniform
   %this.setShaderConst( "$colorMod", "1.0 0.9 1.0 1.0" );
   // example float1 uniform
   %this.setShaderConst( "$strength", "3.0" );
   // example integer uniform
   %this.setShaderConst( "$loops", "5" );}
```

void PostEffect::**setShaderConsts**()
Called immediate before processing this effect. This is the user's chance to set the value of shader uniforms (constants).

void PostEffect::**setShaderMacro**(string *key*, string *value*)
Adds a macro to the effect's shader or sets an existing one's value. This will usually be called within the onAdd or preProcess callback.

Parameters

- **key** – lval of the macro.
- **value** – rval of the macro, or may be empty.

Example:

```
function MyPfx::onAdd( %this )
{
   %this.setShaderMacro( "NUM_SAMPLES", "10" );
   %this.setShaderMacro( "HIGH_QUALITY_MODE" );
```

```
        // In the shader looks like... // #define NUM_SAMPLES 10// #define HIGH_QUALITY_MODE
    }
```

void PostEffect::**setTexture**(int *index*, string *filePath*)
This is used to set the texture file and load the texture on a running effect. If the texture file is not different from the current file nothing is changed. If the texture cannot be found a null texture is assigned.

Parameters

- **index** – The texture stage index.
- **filePath** – The file name of the texture to set.

bool PostEffect::**toggle**()
Toggles the effect between enabled / disabled.

Returns True if effect is enabled.

Fields

bool PostEffect::**allowReflectPass**
Is this effect processed during reflection render passes.

bool PostEffect::**isEnabled**
Is the effect on.

bool PostEffect::**oneFrameOnly**
Allows you to turn on a PostEffect for only a single frame.

bool PostEffect::**onThisFrame**
Allows you to turn on a PostEffect for only a single frame.

string PostEffect::**renderBin**
Name of a renderBin, used if renderTime is PFXBeforeBin or PFXAfterBin.

float PostEffect::**renderPriority**
PostEffects are processed in DESCENDING order of renderPriority if more than one has the same renderBin/Time.

PFXRenderTime PostEffect::**renderTime**
When to process this effect during the frame.

string PostEffect::**shader**
Name of a GFXShaderData for this effect.

bool PostEffect::**skip**
Skip processing of this PostEffect and its children even if its parent is enabled. Parent and sibling PostEffects in the chain are still processed.

GFXStateBlockData PostEffect::**stateBlock**
Name of a GFXStateBlockData for this effect.

string PostEffect::**target**
String identifier of this effect's target texture.

PFXTargetClear PostEffect::**targetClear**
Describes when the target texture should be cleared.

ColorF PostEffect::**targetClearColor**
Color to which the target texture is cleared before rendering.

string PostEffect::**targetDepthStencil**
Optional string identifier for this effect's target depth/stencil texture.

GFXFormat PostEffect::**targetFormat**
Format of the target texture, not applicable if writing to the backbuffer.

Point2F PostEffect::**targetScale**
If targetSize is zero this is used to set a relative size from the current target.

Point2I PostEffect::**targetSize**
If non-zero this is used as the absolute target size.

PFXTargetViewport PostEffect::**targetViewport**
Specifies how the viewport should be set up for a target texture.

filename PostEffect::**texture**[6]
Input textures to this effect (samplers).

TheoraTextureObject Definition of a named texture target playing a Theora video.

Inherit: SimObject

Description TheoraTextureObject defines a named texture target that may play back a Theora video. This texture target can, for example, be used by materials to texture objects with videos.

Example:

```
// The object that provides the video texture and controls its playback.
singleton TheoraTextureObject( TheVideo )
{
   // Unique name for the texture target for referencing in materials.
   texTargetName = "video";

   // Path to the video file.
   theoraFile = "./MyVideo.ogv";
};

// Material that uses the video texture.
singleton Material( TheVideoMaterial )
{
   // This has to reference the named texture target defined by the
   // TheoraTextureObjects texTargetName property.  Prefix with # to
   // identify as texture target reference.
   diffuseMap[ 0 ] = "#video";
};
```

Methods

void TheoraTextureObject::**pause**()
Pause playback of the video.

void TheoraTextureObject::**play**()
Start playback of the video.

void TheoraTextureObject::**stop**()
Stop playback of the video.

Fields

bool TheoraTextureObject::**loop**
Should the video loop.

SFXDescription `TheoraTextureObject::`**`SFXDescription`**
Sound description to use for the video's audio channel. If not set, will use a default one.

string `TheoraTextureObject::`**`texTargetName`**
Name of the texture target by which the texture can be referenced in materials.

filename `TheoraTextureObject::`**`theoraFile`**
Theora video file to play.

Enumeration

`enum PFXRenderTime`
When to process this effect during the frame.

Parameters

- **`PFXBeforeBin`** – Before a RenderInstManager bin.
- **`PFXAfterBin`** – After a RenderInstManager bin.
- **`PFXAfterDiffuse`** – After the diffuse rendering pass.
- **`PFXEndOfFrame`** – When the end of the frame is reached.
- **`PFXTexGenOnDemand`** – This PostEffect is not processed by the manager. It will generate its texture when it is requested.

`enum PFXTargetClear`
Describes when the target texture should be cleared.

Parameters

- **`PFXTargetClear_None`** – Never clear the PostEffect target.
- **`PFXTargetClear_OnCreate`** – Clear once on create.
- **`PFXTargetClear_OnDraw`** – Clear before every draw.

`enum PFXTargetViewport`
Specifies how the viewport should be set up for a PostEffect's target.

Parameters

- **`PFXTargetViewport_TargetSize`** – Set viewport to match target size (default).
- **`PFXTargetViewport_GFXViewport`** – Use the current GFX viewport (scaled to match target size).
- **`PFXTargetViewport_NamedInTexture0`** – Use the input texture 0 if it is named (scaled to match target size), otherwise revert to PFXTargetViewport_TargetSize if there is none.

Functions

void **`addGlobalShaderMacro`**(string *name*, string *value*)
Adds a global shader macro which will be merged with the script defined macros on every shader. The macro will replace the value of an existing macro of the same name. For the new macro to take effect all the shaders in the system need to be reloaded.

void **beginSampling**()
Takes a string informing the backend where to store sample data and optionally a name of the specific logging backend to use. The default is the CSV backend. In most cases, the logging store will be a file name.

Example:

```
beginSampling( "mysamples.csv" );
```

void **enableSamples**()
Enable sampling for all keys that match the given name pattern. Slashes are treated as separators.

int **getActiveDDSFiles**()
Returns the count of active DDSs files in memory.

String **getBitmapInfo**(string *filename*)
Returns image info in the following format: width TAB height TAB bytesPerPixel. It will return an empty string if the file is not found.

void **initDisplayDeviceInfo**()
Initializes variables that track device and vendor information/IDs.

void **playJournalToVideo**(string *journalFile*, string *videoFile*, string *encoder*, float *framerate*, Point2I *resolution*)
Load a journal file and capture it video.

void **removeGlobalShaderMacro**(string *name*)
Removes an existing global macro by name.

void **startVideoCapture**(GuiCanvas *canvas*, string *filename*, string *encoder*, float *framerate*, Point2I *resolution*)
Begins a video capture session.

void **stopSampling**()
Stops the rendering sampler.

void **stopVideoCapture**()
Stops the video capture session.

Variables

bool $pref::imposter::canShadow
User preference which toggles shadows from imposters. Defaults to true.

float $pref::TS::detailAdjust
User perference for scaling the TSShape level of detail. The smaller the value the closer the camera must get to see the highest LOD. This setting can have a huge impact on performance in mesh heavy scenes. The default value is 1.

bool $Scene::disableTerrainOcclusion
Used to disable the somewhat expensive terrain occlusion testing.

bool $Scene::disableZoneCulling
If true, zone culling will be disabled and the scene contents will only be culled against the root frustum.

int $TSControl::frameCount
The number of frames that have been rendered since this control was created.

int $pref::Reflect::frameLimitMS
ReflectionManager tries not to spend more than this amount of time updating reflections per frame.

int $Sampler::frequency
Samples taken every nth frame.

bool $Scene::lockCull
Debug tool which locks the frustum culling to the current camera location.

int $pref::TS::maxInstancingVerts
Enables mesh instancing on non-skin meshes that have less that this count of verts. The default value is 200. Higher values can degrade performance.

int $Scene::maxOccludersPerZone
Maximum number of occluders that will be concurrently allowed into the scene culling state of any given zone.

float $Scene::occluderMinHeightPercentage
TODO.

float $Scene::occluderMinWidthPercentage
TODO.

float $pref::Reflect::refractTexScale
RefractTex has dimensions equal to the active render target scaled in both x and y by this float.

bool $Scene::renderBoundingBoxes
If true, the bounding boxes of objects will be displayed.

int $pref::TS::skipLoadDLs
User perference which causes TSShapes to skip loading higher lods. This potentialy reduces the GPU resources and materials generated as well as limits the LODs rendered. The default value is 0.

int $pref::TS::skipRenderDLs
User perference which causes TSShapes to skip rendering higher lods. This will reduce the number of draw calls and triangles rendered and improve rendering performance when proper LODs have been created for your models. The default value is 0.

float $pref::TS::smallestVisiblePixelSize
User perference which sets the smallest pixel size at which TSShapes will skip rendering. This will force all shapes to stop rendering when they get smaller than this size. The default value is -1 which disables it.

float $pref::windEffectRadius
Radius to affect the wind.

Font

Various helpers for working with fonts from script.

Functions

void **dumpFontCacheStatus**()
Dumps to the console a full description of all cached fonts, along with info on the codepoints each contains.

void **duplicateCachedFont**(string *oldFontName*, int *oldFontSize*, string *newFontName*)
Copy the specified old font to a new name. The new copy will not have a platform font backing it, and so will never have characters added to it. But this is useful for making copies of fonts to add postprocessing effects to via exportCachedFont.

Parameters

- **oldFontName** – The name of the font face to copy.
- **oldFontSize** – The size of the font to copy.
- **newFontName** – The name of the new font face.

void **exportCachedFont** (string *faceName*, int *fontSize*, string *fileName*, int *padding*, int *kerning*)
Export specified font to the specified filename as a PNG. The image can then be processed in Photoshop or another tool and reimported using importCachedFont. Characters in the font are exported as one long strip.

Parameters

- **faceName** – The name of the font face.
- **fontSize** – The size of the font in pixels.
- **fileName** – The file name and path for the output PNG.
- **padding** – The padding between characters.
- **kerning** – The kerning between characters.

void **importCachedFont** (string *faceName*, int *fontSize*, string *fileName*, int *padding*, int *kerning*)
Import an image strip from exportCachedFont. Call with the same parameters you called exportCachedFont.

Parameters

- **faceName** – The name of the font face.
- **fontSize** – The size of the font in pixels.
- **fileName** – The file name and path for the input PNG.
- **padding** – The padding between characters.
- **kerning** – The kerning between characters.

void **populateAllFontCacheRange** (int *rangeStart*, int *rangeEnd*)
Populate the font cache for all fonts with Unicode code points in the specified range.

Parameters

- **rangeStart** – The start Unicode point.
- **rangeEnd** – The end Unicode point.

void **populateAllFontCacheString** (string *string*)
Populate the font cache for all fonts with characters from the specified string.

void **populateFontCacheRange** (string *faceName*, int *fontSize*, int *rangeStart*, int *rangeEnd*)
Populate the font cache for the specified font with Unicode code points in the specified range.

Parameters

- **faceName** – The name of the font face.
- **fontSize** – The size of the font in pixels.
- **rangeStart** – The start Unicode point.
- **rangeEnd** – The end Unicode point.

void **populateFontCacheString** (string *faceName*, int *fontSize*, string *string*)
Populate the font cache for the specified font with characters from the specified string.

Parameters

- **faceName** – The name of the font face.
- **fontSize** – The size of the font in pixels.
- **string** – The string to populate.

void **writeFontCache** ()
Force all cached fonts to serialize themselves to the cache.

GFX

The low level graphics interface to the engine.

Classes

CubemapData Used to create static or dynamic cubemaps.

Inherit: SimObject

Description This object is used with Material, WaterObject, and other objects for cubemap reflections.

A simple declaration of a static cubemap:

Example:

```
singleton CubemapData( SkyboxCubemap )
{
   cubeFace[0] = "./skybox_1";
   cubeFace[1] = "./skybox_2";
   cubeFace[2] = "./skybox_3";
   cubeFace[3] = "./skybox_4";
   cubeFace[4] = "./skybox_5";
   cubeFace[5] = "./skybox_6";
};
```

Methods

string `CubemapData::getFilename()`
Returns the script filename of where the CubemapData object was defined. This is used by the material editor. Reimplemented from SimObject .

void `CubemapData::updateFaces()`
Update the assigned cubemaps faces.

Fields

filename `CubemapData::cubeFace`[6]
The 6 cubemap face textures for a static cubemap. They are in the following order:

- cubeFace[0] is -X
- cubeFace[1] is +X
- cubeFace[2] is -Z
- cubeFace[3] is +Z
- cubeFace[4] is -Y
- cubeFace[5] is +Y

bool `CubemapData::dynamic`
Set to true if this is a dynamic cubemap. The default is false.

float `CubemapData::dynamicFarDist`
The far clip distance used when rendering to the dynamic cubemap.

float `CubemapData::dynamicNearDist`
The near clip distance used when rendering to the dynamic cubemap.

int CubemapData::**dynamicObjectTypeMask**
The typemask used to filter the objects rendered to the dynamic cubemap.

int CubemapData::**dynamicSize**
The size of each dynamic cubemap face in pixels.

DebugDrawer A debug helper for rendering debug primitives to the scene.

Inherit: SimObject

Description The DebugDrawer is used to render debug primitives to the scene for testing. It is often useful when debugging collision code or complex 3d algorithms to have them draw debug information, like culling hulls or bounding volumes, normals, simple lines, and so forth.

A key feature of the DebugDrawer is that each primitive gets a "time to live" (TTL) which allows them to continue to render to the scene for a fixed period of time. You can freeze or resume the system at any time to allow you to examine the output.

Example:

```
DebugDraw.drawLine( %player.getMuzzlePoint( 0 ), %hitPoint );
DebugDraw.setLastTTL( 5000 ); // 5 seconds.
```

The DebugDrawer renders solely in world space and all primitives are rendered with the cull mode disabled.

Methods

void DebugDrawer::**drawBox**(Point3F *a*, Point3F *b*, ColorF *color*)
Draws an axis aligned box primitive within the two 3d points.

void DebugDrawer::**drawLine**(Point3F *a*, Point3F *b*, ColorF *color*)
Draws a line primitive between two 3d points.

void DebugDrawer::**setLastTTL**(int *ms*)
Sets the "time to live" (TTL) for the last rendered primitive.

void DebugDrawer::**setLastZTest**(bool *enabled*)
Sets the z buffer reading state for the last rendered primitive.

void DebugDrawer::**toggleDrawing**()
Toggles the rendering of DebugDrawer primitives.

void DebugDrawer::**toggleFreeze**()
Toggles freeze mode which keeps the currently rendered primitives from expiring.

GFXCardProfiler Provides a device independent wrapper around both the capabilities reported by the card/drivers and the exceptions recorded in various scripts.

Description The GFXCardProfiler provides a device independent wrapper around both the capabilities reported by the card/drivers and the exceptions recorded in various scripts.

The materials system keeps track of most caps-related rendering optimizations and/or workarounds, but it is occasionally necessary to expose capability information to higher level code (for instance, if some feature depends on a specific subset of render functionality) or to keep track of exceptions.

The proper way to fix this is to get the IHV to release fixed drivers and/or move to a single consistent rendering path that works. Of course, when you're releasing a game, especially on a timeline (or with a less than infinite budget) this isn't always a valid solution.

It's also often convenient to be able to tweak performance/detail settings based on the identified card type.

GFXCardProfiler addresses both these needs by providing two data retrieval methods and a generic interface for querying capability strings.

Note: The GFXCardProfiler is at heart a system for implementing WORKAROUNDS. It is not guaranteed to work in all cases. The capability strings it responds to are specific to each implementation. You should be EXTREMELY careful when working with this functionality. When used in moderation it can be a project-saver, but if used to excess or without forethought it can lead to complex, hard-to-maintain code.

The first data retrieval method that the GFXCardProfiler supports is a card-specific capability query. This is implemented by each subclass. In the case of DirectX, this means using the built-in capability query. For OpenGL or other APIs, more exotic methods may be necessary. The goal of this method is to retrieve some reasonable defaults that can be overridden later if necessary.

The second data retrieval method is script based. In ./profile a collection of script files are stored. They are named in one of the forms:

Renderer.cs Renderer.VendorString.CardString.cs Renderer.VendorString.CardString.cs Renderer.VendorString.CardString.VersionString.card-specific

These files are found and executed from most general to most specific. For instance, say we're working in the D3D renderer with an nVidia GeForce FX 5950, running driver version 53.36. The following files would be found and executed:

D3D.cs D3D.nVidia.cs D3D.nVidia.GeForceFX5950.cs D3D.nVidia.GeForceFX5950.5336.cs

The general rule for turning strings into filename parts is to strip all spaces and punctuation. If a file is not found, no error is reported; it is assumed that the absence of a file means all is well.

Several functions are made available to allow simple logic in the script functions (for instance, to enable a workaround for a given range of driver versions). They are:

- GFXCardProfiler::getRenderer()
- GFXCardProfiler::getVendor()
- GFXCardProfiler::getCard()
- GFXCardProfiler::getVersion()

In addition, specific subclasses may expose other values (for instance, chipset IDs). These are made available as static members of the specific subclass. For instance, a D3D-specific chipset query may be made available as GFXD3DCardProfiler::getChipset().

Finally, once a script file has reached a determination they may indicate their settings to the GFXCardProfiler by calling GFXCardProfiler::setCapability(). For instance,

```
    // Indicate we can show the color red.
GFXCardProfiler::setCapability("supportsRed", true);
```

GFXCardProfiler may be queried from script by calling GFXCardProfiler::queryProfile() - for instance:

```
GFXCardProfiler::queryProfile("supportsRed", false); // Query with default.
```

GFXCardProfilerAPI This class is the interface between TorqueScript and GFXCardProfiler.

Description You will not actually declare GFXCardProfilerAPI in TorqueScript. It exists solely to give access to the GFXCardProfiler's querying functions, such as GFXCardProfiler::getRenderer.

Example:

```
// Example of accessing GFXCardProfiler function from script
// Notice you are not using the API version
%videoMem = GFXCardProfiler::getVideoMemoryMB();
```

Methods

static String GFXCardProfilerAPI::**getCard**()
Returns the card name.

static String GFXCardProfilerAPI::**getRenderer**()
Returns the renderer name. For example D3D9 or OpenGL.

static String GFXCardProfilerAPI::**getVendor**()
Returns the card vendor name.

static String GFXCardProfilerAPI::**getVersion**()
Returns the driver version string.

static int GFXCardProfilerAPI::**getVideoMemoryMB**()
Returns the amount of video memory in megabytes.

static int GFXCardProfilerAPI::**queryProfile**(string *name*, int *defaultValue*)
Used to query the value of a specific card capability.

Parameters

- **name** – The name of the capability being queried.
- **defaultValue** – The value to return if the capability is not defined.

static void GFXCardProfilerAPI::**setCapability**(string *name*, int *value*)
Used to set the value for a specific card capability.

Parameters

- **name** – The name of the capability being set.
- **value** – The value to set for that capability.

GFXInit Functions for tracking GFX adapters and initializing them into devices.

Description Functions for tracking GFX adapters and initializing them into devices.

Methods

static void GFXInit::**createNullDevice**()
Create the NULL graphics device used for testing or headless operation.

static String GFXInit::**getAdapterMode**(int *index*, int *modeIndex*)
Gets the details of the specified adapter mode.

Parameters

- **index** – Index of the adapter to query.
- **modeIndex** – Index of the mode to get data from.

Returns A video mode string in the format 'width height fullscreen bitDepth refreshRate aaLevel'.

static int GFXInit::**getAdapterModeCount** (int *index*)
Gets the number of modes available on the specified adapter.

Parameters **index** – Index of the adapter to get modes from.

Returns The number of video modes supported by the adapter or -1 if the given adapter was not found.

static String GFXInit::**getAdapterName** (int *index*)
Returns the name of the graphics adapter.

Parameters **index** – The index of the adapter.

static String GFXInit::**getAdapterOutputName** (int *index*)
Returns the name of the graphics adapter's output display device.

Parameters **index** – The index of the adapter.

static float GFXInit::**getAdapterShaderModel** (int *index*)
Returns the supported shader model of the graphics adapter or -1 if the index is bad.

Parameters **index** – The index of the adapter.

static GFXAdapterType GFXInit::**getAdapterType** (int *index*)
Returns the type (D3D9, D3D8, GL, Null) of a graphics adapter.

Parameters **index** – The index of the adapter.

static int GFXInit::**getDefaultAdapterIndex** ()
Returns the index of the default graphics adapter. This is the graphics device which will be used to initialize the engine.

GFXSamplerStateData A sampler state used by GFXStateBlockData.

Inherit: SimObject

Description The samplers define how a texture will be sampled when used from the shader or fixed function device.

Example:

```
singleton GFXSamplerStateData(SamplerClampLinear)
{
   textureColorOp = GFXTOPModulate;
   addressModeU = GFXAddressClamp;
   addressModeV = GFXAddressClamp;
   addressModeW = GFXAddressClamp;
   magFilter = GFXTextureFilterLinear;
   minFilter = GFXTextureFilterLinear;
   mipFilter = GFXTextureFilterLinear;
};
```

There are a few predefined samplers in the core scripts which you can use with GFXStateBlockData for the most common rendering cases:

- SamplerClampLinear
- SamplerClampPoint
- SamplerWrapLinear
- SamplerWrapPoint

Fields

GFXTextureAddressMode GFXSamplerStateData::**addressModeU**
The texture address mode for the u coordinate. The default is GFXAddressWrap.

GFXTextureAddressMode GFXSamplerStateData::**addressModeV**
The texture address mode for the v coordinate. The default is GFXAddressWrap.

GFXTextureAddressMode GFXSamplerStateData::**addressModeW**
The texture address mode for the w coordinate. The default is GFXAddressWrap.

GFXTextureArgument GFXSamplerStateData::**alphaArg1**
The first alpha argument for the texture stage. The default value is GFXTATexture.

GFXTextureArgument GFXSamplerStateData::**alphaArg2**
The second alpha argument for the texture stage. The default value is GFXTADiffuse.

GFXTextureArgument GFXSamplerStateData::**alphaArg3**
The third alpha channel selector operand for triadic operations (multiply, add, and linearly interpolate). The default value is GFXTACurrent.

GFXTextureOp GFXSamplerStateData::**alphaOp**
The texture alpha blending operation. The default value is GFXTOPModulate.

GFXTextureArgument GFXSamplerStateData::**colorArg1**
The first color argument for the texture stage. The default value is GFXTACurrent.

GFXTextureArgument GFXSamplerStateData::**colorArg2**
The second color argument for the texture stage. The default value is GFXTATexture.

GFXTextureArgument GFXSamplerStateData::**colorArg3**
The third color argument for triadic operations (multiply, add, and linearly interpolate). The default value is GFXTACurrent.

GFXTextureFilterType GFXSamplerStateData::**magFilter**
The texture magnification filter. The default is GFXTextureFilterLinear.

int GFXSamplerStateData::**maxAnisotropy**
The maximum texture anisotropy. The default value is 1.

GFXTextureFilterType GFXSamplerStateData::**minFilter**
The texture minification filter. The default is GFXTextureFilterLinear.

GFXTextureFilterType GFXSamplerStateData::**mipFilter**
The texture mipmap filter used during minification. The default is GFXTextureFilterLinear.

float GFXSamplerStateData::**mipLODBias**
The mipmap level of detail bias. The default value is zero.

GFXTextureArgument GFXSamplerStateData::**resultArg**
The selection of the destination register for the result of this stage. The default is GFXTACurrent.

GFXTextureOp GFXSamplerStateData::**textureColorOp**
The texture color blending operation. The default value is GFXTOPDisable which disables the sampler.

GFXTextureTransformFlags GFXSamplerStateData::**textureTransform**
Sets the texture transform state. The default is GFXTTFFDisable.

GFXStateBlockData A state block description for rendering.

Inherit: SimObject

Description This object is used with ShaderData in CustomMaterial and PostEffect to define the render state.

Example:

```
singleton GFXStateBlockData( PFX_DOFDownSampleStateBlock )
{
   zDefined = true;
   zEnable = false;
   zWriteEnable = false;

   samplersDefined = true;
   samplerStates[0] = SamplerClampLinear;
   samplerStates[1] = SamplerClampPoint;

   // Copy the clamped linear sampler, but change
   // the u coord to wrap for this special case.
   samplerStates[2] = newGFXSamplerStateData( : SamplerClampLinear )
   {
      addressModeU = GFXAddressWrap;
   };
};
```

Fields

bool GFXStateBlockData::**alphaDefined**
Set to true if the alpha test state is not all defaults.

bool GFXStateBlockData::**alphaTestEnable**
Enables per-pixel alpha testing. The default is false.

GFXCmpFunc GFXStateBlockData::**alphaTestFunc**
The test function used to accept or reject a pixel based on its alpha value. The default is GFXCmpGreaterEqual.

int GFXStateBlockData::**alphaTestRef**
The reference alpha value against which pixels are tested. The default is zero.

bool GFXStateBlockData::**blendDefined**
Set to true if the alpha blend state is not all defaults.

GFXBlend GFXStateBlockData::**blendDest**
The destination blend state. The default is GFXBlendZero.

bool GFXStateBlockData::**blendEnable**
Enables alpha blending. The default is false.

GFXBlendOp GFXStateBlockData::**blendOp**
The arithmetic operation applied to alpha blending. The default is GFXBlendOpAdd.

GFXBlend GFXStateBlockData::**blendSrc**
The source blend state. The default is GFXBlendOne.

bool GFXStateBlockData::**colorWriteAlpha**
Enables alpha channel writes. The default is true.

bool GFXStateBlockData::**colorWriteBlue**
Enables blue channel writes. The default is true.

bool GFXStateBlockData::**colorWriteDefined**
Set to true if the color write state is not all defaults.

bool GFXStateBlockData::**colorWriteGreen**
Enables green channel writes. The default is true.

bool GFXStateBlockData::**colorWriteRed**
Enables red channel writes. The default is true.

bool GFXStateBlockData::**cullDefined**
Set to true if the culling state is not all defaults.

GFXCullMode GFXStateBlockData::**cullMode**
Defines how back facing triangles are culled if at all. The default is GFXCullCCW.

bool GFXStateBlockData::**ffLighting**
Enables fixed function lighting when rendering without a shader on geometry with vertex normals. The default is false.

bool GFXStateBlockData::**samplersDefined**
Set to true if the sampler states are not all defaults.

GFXSamplerStateData GFXStateBlockData::**samplerStates**[16]
The array of texture sampler states.

bool GFXStateBlockData::**separateAlphaBlendDefined**
Set to true if the seperate alpha blend state is not all defaults.

GFXBlend GFXStateBlockData::**separateAlphaBlendDest**
The destination blend state. The default is GFXBlendZero.

bool GFXStateBlockData::**separateAlphaBlendEnable**
Enables the separate blend mode for the alpha channel. The default is false.

GFXBlendOp GFXStateBlockData::**separateAlphaBlendOp**
The arithmetic operation applied to separate alpha blending. The default is GFXBlendOpAdd.

GFXBlend GFXStateBlockData::**separateAlphaBlendSrc**
The source blend state. The default is GFXBlendOne.

bool GFXStateBlockData::**stencilDefined**
Set to true if the stencil state is not all defaults.

bool GFXStateBlockData::**stencilEnable**
Enables stenciling. The default is false.

GFXStencilOp GFXStateBlockData::**stencilFailOp**
The stencil operation to perform if the stencil test fails. The default is GFXStencilOpKeep.

GFXCmpFunc GFXStateBlockData::**stencilFunc**
The comparison function to test the reference value to a stencil buffer entry. The default is GFXCmpNever.

int GFXStateBlockData::**stencilMask**
The mask applied to the reference value and each stencil buffer entry to determine the significant bits for the stencil test. The default is 0xFFFFFFFF.

GFXStencilOp GFXStateBlockData::**stencilPassOp**
The stencil operation to perform if both the stencil and the depth tests pass. The default is GFXStencilOpKeep.

int GFXStateBlockData::**stencilRef**
The reference value for the stencil test. The default is zero.

int GFXStateBlockData::**stencilWriteMask**
The write mask applied to values written into the stencil buffer. The default is 0xFFFFFFFF.

GFXStencilOp GFXStateBlockData::**stencilZFailOp**
The stencil operation to perform if the stencil test passes and the depth test fails. The default is GFXStencilOpKeep.

ColorI GFXStateBlockData::**textureFactor**
The color used for multiple-texture blending with the GFXTATFactor texture-blending argument or the GFXTOPBlendFactorAlpha texture-blending operation. The default is opaque white (255, 255, 255, 255).

bool GFXStateBlockData::**vertexColorEnable**
Enables fixed function vertex coloring when rendering without a shader. The default is false.

float GFXStateBlockData::**zBias**
A floating-point bias used when comparing depth values. The default is zero.

bool GFXStateBlockData::**zDefined**
Set to true if the depth state is not all defaults.

bool GFXStateBlockData::**zEnable**
Enables z-buffer reads. The default is true.

GFXCmpFunc GFXStateBlockData::**zFunc**
The depth comparision function which a pixel must pass to be written to the z-buffer. The default is GFXCmpLessEqual.

float GFXStateBlockData::**zSlopeBias**
An additional floating-point bias based on the maximum depth slop of the triangle being rendered. The default is zero.

bool GFXStateBlockData::**zWriteEnable**
Enables z-buffer writes. The default is true.

Material A material in Torque 3D is a data structure that describes a surface.

Inherit: SimObject

Description It contains many different types of information for rendering properties. Torque 3D generates shaders from Material definitions. The shaders are compiled at runtime and output into the example/shaders directory. Any errors or warnings generated from compiling the procedurally generated shaders are output to the console as well as the output window in the Visual C IDE.

Example:

```
singleton Material(DECAL_scorch)
{
   baseTex[0] = "./scorch_decal.png";
   vertColor[ 0 ] = true;

   translucent = true;
   translucentBlendOp = None;
   translucentZWrite = true;
   alphaTest = true;
   alphaRef = 84;
};
```

Fields

int Material::**alphaRef**
The alpha reference value for alpha testing. Must be between 0 to 255.

bool Material::**alphaTest**
Enables alpha test when rendering the material.

MaterialAnimType Material::**animFlags**[4]
The types of animation to play on this material.

filename Material::**baseTex**[4]
For backwards compatibility.

bool Material::**bumpAtlas**[4]

filename Material::**bumpTex**[4]
For backwards compatibility.

bool Material::**castShadows**
If set to false the lighting system will not cast shadows from this material.

Point2I Material::**cellIndex**[4]

Point2I Material::**cellLayout**[4]

int Material::**cellSize**[4]

ColorF Material::**colorMultiply**[4]
For backwards compatibility.

string Material::**cubemap**
The name of a CubemapData for environment mapping.

SFXTrack Material::**customFootstepSound**
The sound to play when the player walks over the material. If this is set, it overrides footstepSoundId . This field is useful for directly assigning custom footstep sounds to materials without having to rely on the PlayerData sound assignment. Be aware that materials are client-side objects. This means that the SFXTracks assigned to materials must be client-side, too.

SFXTrack Material::**customImpactSound**
The sound to play when the player impacts on the surface with a velocity equal or greater than PlayerData::groundImpactMinSpeed . If this is set, it overrides impactSoundId . This field is useful for directly assigning custom impact sounds to materials without having to rely on the PlayerData sound assignment. Be aware that materials are client-side objects. This means that the SFXTracks assigned to materials must be client-side, too.

filename Material::**detailMap**[4]
A typically greyscale detail texture additively blended into the material.

filename Material::**detailNormalMap**[4]
A second normal map texture applied at the detail scale. You can use the DXTnm format only when per-pixel specular highlights are disabled.

float Material::**detailNormalMapStrength**[4]
Used to scale the strength of the detail normal map when blended with the base normal map.

Point2F Material::**detailScale**[4]
The scale factor for the detail map.

filename Material::**detailTex**[4]
For backwards compatibility.

ColorF Material::**diffuseColor**[4]
This color is multiplied against the diffuse texture color. If no diffuse texture is present this is the material color.

filename Material::**diffuseMap**[4]
The diffuse color texture map.

bool Material::**doubleSided**
Disables backface culling casing surfaces to be double sided. Note that the lighting on the backside will be a mirror of the front side of the surface.

void Material::**dumpInstances**
Dumps a formatted list of the currently allocated material instances for this material to the console.

bool Material::**dynamicCubemap**
Enables the material to use the dynamic cubemap from the ShapeBase object its applied to.

ColorF Material::**effectColor**[2]
If showDust is true, this is the set of colors to use for the ParticleData of the dust emitter.

bool Material::**emissive**[4]
Enables emissive lighting for the material.

filename Material::**envMap**[4]
The name of an environment map cube map to apply to this material.

filename Material::**envTex**[4]
For backwards compatibility.

void Material::**flush**
Flushes all material instances that use this material.

int Material::**footstepSoundId**
What sound to play from the PlayerData sound list when the player walks over the material. -1 (default) to not play any sound. The IDs are:

- 0:
- PlayerData::FootSoftSound
- 1:
- PlayerData::FootHardSound
- 2:
- PlayerData::FootMetalSound
- 3:
- PlayerData::FootSnowSound
- 4:
- PlayerData::FootShallowSound
- 5:
- PlayerData::FootWadingSound
- 6:
- PlayerData::FootUnderwaterSound
- 7:
- PlayerData::FootBubblesSound
- 8:
- PlayerData::movingBubblesSound
- 9:
- PlayerData::waterBreathSound
- 10:
- PlayerData::impactSoftSound

- 11:
- PlayerData::impactHardSound
- 12:
- PlayerData::impactMetalSound
- 13:
- PlayerData::impactSnowSound
- 14:
- PlayerData::impactWaterEasy
- 15:
- PlayerData::impactWaterMedium
- 16:
- PlayerData::impactWaterHard
- 17:
- PlayerData::exitingWater

string `Material::`**`getAnimFlags`**

string `Material::`**`getFilename`**
Get filename of material.

bool `Material::`**`glow`**[4]
Enables rendering this material to the glow buffer.

int `Material::`**`impactSoundId`**
What sound to play from the PlayerData sound list when the player impacts on the surface with a velocity equal or greater than PlayerData::groundImpactMinSpeed . For a list of IDs, see footstepSoundId

bool `Material::`**`isAutoGenerated`**
Returns true if this Material was automatically generated by MaterialList::mapMaterials().

filename `Material::`**`lightMap`**[4]
The lightmap texture used with pureLight.

string `Material::`**`mapTo`**
Used to map this material to the material name used by TSShape.

float `Material::`**`minnaertConstant`**[4]
The Minnaert shading constant value. Must be greater than 0 to enable the effect.

filename `Material::`**`normalMap`**[4]
The normal map texture. You can use the DXTnm format only when per-pixel specular highlights are disabled, or a specular map is in use.

filename `Material::`**`overlayMap`**[4]
A secondary diffuse color texture map which will use the second texcoord of a mesh.

filename `Material::`**`overlayTex`**[4]
For backwards compatibility.

float `Material::`**`parallaxScale`**[4]
Enables parallax mapping and defines the scale factor for the parallax effect. Typically this value is less than 0.4 else the effect breaks down.

bool Material::**pixelSpecular**[4]
This enables per-pixel specular highlights controlled by the alpha channel of the normal map texture. Note that if pixel specular is enabled the DXTnm format will not work with your normal map, unless you are also using a specular map.

bool Material::**planarReflection**

void Material::**reload**
Reloads all material instances that use this material.

Point2F Material::**rotPivotOffset**[4]
The piviot position in UV coordinates to center the rotation animation.

float Material::**rotSpeed**[4]
The speed to rotate the texture in degrees per second when rotation animation is enabled.

Point2F Material::**scrollDir**[4]
The scroll direction in UV space when scroll animation is enabled.

float Material::**scrollSpeed**[4]
The speed to scroll the texture in UVs per second when scroll animation is enabled.

float Material::**sequenceFramePerSec**[4]
The number of frames per second for frame based sequence animations if greater than zero.

float Material::**sequenceSegmentSize**[4]
The size of each frame in UV units for sequence animations.

void Material::**setAutoGenerated**
setAutoGenerated(bool isAutoGenerated): Set whether or not the Material is autogenerated.

bool Material::**showDust**
Whether to emit dust particles from a shape moving over the material. This is, for example, used by vehicles or players to decide whether to show dust trails.

bool Material::**showFootprints**
Whether to show player footprint decals on this material.

ColorF Material::**specular**[4]
The color of the specular highlight when not using a specularMap.

filename Material::**specularMap**[4]
The specular map texture. The RGB channels of this texture provide a per-pixel replacement for the 'specular' parameter on the material. If this texture contains alpha information, the alpha channel of the texture will be used as the gloss map. This provides a per-pixel replacement for the 'specularPower' on the material.

float Material::**specularPower**[4]
The hardness of the specular highlight when not using a specularMap.

float Material::**specularStrength**[4]
The strength of the specular highlight when not using a specularMap.

bool Material::**subSurface**[4]
Enables the subsurface scattering approximation.

ColorF Material::**subSurfaceColor**[4]
The color used for the subsurface scattering approximation.

float Material::**subSurfaceRolloff**[4]
The 0 to 1 rolloff factor used in the subsurface scattering approximation.

filename Material::**toneMap**[4]
The tonemap texture used with pureLight.

bool Material::**translucent**
If true this material is translucent blended.

MaterialBlendOp Material::**translucentBlendOp**
The type of blend operation to use when the material is translucent.

bool Material::**translucentZWrite**
If enabled and the material is translucent it will write into the depth buffer.

bool Material::**useAnisotropic**[4]
Use anisotropic filtering for the textures of this stage.

bool Material::**vertColor**[4]
If enabled, vertex colors are premultiplied with diffuse colors.

bool Material::**vertLit**[4]
If true the vertex color is used for lighting.

float Material::**waveAmp**[4]
The wave amplitude when wave animation is enabled.

float Material::**waveFreq**[4]
The wave frequency when wave animation is enabled.

MaterialWaveType Material::**waveType**[4]
The type of wave animation to perform when wave animation is enabled.

PfxVis Singleton class that exposes ConsoleStaticFunctions for debug visualizing PostEffects.

Description Singleton class that exposes ConsoleStaticFunctions for debug visualizing PostEffects.

Example:

```
// Script interface...
PfxVis::open( PostEffect )
// Multiple PostEffects can be visualized at the same time
PfxVis::clear()
// Clear all visualizer windows
PfxVis::hide()
// Hide all windows (are not destroyed)
PfxVis::show()
```

Methods

void PfxVis::**clear**()
Close all visualization windows.

Example:

```
PfxVis::clear();
```

void PfxVis::**hide**()
Hide all visualization windows (they are not destroyed).

Example:

```
PfxVis::hide();
```

void PfxVis::**onWindowClosed**(GuiWindowCtrl *ctrl*)
Callback when a visualization window is closed.

Parameters **ctrl** – Name of the GUI control being closed

Example:

```
PfxVis::onWindowClosed( VisWindow )
```

void PfxVis::**open**(PostEffect *effect*, bool *clear*)
Open visualization windows for all input and target textures.

Parameters

- **effect** – Name of the PostEffect to open
- **clear** – True to close all visualization windows before opening the effect

Example:

```
// Multiple PostEffects can be visualized at the same timePfxVis::open( PostEffect )
```

void PfxVis::**show**()
Show all visualization windows.

Example:

```
PfxVis::show();
```

RenderFormatToken Used to change the render target format when rendering in AL.

Inherit: RenderPassStateToken

Description RenderFormatToken is an implementation which changes the format of the back buffer and/or the depth buffer.

The RenderPassStateBin manager changes the rendering state associated with this token. In stock Torque 3D, a single example exists in the way of AL_FormatToken (found in renderManager.cs). In that script file, all the render managers are intialized, and a single RenderFormatToken is used. This implementation basically exists to ensure Advanced Lighting works with MSAA.

The actions for this token toggle the format of the back/depth buffers and it lets you specify a custom shader to "copy" the data so it can be reformatted or altered. This is done through the variables copyEffect and resolveEffect (which are post processes just like fog or glow)

Example:

```
// This token, and the associated render managers, ensure that driver MSAA does not get used for Adva
{
   enabled = "false";

   format = "GFXFormatR8G8B8A8";
   depthFormat = "GFXFormatD24S8";
   aaLevel = 0; // -1 = match backbuffer
   // The contents of the back buffer before this format token is executed
   // is provided in
   $inTexcopyEffect = "AL_FormatCopy";

   // The contents of the render target created by this format token is
   // provided in
   $inTexresolveEffect = "AL_FormatCopy";
};
```

Fields

int `RenderFormatToken::`**`aaLevel`**
Anti-ailiasing level for the this token. 0 disables, -1 uses adapter default.

PostEffect `RenderFormatToken::`**`copyEffect`**
This PostEffect will be run when the render target is changed to the format specified by this token. It is used to copy/format data into the token rendertarget.

GFXFormat `RenderFormatToken::`**`depthFormat`**
Sets the depth/stencil buffer format for this token.

GFXFormat `RenderFormatToken::`**`format`**
Sets the color buffer format for this token.

PostEffect `RenderFormatToken::`**`resolveEffect`**
This PostEffect will be run when the render target is changed back to the format active prior to this token. It is used to copy/format data from the token rendertarget to the backbuffer.

Description

In Torque the GFX layer provides access to abstracted low level graphics concepts. From script you have limited access to graphics rendering as it is usually too slow to do individual draw calls thru the scripting interface. For drawing its usually better to use the higher level gameplay objects.

Note: Detailed technical descriptions of when to use specific GFXStateBlockData fields, how GFXBlendOp works, or other interfaces of that nature are outside the scope of this manual. Since Torque is based on DirectX and OpenGL any reference documents for those APIs will provide the background needed to learn about rendering.

Enumeration

`enum GFXAdapterType`
Back-end graphics API used by the GFX subsystem.

Parameters

- **`OpenGL`** – OpenGL.
- **`D3D8`** – Direct3D 8.
- **`D3D9`** – Direct3D 9.
- **`NullDevice`** – Null device for dedicated servers.
- **`Xenon`** – Direct3D 9 on Xbox 360.

`enum GFXBlend`
The supported blend modes.

Parameters

- **`GFXBlendZero`** – (0, 0, 0, 0)
- **`GFXBlendOne`** – (1, 1, 1, 1)
- **`GFXBlendSrcColor`** – (Rs, Gs, Bs, As)
- **`GFXBlendInvSrcColor`** – (1 - Rs, 1 - Gs, 1 - Bs, 1 - As)
- **`GFXBlendSrcAlpha`** – (As, As, As, As)

- **GFXBlendInvSrcAlpha** – (1 - As, 1 - As, 1 - As, 1 - As)
- **GFXBlendDestAlpha** – (Ad Ad Ad Ad)
- **GFXBlendInvDestAlpha** – (1 - Ad 1 - Ad 1 - Ad 1 - Ad)
- **GFXBlendDestColor** – (Rd, Gd, Bd, Ad)
- **GFXBlendInvDestColor** – (1 - Rd, 1 - Gd, 1 - Bd, 1 - Ad)
- **GFXBlendSrcAlphaSat** – (f, f, f, 1) where f = min(As, 1 - Ad)

enum GFXBlendOp

The blend operators.

Parameters

- **GFXBlendOpAdd** –
- **GFXBlendOpSubtract** –
- **GFXBlendOpRevSubtract** –
- **GFXBlendOpMin** –
- **GFXBlendOpMax** –

enum GFXCmpFunc

The supported comparison functions.

Parameters

- **GFXCmpNever** –
- **GFXCmpLess** –
- **GFXCmpEqual** –
- **GFXCmpLessEqual** –
- **GFXCmpGreater** –
- **GFXCmpNotEqual** –
- **GFXCmpGreaterEqual** –
- **GFXCmpAlways** –

enum GFXCullMode

The render cull modes.

Parameters

- **GFXCullNone** –
- **GFXCullCW** –
- **GFXCullCCW** –

enum GFXFormat

The texture formats.

Parameters

- **GFXFormatR8G8B8** –
- **GFXFormatR8G8B8A8** –
- **GFXFormatR8G8B8X8** –

- **GFXFormatR32F** –
- **GFXFormatR5G6B5** –
- **GFXFormatR5G5B5A1** –
- **GFXFormatR5G5B5X1** –
- **GFXFormatA4L4** –
- **GFXFormatA8L8** –
- **GFXFormatA8** –
- **GFXFormatL8** –
- **GFXFormatDXT1** –
- **GFXFormatDXT2** –
- **GFXFormatDXT3** –
- **GFXFormatDXT4** –
- **GFXFormatDXT5** –
- **GFXFormatD32** –
- **GFXFormatD24X8** –
- **GFXFormatD24S8** –
- **GFXFormatD24FS8** –
- **GFXFormatD16** –
- **GFXFormatR32G32B32A32F** –
- **GFXFormatR16G16B16A16F** –
- **GFXFormatL16** –
- **GFXFormatR16G16B16A16** –
- **GFXFormatR16G16** –
- **GFXFormatR16F** –
- **GFXFormatR16G16F** –
- **GFXFormatR10G10B10A2** –

enum GFXStencilOp

The stencil operators.

Parameters

- **GFXStencilOpKeep** –
- **GFXStencilOpZero** –
- **GFXStencilOpReplace** –
- **GFXStencilOpIncrSat** –
- **GFXStencilOpDecrSat** –
- **GFXStencilOpInvert** –
- **GFXStencilOpIncr** –

- **GFXStencilOpDecr** –

enum GFXTextureAddressMode
The texture address modes.

Parameters

- **GFXAddressWrap** –
- **GFXAddressMirror** –
- **GFXAddressClamp** –
- **GFXAddressBorder** –
- **GFXAddressMirrorOnce** –

enum GFXTextureArgument
The texture arguments.

Parameters

- **GFXTADiffuse** –
- **GFXTACurrent** –
- **GFXTATexture** –
- **GFXTATFactor** –
- **GFXTASpecular** –
- **GFXTATemp** –
- **GFXTAConstant** –
- **OneMinus** –
- **AlphaReplicate** –

enum GFXTextureFilterType
The texture filter types.

Parameters

- **GFXTextureFilterNone** –
- **GFXTextureFilterPoint** –
- **GFXTextureFilterLinear** –
- **GFXTextureFilterAnisotropic** –
- **GFXTextureFilterPyramidalQuad** –
- **GFXTextureFilterGaussianQuad** –

enum GFXTextureOp
The texture operators.

Parameters

- **GFXTOPDisable** –
- **GFXTOPSelectARG1** –
- **GFXTOPSelectARG2** –
- **GFXTOPModulate** –

- **GFXTOPModulate2X** –
- **GFXTOPModulate4X** –
- **GFXTOPAdd** –
- **GFXTOPAddSigned** –
- **GFXTOPAddSigned2X** –
- **GFXTOPSubtract** –
- **GFXTOPAddSmooth** –
- **GFXTOPBlendDiffuseAlpha** –
- **GFXTOPBlendTextureAlpha** –
- **GFXTOPBlendFactorAlpha** –
- **GFXTOPBlendTextureAlphaPM** –
- **GFXTOPBlendCURRENTALPHA** –
- **GFXTOPPreModulate** –
- **GFXTOPModulateAlphaAddColor** –
- **GFXTOPModulateColorAddAlpha** –
- **GFXTOPModulateInvAlphaAddColor** –
- **GFXTOPModulateInvColorAddAlpha** –
- **GFXTOPBumpEnvMap** –
- **GFXTOPBumpEnvMapLuminance** –
- **GFXTOPDotProduct3** –
- **GFXTOPLERP** –

enum GFXTextureTransformFlags
The texture transform state flags.

Parameters

- **GFXTTFDisable** –
- **GFXTTFFCoord1D** –
- **GFXTTFFCoord2D** –
- **GFXTTFFCoord3D** –
- **GFXTTFFCoord4D** –
- **GFXTTFProjected** –

enum MaterialAnimType
The type of animation effect to apply to this material.

Parameters

- **Scroll** – Scroll the material along the X/Y axis.
- **Rotate** – Rotate the material around a point.
- **Wave** – Warps the material with an animation using Sin, Triangle or Square mathematics.
- **Scale** – Scales the material larger and smaller with a pulsing effect.

- **Sequence** – Enables the material to have multiple frames of animation in its imagemap.

enum MaterialBlendOp
The type of graphical blending operation to apply to this material.

Parameters

- **None** – Disable blending for this material.
- **Mul** – Multiplicative blending.
- **Add** – Adds the color of the material to the frame buffer with full alpha for each pixel.
- **AddAlpha** – The color is modulated by the alpha channel before being added to the frame buffer.
- **Sub** – Subtractive Blending. Reverses the color model, causing dark colors to have a stronger visual effect.
- **LerpAlpha** – Linearly interpolates between Material color and frame buffer color based on alpha.

enum MaterialWaveType
When using the Wave material animation, one of these Wave Types will be used to determine the type of wave to display.

Parameters

- **Sin** – Warps the material along a curved Sin Wave.
- **Triangle** – Warps the material along a sharp Triangle Wave.
- **Square** – Warps the material along a wave which transitions between two oppposite states. As a Square Wave, the transition is quick and sudden.

Functions

void **cleanupTexturePool**()
Release the unused pooled textures in texture manager freeing up video memory.

void **clearGFXResourceFlags**()
Clears the flagged state on all allocated GFX resources. See flagCurrentGFXResources for usage details.

void **describeGFXResources**(string *resourceTypes*, string *filePath*, bool *unflaggedOnly*)
Dumps a description of GFX resources to a file or the console.

•texture

•texture target

•window target

•vertex buffers

•primitive buffers

•fences

•cubemaps

•shaders

•stateblocks

Parameters

- **resourceTypes** – A space seperated list of resource types or an empty string for all resources.
- **filePath** – A file to dump the list to or an empty string to write to the console.
- **unflaggedOnly** – If true only unflagged resources are dumped. See flagCurrentGFXResources.

void **describeGFXStateBlocks**(string *filePath*)
Dumps a description of all state blocks.

Parameters **filePath** – A file to dump the state blocks to or an empty string to write to the console.

void **dumpRandomNormalMap**()
Creates a 64x64 normal map texture filled with noise. The texture is saved to randNormTex.png in the location of the game executable.

void **dumpTextureObjects**()
Dumps a list of all active texture objects to the console.

void **flagCurrentGFXResources**()
Flags all currently allocated GFX resources. Used for resource allocation and leak tracking by flagging current resources then dumping a list of unflagged resources at some later point in execution.

void **flushTextureCache**()
Releases all textures and resurrects the texture manager.

static int GFXInit::**getAdapterCount**()
Return the number of graphics adapters available.

GFXFormat **getBestHDRFormat**()
Returns the best texture format for storage of HDR data for the active device.

Point3F **getDesktopResolution**()
Returns the width, height, and bitdepth of the screen/desktop.

string **getDisplayDeviceInformation**()
Get the string describing the active GFX device.

String **getDisplayDeviceList**()
Returns a tab-seperated string of the detected devices across all adapters.

float **getPixelShaderVersion**()
Returns the pixel shader version for the active device.

String **getTextureProfileStats**()
Returns a list of texture profiles in the format: ProfileName TextureCount TextureMB.

void **listGFXResources**(bool *unflaggedOnly*)
Returns a list of the unflagged GFX resources. See flagCurrentGFXResources for usage details.

void **reloadTextures**()
Reload all the textures from disk.

void **screenShot**(string *file*, string *format*, int *tileCount*, float *tileOverlap*)
Takes a screenshot with optional tiling to produce huge screenshots.

Parameters

- **file** – The output image file path.
- **format** – Either JPEG or PNG.

- **tileCount** – If greater than 1 will tile the current screen size to take a large format screenshot.
- **tileOverlap** – The amount of horizontal and vertical overlap between the tiles used to remove tile edge artifacts from post effects.

void **setPixelShaderVersion**(float *version*)
Sets the pixel shader version for the active device. This can be used to force a lower pixel shader version than is supported by the device for testing or performance optimization.

Parameters **version** – The floating point shader version number.

void **setReflectFormat**(GFXFormat *format*)
Set the reflection texture format.

Variables

bool $gfx::disableOcclusionQuery
Debug helper that disables all hardware occlusion queries causing them to return only the visibile state.

bool $pref::Video::disableVerticalSync
Disables vertical sync on the active device.

bool $gfx::disassembleAllShaders
On supported devices this will dump shader disassembly to the procedural shader folder.

float $pref::Video::forcedPixVersion
Will force the shader model if the value is positive and less than the shader model supported by the active device. Use 0 for fixed function.

string $pref::Video::missingTexturePath
The file path of the texture to display when the requested texture is missing.

int $pref::Video::textureReductionLevel
The number of mipmap levels to drop on loaded textures to reduce video memory usage. It will skip any textures that have been defined as not allowing down scaling.

string $pref::Video::unavailableTexturePath
The file path of the texture to display when the requested texture is unavailable. Often this texture is used by GUI controls to indicate that the request image is unavailable.

string $pref::Video::warningTexturePath
The file path of the texture used to warn the developer.

bool $gfx::wireframe
Used to toggle wireframe rendering at runtime.

Materials

Classes, structures, functions, and variables related to Torque 3D's material system.

Classes

CustomMaterial Material object which provides more control over surface properties.

Inherit: Material

Description CustomMaterials allow the user to specify their own shaders via the ShaderData datablock. Because CustomMaterials are derived from Materials, they can hold a lot of the same properties. It is up to the user to code how these properties are used.

Example:

```
singleton CustomMaterial( WaterBasicMat )
{
   sampler["reflectMap"] = "$reflectbuff";
   sampler["refractBuff"] = "$backbuff";

   cubemap = NewLevelSkyCubemap;
   shader = WaterBasicShader;
   stateBlock = WaterBasicStateBlock;
   version = 2.0;
};
```

Fields

Material `CustomMaterial::`**`fallback`**
: Alternate material for targeting lower end hardware. If the CustomMaterial requires a higher pixel shader version than the one it's using, it's fallback Material will be processed instead. If the fallback material wasn't defined, Torque 3D will assert and attempt to use a very basic material in it's place.

bool `CustomMaterial::`**`forwardLit`**
: Determines if the material should recieve lights in Basic Lighting. Has no effect in Advanced Lighting.

string `CustomMaterial::`**`shader`**
: Name of the ShaderData to use for this effect.

GFXStateBlockData `CustomMaterial::`**`stateBlock`**
: Name of a GFXStateBlockData for this effect.

string `CustomMaterial::`**`target`**
: String identifier of this material's target texture.

float `CustomMaterial::`**`version`**
: Specifies pixel shader version for hardware. Valid pixel shader versions include 2.0, 3.0, etc.

Functions

void **`addMaterialMapping`**(string *texName*, string *matName*)
: Maps the given texture to the given material. Generates a console warning before overwriting. Material maps are used by terrain and interiors for triggering effects when an object moves onto a terrain block or interior surface using the associated texture.

string **`getMaterialMapping`**(string *texName*)
: Returns the name of the material mapped to this texture. If no materials are found, an empty string is returned.

  Parameters **`texName`** – Name of the texture

Variables

`int $pref::Video::defaultAnisotropy`
: Global variable defining the default anisotropy value. Controls the default anisotropic texture filtering level for all materials, including the terrain. This value can be changed at runtime to see its affect without reloading.

void **dumpMaterialInstances**
Dumps a formatted list of currently allocated material instances to the console.

void **reInitMaterials**
Flushes all procedural shaders and re-initializes all active material instances.

Shaders

Classes, structures, functions, and variables related to Torque 3D's shader system.

Classes

ShaderData Special type of data block that stores information about a handwritten shader.

Inherit: SimObject

Description To use hand written shaders, a ShaderData datablock must be used. This datablock refers only to the vertex and pixel shader filenames and a hardware target level. Shaders are API specific, so DirectX and OpenGL shaders must be explicitly identified.

Example:

```
// Used for the procedural clould system
singleton ShaderData( CloudLayerShader )
{
   DXVertexShaderFile   = "shaders/common/cloudLayerV.hlsl";
   DXPixelShaderFile    = "shaders/common/cloudLayerP.hlsl";
   OGLVertexShaderFile = "shaders/common/gl/cloudLayerV.glsl";
   OGLPixelShaderFile = "shaders/common/gl/cloudLayerP.glsl";
   pixVersion = 2.0;
};
```

Methods

void ShaderData::**reload**()
Rebuilds all the vertex and pixel shader instances created from this ShaderData .

Example:

```
// Rebuild the shader instances from ShaderData CloudLayerShader
CloudLayerShader.reload();
```

Fields

string ShaderData::**defines**
String of case-sensitive defines passed to the shader compiler. The string should be delimited by a semicolon, tab, or newline character.

Example:

```
singleton ShaderData( FlashShader )
{
DXVertexShaderFile   = "shaders/common/postFx/flashV.hlsl";
DXPixelShaderFile    = "shaders/common/postFx/flashP.hlsl";

 //Define setting the color of WHITE_COLOR.defines = "WHITE_COLOR=float4(1.0,1.0,1.0,0.0)";
```

```
    pixVersion = 2.0
    }
```

filename ShaderData::**DXPixelShaderFile**
: Path to the DirectX pixel shader file to use for this ShaderData . It must contain only one program and no vertex shader, just the pixel shader. It can be either an HLSL or assembly level shader. HLSL's must have a filename extension of .hlsl, otherwise its assumed to be an assembly file.

filename ShaderData::**DXVertexShaderFile**
: Path to the DirectX vertex shader file to use for this ShaderData . It must contain only one program and no pixel shader, just the vertex shader.It can be either an HLSL or assembly level shader. HLSL's must have a filename extension of .hlsl, otherwise its assumed to be an assembly file.

filename ShaderData::**OGLPixelShaderFile**
: Path to an OpenGL pixel shader file to use for this ShaderData . It must contain only one program and no vertex shader, just the pixel shader.

filename ShaderData::**OGLVertexShaderFile**
: Path to an OpenGL vertex shader file to use for this ShaderData . It must contain only one program and no pixel shader, just the vertex shader.

float ShaderData::**pixVersion**
: Indicates target level the shader should be compiled. Valid numbers at the time of this writing are 1.1, 1.4, 2.0, and 3.0. The shader will not run properly if the hardware does not support the level of shader compiled.

bool ShaderData::**useDevicePixVersion**
: If true, the maximum pixel shader version offered by the graphics card will be used. Otherwise, the script-defined pixel shader version will be used.

Lighting

The script functionality related to the lighting systems and lights.

Classes

AdvancedLightBinManager Rendering Manager responsible for lighting, shadows, and global variables affecing both.

Inherit: RenderTexTargetBinManager

Description Should not be exposed to TorqueScript as a game object, meant for internal use only

LightBase This is the base class for light objects.

Inherit: SceneObject

Description It is *NOT* intended to be used directly in script, but exists to provide the base member variables and generic functionality. You should be using the derived classes PointLight and SpotLight, which can be declared in TorqueScript or added from the World Editor.

For this class, we only add basic lighting options that all lighting systems would use. The specific lighting system options are injected at runtime by the lighting system itself.

Methods

void LightBase::**playAnimation**()

Plays the light animation assigned to this light with the existing LightAnimData datablock.

Example:

```
// Play the animation assigned to this light
CrystalLight.playAnimation();
```

void LightBase::**playAnimation**(LightAnimData *anim*)

Plays the light animation on this light using a new LightAnimData . If no LightAnimData is passed the existing one is played.

Parameters **anim** – Name of the LightAnimData datablock to be played

Example:

```
// Play the animation using a new LightAnimData datablock
CrystalLight.playAnimation(SubtlePulseLightAnim);
```

void LightBase::**setLightEnabled**(bool *state*)

Toggles the light on and off.

Parameters **state** – Turns the light on (true) or off (false)

Example:

```
// Disable the light
CrystalLight.setLightEnabled(false);

// Renable the light
CrystalLight.setLightEnabled(true);
```

Fields

bool LightBase::**animate**

Toggles animation for the light on and off.

float LightBase::**animationPeriod**

The length of time in seconds for a single playback of the light animation (must be gt 0).

float LightBase::**animationPhase**

The phase used to offset the animation start time to vary the animation of nearby lights.

LightAnimData LightBase::**animationType**

Datablock containing light animation information (LightAnimData).

Point3F LightBase::**attenuationRatio**

The proportions of constant, linear, and quadratic attenuation to use for the falloff for point and spot lights.

float LightBase::**brightness**

Adjusts the lights power, 0 being off completely.

bool LightBase::**castShadows**

Enables/disabled shadow casts by this light.

ColorF LightBase::**color**

Changes the base color hue of the light.

filename LightBase::**cookie**

A custom pattern texture which is projected from the light.

float LightBase::**fadeStartDistance**

Start fading shadows out at this distance. 0 = auto calculate this distance.

float LightBase::**flareScale**
Globally scales all features of the light flare.

LightFlareData LightBase::**flareType**
Datablock containing light flare information (LightFlareData).

bool LightBase::**includeLightmappedGeometryInShadow**
This light should render lightmapped geometry during its shadow-map update (ignored if 'representedInLightmap' is false).

bool LightBase::**isEnabled**
Enables/Disables the object rendering and functionality in the scene.

bool LightBase::**lastSplitTerrainOnly**
This toggles only terrain being rendered to the last split of a PSSM shadow map.

float LightBase::**logWeight**
The logrithmic PSSM split distance factor.

int LightBase::**numSplits**
The logrithmic PSSM split distance factor.

Point4F LightBase::**overDarkFactor**
The ESM shadow darkening factor.

void LightBase::**pauseAnimation**
Stops the light animation.

float LightBase::**priority**
Used for sorting of lights by the light manager. Priority determines if a light has a stronger effect than, those with a lower value.

bool LightBase::**representedInLightmap**
This light is represented in lightmaps (static light, default: false).

ColorF LightBase::**shadowDarkenColor**
The color that should be used to multiply-blend dynamic shadows onto lightmapped geometry (ignored if 'representedInLightmap' is false).

float LightBase::**shadowDistance**
The distance from the camera to extend the PSSM shadow.

float LightBase::**shadowSoftness**

ShadowType LightBase::**shadowType**
The type of shadow to use on this light.

int LightBase::**texSize**
The texture size of the shadow map.

LightDescription A helper datablock used by classes (such as shapebase) that submit lights to the scene but do not use actual "LightBase" objects.

Inherit: SimDataBlock

Description A helper datablock used by classes (such as shapebase) that submit lights to the scene but do not use actual "LightBase" objects.

This datablock stores the properties of that light as fields that can be initialized from script.

Example:

```
// Declare a light description to be used on a rocket launcher projectile
datablock LightDescription(RocketLauncherLightDesc)
{
   range = 4.0;
   color = "1 1 0";
   brightness = 5.0;
   animationType = PulseLightAnim;
   animationPeriod = 0.25;
};

// Declare a ProjectileDatablock which uses the light description
datablock ProjectileData(RocketLauncherProjectile)
{
   lightDesc = RocketLauncherLightDesc;

   projectileShapeName = "art/shapes/weapons/SwarmGun/rocket.dts";

   directDamage = 30;
   radiusDamage = 30;
   damageRadius = 5;
   areaImpulse = 2500;

   // ... remaining ProjectileData fields not listed for this example
};
```

Methods

void LightDescription::**apply**()

Force an inspectPostApply call for the benefit of tweaking via the console. Normally this functionality is only exposed to objects via the World Editor, once changes have been made. Exposing apply to script allows you to make changes to it on the fly without the World Editor.

Example:

```
// Change a property of the light description
RocketLauncherLightDesc.brightness = 10;

// Make it so
RocketLauncherLightDesc.apply();
```

Fields

float LightDescription::**animationPeriod**

The length of time in seconds for a single playback of the light animation.

float LightDescription::**animationPhase**

The phase used to offset the animation start time to vary the animation of nearby lights.

LightAnimData LightDescription::**animationType**

Datablock containing light animation information (LightAnimData).

Point3F LightDescription::**attenuationRatio**

The proportions of constant, linear, and quadratic attenuation to use for the falloff for point and spot lights.

float LightDescription::**brightness**

Adjusts the lights power, 0 being off completely.

bool LightDescription::**castShadows**

Enables/disabled shadow casts by this light.

ColorF LightDescription::**color**
Changes the base color hue of the light.

filename LightDescription::**cookie**
A custom pattern texture which is projected from the light.

float LightDescription::**fadeStartDistance**
Start fading shadows out at this distance. 0 = auto calculate this distance.

float LightDescription::**flareScale**
Globally scales all features of the light flare.

LightFlareData LightDescription::**flareType**
Datablock containing light flare information (LightFlareData).

bool LightDescription::**includeLightmappedGeometryInShadow**
This light should render lightmapped geometry during its shadow-map update (ignored if 'representedInLightmap' is false).

bool LightDescription::**lastSplitTerrainOnly**
This toggles only terrain being rendered to the last split of a PSSM shadow map.

float LightDescription::**logWeight**
The logrithmic PSSM split distance factor.

int LightDescription::**numSplits**
The logrithmic PSSM split distance factor.

Point4F LightDescription::**overDarkFactor**
The ESM shadow darkening factor.

float LightDescription::**range**
Controls the size (radius) of the light.

bool LightDescription::**representedInLightmap**
This light is represented in lightmaps (static light, default: false).

ColorF LightDescription::**shadowDarkenColor**
The color that should be used to multiply-blend dynamic shadows onto lightmapped geometry (ignored if 'representedInLightmap' is false).

float LightDescription::**shadowDistance**
The distance from the camera to extend the PSSM shadow.

float LightDescription::**shadowSoftness**

ShadowType LightDescription::**shadowType**
The type of shadow to use on this light.

int LightDescription::**texSize**
The texture size of the shadow map.

LightFlareData Defines a light flare effect usable by scene lights.

Inherit: SimDataBlock

Description LightFlareData is a datablock which defines a type of flare effect. This may then be referenced by other classes which support the rendering of a flare: Sun, ScatterSky, LightBase.

A flare contains one or more elements defined in the element* named fields of LightFlareData, with a maximum of ten elements. Each element is rendered as a 2D sprite in screenspace.

Example:

```
// example from Full Template, core/art/datablocks/lights.cs
datablock LightFlareData( LightFlareExample0 )
{
   overallScale = 2.0;
   flareEnabled = true;
   renderReflectPass = true;
   flareTexture = "./../special/lensFlareSheet1";
   occlusionRadius = 0.25;

   elementRect[0] = "0 512 512 512";
   elementDist[0] = 0.0;
   elementScale[0] = 0.5;
   elementTint[0] = "1.0 1.0 1.0";
   elementRotate[0] = false;
   elementUseLightColor[0] = false;

   elementRect[1] = "512 0 512 512";
   elementDist[1] = 0.0;
   elementScale[1] = 2.0;
   elementTint[1] = "0.5 0.5 0.5";
   elementRotate[1] = false;
   elementUseLightColor[1] = false;
};
```

The elementDist field defines where along the flare's beam the element appears. A distance of 0.0 is directly over the light source, a distance of 1.0 is at the screen center, and a distance of 2.0 is at the position of the light source mirrored across the screen center.

Methods

void `LightFlareData::`**`apply`**`()`
Intended as a helper to developers and editor scripts. Force trigger an inspectPostApply

Fields

float `LightFlareData::`**`elementDist`**[20]
Where this element appears along the flare beam.

RectF `LightFlareData::`**`elementRect`**[20]
A rectangle specified in pixels of the flareTexture image.

bool `LightFlareData::`**`elementRotate`**[20]
Defines if this element orients to point along the flare beam or if it is always upright.

float `LightFlareData::`**`elementScale`**[20]
Size scale applied to this element.

ColorF `LightFlareData::`**`elementTint`**[20]
Used to modulate this element's color if elementUseLightColor is false.

bool `LightFlareData::`**`elementUseLightColor`**[20]
If true this element's color is modulated by the light color. If false, elementTint will be used.

bool `LightFlareData::`**`flareEnabled`**
Allows the user to disable this flare globally for any lights referencing it.

filename `LightFlareData::`**`flareTexture`**
The texture / sprite sheet for this flare.

float LightFlareData::**occlusionRadius**
If positive an occlusion query is used to test flare visibility, else it uses simple raycasts.

float LightFlareData::**overallScale**
Size scale applied to all elements of the flare.

bool LightFlareData::**renderReflectPass**
If false the flare does not render in reflections, else only non-zero distance elements are rendered.

PointLight Lighting object that radiates light in all directions.

Inherit: LightBase

Description PointLight is one of the two types of lighting objects that can be added to a Torque 3D level, the other being SpotLight. Unlike directional or conical light, the PointLight emits lighting in all directions. The attenuation is controlled by a single variable: LightObject::radius.

Example:

```
// Declaration of a point light in script, or created by World EditornewPointLight(CrystalLight)
{
   radius = "10";
   isEnabled = "1";
   color = "1 0.905882 0 1";
   brightness = "0.5";
   castShadows = "1";
   priority = "1";
   animate = "1";
   animationType = "SubtlePulseLightAnim";
   animationPeriod = "3";
   animationPhase = "3";
   flareScale = "1";
   attenuationRatio = "0 1 1";
   shadowType = "DualParaboloidSinglePass";
   texSize = "512";
   overDarkFactor = "2000 1000 500 100";
   shadowDistance = "400";
   shadowSoftness = "0.15";
   numSplits = "1";
   logWeight = "0.91";
   fadeStartDistance = "0";
   lastSplitTerrainOnly = "0";
   splitFadeDistances = "10 20 30 40";
   representedInLightmap = "0";
   shadowDarkenColor = "0 0 0 -1";
   includeLightmappedGeometryInShadow = "1";
   position = "-61.3866 1.69186 5.1464";
   rotation = "1 0 0 0";
};
```

Fields
float PointLight::**radius**
Controls the falloff of the light emission.

SpotLight Lighting object which emits conical light in a direction.

Inherit: LightBase

Description SpotLight is one of the two types of lighting objects that can be added to a Torque 3D level, the other being PointLight. Unlike directional or point lights, the SpotLights emits lighting in a specific direction within a cone. The distance of the cone is controlled by the SpotLight::range variable.

Example:

```
// Declaration of a point light in script, or created by World EditornewSpotLight(SampleSpotLight)
{
   range = "10";
   innerAngle = "40";
   outerAngle = "45";
   isEnabled = "1";
   color = "1 1 1 1";
   brightness = "1";
   castShadows = "0";
   priority = "1";
   animate = "1";
   animationPeriod = "1";
   animationPhase = "1";
   flareType = "LightFlareExample0";
   flareScale = "1";
   attenuationRatio = "0 1 1";
   shadowType = "Spot";
   texSize = "512";
   overDarkFactor = "2000 1000 500 100";
   shadowDistance = "400";
   shadowSoftness = "0.15";
   numSplits = "1";
   logWeight = "0.91";
   fadeStartDistance = "0";
   lastSplitTerrainOnly = "0";
   representedInLightmap = "0";
   shadowDarkenColor = "0 0 0 -1";
   includeLightmappedGeometryInShadow = "0";
   position = "-29.4362 -5.86289 5.58602";
   rotation = "1 0 0 0";
};
```

Fields

float SpotLight::**innerAngle**
float SpotLight::**outerAngle**

float SpotLight::**range**

Functions

string **getActiveLightManager**()
Returns the active light manager name.

String **getLightManagerNames**()
Returns a tab seperated list of light manager names.

bool **lightScene**(string *completeCallbackFn*, string *mode*)
Will generate static lighting for the scene if supported by the active light manager. If mode is “forceAlways”,

the lightmaps will be regenerated regardless of whether lighting cache files can be written to. If mode is "forceWritable", then the lightmaps will be regenerated only if the lighting cache files can be written.

Parameters

- **completeCallbackFn** – The name of the function to execute when the lighting is complete.
- **mode** – One of "forceAlways", "forceWritable" or "loadOnly".

Returns Returns true if the scene lighting process was started.

void **onLightManagerActivate**(string *name*)
A callback called by the engine when a light manager is activated.

Parameters **name** – The name of the light manager being activated.

void **onLightManagerDeactivate**(string *name*)
A callback called by the engine when a light manager is deactivated.

Parameters **name** – The name of the light manager being deactivated.

void **resetLightManager**()
Deactivates and then activates the currently active light manager.This causes most shaders to be regenerated and is often used when global rendering changes have occured.

bool **setLightManager**(string *name*)
Finds and activates the named light manager.

Returns Returns true if the light manager is found and activated.

Variables

bool $Light::renderLightFrustums
Toggles rendering of light frustums when the light is selected in the editor.

bool $Light::renderViz
Toggles visualization of light object's radius or cone.

Advanced Lighting

The script functionality related to the Advanced Lghting system.

Enumeration
enum ShadowFilterMode
The shadow filtering modes for Advanced Lighting shadows.

Parameters

- **None** – Simple point sampled filtering. This is the fastest and lowest quality mode.
- **SoftShadow** – A variable tap rotated poisson disk soft shadow filter. It performs 4 taps to classify the point as in shadow, out of shadow, or along a shadow edge. Samples on the edge get an additional 8 taps to soften them.
- **SoftShadowHighQuality** – A 12 tap rotated poisson disk soft shadow filter. It performs all the taps for every point without any early rejection.

enum ShadowType

Parameters

- **Spot** –
- **PSSM** –
- **Paraboloid** –
- **DualParaboloidSinglePass** –
- **DualParaboloid** –
- **CubeMap** –

Variables

float $pref::PSSM::detailAdjustScale
Scales the model LOD when rendering into the PSSM shadow. Use this to reduce the draw calls when rendering the shadow by having meshes LOD out nearer to the camera than normal.

bool $Shadows::disable
Used by the editor to disable all shadow rendering.

bool $pref::Shadows::disable
Used to disable all shadow rendering.

ShadowFilterMode $pref::shadows::filterMode
The filter mode to use for shadows.

bool $AL::PSSMDebugRender
Enables debug rendering of the PSSM shadows.

float $pref::PSSM::smallestVisiblePixelSize
The smallest pixel size an object can be and still be rendered into the PSSM shadow. Use this to force culling of small objects which contribute little to the final shadow.

float $pref::Shadows::textureScalar
Used to scale the shadow texture sizes. This can reduce the shadow quality and texture memory overhead or increase them.

bool $AL::UseSSAOMask
Used by the SSAO PostEffect to toggle the sampling of ssaomask texture by the light shaders.

Basic Lighting

The script functionality related to the Basic Lghting system.

Variables

int $BasicLightManagerStats::activePlugins
The number of active Basic Lighting SceneObjectLightingPlugin objects this frame.

int $BasicLightManagerStats::elapsedUpdateMs
The number of milliseconds spent this frame updating Basic Lighting shadows.

float $pref::ProjectedShadow::fadeEndPixelSize
A size in pixels at which BL shadows are fully faded out. This should be a smaller value than fadeStartPixelSize.

float $pref::ProjectedShadow::fadeStartPixelSize
A size in pixels at which BL shadows begin to fade out. This should be a larger value than fadeEndPixelSize.

float $BasicLightManager::shadowFilterDistance
The maximum distance in meters that projected shadows will get soft filtering.

int $BasicLightManagerStats::shadowsUpdated
The number of Basic Lighting shadows updated this frame.

Render Binning

The render sorting and batching system.

Classes

RenderBinManager The abstract base for all render bins.

Inherit: SimObject

Description The render bins are used by the engine as a high level method to order and batch rendering operations.

Methods
string RenderBinManager::**getBinType**()
Returns the bin type string.

Fields
string RenderBinManager::**binType**
Sets the render bin type which limits what render instances are added to this bin.
float RenderBinManager::**processAddOrder**
Defines the order for adding instances in relation to other bins.

float RenderBinManager::**renderOrder**
Defines the order for rendering in relation to other bins.

RenderGlowMgr A render bin for the glow pass.

Inherit: RenderTexTargetBinManager

Description When the glow buffer PostEffect is enabled this bin gathers mesh render instances with glow materials and renders them to the glowbuffer offscreen render target.

This render target is then used by the 'GlowPostFx' PostEffect to blur and render the glowing portions of the screen.

RenderImposterMgr A render bin for batch rendering imposters.

Inherit: RenderBinManager

Description This render bin gathers imposter render instances and renders them in large batches.

You can type 'metrics(imposter)' in the console to see rendering statistics.

RenderMeshMgr A render bin for mesh rendering.

Inherit: RenderBinManager

Description This is the primary render bin in Torque which does most of the work of rendering DTS shapes and arbitrary mesh geometry. It knows how to render mesh instances using materials and supports hardware mesh instancing.

RenderObjectMgr A render bin which uses object callbacks for rendering.

Inherit: RenderBinManager

Description This render bin gathers object render instances and calls its delegate method to perform rendering. It is used infrequently for specialized scene objects which perform custom rendering.

RenderOcclusionMgr A render bin which renders occlusion query requests.

Inherit: RenderBinManager

Description This render bin gathers occlusion query render instances and renders them. It is currently used by light flares and ShapeBase reflection cubemaps.

You can type '$RenderOcclusionMgr::debugRender = true' in the console to see debug rendering of the occlusion geometry.

RenderParticleMgr A render bin which renders particle geometry.

Inherit: RenderTexTargetBinManager

Description This render bin gathers particle render instances, sorts, and renders them. It is currently used by ParticleEmitter and LightFlareData.

RenderPassManager A grouping of render bin managers which forms a render pass.

Inherit: SimObject

Description The render pass is used to order a set of RenderBinManager objects which are used when rendering a scene. This class does little work itself other than managing its list of render bins.

In 'core/scripts/client/renderManager.cs' you will find the DiffuseRenderPassManager which is used by the C++ engine to render the scene.

Methods

void `RenderPassManager::addManager`(RenderBinManager *renderBin*)
Add as a render bin manager to the pass.

RenderBinManager `RenderPassManager::getManager`(int *index*)
Returns the render bin manager at the index or null if the index is out of range.

int `RenderPassManager::getManagerCount`()
Returns the total number of bin managers.

void `RenderPassManager::removeManager`(RenderBinManager *renderBin*)
Removes a render bin manager.

RenderPassStateBin A non-rendering render bin used to enable/disable a RenderPassStateToken.

Inherit: RenderBinManager

Description This is a utility RenderBinManager which does not render any render instances. Its only used to define a point in the render bin order at which a RenderPassStateToken is triggered.

Fields
RenderPassStateToken `RenderPassStateBin::stateToken`

RenderPassStateToken Abstract base class for RenderFormatToken, used to manipulate what goes on in the render manager.

Inherit: SimObject

Description You cannot actually instantiate RenderPassToken, only its child: RenderFormatToken. RenderFormatToken is an implementation which changes the format of the back buffer and/or the depth buffer.

The RenderPassStateBin manager changes the rendering state associated with a token it is declared with. In stock Torque 3D, a single example exists in the way of AL_FormatToken (found in renderManager.cs). In that script file, all the render managers are intialized, and a single RenderFormatToken is used. This implementation basically exists to ensure Advanced Lighting works with MSAA.

Methods
void `RenderPassStateToken::disable()`
Disables the token.
void `RenderPassStateToken::enable()`
Enables the token.
void `RenderPassStateToken::toggle()`
Toggles the token from enabled to disabled or vice versa.

Fields
bool `RenderPassStateToken::enabled`
Enables or disables this token.

RenderPrePassMgr The render bin which performs a z+normals prepass used in Advanced Lighting.

Inherit: RenderTexTargetBinManager

Description This render bin is used in Advanced Lighting to gather all opaque mesh render instances and render them to the g-buffer for use in lighting the scene and doing effects.

PostEffect and other shaders can access the output of this bin by using the prepass texture target name. See the edge anti-aliasing post effect for an example.

RenderTerrainMgr A render bin for terrain mesh rendering.

Inherit: RenderBinManager

Description This bin renders terrain render instances from a TerrainBlock. Normally a mesh would render via the RenderMeshMgr, but terrain uses a TerrainMaterial designed for multi-layered surfaces which this bin can processs.

RenderTexTargetBinManager An abstract base class for render bin managers that render to a named textue target.

Inherit: RenderBinManager

Description This bin itself doesn't do any rendering work. It offers functionality to manage a texture render target which derived render bin classes can render into.

RenderTranslucentMgr A render bin for rendering translucent meshes.

Inherit: RenderBinManager

Description This bin is used to render translucent render mesh instances and render object instances. It is generally ordered late in the RenderPassManager after all opaque geometry bins.

Description

In Torque we use a binning system to do the initial ordering and batching of rendering operations.

When rendering a pass is made thru all the game objects visible in the scene. The game objects will each submit one or more RenderInst to the RenderPassManager. The pass manager maintains an ordered list of RenderBinManagers each which get a chance to consume the RenderInst.

After all the game objects have been processed the RenderPassManager lets each bin sort then render the RenderInsts they contain.

Currently from script you can only define and change the order of the bins in the RenderPassManager. To create new types of bins or add new rendering methods you will need C++ source access.

See also:

The file corescriptsclientrenderManager.cs.

Enumeration

enum RenderTexTargetSize

What size to render the target texture. Sizes are based on the Window the render is occuring in.

Parameters

- **windowsize** – Render to the size of the window.
- **windowsizescaled** – Render to the size of the window, scaled to the render target's size.
- **fixedsize** – Don't scale the target texture, and render to its default size.

Variables

bool `RenderOcclusionMgr::`**`debugRender`**[static, inherited]
A debugging feature which renders the occlusion volumes to the scene.

bool `RenderTerrainMgr::`**`renderWireframe`**[static, inherited]
Used to enable wireframe rendering on terrain for debugging.

5.3.7 Examples

Examples

Classes

RenderMeshExample An example scene object which renders a mesh.

Inherit: SceneObject

Description This class implements a basic SceneObject that can exist in the world at a 3D position and render itself. There are several valid ways to render an object in Torque. This class implements the preferred rendering method which is to submit a MeshRenderInst along with a Material, vertex buffer, primitive buffer, and transform and allow the RenderMeshMgr handle the actual setup and rendering for you.

See the C++ code for implementation details.

Methods
void `RenderMeshExample::`**`postApply`**`()`
A utility method for forcing a network update.

Fields
string `RenderMeshExample::`**`Material`**
The name of the material used to render the mesh.

RenderObjectExample An example scene object which renders using a callback.

Inherit: SceneObject

Description This class implements a basic SceneObject that can exist in the world at a 3D position and render itself. Note that RenderObjectExample handles its own rendering by submitting itself as an ObjectRenderInst (see renderInstance enderPassmanager.h) along with a delegate for its render() function. However, the preffered rendering method in the engine is to submit a MeshRenderInst along with a Material, vertex buffer, primitive buffer, and transform and allow the RenderMeshMgr handle the actual rendering. You can see this implemented in RenderMeshExample.

See the C++ code for implementation details.

RenderShapeExample An example scene object which renders a DTS.

Inherit: SceneObject

Description This class implements a basic SceneObject that can exist in the world at a 3D position and render itself. There are several valid ways to render an object in Torque. This class makes use of the 'TS' (three space) shape system. TS manages loading the various mesh formats supported by Torque as well was rendering those meshes (including LOD and animation...though this example doesn't include any animation over time).

See the C++ code for implementation details.

Fields

filename `RenderShapeExample::shapeFile`
The path to the DTS shape file.

5.4 Tutorials

5.4.1 Simple

Echo

Start by running a Torque 3D project. Once the game is up, open the console by pressing the tilde (~) key. In the console, type the following:

```
echo("Hello World");

OUTPUT: Hello World
```

Now, let's make use of the second parameter. Passing in a value for the second argument will append it to your text:

```
echo("Hello World", 3);

OUTPUT: Hello World3
```

Notice how there is no space between World and 3. The optional text is appended exactly how you type it. If you want, you can include your own white space to format the output:

```
echo("Hello World: ", 5);

OUTPUT: Hello World: 5
```

As you can see, the colon and space are included in the output. 5 is still appended, but does not ignore the whitespace. In addition to echo(...), there are two other output functions you will find useful. Their syntax and functionality are nearly identical to echo, but the output is different.

The two functions I'm referring to are warn(...) and error(...). You can post a message in the console and log the same way you echo:

```
warn("Be careful. Something bad might happen");

error("Something has gone horribly wrong");

OUTPUT:
Be careful. Something bad might happen (teal color)
Something has gone horribly wrong (red color)
```

You can use these functions to output multicolored text to the console, which will help you identify problems with your scripts.

Creating the Script

There is no real reason to have a script full of echo statements. You will want to use echo(...) while debugging your other functions. However, as an example, you can create a script consisting only of output statements.

First, we need to create a new script:

1. Navigate to your project's game/scripts/client directory.
2. Create a new script file called "Output.cs". In Torsion, right click on the directory, click the "New Script" option, then name your script. On Windows or OS X, create a new text file and change the extension to .cs
3. Open your new script using a text editor or Torsion.

Add the following code to the script:

```
//-----------------------------------------------------------------------------
// Torque 3D
// Copyright (C) GarageGames.com 2000 - 2009 All Rights Reserved
//-----------------------------------------------------------------------------

// Create a nice border effect around these echos, makes it easier to find
echo("************************************************************");
echo("************************************************************");

// Standard use
echo("Hello");
echo("World");
echo("Hello World");

// With escape commands
echo("H\ne\nl\nl\no\nW\no\nr\nl\nd\n");

// Appending
echo("Hello World", 1);
echo("Hello World ", 2);
echo("Hello World: ", 3);

// Warning
warn("Warning! Watch for teal text");

// Error
error("Something has gone horribly wrong");
echo("************************************************************");
echo("************************************************************");
```

Save the script now.

Testing the Script

Open game/scripts/client/init.cs and locate the initClient() function. At the end of that function, execute your new script by typing the following:

```
exec("./Output.cs");
```

Run your game, then open the console by pressing tilde (~). Look for the long string of asterisks (*), and you will find your echo statements. Note: you may need to scroll up to find the echo statements.

Calling Functions

Once a function is written, you can call it from script or the console. You only need to know the name of the function and its parameters, if it has any. The echo function is the easiest method to start with. It is a stock ConsoleFunction which accepts up to 2 parameters:

```
// Print "Hello World" in the console
// Only passing in 1 argument
echo("Hello World");
```

The echo command can actually make use of 2 arguments, depending on your goal:

```
// Print "HelloWorld" in the console
// Passing in 2 arguments

%hello = "Hello";
%world = "World";

echo(%hello, %world);
```

If your function does not use arguments, you do not have to type anything in the parenthesis:

```
// Function declaration
function CreateLevels()
{
   echo("Levels Created");
}

// Calling the function
CreateLevels();
```

The last way to call a method is invoking a member function. You can call the member functions of an object, such as a Player, using a scoping symbol:

```
// Player function "doSomething"
// %this - The Player class/object
// %action - String to print out
function Player::doSomething(%this, %action)
{
   echo(%action);
}

// Create a player object
%myPlayer = new Player(){...};

// Call "doSomething" member function
%myPlayer.doSomething("Dance");
```

Creating the Script

Now that you know how to declare and call functions, we can create a few examples from scratch. First, we need to create a new script:

1. Navigate to your project's game/scripts/client directory.
2. Create a new script file called "sampleFunctions". In Torsion, right click on the directory, click the "New Script" option, then name your script. On Windows or OS X, create a new text file and change the extension to .cs
3. Open your new script using a text editor or Torsion.

Before writing any actual script code, we should go ahead and tell the game it should load the script. Open game/scripts/client/init.cs. Scroll down to the initClient function. Under the // Client scripts section, add the following:

Execute our new script:

```
exec("./sampleFunctions.cs");
```

Now, let's write an extremely simple function that prints a message to the console. The echo(...) function already performs this, but we are going to create a more intuitively named method to work with. Type the following in the script:

```
// Print a message to the console
// Kind of repetitiously redundant
// %message - The message to print
function printMessage(%message)
{
   echo(%message);
}
```

To test your new script:

1. Save
2. Run your game
3. Open the console by pressing the tilde (~) key
4. Type the following, pressing enter after each line:

```
    printMessage("Of melodies pure and true,");
    printMessage("Sayin, this is my message to you-ou-ou");
```

Fairly straight forward. From here on, it will be assumed you know how to save your script, run the game, and call functions in the console. Next, let's create a function that takes multiple parameters. Write the following code in your script:

```
// Print two separate strings to the console
// Equally redundant in equality
// %part1 - First part of message
// %part2 - Second part of message
function printAdvancedMessage(%part1, %part2)
{
   echo(%part1, %part2);
}
```

Run the game and type the following in the console:

```
printAdvancedMessage("Singin: dont worry about a thing,", "\ncause every little thing gonna be all ri
```

In a single function call, the above code will write out two separate lyrics on different lines. Every game always has at least one initialization function. Some even have multiple inits. We can write a function that creates and initializes a few game specific variables. Note, that the variables used here are completely new and not used by stock Torque 3D projects:

```
// Change global game variables to default values
function resetGameVariables()
{
   // Game's name
   $GameName = "Blank";
```

```
   // Player's name
   $PlayerName = "Player";

   // Game play type
   $GameType = "Default";
}
```

The above code simply declares three global variables and sets them to default values. Every time this function is called, the same logic will execute. If you were to call this in the console, you will not see anything for output. Let's add a function to do this:

```
// Print our game's information to the console
function printGameInformation()
{
   echo("Game Name: ", $GameName);
   echo("Player's Name: ", $PlayerName);
   echo("Game Type: ", %gameType);
}
```

Save your new script and run the game. In the console, you will need to call the init function before the print function. Invoke the functions in this order:

```
resetGameVariables();
printGameInformation();
```

Instead of manually setting each variable in the console, we can write a "set" function for our game variables. Add the following to your script:

```
// Set the global game variables
// %gameName - Game's name
// %playerName - Player's name
// %gameType - Game play type
function setGameVariables(%gameName, %playerName, %gameType)
{
   $GameName = %gameName;
   $PlayerName = %playerName;
   $GameType = %gameType;
}
```

Now, you can set your game variables to whatever you wish through a single function call:

```
setGameVariables("Ars Moriendi", "Mich", "Survival Horror");

printGameInformation();

resetGameVariables();

printGameInformation();
```

We will get into creating member functions in a later section of the script documentation. For now, you should know enough about functions to move on.

Math

There are two types of variables you can declare and use in TorqueScript: local and global. Both are created and referenced similarly:

```
%localVariable = 1;
$globalVariable = 2;
```

As you can see, local variable names are preceded by the percent sign (%). Global variables are preceded by the dollar sign ($). Both types can be used in the same manner: operations, functions, equations, etc. The main difference has to do with how they are scoped.

In programming, scoping refers to where in memory a variable exists during its life. A local variable is meant to only exist in specific blocks of code, and its value is discarded when you leave that block. Global variables are meant to exist and hold their value during your entire programs execution.

Creating the Script

First, we need to create a new script:

1. Navigate to your project's game/scripts/client directory.
2. Create a new script file called "maths". In Torsion, right click on the directory, click the "New Script" option, then name your script. On Windows or OS X, create a new text file and change the extension to .cs.
3. Open your new script using a text editor or Torsion.

Before writing any actual script code, we should go ahead and tell the game it should load the script. Open game/scripts/client/init.cs. Scroll down to the initClient function. Under the // Client scripts section, add the following:

Execute our new script:

```
exec("./maths.cs");
```

Standard arithmetic operators are the easiest to script. Start by adding this function to your new script:

```
// Print the sum of %a and %b
function addValues(%a, %b)
{
   %sum = %a + %b;

   echo("Sum of " @ %a @ " + " @ %b @ ": ", %sum);
}
```

This simple function takes in two numerical arguments. A new variable, %sum, holds the result of adding the two arguments together. Finally, an echo(...) statement is formatted to print the original values (%a and %b) and the sum (%sum of the two).

To test your new script:

1. Save the script
2. Run your game
3. Open the console by pressing the tilde (~) key
4. Type the following, pressing enter after each line:

```
   addValues(1,1);
   addValues(2,3);
   addValues(-3,2);
```

Your console output should look like this:

```
Sum of 1 + 1: 2
Sum of 2 + 3: 5
Sum of -3 + 2: -1
```

As you can see, you can use positive or negative numbers. You can also use floating point (decimal) values if you wish. Add the following script code to test the other basic arithmetic operations:

```
// Print the difference between %a and %b
function subtractValues(%a, %b)
{
   %difference = %a - %b;

   echo("Difference between " @ %a @ " - " @ %b @ ": ", %difference);
}

// Print the product of %a and %b
function multiplyValues(%a, %b)
{
   %product = %a * %b;

   echo("Product of " @ %a @ " * " @ %b @ ": ", %product);
}

// Print the quotient of %a and %b
function divideValues(%a, %b)
{
   %quotient = %a / %b;

   echo("Quotient of " @ %a @ " / " @ %b @ ": ", %quotient);
}

// Print remainder of %a divided by %b
function moduloValue(%a, %b)
{
   %remainder = %a % %b;

   echo("Remainder of " @ %a @ " % " @ %b @ ": ", %remainder);
}
```

You will use the same process of scripting, saving, running the game, and calling the functions via the console that has been previously discussed above. Another way of manipulating values involves more complex operators. Standard additions, subtraction, etc, use two operators: assignment (=) and arithmetic (+, -, *, etc).

You can increase or decrease the value of a variable by using the auto-increment and auto-decrement operators. As soon as the operation completes, the variable is permanently changed. You do not need to use an assignment operator in this case. Use the following script code to test it out:

```
// Print the increment of %a
function incrementValue(%a)
{
   %original = %a;
   %a++;

   echo("Single increment of " @ %original @ ": ", %a);
}

// Print the decrement of %a
function decrementValue(%a)
{
```

```
   %original = %a;
   %a--;

   echo("Single decrement of " @ %original @ ": ", %a);
}
```

As you can see, the original value of %a had to be stored before the increment/decrement operation was applied. The ++ and – automatically adjust the variable for you. Another non-basic manipulation involves combining the assignment operator with an arithmetic operator:

```
// Print the result of a+=b
function addToValue(%a, %b)
{
   %original = %a;
   %a += %b;

   echo("Sum of " @ %original @ " += " @ %b @ ": ", %a);
}
```

In the above example, the + and = are combined together for a single operation. In simple terms, %a += %b can be verbalized as "A equals itself plus B." Unlike the addValue(...) function written earlier, a third variable is not used in this equation. This operation can be applied to the other arithmetic operators.

The last topic we will cover in this guide is comparison operators. As the name implies, these operators will compare two values together and produce a boolean (1 or 0) based on the results. Add the following function to see the first example:

```
// Compare %a to %b, then print the relation
function compareValues(%a, %b)
{
   if(%a > %b)
      echo("A is greater than B");
}
```

The above code is very straight forward. The values of %a and %b are compared to each other to see which is higher. Test the comparison code in the console using the following:

```
compareValues(2,1);
compareValues(3,2);
compareValues(1,2);
compareValues(0,0);
```

The output should be the following:

```
A is greater than B
A is greater than B
<no output>
<no output>
```

The first two calls will prove the comparison as "true", and print out the message. The comparison results to false on the last two calls, so nothing will be printed. The rest of the function showing off the comparison operators can be copied over what you currently have:

```
// Compare %a to %b, then print the relation
function compareValues(%a, %b)
{
   // Printing symbols just as a decorator
   // Makes it easier to isolate the print out
   echo("\n====================================");
```

```
   // Print out the value of %a and %b
   echo("\nValue of A: ", %a);
   echo("Value of B: ", %b);

   if(!%a)
      echo("\nA is a zero value\n");
   else
      echo("\nA is a non-zero value\n");

   if(!%b)
      echo("B is a zero value\n");
   else
      echo("B is a non-zero value\n");

   if(%a && %b)
      echo("Both A and B are non-zero values\n");

   if(%a || %b)
      echo("Either A or B is a non-zero value\n");

   if(%a == %b)
      echo("A is exactly equal to B\n");

   if(%a != %b)
      echo("A is not equal to B\n");

   if(%a < %b)
      echo("A is less than B");
   else if(%a <= %b)
      echo("A is less than or equal to B");

   if(%a > %b)
      echo("A is greater than B");
   else if(%a >= %b)
      echo("A is greater than or equal to B");

   // Printing symbols just as a decorator
   // Makes it easier to isolate the print out
   echo("\n====================================");
}
```

I have added "decorator text" to help separate console output and make the output easier to read. Notice that each operation uses an if(...) statement to compare. Remember, the if(...) code is based on checking for a 1 (true) or 0 (false) value. This is all a comparison operation will return.

String Manipulation

Text, such as names or phrases, are supported as strings. Numbers can also be stored in string format. Standard strings are stored in double-quotes:

```
"abcd"    (string)
```

Example:

```
$UserName = "Heather";
```

Strings with single quotes are called "tagged strings.":

```
'abcd'  (tagged string)
```

Tagged strings are special in that they contain string data, but also have a special numeric tag associated with them. Tagged strings are used for sending string data across a network. The value of a tagged string is only sent once, regardless of how many times you actually do the sending.

On subsequent sends, only the tag value is sent. Tagged values must be de-tagged when printing. You will not need to use a tagged string often unless you are in need of sending strings across a network often, like a chat system.

There are special values you can use to concatenate strings and variables. Concatenation refers to the joining of multiple values into a single variable. The following is the basic syntax:

```
"string 1" operation "string 2"
```

You can use string operators similarly to how you use mathematical operators (=, +, -, *). You have four operators at your disposal:

```
@       (concatenates two strings)
TAB     (concatenation with tab)
SPC     (concatenation with space)
NL      (newline)
```

Creating the Script

First, we need to create a new script:

1. Navigate to your project's game/scripts/client directory.
2. Create a new script file called "stringManip". In Torsion, right click on the directory, click the "New Script" option, then name your script. On Windows or OS X, create a new text file and change the extension to .cs.
3. Open your new script using a text editor or Torsion.

Before writing any actual script code, we should go ahead and tell the game it should load the script. Open game/scripts/client/init.cs. Scroll down to the initClient function. Under the // Client scripts section, add the following:

Execute our new script:

```
exec("./stringManip.cs");
```

In the new script, define three global variables at the very top as shown in the following code:

```
$PlayerName = "Player";
$GameName = "Default";
$BattleCry = "Hello World";
```

These are the strings that will be manipulated in this script. To test one of the variables, write the following function:

```
// Print player name string
function printPlayerName()
{
   echo($PlayerName);
}
```

The printPlayerName() function simply prints out the string value held by $PlayerName to the console. To test your new script:

1. Save the script
2. Run your game

3. Open the console by pressing the tilde (~) key
4. Type the following, and press enter:

```
   printPlayerName();
```

The output is extremely basic. All you will see is the string held by the variable. We can perform some string manipulation to print out something more descriptive. Change the function code to the following:

```
// Print player name string
function printPlayerName()
{
   // Concatenate "Player's Name" with the variable
   // Containing the name
   echo("Player's Name: " @ $PlayerName);
}
```

Now, when you call the function you will see the following output:

```
Player's Name: Default
```

This kind of string formatting and manipulation will make debugging and management a lot easier. Add the following code to achieve the same affect for the $GameName variable:

```
// Print game name string
function printGameName()
{
   // Concatenate "Game Name" with the variable
   // Containing the name
   echo("Game Name: " @ $GameName);
}
```

We will do something slightly different with the battle cry. You can store the result of a string manipulation in a variable before you use it. This will come in handy for saving permanent changes for strings and numbers. Use the following code to create a new function:

```
// Print battle cry string
function printBattleCry()
{
   // Concatenate the string in $PlayerName
   // with the static string yelled: "
   %message = $PlayerName @ " yelled: \"";

   // Concatenate the value of %message with
   // the string in $BattleCry and the " symbol
   // Store the results in the %message variable
   %message = %message @ $BattleCry @ "\"";

   // Print the new string after it
   // has been manipulated
   echo(%message);
}
```

The printBattleCry() function starts by defining a new local variable (%message) and assigning it the value of the $PlayerName concatenated with a static string. The second line concatenates the new %message variable with the contents of $BattleCry, and wraps the quotation mark around the actual phrase. In the same line, the %message variable is replaced with itself + the concatenated string.

Let's go ahead and create a function to print all of the variables out with a little decoration. Add the following to your script:

```
// Print all the game strings using a single function
function printGameStrings()
{
   echo("\n*********************************");
   echo("*         GAME STATS              *");
   echo("*********************************\n");

   echo("Game Name: " @ $GameName);
   echo("Player's Name: " @ $PlayerName);
   echo($PlayerName @ " battle cry: \"" @ $BattleCry @ "\"");
}
```

When you call this function in the console, you will get the following output:

```
*********************************
*         GAME STATS              *
*********************************

Game Name: Default
Player's Name: Player
Player battle cry: "Hello World"
```

So far we have been concatenating and printing out strings. You can also assign string values using the assignment operator (=), and compare string values using the string equality operator ($=).

The following function uses the operators to adjust the game string variables:

```
// Set game strings with other strings
// %playerName will be assigned to $PlayerName
// %gameName will be assigned to $GameName
// %battleCry will be assigned to $BattleCry
function setGameStrings(%playerName, %gameName, %battleCry)
{
   // Check to see if the two strings are identical
   // If so, do nothing and print a message.
   // Otherwise, assign the new string
   if($PlayerName $= %playerName)
      echo("New player name is identical. Doing nothing");
   else
      $PlayerName = %playerName;
}
```

The above function takes in three variables containing strings, one of which is used initially. The first if(...) check compares $PlayerName to %playerName. If the two are identical, the assignment of a new value will not occur. A message will be printed to console instead.

You can also apply the logical NOT (!) operator to a comparison to achieve the opposite test:

```
// Check to see if the two strings are different
// If so, assign the new string
// Otherwise, do nothing and print a message.
if($GameName !$= %gameName)
   $GameName = %gameName;
else
   echo("Game name is identical. Doing nothing");
```

In this check, if the two strings are NOT the same, then the new value assignment will occur. Otherwise, a message is printed to the console. You can go ahead and add the last portion of the code handling the %battleCry assignment:

```
// Check to see if the two strings are identical
// If so, do nothing and print a message.
// Otherwise, assign the new string
if($BattleCry $= %battleCry)
  echo("Battle cry is identical. Doing nothing");
else
  $BattleCry = %battleCry;
```

Looping Structures

There are two types of used in TorqueScript: for and while loops. A for loop repeats a statement or block of code for a set number of iterations. A while loop on the other hand repeats a statement or block of code as long as an expression given to the while loop remains true.

for Loop Syntax:

```
for(expression0; expression1; expression2)
{
    statement(s);
}
```

One way to label the expressions in this syntax are (startExpression; testExpression; countExpression). Each expression is separated by a semi-colon.

while Loop Syntax:

```
while(expression)
{
    statements;
}
```

As soon as the expression is met, the while loop will terminate.

Example For Loop:

```
for(%count = 0; %count < 3; %count++)
{
    echo(%count);
}

OUTPUT:
0
1
2
```

While Loop:

```
%countLimit = 0;

while(%countLimit <= 5)
{
   echo("Still in loop");
   %count++;
}
echo("Loop was terminated");

OUTPUT:
Still in loop
Still in loop
```

```
Still in loop
Still in loop
Still in loop
Loop was terminated
```

Creating the Script

First, we need to create a new script:

1. Navigate to your project's game/scripts/client directory.
2. Create a new script file called "loops". In Torsion, right click on the directory, click the "New Script" option, then name your script. On Windows or OS X, create a new text file and change the extension to .cs.
3. Open your new script using a text editor or Torsion.

Before writing any actual script code, we should go ahead and tell the game it should load the script. Open game/scripts/client/init.cs. Scroll down to the initClient function. Under the // Client scripts section, add the following:

Execute our new script:

```
exec("./loops.cs");
```

We will start with a very basic loop. Add the following to your script:

```
// Print 0 -> %count in the console
function printNumbers(%count)
{
   for(%i = 0; %i < %count; %i++)
   {
      echo(%i);
   }
}
```

The above function takes in a single argument (%count). The for(...) loop uses three expressions. The first is declaration expression. This is the setup for the loop. In this example, the iterator is defined. The iterator is the variable that will change each loop.

The second expression sets up the condition that will cause the loop to terminate. In the above code, when the iterator is no longer less than the %count variable, the loop will end. Finally, the third expression is the logic that occurs after each loop. In our example, we increment the count of our iterator.

The main logic is enclosed in brackets after the loop declaration. In above code, the iterator (%i) is printed to the console each loop. To test your new script:

1. Save the script
2. Run your game
3. Open the console by pressing the tilde (~) key
4. Type the following, and press enter:

```
   printNumbers(10);
```

Your output should look like the following:

```
0
1
2
```

```
3
4
5
6
7
8
9
```

As expected, the iterator is printed to the console then incremented by 1. Notice that it stops when it gets to 9, even though 10 was passed in. Look at the second expression's logic again:

```
%i < %count;
```

When %i reaches 10, then it is equal to the %count passed in which is also 10. 10 is not less than 10. As soon as that expression failed, the loop terminated. To get the full ten count, modify the function to use a different logic check:

```
function printNumbers(%count)
{
   for(%i = 0; %i <= %count; %i++)
   {
      echo(%i);
   }
}
```

Now, when you call the following code in the console:

```
printNumbers(10);
```

Your output should be:

```
0
1
2
3
4
5
6
7
8
9
10
```

You can apply different modifiers to your iterator. You do not always have to use an incremental counter. Add the following function to your script:

```
// Print %startCount -> 0 in the console
function countdown(%startCount)
{
   for(%i = %startCount; %i >= 0; %i--)
   {
      echo(%i);
   }
}
```

Save and run. Now you can see a countdown from a base number, as the following shows:

```
countdown(5);
```

Output:

```
5
4
3
2
1
0
```

An important keyword to remember when working with for(...) loops is continue. The continue keyword will cause a loop to immediately skip to the next iteration, similar to how the return keyword works in a function. Add the following function to see it work:

```
// Print 0 -> %count, except %skipNumber, in the console
function skipCount(%count, %skipNumber)
{
   for(%i = 0; %i <= %count; %i++)
   {
      if(%i == %skipNumber)
         continue;

      echo(%i);
   }
}
```

In the above code, when the iterator (%i) exactly matches the %skipNumber variable, the loop immediately goes to the next iteration. This ignores the echo(...) command on the next line. Try calling this in the console:

```
skipCount(5, 4);
```

The output should be:

```
0
1
2
3
5
```

Instead of terminating soon as the iterator reached 4, a continue keyword was used to skip to the next loop iteration. If a less complex loop is desired, the while(...) structure will be handy.

Add the following function to your script:

```
// Increase %count incrementally until it is no
// longer less than %breakNumber
function whileExample(%count, %breakNumber)
{
   // While the count is less than the breaknumber
   while(%count < %breakNumber)
   {
      // Print the count
      echo(%count);

      // Increase the count
      %count++;
   }
}
```

In this new function, the loop will check the expression in the parenthesis each time it completes an iteration. The body of the loop, contained in the brackets, simply prints the %count variable and then increases. You must be careful with loops, especially while(...) structures. The wrong use of variables can result in an infinite loop which will freeze your game.

Break is another keyword that affects looping structures. It will immediately terminate the loop. The following function shows proper use of a while loop avoiding infinite cycling:

```
// Increase %iterator until it is equal to
// %conditional. When it is, break out of
// the infinite loop
function breakOut(%iterator, %conditional)
{
   // If iterator is less than conditional
   // we will be stuck in an infinite loop
   // Error out and exit function.
   if(%iterator > %conditional)
   {
      error("Iterator is greater than conditional, try again");
      return;
   }

   // Loop infinitely until a condition is met
   while(true)
   {
      // Condition has been met, break out.
      if(%iterator == %conditional)
         break;

      echo(%iterator);

      %iterator++;
   }
}
```

Before the loop even starts, an if(...) check is made to make sure the variables used by the loop will insure a proper break. The goal of the loop is to continue iterating until the %iterator variable is equal to the %conditional.

The while(true) syntax creates the "infinite" loop. However, it will not loop infinitely since a break keyword is used. Once the %iterator is equal to the %conditional, a break is called. Otherwise, the %iterator is printed to the console and then increased.

To see the output, call the following in the console (pressing enter after each line):

```
breakOut(10,1);
breakOut(10,10);
breakOut(0, 10);
```

Output:

```
Iterator is greater than conditional, try again

0
1
2
3
4
5
6
7
8
9
```

The first call gives you the error message. The second call immediately causes the loop to terminate since the two variables are already equal. The last call provides the proper output of the function.

The last concept we will cover is nested loops. These are loops within other loops. For the next example, the terminology should be addressed first. The first loop is identical to the structures you have created in the past.

The nested loop is declared inside the first loop. Remember, it is important to be smart about your variable names. You can name your iterators anything you want, such as using %iterator instead of %i. If you go with the longer name, then it would make sense to name your second iterator something like "%iteratorTwo".

The naming convention for loop iterators is preferential. The use of %i typically stands for iterator. In quite a few programming primers (such as the ones this writer has read), the second iterator is often named %j. For these simple examples, you can get away with this. In more complex or critical loops, you might want to name your iterators based on what the loop does.

Add the following function:

```
// Run a nested loop
// Print messages, color based on level
function nestedLoops()
{
   // Max iteration for first loop
   %firstCount = 10;

   // Execute first loop %firstCount times
   for(%i = 0; %i < %firstCount; %i++)
   {
      // Print in teal
      warn("Running main loop: " @ %i);
   }
}
```

Run the function in the console, and you should see the following printed in a teal color:

```
Running main loop: 0
Running main loop: 1
Running main loop: 2
Running main loop: 3
Running main loop: 4
Running main loop: 5
Running main loop: 6
Running main loop: 7
Running main loop: 8
Running main loop: 9
```

For the nested loop, we will stick with a pattern. A second count variable should be declared, and the nested loop should perform a similar operation. Modify the function to use this pattern:

```
// Run a nested loop
// Print messages, color based on level
function nestedLoops()
{
   // Max iteration for first loop
   %firstCount = 10;

   // Max iteration for nested loop
   %secondCount = 2;

   // Execute first loop %firstCount times
   for(%i = 0; %i < %firstCount; %i++)
   {
      // Execute nested loop %secondCount times
      for(%j = 0; %j < %secondCount; %j++)
```

```
      {
         // Print in red
         error("Running nested loop: " @ %j);
      }
      // Print in teal
      warn("Running main loop: " @ %i);
   }
}
```

Run this function again to see the new output:

```
Running nested loop: 0
Running nested loop: 1
Running main loop: 0
Running nested loop: 0
Running nested loop: 1
Running main loop: 1
Running nested loop: 0
Running nested loop: 1
Running main loop: 2
Running nested loop: 0
Running nested loop: 1
Running main loop: 3
Running nested loop: 0
Running nested loop: 1
Running main loop: 4
Running nested loop: 0
Running nested loop: 1
Running main loop: 5
Running nested loop: 0
Running nested loop: 1
Running main loop: 6
Running nested loop: 0
Running nested loop: 1
Running main loop: 7
Running nested loop: 0
Running nested loop: 1
Running main loop: 8
Running nested loop: 0
Running nested loop: 1
Running main loop: 9
```

Your console output will be color-coded. The main loop output should still be teal, and the nested loop output should be red. Here is the breakdown of the full loop:

1. First loop starts
2. Main iterator (%i) starts at 0
3. Nested loop starts
4. Second iterator (%j) starts at 0
5. Print second iterator (0)
6. Increment second iterator
7. Print second iterator (1)
8. End nested loop
9. Print first iterator

10. Increment first loop
11. Go back to step 3, repeat until first loop ends

Based on the default values, the nested loop will execute 10 times. Its iterator will reset each time the first loop iterates. Try adjusting the %firstCount and %secondCount variables to see the varying outputs if you are still trying to understand the concept.

Array Manipulation

Arrays are data structures used to store consecutive values of the same data type. Arrays can be single-dimension or multidimensional:

```
$TestArray[n]   (Single-dimension)
$TestArray[m,n] (Multidimensional)
$TestArray[m_n] (Multidimensional)
```

If you have a list of similar variables you wish to store together, try using an array to save time and create cleaner code. The syntax displayed above uses the letters 'n' and 'm' to represent where you will input the number of elements in an array. The following example shows code that could benefit from an array:

```
$userNames[0] = "Heather";
$userNames[1] = "Nikki";
$userNames[2] = "Mich";

echo($userNames[0]);
echo($userNames[1]);
echo($userNames[2]);
```

Creating the Script

First, we need to create a new script:

1. Navigate to your project's game/scripts/client directory.
2. Create a new script file called "arrays". In Torsion, right click on the directory, click the "New Script" option, then name your script. On Windows or OS X, create a new text file and change the extension to .cs.
3. Open your new script using a text editor or Torsion.

Before writing any actual script code, we should go ahead and tell the game it should load the script. Open game/scripts/client/init.cs. Scroll down to the initClient function. Under the // Client scripts section, add the following:

Execute our new script:

```
exec("./arrays.cs");
```

This new script is going to work with two different arrays: $names and $board. $names is a single dimensional array containing strings. $board is a two dimensional array, also containing strings. The two are not related, but show off different uses of arrays. Let's create the initialization function:

```
// Set up all of the arrays
// with default values
function initArrays()
{
   // Initialize single dimensional array
   // containing a list of names
```

```
   $names[0] = "Heather";
   $names[1] = "Nikki";
   $names[2] = "Mich";

   // Initialize two dimensional array
   // containing symbols for a
   // tic-tac-toe game

   // Row one values
   $board[0,0] = "_";
   $board[0,1] = "_";
   $board[0,2] = "_";

   // Row two values
   $board[1,0] = "_";
   $board[1,1] = "_";
   $board[1,2] = "_";

   // Row three values
   $board[2,0] = "_";
   $board[2,1] = "_";
   $board[2,2] = "_";
}
```

The above code defines the two arrays ($names and $board). $names is given three strings representing people's names. $board is setup like a tic-tac-toe board. It will be making use of "X"s and "O"s, but for now the blank value is "_".

Instead of manually calling this function every time the game is run, we can call the function on game initialization. Open game/scripts/client/init.cs. Scroll down to the initClient function. Under this code:

```
exec("./arrays.cs");
```

Add the following:

```
initArrays();
```

Save the arrays.cs and init.cs scripts. Since there is nothing to see yet, create a function that will print out the values of $names:

```
// Print out all the values
// in the $names array
function printNames()
{
   // Print each name using
   // hard coded values (0,1,2)
   echo("0:" @ $names[0]);
   echo("1:" @ $names[1]);
   echo("2:" @ $names[2]);
}
```

To test your new script:

1. Save the script
2. Run your game
3. Open the console by pressing the tilde (~) key
4. Type the following, and press enter:

```
    printNames();
```

The output is extremely basic. All you will see is the strings held by the array, by index:

```
0: Heather
1: Nikki
2: Mich
```

This is a good start, but what if the array has 1000 elements? An optimization for this function would be to make use of a looping structure. Modify the printNames() function to use the following code:

```
function printNames()
{
   // Iterate through the names
   // array and print the values
   for(%i = 0; %i < 3; %i++)
      echo(%i @ ": " @ $names[%i]);
}
```

Instead of having three (or 1000) echo statements, you only have to script two lines. The above code iterates through the elements of the $names array using a for(...) loop. To change an individual element, add the following function to your script:

```
// Change the value of an array item
// %id = index to change
// %name = the new value
function setNames(%id, %name)
{
   // Our array only contains three elements:
   // [0] [1] [2]
   // If anything other than 0, 1, or 2 is
   // passed in, inform the user of an error
   if(%id > 2 || %id < 0)
   {
      error("Index " @ %id @ " out of range");
      error("Please use 0 - 2 as the %id");
   }
   else
      $names[%id] = %name;
}
```

To use this function, run the game and open the console. The first variable determines which array index is changing, and the second variable is the new string (name) to use. Example usage:

```
setNames(0, "Brad");
```

If you try to pass in any other numbers besides 0, 1, or 2, you will get an error message letting you know you have tried to access outside of the array bounds. Moving on, the script needs functions for printing, manipulating, and testing the $board array.

To print out just the values in order, add the following function:

```
// Print out the the values
// in the $board array
function printBoardValues()
{
   // %i loops through rows
   for(%i = 0; %i < 3; %i++)
   {
      // %j loops through columns
```

```
      for(%j = 0; %j < 3; %j++)
      {
         // Print the value of the [%i,%j]
         echo("[" @ %i @ "," @ %j @ "]: " @ $board[%i, %j]);
      }
   }
}
```

The above code uses the concept of nested loops. Nested loops are simply loops within other loops. Notice there are two for(...) structures set up. This allows the iteration of each row and column, which is necessary with a two-dimensional array. Calling this function will result in the following output:

```
[0,0]: _
[0,1]: _
[0,2]: _
[1,0]: _
[1,1]: _
[1,2]: _
[2,0]: _
[2,1]: _
[2,2]: _
```

As you can see, the function prints the current index and the value it contains. Being a tic-tac-toe board, it might help to visualize the board based on value locations. The following function will print the values of $board in a relative format:

```
// Print tic-tac-toe board
// in a relative format
function printBoard()
{
   // Print out an entre row in 1 echo
   echo($board[0,0] @" "@ $board[0,1] @" "@ $board[0,2]);
   echo($board[1,0] @" "@ $board[1,1] @" "@ $board[1,2]);
   echo($board[2,0] @" "@ $board[2,1] @" "@ $board[2,2]);
}
```

The initial output without changing the values will look like this:

```
_ _ _
_ _ _
_ _ _
```

If you have never played tic-tac-toe, each player takes a turn putting an X or O in one of the board positions. When three X's or O's are lined up, a player wins. The alignment can be three in a row, three in a column, or three diagonally. We can simulate this game play, but we will only work with rows.

We are going to change this function a few times, but we will start with the shell:

```
// Set a specific value in the array
// to an X or O
function setBoardValue(%row, %column, %value)
{
   // Make sure "X" or "O" was passed in
   if(%value !$= "X" && %value !$= "O")
   {
      echo("Invalid entry:\nPlease use \'X\' or \'O\'");
      return;
   }
}
```

The user will input a row index (%row), a column index (%column), and a value (%value) represented by an "X" or "O" string. If anything other than a capital X or capital O are passed in, the function will throw an error message and exit. If the function gets past the check, the value is assigned:

```
// Set a specific value in the array
// to an X or O
function setBoardValue(%row, %column, %value)
{
   // Make sure "X" or "O" was passed in
   if(%value !$= "X" && %value !$= "O")
   {
      echo("Invalid entry:\nPlease use \'X\' or \'O\'");
      return;
   }

   // Set the board value
   $board[%row, %column] = %value;
}
```

Save the script and run. Call the following functions, in order, to see the results:

```
printBoard();
setBoardValue(0,0,"X");
setBoardValue(0,1,"O");
printBoard();
```

Your output should look like the following:

```
_ _ _
_ _ _
_ _ _

X O _
_ _ _
_ _ _
```

To reset back to the default values, you can create a function that iterates through the array:

```
// Set all values of $board
// array back to "nothing"
// In this case, nothing is _
function resetBoard()
{
   // %i loops through rows
   for(%i = 0; %i < 3; %i++)
   {
      // %j loops through columns
      for(%j = 0; %j < 3; %j++)
      {
         // Set value to _
         $board[%i, %j] = "_";
      }
   }
}
```

Now, any normal game will have a victory condition. Enable to win, a row must contain three of the same value. Creating a function for this is quite simple using array access and string comparisons:

```
// Compare the values of each array
// item in a row
```

```
// If row contains the same values
// Return true for a victory
// Return false if values are different
function checkForWin()
{
   // Make sure at least the first symbol is X or O
   // Then compare the three values of a row

   // Row 1
   if($board[0,0] !$= "_" && $board[0,0] $= $board[0,1] && $board[0,1] $= $board[0,2])
      return true;

   // Row 2
   if($board[1,0] !$= "_" && $board[1,0] $= $board[1,1] && $board[1,1] $= $board[1,2])
      return true;

   // Row 3
   if($board[2,0] !$= "_" && $board[2,0] $= $board[2,1] && $board[2,1] $= $board[2,2])
      return true;

   return false;
}
```

The checkForWin() function will return true if any of the three if(...) statements pass. If there is no win condition, the function will return false. In a previous guide, you learned about the $= operator. Alternatively, you can use a function to compare two strings: strcmp(...).

The strcmp(...) function takes in two string, compares the two, then return a 1 or 0 based on the comparison. If the two strings are the same, it will return a 0. If the two strings are different, it will return a 1.

Example:

```
%string1 = "Hello";
%string2 = "Hello";
%string3 = "World";

// Returns 0
strcmp(%string1, %string2);

// Returns 1
strcmp(%string1, %string3);
```

We can replace the $= operators in the checkForWin() function using a different set of operators. Comment out the first chunk of code, and replace it with the following:

```
function checkForWin()
{
   // Make sure at least the first symbol is X or O
   // Then compare the three values of a row
   //if($board[0,0] !$= "_" && $board[0,0] $= $board[0,1] && $board[0,1] $= $board[0,2])
      //return true;
      //
   //if($board[1,0] !$= "_" && $board[1,0] $= $board[1,1] && $board[1,1] $= $board[1,2])
      //return true;
      //
   //if($board[2,0] !$= "_" && $board[2,0] $= $board[2,1] && $board[2,1] $= $board[2,2])
      //return true;

   if($board[0,0] !$= "_" && !strcmp($board[0,0], $board[0,1]) && !strcmp($board[0,1], $board[0,2]))
```

```
      return true;

   if($board[0,0] !$= "_" && !strcmp($board[1,0], $board[1,1]) && !strcmp($board[1,1], $board[1,2]))
      return true;

   if($board[0,0] !$= "_" && !strcmp($board[2,0], $board[2,1]) && !strcmp($board[2,1], $board[2,2]))
      return true;

   return false;
}
```

Let's break down the if(...) statements to see what is going on:

```
if($board[0,0] !$= "_" &&)
```

The first part checks to see if the row contains a blank entry ("_"). If this is true, then there is no point checking for anything else. The row does not have three similar values, so the function can move on to check the rest of the rows:

```
!strcmp($board[0,0], $board[0,1])
```

If the first check succeeds, the values of the row's first and second column are compared. If they are the same, a 0 is returned. Instead of catching the return value in a variable and testing it, we can just use the logical NOT (!) operator.

If the first two columns are the same, we can just compare the third column to one of the others. There is no point in making three string comparisons:

```
&& !strcmp($board[0,1], $board[0,2])
```

There are most likely more optimized ways to check for this kind of situation, but the above code demonstrates multiple syntactical approaches and comparisons. We can now have a way to check for a victory condition. Go back into the setBoardValue(...) function and add the win check:

```
function setBoardValue(%row, %column, %value)
{
   // Make sure "X" or "O" was passed in
   if(%value !$= "X" && %value !$= "O")
   {
      echo("Invalid entry:\nPlease use \'X\' or \'O\'");
      return;
   }

   // Set the board value
   $board[%row, %column] = %value;

   // Check to see if we have the same
   // three values in a row
   if(checkForWin())
   {
      // Entire row matched
      // Print a victory message
      echo("\n**********************");
      echo("*    Win Condition!  *");
      echo("**********************\n");

      // Print the board
      printBoard();

      // Reset the game
      echo("\nResetting board");
```

```
      resetBoard();
   }
}
```

Remember, the checkForWin() functions returns a true if the game has been won. The first portion of the code prints a message about the victory. After that, the board is printed to show what row won, and then resets the game.

While this version of the game is very rudimentary, you should be able to expand it by checking for columns and diagonals. There is plenty of room for optimization and more functions to make the game easier. However, this is not necessary to learning a powerful game engine like Torque 3D.

Switch Statements

There are two types of switch statements used in TorqueScript. switch(...) is used to compare numerical values and switch$(...) is used to compare strings.

Standard switch statements use numerical values to determine which case to execute.

switch Syntax:

```
switch(<numeric expression>)
{
   case value0:
       statements;
   case value1:
       statements;
   case value3:
       statements;
   default:
       statements;
}
```

Switch statements requiring string comparison use the switch$ syntax.

switch$ Syntax:

```
switch$ (<string expression>)
{
   case "string value 0":
                statements;
   case "string value 1":
                statements;
...
   case "string value N":
                statements;
   default:
                statements;
}
```

Creating the Script

First, we need to create a new script:

1. Navigate to your project's game/scripts/client directory.
2. Create a new script file called "switch". In Torsion, right click on the directory, click the "New Script" option, then name your script. On Windows or OS X, create a new text file and change the extension to .cs.

3. Open your new script using a text editor or Torsion.

Before writing any actual script code, we should go ahead and tell the game it should load the script. Open game/scripts/client/init.cs. Scroll down to the initClient function. Under the // Client scripts section, add the following:

Execute our new script:

```
exec("./switch.cs");
```

The first function we are going to write will take in a numerical argument. This number will be checked for a specific value, and a message will be printed based on the value comparison. Create the following function in your script:

```
// Print a message to a console based on
// the amount of ammo a weapon has
// %ammoCount - Ammo count (obviously)
function checkAmmoCount(%ammoCount)
{
   // If the ammo is at 0, we are out of ammo
   // If the ammo is at 1, we are at the end of the clip
   // If the ammo is at 100, we have a full clip
   // If the ammo is anything else, we do not care
   if(%ammoCount == 0)
      echo("Out of ammo, time to reload");
   else if(%ammoCount == 1)
      echo("Almost out of ammo, warn user");
   else if(%ammoCount == 100)
      echo("Full ammo count");
   else
      echo("Doing nothing");
}
```

To test your new script:

1. Save the script
2. Run your game
3. Open the console by pressing the tilde (~) key
4. Type the following, press enter after each line:

```
    checkAmmoCount(0);
    checkAmmoCount(1);
    checkAmmoCount(100);
    checkAmmoCount(44);
```

Your console output should be the following:

```
Out of ammo, time to reload

Almost out of ammo, warn user

Full ammo count

Do nothing
```

Instead of using four separate if/else checks, we can use a single switch statement to handle all of the cases. Change the checkAmmoCount(...) function to use the following code:

```
// Print a message to a console based on
// the amount of ammo a weapon has
```

```
// %ammoCount - Ammo count (obviously)
function checkAmmoCount(%ammoCount)
{
   // If the ammo is at 0, we are out of ammo
   // If the ammo is at 1, we are at the end of the clip
   // If the ammo is at 100, we have a full clip
   // If the ammo is anything else, we do not care
   switch(%ammoCount)
   {
      case 0:
         echo("Out of ammo, time to reload");
      case 1:
         echo("Almost out of ammo, warn user");
      case 100:
         echo("Full ammo count");
      default:
         echo("Doing nothing");
   }
}
```

The switch is declared using the switch(%ammoCount){...} syntax. The test value is kept in the parenthesis, and the cases are defined in the brackets. Each case you wish to check for is defined by the keyword case, the value, and a colon (case: 0).

You can write as few or as many lines of TorqueScript code between cases as you need to handle each numerical value. The default keyword is used when you want to handle a value that does not have a defined case. Without the default case, any other value besides was is defined as a case will be ignored.

If you test the function as you did previously, you should get the same result:

```
checkAmmoCount(0);
checkAmmoCount(1);
checkAmmoCount(100);
checkAmmoCount(44);
```

Result:

```
Out of ammo, time to reload

Almost out of ammo, warn user

Full ammo count

Do nothing
```

Testing strings in switch statements requires a small syntactical change. There are multiple ways to perform a string comparison. Write the following function in your script:

```
// Check to see if a person's name is
// a known user
// %userName - String containing person's name
function matchNames(%userName)
{
   if(!strcmp(%userName, "Heather"))
      echo("User Found: " @ %userName);
   else if(%userName $= "Mich")
      echo("User Found: " @ %userName);
   else if(%userName $= "Nikki")
      echo("User Found: " @ %userName);
```

```
   else
      echo("User " @ %userName @ " not found");
}
```

The above code defines a function which takes in a string as an argument, then performs three separate string comparison to find a result. The first if(...) check uses the strcmp function to check the %userName variable against a static string ("Heather").

The two other checks use the basic $= string equality operator. Finally, an else statement exists to inform the system that no user was found. Run the script and type the following to test the function:

```
matchNames("Heather");
matchNames("Mich");
matchNames("Nikki");
matchNames("Brad");
```

Output:

```
User Found: Heather
User Found: Mich
User Found: Nikki
User Brad not found
```

Instead of four separate if/else string comparison statements, a single switch can clean the code up greatly. Replace the matchNames(...) function with the following:

```
// Check to see if a person's name is
// a known user
// %userName - String containing person's name
function matchNames(%userName)
{
   switch$(%userName)
   {
      case "Heather":
         echo("User Found: " @ %userName);
      case "Mich":
         echo("User Found: " @ %userName);
      case "Nikki":
         echo("User Found: " @ %userName);
      default:
         echo("User: " @ %userName @ " not found");
   }
}
```

Just like the switch statement used in the checkAmmoCount(...) function, the above code starts with the switch$ keyword. This is followed by the string we are testing, held in the parenthesis. Instead of numerical values, the case keywords are followed by a strings.

In the above example, the case statements are comparing the test (%userName) against string literals. String literals are raw text displayed in code between quotations. If you have variables that contain a string value to test against, you can use those instead.

As with a numerical switch statement, you can write your logic in between the case statements.

5.4.2 Advanced

Player Class

Player Datablock

Shapebase Class

Turrest

Weapons

Proximity Mines

Camera Modes

RTS Prototype

TShapeConstructor

Engine to Script

Projectiles

Networking

CHAPTER 6

Engine

CHAPTER 7

License

A

ActionMap::bind (C++ function), 733
ActionMap::bindCmd (C++ function), 734
ActionMap::bindObj (C++ function), 735
ActionMap::getBinding (C++ function), 736
ActionMap::getCommand (C++ function), 736
ActionMap::getDeadZone (C++ function), 736
ActionMap::getScale (C++ function), 737
ActionMap::isInverted (C++ function), 737
ActionMap::pop (C++ function), 737
ActionMap::push (C++ function), 737
ActionMap::save (C++ function), 737
ActionMap::unbind (C++ function), 738
ActionMap::unbindObj (C++ function), 738
activateDirectInput (C++ function), 745
activatePackage (C++ function), 317
addBadWord (C++ function), 510
addGlobalShaderMacro (C++ function), 826
addMaterialMapping (C++ function), 853
addTaggedString (C++ function), 772
AIClient::getAimLocation (C++ member), 652
AIClient::getLocation (C++ member), 652
AIClient::getMoveDestination (C++ member), 652
AIClient::getTargetObject (C++ member), 652
AIClient::missionCycleCleanup (C++ member), 652
AIClient::move (C++ member), 652
AIClient::moveForward (C++ member), 652
AIClient::setAimLocation (C++ member), 652
AIClient::setMoveDestination (C++ member), 652
AIClient::setMoveSpeed (C++ member), 652
AIClient::setTargetObject (C++ member), 652
AIClient::stop (C++ member), 652
aiConnect (C++ function), 656
AIConnection::getAddress (C++ member), 654
AIConnection::getFreeLook (C++ member), 654
AIConnection::getMove (C++ function), 653
AIConnection::getTrigger (C++ function), 653
AIConnection::setFreeLook (C++ function), 653
AIConnection::setMove (C++ function), 653
AIConnection::setTrigger (C++ function), 653
AIPlayer::clearAim (C++ function), 523
AIPlayer::getAimLocation (C++ function), 523
AIPlayer::getAimObject (C++ function), 523
AIPlayer::getMoveDestination (C++ function), 523
AIPlayer::getMoveSpeed (C++ function), 523
AIPlayer::mMoveTolerance (C++ member), 524
AIPlayer::moveStuckTestDelay (C++ member), 524
AIPlayer::moveStuckTolerance (C++ member), 524
AIPlayer::setAimLocation (C++ function), 523
AIPlayer::setAimObject (C++ function), 523
AIPlayer::setMoveDestination (C++ function), 523
AIPlayer::setMoveSpeed (C++ function), 524
AIPlayer::stop (C++ function), 524
AITurretShape::activateTurret (C++ function), 529
AITurretShape::addToIgnoreList (C++ function), 529
AITurretShape::deactivateTurret (C++ function), 529
AITurretShape::getTarget (C++ function), 530
AITurretShape::getWeaponLeadVelocity (C++ function), 530
AITurretShape::hasTarget (C++ function), 530
AITurretShape::recenterTurret (C++ function), 530
AITurretShape::removeFromIgnoreList (C++ function), 530
AITurretShape::resetTarget (C++ function), 530
AITurretShape::setAllGunsFiring (C++ function), 530
AITurretShape::setGunSlotFiring (C++ function), 530
AITurretShape::setTurretState (C++ function), 530
AITurretShape::setWeaponLeadVelocity (C++ function), 530
AITurretShape::startScanForTargets (C++ function), 530
AITurretShape::startTrackingTarget (C++ function), 530
AITurretShape::stopScanForTargets (C++ function), 530
AITurretShape::stopTrackingTarget (C++ function), 531
AITurretShapeData::maxScanDistance (C++ member), 531
AITurretShapeData::maxScanHeading (C++ member), 531
AITurretShapeData::maxScanPitch (C++ member), 531
AITurretShapeData::maxWeaponRange (C++ member), 531

AITurretShapeData::scanTickFrequency (C++ member), 531
AITurretShapeData::scanTickFrequencyVariance (C++ member), 531
AITurretShapeData::stateDirection (C++ member), 531
AITurretShapeData::stateFire (C++ member), 531
AITurretShapeData::stateName (C++ member), 531
AITurretShapeData::stateScaleAnimation (C++ member), 531
AITurretShapeData::stateScan (C++ member), 531
AITurretShapeData::stateScript (C++ member), 531
AITurretShapeData::stateSequence (C++ member), 531
AITurretShapeData::stateTimeoutValue (C++ member), 531
AITurretShapeData::stateTransitionOnActivated (C++ member), 531
AITurretShapeData::stateTransitionOnAtRest (C++ member), 532
AITurretShapeData::stateTransitionOnDeactivated (C++ member), 532
AITurretShapeData::stateTransitionOnNoTarget (C++ member), 532
AITurretShapeData::stateTransitionOnNotAtRest (C++ member), 532
AITurretShapeData::stateTransitionOnTarget (C++ member), 532
AITurretShapeData::stateTransitionOnTimeout (C++ member), 532
AITurretShapeData::stateWaitForTimeout (C++ member), 532
AITurretShapeData::trackLostTargetTime (C++ member), 532
AITurretShapeData::weaponLeadVelocity (C++ member), 532
allowConnections (C++ member), 775
ArrayObject::add (C++ function), 318
ArrayObject::append (C++ function), 318
ArrayObject::caseSensitive (C++ member), 322
ArrayObject::count (C++ function), 318
ArrayObject::countKey (C++ function), 318
ArrayObject::countValue (C++ function), 318
ArrayObject::crop (C++ function), 318
ArrayObject::duplicate (C++ function), 318
ArrayObject::echo (C++ function), 319
ArrayObject::empty (C++ function), 319
ArrayObject::erase (C++ function), 319
ArrayObject::getCurrent (C++ function), 319
ArrayObject::getIndexFromKey (C++ function), 319
ArrayObject::getIndexFromValue (C++ function), 319
ArrayObject::getKey (C++ function), 319
ArrayObject::getValue (C++ function), 319
ArrayObject::insert (C++ function), 319
ArrayObject::key (C++ member), 322
ArrayObject::moveFirst (C++ function), 319
ArrayObject::moveLast (C++ function), 320
ArrayObject::moveNext (C++ function), 320
ArrayObject::movePrev (C++ function), 320
ArrayObject::pop_back (C++ function), 320
ArrayObject::pop_front (C++ function), 320
ArrayObject::push_back (C++ function), 320
ArrayObject::push_front (C++ function), 320
ArrayObject::setCurrent (C++ function), 320
ArrayObject::setKey (C++ function), 320
ArrayObject::setValue (C++ function), 320
ArrayObject::sort (C++ function), 320
ArrayObject::sorta (C++ function), 320
ArrayObject::sortd (C++ function), 320
ArrayObject::sortf (C++ function), 321
ArrayObject::sortfd (C++ function), 321
ArrayObject::sortfk (C++ function), 321
ArrayObject::sortfkd (C++ function), 321
ArrayObject::sortk (C++ function), 321
ArrayObject::sortka (C++ function), 321
ArrayObject::sortkd (C++ function), 321
ArrayObject::sortn (C++ function), 321
ArrayObject::sortna (C++ function), 321
ArrayObject::sortnd (C++ function), 321
ArrayObject::sortnk (C++ function), 321
ArrayObject::sortnka (C++ function), 321
ArrayObject::sortnkd (C++ function), 322
ArrayObject::uniqueKey (C++ function), 322
ArrayObject::uniqueValue (C++ function), 322

B

backtrace (C++ function), 307
BanList::add (C++ function), 770
BanList::addAbsolute (C++ function), 770
BanList::isBanned (C++ function), 771
BanList::removeBan (C++ function), 771
BarrelDistortionPostEffect::hmdIndex (C++ member), 822
BarrelDistortionPostEffect::scaleOutput (C++ member), 822
BarrelDistortionPostEffect::sensorIndex (C++ member), 822
BasicClouds::height (C++ member), 683
BasicClouds::layerEnabled (C++ member), 683
BasicClouds::texDirection (C++ member), 683
BasicClouds::texOffset (C++ member), 683
BasicClouds::texScale (C++ member), 683
BasicClouds::texSpeed (C++ member), 683
BasicClouds::texture (C++ member), 683
beginSampling (C++ function), 826
buildTaggedString (C++ function), 772

C

calcExplosionCoverage (C++ function), 649
call (C++ function), 324

Camera::angularDrag (C++ member), 726
Camera::angularForce (C++ member), 726
Camera::autoFitRadius (C++ function), 722
Camera::brakeMultiplier (C++ member), 726
Camera::controlMode (C++ member), 726
Camera::drag (C++ member), 726
Camera::extendedMovePosRotIndex (C++ member), 730
Camera::force (C++ member), 726
Camera::getAngularVelocity (C++ function), 722
Camera::getMode (C++ function), 722
Camera::getOffset (C++ function), 722
Camera::getPosition (C++ function), 723
Camera::getRotation (C++ function), 723
Camera::getVelocity (C++ function), 723
Camera::isEditOrbitMode (C++ function), 723
Camera::isRotationDamped (C++ function), 723
Camera::lookAt (C++ function), 723
Camera::mass (C++ member), 726
Camera::movementSpeed (C++ member), 730
Camera::newtonMode (C++ member), 726
Camera::newtonRotation (C++ member), 726
Camera::setAngularDrag (C++ function), 723
Camera::setAngularForce (C++ function), 723
Camera::setAngularVelocity (C++ function), 723
Camera::setBrakeMultiplier (C++ function), 723
Camera::setDrag (C++ function), 723
Camera::setEditOrbitMode (C++ function), 723
Camera::setEditOrbitPoint (C++ function), 723
Camera::setFlyForce (C++ function), 724
Camera::setFlyMode (C++ function), 724
Camera::setMass (C++ function), 724
Camera::setNewtonFlyMode (C++ function), 724
Camera::setOffset (C++ function), 724
Camera::setOrbitMode (C++ function), 724
Camera::setOrbitObject (C++ function), 724
Camera::setOrbitPoint (C++ function), 725
Camera::setRotation (C++ function), 725
Camera::setSpeedMultiplier (C++ function), 725
Camera::setTrackObject (C++ function), 725
Camera::setValidEditOrbitPoint (C++ function), 725
Camera::setVelocity (C++ function), 725
Camera::speedMultiplier (C++ member), 726
cleanupTexturePool (C++ function), 850
clearGFXResourceFlags (C++ function), 850
closeNetPort (C++ function), 772
CloudLayer::baseColor (C++ member), 684
CloudLayer::coverage (C++ member), 684
CloudLayer::exposure (C++ member), 684
CloudLayer::height (C++ member), 684
CloudLayer::texDirection (C++ member), 684
CloudLayer::texScale (C++ member), 684
CloudLayer::texSpeed (C++ member), 684
CloudLayer::texture (C++ member), 684
CloudLayer::windSpeed (C++ member), 684
cls (C++ function), 305
collapseEscape (C++ function), 365
commandToClient (C++ function), 772
commandToServer (C++ function), 773
compile (C++ function), 324
ConsoleLogger::attach (C++ function), 309
ConsoleLogger::detach (C++ function), 310
ConsoleLogger::level (C++ member), 310
containerBoxEmpty (C++ function), 511
containerFindFirst (C++ function), 511
containerFindNext (C++ function), 511
containerRayCast (C++ function), 511
containerSearchCurrDist (C++ function), 511
containerSearchCurrRadiusDist (C++ function), 512
containerSearchNext (C++ function), 512
containsBadWords (C++ function), 512
ConvexShape::Material (C++ member), 708
ConvexShape::surface (C++ member), 708
countBits (C++ function), 377
createPath (C++ function), 350
CubemapData::cubeFace (C++ member), 830
CubemapData::dynamic (C++ member), 830
CubemapData::dynamicFarDist (C++ member), 830
CubemapData::dynamicNearDist (C++ member), 830
CubemapData::dynamicObjectTypeMask (C++ member), 830
CubemapData::dynamicSize (C++ member), 831
CubemapData::getFilename (C++ function), 830
CubemapData::updateFaces (C++ function), 830
CustomMaterial::fallback (C++ member), 853
CustomMaterial::forwardLit (C++ member), 853
CustomMaterial::shader (C++ member), 853
CustomMaterial::stateBlock (C++ member), 853
CustomMaterial::target (C++ member), 853
CustomMaterial::version (C++ member), 853

D

deactivateDirectInput (C++ function), 745
deactivatePackage (C++ function), 317
Debris::init (C++ function), 626
Debris::lifetime (C++ member), 626
DebrisData::baseRadius (C++ member), 627
DebrisData::bounceVariance (C++ member), 627
DebrisData::elasticity (C++ member), 627
DebrisData::emitters (C++ member), 627
DebrisData::explodeOnMaxBounce (C++ member), 627
DebrisData::Explosion (C++ member), 627
DebrisData::fade (C++ member), 627
DebrisData::friction (C++ member), 627
DebrisData::gravModifier (C++ member), 627
DebrisData::ignoreWater (C++ member), 627
DebrisData::lifetime (C++ member), 627
DebrisData::lifetimeVariance (C++ member), 627
DebrisData::maxSpinSpeed (C++ member), 627

DebrisData::minSpinSpeed (C++ member), 627
DebrisData::numBounces (C++ member), 627
DebrisData::shapeFile (C++ member), 627
DebrisData::snapOnMaxBounce (C++ member), 628
DebrisData::staticOnMaxBounce (C++ member), 628
DebrisData::terminalVelocity (C++ member), 628
DebrisData::texture (C++ member), 628
DebrisData::useRadiusMass (C++ member), 628
DebrisData::velocity (C++ member), 628
DebrisData::velocityVariance (C++ member), 628
debug (C++ function), 307
DebugDrawer::drawBox (C++ function), 831
DebugDrawer::drawLine (C++ function), 831
DebugDrawer::setLastTTL (C++ function), 831
DebugDrawer::setLastZTest (C++ function), 831
DebugDrawer::toggleDrawing (C++ function), 831
DebugDrawer::toggleFreeze (C++ function), 831
debugDumpAllObjects (C++ function), 307
debugEnumInstances (C++ function), 305
debugv (C++ function), 307
DecalData::clippingAngle (C++ member), 629
DecalData::fadeEndPixelSize (C++ member), 629
DecalData::fadeStartPixelSize (C++ member), 629
DecalData::fadeTime (C++ member), 629
DecalData::frame (C++ member), 629
DecalData::lifeSpan (C++ member), 629
DecalData::Material (C++ member), 629
DecalData::postApply (C++ function), 628
DecalData::randomize (C++ member), 629
DecalData::renderPriority (C++ member), 629
DecalData::size (C++ member), 629
DecalData::texCols (C++ member), 629
DecalData::texRows (C++ member), 629
DecalData::textureCoordCount (C++ member), 629
DecalData::textureCoords (C++ member), 629
decalManagerAddDecal (C++ function), 650
decalManagerClear (C++ function), 650
decalManagerDirty (C++ function), 650
decalManagerLoad (C++ function), 651
decalManagerRemoveDecal (C++ function), 651
decalManagerSave (C++ function), 651
DecalRoad::breakAngle (C++ member), 693
DecalRoad::discardAll (C++ member), 693
DecalRoad::EditorOpen (C++ member), 693
DecalRoad::Material (C++ member), 693
DecalRoad::Node (C++ member), 694
DecalRoad::postApply (C++ function), 693
DecalRoad::regenerate (C++ function), 693
DecalRoad::renderPriority (C++ member), 694
DecalRoad::showBatches (C++ member), 694
DecalRoad::showRoad (C++ member), 694
DecalRoad::showSpline (C++ member), 694
DecalRoad::textureLength (C++ member), 694
DecalRoad::updateDelay (C++ member), 694
DecalRoad::wireframe (C++ member), 694
deleteVariables (C++ function), 324
describeGFXResources (C++ function), 850
describeGFXStateBlocks (C++ function), 851
detag (C++ function), 774
disableJoystick (C++ function), 745
disableXInput (C++ function), 745
dispatchMessage (C++ function), 316
dispatchMessageObject (C++ function), 316
displaySplashWindow (C++ function), 776
DNetSetLogging (C++ function), 774
dumpAlloc (C++ function), 307
dumpConsoleClasses (C++ function), 310
dumpConsoleFunctions (C++ function), 310
dumpEngineDocs (C++ function), 305
dumpFontCacheStatus (C++ function), 828
dumpMaterialInstances (C++ member), 853
dumpMemSnapshot (C++ function), 307
dumpNetStats (C++ function), 774
dumpNetStringTable (C++ function), 774
dumpRandomNormalMap (C++ function), 851
dumpStringMemStats (C++ function), 365
dumpTextureObjects (C++ function), 851
dumpUnflaggedAllocs (C++ function), 307
duplicateCachedFont (C++ function), 828

E

echo (C++ function), 310
echoInputState (C++ function), 745
enableJoystick (C++ function), 745
enableSamples (C++ function), 827
enableXInput (C++ function), 745
endsWith (C++ function), 365
error (C++ function), 311
EventManager::dumpEvents (C++ function), 312
EventManager::dumpSubscribers (C++ function), 312
EventManager::isRegisteredEvent (C++ function), 312
EventManager::postEvent (C++ function), 312
EventManager::queue (C++ member), 313
EventManager::registerEvent (C++ function), 312
EventManager::remove (C++ function), 312
EventManager::removeAll (C++ function), 313
EventManager::subscribe (C++ function), 313
EventManager::unregisterEvent (C++ function), 313
excludeOtherInstance (C++ function), 428
exec (C++ function), 324
execPrefs (C++ function), 325
expandEscape (C++ function), 366
expandFilename (C++ function), 350
expandOldFilename (C++ function), 350
ExplosionData::camShakeAmp (C++ member), 631
ExplosionData::camShakeDuration (C++ member), 631
ExplosionData::camShakeFalloff (C++ member), 631
ExplosionData::camShakeFreq (C++ member), 631

ExplosionData::camShakeRadius (C++ member), 631
ExplosionData::Debris (C++ member), 631
ExplosionData::debrisNum (C++ member), 631
ExplosionData::debrisNumVariance (C++ member), 631
ExplosionData::debrisPhiMax (C++ member), 631
ExplosionData::debrisPhiMin (C++ member), 631
ExplosionData::debrisThetaMax (C++ member), 631
ExplosionData::debrisThetaMin (C++ member), 631
ExplosionData::debrisVelocity (C++ member), 632
ExplosionData::debrisVelocityVariance (C++ member), 632
ExplosionData::delayMS (C++ member), 632
ExplosionData::delayVariance (C++ member), 632
ExplosionData::emitter (C++ member), 632
ExplosionData::explosionScale (C++ member), 632
ExplosionData::explosionShape (C++ member), 632
ExplosionData::faceViewer (C++ member), 632
ExplosionData::lifetimeMS (C++ member), 632
ExplosionData::lifetimeVariance (C++ member), 632
ExplosionData::lightEndBrightness (C++ member), 632
ExplosionData::lightEndColor (C++ member), 632
ExplosionData::lightEndRadius (C++ member), 632
ExplosionData::lightNormalOffset (C++ member), 632
ExplosionData::lightStartBrightness (C++ member), 632
ExplosionData::lightStartColor (C++ member), 632
ExplosionData::lightStartRadius (C++ member), 632
ExplosionData::offset (C++ member), 632
ExplosionData::particleDensity (C++ member), 632
ExplosionData::ParticleEmitter (C++ member), 633
ExplosionData::particleRadius (C++ member), 633
ExplosionData::playSpeed (C++ member), 633
ExplosionData::shakeCamera (C++ member), 633
ExplosionData::sizes (C++ member), 633
ExplosionData::soundProfile (C++ member), 633
ExplosionData::subExplosion (C++ member), 633
ExplosionData::times (C++ member), 633
exportCachedFont (C++ function), 828
exportEngineAPIToXML (C++ function), 305

F

fileBase (C++ function), 350
fileCreatedTime (C++ function), 350
fileDelete (C++ function), 350
FileDialog::changePath (C++ member), 328
FileDialog::defaultFile (C++ member), 329
FileDialog::defaultPath (C++ member), 329
FileDialog::Execute (C++ function), 327
FileDialog::fileName (C++ member), 329
FileDialog::filters (C++ member), 329
FileDialog::title (C++ member), 329
fileExt (C++ function), 350
fileModifiedTime (C++ function), 351
fileName (C++ function), 351
FileObject::close (C++ function), 330
FileObject::isEOF (C++ function), 330
FileObject::openForAppend (C++ function), 331
FileObject::openForRead (C++ function), 331
FileObject::openForWrite (C++ function), 331
FileObject::peekLine (C++ function), 331
FileObject::readLine (C++ function), 332
FileObject::writeLine (C++ function), 332
FileObject::writeObject (C++ function), 332, 333
filePath (C++ function), 351
fileSize (C++ function), 351
FileStreamObject::close (C++ function), 334
FileStreamObject::open (C++ function), 335
filterString (C++ function), 513
findFirstFile (C++ function), 353
findFirstFileMultiExpr (C++ function), 354
findNextFile (C++ function), 354
findNextFileMultiExpr (C++ function), 354
firstWord (C++ function), 373
flagCurrentAllocs (C++ function), 307
flagCurrentGFXResources (C++ function), 851
flushTextureCache (C++ function), 851
FlyingVehicle::useCreateHeight (C++ function), 670
FlyingVehicleData::autoAngularForce (C++ member), 670
FlyingVehicleData::autoInputDamping (C++ member), 670
FlyingVehicleData::autoLinearForce (C++ member), 670
FlyingVehicleData::backwardJetEmitter (C++ member), 670
FlyingVehicleData::createHoverHeight (C++ member), 670
FlyingVehicleData::downJetEmitter (C++ member), 670
FlyingVehicleData::engineSound (C++ member), 670
FlyingVehicleData::forwardJetEmitter (C++ member), 670
FlyingVehicleData::horizontalSurfaceForce (C++ member), 671
FlyingVehicleData::hoverHeight (C++ member), 671
FlyingVehicleData::jetSound (C++ member), 671
FlyingVehicleData::maneuveringForce (C++ member), 671
FlyingVehicleData::maxAutoSpeed (C++ member), 671
FlyingVehicleData::minTrailSpeed (C++ member), 671
FlyingVehicleData::rollForce (C++ member), 671
FlyingVehicleData::rotationalDrag (C++ member), 671
FlyingVehicleData::steeringForce (C++ member), 671
FlyingVehicleData::steeringRollForce (C++ member), 671
FlyingVehicleData::trailEmitter (C++ member), 671
FlyingVehicleData::verticalSurfaceForce (C++ member), 671
FlyingVehicleData::vertThrustMultiple (C++ member), 671
fmodDumpDSPInfo (C++ function), 820

fmodDumpMemoryStats (C++ function), 820
Forest::clear (C++ function), 698
Forest::dataFile (C++ member), 698
Forest::disableImposters (C++ member), 700
Forest::drawBounds (C++ member), 700
Forest::drawCells (C++ member), 700
Forest::forceImposters (C++ member), 700
Forest::isDirty (C++ function), 698
Forest::lodReflectScalar (C++ member), 698
Forest::regenCells (C++ function), 698
Forest::saveDataFile (C++ member), 698
ForestBrushElement::elevationMax (C++ member), 699
ForestBrushElement::elevationMin (C++ member), 699
ForestBrushElement::ForestItemData (C++ member), 699
ForestBrushElement::probability (C++ member), 699
ForestBrushElement::rotationRange (C++ member), 699
ForestBrushElement::scaleExponent (C++ member), 699
ForestBrushElement::scaleMax (C++ member), 699
ForestBrushElement::scaleMin (C++ member), 699
ForestBrushElement::sinkMax (C++ member), 699
ForestBrushElement::sinkMin (C++ member), 699
ForestBrushElement::sinkRadius (C++ member), 699
ForestBrushElement::slopeMax (C++ member), 699
ForestBrushElement::slopeMin (C++ member), 699
ForestItemData::branchAmp (C++ member), 699
ForestItemData::collidable (C++ member), 699
ForestItemData::dampingCoefficient (C++ member), 699
ForestItemData::detailAmp (C++ member), 699
ForestItemData::detailFreq (C++ member), 700
ForestItemData::mass (C++ member), 700
ForestItemData::radius (C++ member), 700
ForestItemData::rigidity (C++ member), 700
ForestItemData::shapeFile (C++ member), 700
ForestItemData::tightnessCoefficient (C++ member), 700
ForestItemData::trunkBendScale (C++ member), 700
ForestItemData::windScale (C++ member), 700
ForestWindEmitter::attachToObject (C++ function), 634
ForestWindEmitter::gustFrequency (C++ member), 634
ForestWindEmitter::gustStrength (C++ member), 634
ForestWindEmitter::gustWobbleStrength (C++ member), 634
ForestWindEmitter::gustYawAngle (C++ member), 634
ForestWindEmitter::gustYawFrequency (C++ member), 634
ForestWindEmitter::hasMount (C++ member), 634
ForestWindEmitter::radialEmitter (C++ member), 634
ForestWindEmitter::radius (C++ member), 634
ForestWindEmitter::strength (C++ member), 634
ForestWindEmitter::turbulenceFrequency (C++ member), 634
ForestWindEmitter::turbulenceStrength (C++ member), 634
ForestWindEmitter::windEnabled (C++ member), 634
freeMemoryDump (C++ function), 307
fxFoliageReplicator::AllowedTerrainSlope (C++ member), 701
fxFoliageReplicator::AllowOnStatics (C++ member), 701
fxFoliageReplicator::AllowOnTerrain (C++ member), 701
fxFoliageReplicator::AllowOnWater (C++ member), 701
fxFoliageReplicator::AllowWaterSurface (C++ member), 701
fxFoliageReplicator::AlphaCutoff (C++ member), 701
fxFoliageReplicator::CullResolution (C++ member), 701
fxFoliageReplicator::DebugBoxHeight (C++ member), 701
fxFoliageReplicator::FadeInRegion (C++ member), 701
fxFoliageReplicator::FadeOutRegion (C++ member), 701
fxFoliageReplicator::FixAspectRatio (C++ member), 701
fxFoliageReplicator::FixSizeToMax (C++ member), 701
fxFoliageReplicator::FoliageCount (C++ member), 701
fxFoliageReplicator::FoliageFile (C++ member), 701
fxFoliageReplicator::FoliageRetries (C++ member), 701
fxFoliageReplicator::GroundAlpha (C++ member), 701
fxFoliageReplicator::HideFoliage (C++ member), 701
fxFoliageReplicator::InnerRadiusX (C++ member), 701
fxFoliageReplicator::InnerRadiusY (C++ member), 701
fxFoliageReplicator::LightOn (C++ member), 701
fxFoliageReplicator::LightSync (C++ member), 701
fxFoliageReplicator::lightTime (C++ member), 701
fxFoliageReplicator::MaxHeight (C++ member), 702
fxFoliageReplicator::MaxLuminance (C++ member), 702
fxFoliageReplicator::MaxSwayTime (C++ member), 702
fxFoliageReplicator::MaxWidth (C++ member), 702
fxFoliageReplicator::MinHeight (C++ member), 702
fxFoliageReplicator::MinLuminance (C++ member), 702
fxFoliageReplicator::MinSwayTime (C++ member), 702
fxFoliageReplicator::MinWidth (C++ member), 702
fxFoliageReplicator::OffsetZ (C++ member), 702
fxFoliageReplicator::OuterRadiusX (C++ member), 702
fxFoliageReplicator::OuterRadiusY (C++ member), 702
fxFoliageReplicator::PlacementAreaHeight (C++ member), 702
fxFoliageReplicator::PlacementColour (C++ member), 702
fxFoliageReplicator::RandomFlip (C++ member), 702
fxFoliageReplicator::seed (C++ member), 702
fxFoliageReplicator::ShowPlacementArea (C++ member), 702
fxFoliageReplicator::SwayMagFront (C++ member), 702
fxFoliageReplicator::SwayMagSide (C++ member), 702
fxFoliageReplicator::SwayOn (C++ member), 702
fxFoliageReplicator::SwaySync (C++ member), 702
fxFoliageReplicator::UseCulling (C++ member), 702
fxFoliageReplicator::UseDebugInfo (C++ member), 703
fxFoliageReplicator::useTrueBillboards (C++ member), 703

fxFoliageReplicator::ViewClosest (C++ member), 703
fxFoliageReplicator::ViewDistance (C++ member), 703
fxShapeReplicatedStatic::allowPlayerStep (C++ member), 703
fxShapeReplicatedStatic::collisionType (C++ member), 703
fxShapeReplicatedStatic::decalType (C++ member), 703
fxShapeReplicatedStatic::forceDetail (C++ member), 703
fxShapeReplicatedStatic::meshCulling (C++ member), 703
fxShapeReplicatedStatic::originSort (C++ member), 703
fxShapeReplicatedStatic::playAmbient (C++ member), 703
fxShapeReplicatedStatic::renderNormals (C++ member), 703
fxShapeReplicatedStatic::shapeName (C++ member), 703
fxShapeReplicatedStatic::skin (C++ member), 703
fxShapeReplicator::AlignToTerrain (C++ member), 704
fxShapeReplicator::AllowedTerrainSlope (C++ member), 704
fxShapeReplicator::AllowOnStatics (C++ member), 704
fxShapeReplicator::AllowOnTerrain (C++ member), 704
fxShapeReplicator::AllowOnWater (C++ member), 704
fxShapeReplicator::AllowWaterSurface (C++ member), 704
fxShapeReplicator::HideReplications (C++ member), 704
fxShapeReplicator::InnerRadiusX (C++ member), 704
fxShapeReplicator::InnerRadiusY (C++ member), 704
fxShapeReplicator::Interactions (C++ member), 704
fxShapeReplicator::OffsetZ (C++ member), 704
fxShapeReplicator::OuterRadiusX (C++ member), 704
fxShapeReplicator::OuterRadiusY (C++ member), 704
fxShapeReplicator::PlacementAreaHeight (C++ member), 704
fxShapeReplicator::PlacementColour (C++ member), 704
fxShapeReplicator::seed (C++ member), 704
fxShapeReplicator::ShapeCount (C++ member), 705
fxShapeReplicator::shapeFile (C++ member), 705
fxShapeReplicator::ShapeRetries (C++ member), 705
fxShapeReplicator::ShapeRotateMax (C++ member), 705
fxShapeReplicator::ShapeRotateMin (C++ member), 705
fxShapeReplicator::ShapeScaleMax (C++ member), 705
fxShapeReplicator::ShapeScaleMin (C++ member), 705
fxShapeReplicator::ShowPlacementArea (C++ member), 705
fxShapeReplicator::TerrainAlignment (C++ member), 705

G

GameBase::applyImpulse (C++ function), 532
GameBase::applyRadialImpulse (C++ function), 532
GameBase::boundingBox (C++ member), 533
GameBase::dataBlock (C++ member), 533
GameBase::getDataBlock (C++ function), 532
GameBase::setControl (C++ function), 532
GameBase::setDataBlock (C++ function), 533
GameBaseData::category (C++ member), 534
GameBaseData::onAdd (C++ function), 533
GameBaseData::onMount (C++ function), 534
GameBaseData::onNewDataBlock (C++ function), 534
GameBaseData::onRemove (C++ function), 534
GameBaseData::onUnmount (C++ function), 534
GameConnection::activateGhosting (C++ function), 751
GameConnection::chaseCam (C++ function), 751
GameConnection::clearCameraObject (C++ function), 751
GameConnection::clearDisplayDevice (C++ function), 751
GameConnection::getCameraObject (C++ function), 752
GameConnection::getControlCameraDefaultFov (C++ function), 752
GameConnection::getControlCameraFov (C++ function), 752
GameConnection::getControlObject (C++ function), 752
GameConnection::getControlSchemeAbsoluteRotation (C++ function), 752
GameConnection::getDamageFlash (C++ function), 752
GameConnection::getServerConnection (C++ function), 752
GameConnection::getWhiteOut (C++ function), 752
GameConnection::initialControlSet (C++ function), 752
GameConnection::isAIControlled (C++ function), 752
GameConnection::isControlObjectRotDampedCamera (C++ function), 752
GameConnection::isDemoPlaying (C++ function), 752
GameConnection::isDemoRecording (C++ function), 752
GameConnection::isFirstPerson (C++ function), 752
GameConnection::listClassIDs (C++ function), 752
GameConnection::onConnectionAccepted (C++ function), 752
GameConnection::onConnectionDropped (C++ function), 752
GameConnection::onConnectionError (C++ function), 753
GameConnection::onConnectionTimedOut (C++ function), 753
GameConnection::onConnectRequestRejected (C++ function), 753
GameConnection::onConnectRequestTimedOut (C++ function), 753
GameConnection::onControlObjectChange (C++ function), 753
GameConnection::onDataBlocksDone (C++ function), 753
GameConnection::onDrop (C++ function), 753

GameConnection::onFlash (C++ function), 753
GameConnection::play2D (C++ function), 753
GameConnection::play3D (C++ function), 753
GameConnection::playDemo (C++ function), 754
GameConnection::resetGhosting (C++ function), 754
GameConnection::setBlackOut (C++ function), 754
GameConnection::setCameraObject (C++ function), 754
GameConnection::setConnectArgs (C++ function), 754
GameConnection::setControlCameraFov (C++ function), 754
GameConnection::setControlObject (C++ function), 755
GameConnection::setControlSchemeParameters (C++ function), 755
GameConnection::setFirstPerson (C++ function), 755
GameConnection::setJoinPassword (C++ function), 755
GameConnection::setLagIcon (C++ function), 755
GameConnection::setMissionCRC (C++ function), 755
GameConnection::startRecording (C++ function), 756
GameConnection::stopRecording (C++ function), 756
GameConnection::transmitDataBlocks (C++ function), 756
generateUUID (C++ function), 377
getActiveDDSFiles (C++ function), 827
getActiveLightManager (C++ function), 862
getAppVersionNumber (C++ member), 309
getAppVersionString (C++ member), 309
getBestHDRFormat (C++ function), 851
getBitmapInfo (C++ function), 827
getBoxCenter (C++ function), 355
getBuildString (C++ member), 309
getCategoryOfClass (C++ function), 305
getCompileTimeString (C++ member), 309
getCoreLangTable (C++ function), 786
getCurrentActionMap (C++ function), 745
getCurrentDirectory (C++ function), 351
getDescriptionOfClass (C++ function), 305
getDesktopResolution (C++ function), 851
getDirectoryList (C++ function), 351
getDisplayDeviceInformation (C++ function), 851
getDisplayDeviceList (C++ function), 851
getDSOPath (C++ function), 325
getEngineName (C++ member), 309
getExecutableName (C++ function), 351
getField (C++ function), 373
getFieldCount (C++ function), 374
getFields (C++ function), 374
getFileCount (C++ function), 355
getFileCountMultiExpr (C++ function), 355
getFileCRC (C++ function), 351
getFunctionPackage (C++ function), 317
getLightManagerNames (C++ function), 862
getMainDotCsDir (C++ function), 351
getMaterialMapping (C++ function), 853
getMax (C++ function), 355
getMethodPackage (C++ function), 317
getMin (C++ function), 356
getMissionAreaServerObject (C++ function), 720
getOVRHMDChromaticAbCorrection (C++ function), 513
getOVRHMDCount (C++ function), 513
getOVRHMDCurrentIPD (C++ function), 513
getOVRHMDDisplayDesktopPos (C++ function), 513
getOVRHMDDisplayDeviceId (C++ function), 513
getOVRHMDDisplayDeviceName (C++ function), 513
getOVRHMDDistortionCoefficients (C++ function), 514
getOVRHMDDistortionScale (C++ function), 514
getOVRHMDEyeXOffsets (C++ function), 514
getOVRHMDManufacturer (C++ function), 514
getOVRHMDProductName (C++ function), 514
getOVRHMDProfileIPD (C++ function), 514
getOVRHMDResolution (C++ function), 514
getOVRHMDVersion (C++ function), 514
getOVRHMDXCenterOffset (C++ function), 514
getOVRHMDYFOV (C++ function), 514
getOVRSensorAcceleration (C++ function), 515
getOVRSensorAngVelocity (C++ function), 515
getOVRSensorCount (C++ function), 515
getOVRSensorEulerRotation (C++ function), 515
getOVRSensorGravityCorrection (C++ function), 515
getOVRSensorMagnetometer (C++ function), 515
getOVRSensorMagnetometerCalibrated (C++ function), 515
getOVRSensorPredictionTime (C++ function), 515
getOVRSensorYawCorrection (C++ function), 515
getPackageList (C++ function), 317
getPixelShaderVersion (C++ function), 851
getRandom (C++ function), 364
getRandomSeed (C++ function), 365
getRazerHydraControllerPos (C++ function), 515
getRazerHydraControllerRot (C++ function), 516
getRazerHydraControllerTransform (C++ function), 516
getRealTime (C++ function), 777
getRecord (C++ function), 374
getRecordCount (C++ function), 374
getRecords (C++ function), 375
getSimTime (C++ function), 777
getSubStr (C++ function), 366
getTag (C++ function), 774
getTaggedString (C++ function), 774
getTerrainHeight (C++ function), 697
getTerrainHeightBelowPosition (C++ function), 697
getTerrainUnderWorldPoint (C++ function), 697
getTextureProfileStats (C++ function), 851
getTrailingNumber (C++ function), 366
getVariable (C++ function), 325
getVersionNumber (C++ member), 309
getVersionString (C++ member), 309
getWebDeployment (C++ function), 777

getWord (C++ function), 375
getWordCount (C++ function), 375
getWords (C++ function), 375
getWorkingDirectory (C++ function), 351
getXInputState (C++ function), 746
GFXCardProfilerAPI::getCard (C++ function), 833
GFXCardProfilerAPI::getRenderer (C++ function), 833
GFXCardProfilerAPI::getVendor (C++ function), 833
GFXCardProfilerAPI::getVersion (C++ function), 833
GFXCardProfilerAPI::getVideoMemoryMB (C++ function), 833
GFXCardProfilerAPI::queryProfile (C++ function), 833
GFXCardProfilerAPI::setCapability (C++ function), 833
GFXInit::createNullDevice (C++ function), 833
GFXInit::getAdapterCount (C++ function), 851
GFXInit::getAdapterMode (C++ function), 833
GFXInit::getAdapterModeCount (C++ function), 833
GFXInit::getAdapterName (C++ function), 834
GFXInit::getAdapterOutputName (C++ function), 834
GFXInit::getAdapterShaderModel (C++ function), 834
GFXInit::getAdapterType (C++ function), 834
GFXInit::getDefaultAdapterIndex (C++ function), 834
GFXSamplerStateData::addressModeU (C++ member), 835
GFXSamplerStateData::addressModeV (C++ member), 835
GFXSamplerStateData::addressModeW (C++ member), 835
GFXSamplerStateData::alphaArg1 (C++ member), 835
GFXSamplerStateData::alphaArg2 (C++ member), 835
GFXSamplerStateData::alphaArg3 (C++ member), 835
GFXSamplerStateData::alphaOp (C++ member), 835
GFXSamplerStateData::colorArg1 (C++ member), 835
GFXSamplerStateData::colorArg2 (C++ member), 835
GFXSamplerStateData::colorArg3 (C++ member), 835
GFXSamplerStateData::magFilter (C++ member), 835
GFXSamplerStateData::maxAnisotropy (C++ member), 835
GFXSamplerStateData::minFilter (C++ member), 835
GFXSamplerStateData::mipFilter (C++ member), 835
GFXSamplerStateData::mipLODBias (C++ member), 835
GFXSamplerStateData::resultArg (C++ member), 835
GFXSamplerStateData::textureColorOp (C++ member), 835
GFXSamplerStateData::textureTransform (C++ member), 835
GFXStateBlockData::alphaDefined (C++ member), 836
GFXStateBlockData::alphaTestEnable (C++ member), 836
GFXStateBlockData::alphaTestFunc (C++ member), 836
GFXStateBlockData::alphaTestRef (C++ member), 836
GFXStateBlockData::blendDefined (C++ member), 836
GFXStateBlockData::blendDest (C++ member), 836
GFXStateBlockData::blendEnable (C++ member), 836
GFXStateBlockData::blendOp (C++ member), 836
GFXStateBlockData::blendSrc (C++ member), 836
GFXStateBlockData::colorWriteAlpha (C++ member), 836
GFXStateBlockData::colorWriteBlue (C++ member), 836
GFXStateBlockData::colorWriteDefined (C++ member), 836
GFXStateBlockData::colorWriteGreen (C++ member), 836
GFXStateBlockData::colorWriteRed (C++ member), 836
GFXStateBlockData::cullDefined (C++ member), 837
GFXStateBlockData::cullMode (C++ member), 837
GFXStateBlockData::ffLighting (C++ member), 837
GFXStateBlockData::samplersDefined (C++ member), 837
GFXStateBlockData::samplerStates (C++ member), 837
GFXStateBlockData::separateAlphaBlendDefined (C++ member), 837
GFXStateBlockData::separateAlphaBlendDest (C++ member), 837
GFXStateBlockData::separateAlphaBlendEnable (C++ member), 837
GFXStateBlockData::separateAlphaBlendOp (C++ member), 837
GFXStateBlockData::separateAlphaBlendSrc (C++ member), 837
GFXStateBlockData::stencilDefined (C++ member), 837
GFXStateBlockData::stencilEnable (C++ member), 837
GFXStateBlockData::stencilFailOp (C++ member), 837
GFXStateBlockData::stencilFunc (C++ member), 837
GFXStateBlockData::stencilMask (C++ member), 837
GFXStateBlockData::stencilPassOp (C++ member), 837
GFXStateBlockData::stencilRef (C++ member), 837
GFXStateBlockData::stencilWriteMask (C++ member), 837
GFXStateBlockData::stencilZFailOp (C++ member), 837
GFXStateBlockData::textureFactor (C++ member), 837
GFXStateBlockData::vertexColorEnable (C++ member), 838
GFXStateBlockData::zBias (C++ member), 838
GFXStateBlockData::zDefined (C++ member), 838
GFXStateBlockData::zEnable (C++ member), 838
GFXStateBlockData::zFunc (C++ member), 838
GFXStateBlockData::zSlopeBias (C++ member), 838
GFXStateBlockData::zWriteEnable (C++ member), 838
gotoWebPage (C++ function), 777
GroundCover::billboardUVs (C++ member), 705
GroundCover::clumpExponent (C++ member), 705
GroundCover::clumpRadius (C++ member), 705
GroundCover::dissolveRadius (C++ member), 705
GroundCover::gridSize (C++ member), 705
GroundCover::invertLayer (C++ member), 705

GroundCover::layer (C++ member), 705
GroundCover::lockFrustum (C++ member), 705
GroundCover::Material (C++ member), 705
GroundCover::maxBillboardTiltAngle (C++ member), 706
GroundCover::maxClumpCount (C++ member), 706
GroundCover::maxElements (C++ member), 706
GroundCover::maxElevation (C++ member), 706
GroundCover::maxSlope (C++ member), 706
GroundCover::minClumpCount (C++ member), 706
GroundCover::minElevation (C++ member), 706
GroundCover::noBillboards (C++ member), 706
GroundCover::noShapes (C++ member), 706
GroundCover::probability (C++ member), 706
GroundCover::radius (C++ member), 706
GroundCover::reflectScale (C++ member), 706
GroundCover::renderCells (C++ member), 706
GroundCover::renderedBatches (C++ member), 707
GroundCover::renderedBillboards (C++ member), 707
GroundCover::renderedCells (C++ member), 707
GroundCover::renderedShapes (C++ member), 707
GroundCover::seed (C++ member), 706
GroundCover::shapeCullRadius (C++ member), 706
GroundCover::shapeFilename (C++ member), 706
GroundCover::shapesCastShadows (C++ member), 706
GroundCover::sizeExponent (C++ member), 706
GroundCover::sizeMax (C++ member), 706
GroundCover::sizeMin (C++ member), 706
GroundCover::windDirection (C++ member), 706
GroundCover::windGustFrequency (C++ member), 707
GroundCover::windGustLength (C++ member), 707
GroundCover::windGustStrength (C++ member), 707
GroundCover::windScale (C++ member), 707
GroundCover::windTurbulenceFrequency (C++ member), 707
GroundCover::windTurbulenceStrength (C++ member), 707
GroundCover::zOffset (C++ member), 707
GroundPlane::Material (C++ member), 694
GroundPlane::postApply (C++ function), 694
GroundPlane::scaleU (C++ member), 694
GroundPlane::scaleV (C++ member), 694
GroundPlane::squareSize (C++ member), 694
GuiAutoScrollCtrl::childBorder (C++ member), 451
GuiAutoScrollCtrl::isLooping (C++ member), 451
GuiAutoScrollCtrl::onComplete (C++ function), 451
GuiAutoScrollCtrl::onReset (C++ function), 451
GuiAutoScrollCtrl::onStart (C++ function), 451
GuiAutoScrollCtrl::onTick (C++ function), 451
GuiAutoScrollCtrl::reset (C++ function), 451
GuiAutoScrollCtrl::resetDelay (C++ member), 451
GuiAutoScrollCtrl::scrollDirection (C++ member), 451
GuiAutoScrollCtrl::scrollOutOfSight (C++ member), 451
GuiAutoScrollCtrl::scrollSpeed (C++ member), 451
GuiAutoScrollCtrl::startDelay (C++ member), 451
GuiBitmapButtonCtrl::autoFitExtents (C++ member), 430
GuiBitmapButtonCtrl::bitmap (C++ member), 430
GuiBitmapButtonCtrl::bitmapMode (C++ member), 430
GuiBitmapButtonCtrl::onAltClick (C++ function), 430
GuiBitmapButtonCtrl::onCtrlClick (C++ function), 430
GuiBitmapButtonCtrl::onDefaultClick (C++ function), 430
GuiBitmapButtonCtrl::onShiftClick (C++ function), 430
GuiBitmapButtonCtrl::setBitmap (C++ function), 430
GuiBitmapButtonCtrl::useModifiers (C++ member), 430
GuiBitmapButtonCtrl::useStates (C++ member), 430
GuiBitmapCtrl::bitmap (C++ member), 435
GuiBitmapCtrl::setBitmap (C++ function), 435
GuiBitmapCtrl::setValue (C++ function), 435
GuiBitmapCtrl::wrap (C++ member), 435
GuiButtonBaseCtrl::buttonType (C++ member), 432
GuiButtonBaseCtrl::getText (C++ function), 431
GuiButtonBaseCtrl::groupNum (C++ member), 432
GuiButtonBaseCtrl::onClick (C++ function), 431
GuiButtonBaseCtrl::onDoubleClick (C++ function), 431
GuiButtonBaseCtrl::onMouseDown (C++ function), 431
GuiButtonBaseCtrl::onMouseDragged (C++ function), 431
GuiButtonBaseCtrl::onMouseEnter (C++ function), 431
GuiButtonBaseCtrl::onMouseLeave (C++ function), 431
GuiButtonBaseCtrl::onMouseUp (C++ function), 431
GuiButtonBaseCtrl::onRightClick (C++ function), 431
GuiButtonBaseCtrl::performClick (C++ function), 431
GuiButtonBaseCtrl::resetState (C++ function), 432
GuiButtonBaseCtrl::setStateOn (C++ function), 432
GuiButtonBaseCtrl::setText (C++ function), 432
GuiButtonBaseCtrl::setTextID (C++ function), 432
GuiButtonBaseCtrl::text (C++ member), 432
GuiButtonBaseCtrl::textID (C++ member), 432
GuiButtonBaseCtrl::useMouseEvents (C++ member), 432
GuiCanvas::alwaysHandleMouseButtons (C++ member), 391
GuiCanvas::clientToScreen (C++ function), 385
GuiCanvas::cursorOff (C++ function), 385
GuiCanvas::cursorOn (C++ function), 386
GuiCanvas::findFirstMatchingMonitor (C++ function), 386
GuiCanvas::getContent (C++ function), 386
GuiCanvas::getCursorPos (C++ function), 386
GuiCanvas::getExtent (C++ function), 386
GuiCanvas::getMode (C++ function), 386
GuiCanvas::getModeCount (C++ function), 386
GuiCanvas::getMonitorCount (C++ function), 387
GuiCanvas::getMonitorName (C++ function), 387
GuiCanvas::getMonitorRect (C++ function), 387
GuiCanvas::getMouseControl (C++ function), 387

GuiCanvas::getVideoMode (C++ function), 387
GuiCanvas::getWindowPosition (C++ function), 387
GuiCanvas::hideCursor (C++ function), 387
GuiCanvas::isCursorOn (C++ function), 387
GuiCanvas::isCursorShown (C++ function), 387
GuiCanvas::isFullscreen (C++ function), 388
GuiCanvas::isMaximized (C++ function), 388
GuiCanvas::isMinimized (C++ function), 388
GuiCanvas::maximizeWindow (C++ function), 388
GuiCanvas::minimizeWindow (C++ function), 388
GuiCanvas::numFences (C++ member), 391
GuiCanvas::popDialog (C++ function), 388
GuiCanvas::popLayer (C++ function), 388
GuiCanvas::pushDialog (C++ function), 388
GuiCanvas::renderFront (C++ function), 389
GuiCanvas::repaint (C++ function), 389
GuiCanvas::reset (C++ function), 389
GuiCanvas::restoreWindow (C++ function), 389
GuiCanvas::screenToClient (C++ function), 389
GuiCanvas::setContent (C++ function), 389
GuiCanvas::setCursor (C++ function), 389
GuiCanvas::setCursorPos (C++ function), 389, 390
GuiCanvas::setFocus (C++ function), 390
GuiCanvas::setVideoMode (C++ function), 390
GuiCanvas::setWindowPosition (C++ function), 390
GuiCanvas::setWindowTitle (C++ function), 390
GuiCanvas::showCursor (C++ function), 390
GuiCanvas::toggleFullscreen (C++ function), 390
GuiCheckBoxCtrl::isStateOn (C++ function), 433
GuiCheckBoxCtrl::setStateOn (C++ function), 433
GuiChunkedBitmapCtrl::bitmap (C++ member), 474
GuiChunkedBitmapCtrl::setBitmap (C++ function), 474
GuiChunkedBitmapCtrl::tile (C++ member), 474
GuiChunkedBitmapCtrl::useVariable (C++ member), 474
GuiClockHud::fillColor (C++ member), 491
GuiClockHud::frameColor (C++ member), 491
GuiClockHud::getTime (C++ function), 491
GuiClockHud::setReverseTime (C++ function), 491
GuiClockHud::setTime (C++ function), 491
GuiClockHud::showFill (C++ member), 491
GuiClockHud::showFrame (C++ member), 491
GuiClockHud::textColor (C++ member), 491
GuiConsole::onMessageSelected (C++ function), 391
GuiConsoleEditCtrl::useSiblingScroller (C++ member), 392
GuiContainer::anchorBottom (C++ member), 451
GuiContainer::anchorLeft (C++ member), 451
GuiContainer::anchorRight (C++ member), 451
GuiContainer::anchorTop (C++ member), 452
GuiContainer::docking (C++ member), 452
GuiContainer::margin (C++ member), 452
GuiContainer::padding (C++ member), 452
GuiControl::accelerator (C++ member), 397
GuiControl::active (C++ member), 397
GuiControl::addGuiControl (C++ function), 393
GuiControl::altCommand (C++ member), 397
GuiControl::clearFirstResponder (C++ function), 393
GuiControl::command (C++ member), 397
GuiControl::controlIsChild (C++ function), 393
GuiControl::extent (C++ member), 397
GuiControl::findHitControl (C++ function), 393
GuiControl::findHitControls (C++ function), 393
GuiControl::getAspect (C++ function), 393
GuiControl::getCenter (C++ function), 393
GuiControl::getExtent (C++ function), 393
GuiControl::getFirstResponder (C++ function), 393
GuiControl::getGlobalCenter (C++ function), 394
GuiControl::getGlobalPosition (C++ function), 394
GuiControl::getMinExtent (C++ function), 394
GuiControl::getParent (C++ function), 394
GuiControl::getPosition (C++ function), 394
GuiControl::getRoot (C++ function), 394
GuiControl::getValue (C++ member), 397
GuiControl::horizSizing (C++ member), 397
GuiControl::hovertime (C++ member), 397
GuiControl::isActive (C++ member), 397
GuiControl::isAwake (C++ function), 394
GuiControl::isContainer (C++ member), 397
GuiControl::isFirstResponder (C++ function), 394
GuiControl::isMouseLocked (C++ function), 394
GuiControl::isVisible (C++ function), 394
GuiControl::langTableMod (C++ member), 397
GuiControl::makeFirstResponder (C++ function), 394
GuiControl::minExtent (C++ member), 397
GuiControl::modal (C++ member), 398
GuiControl::onAction (C++ function), 394
GuiControl::onActive (C++ function), 394
GuiControl::onAdd (C++ function), 395
GuiControl::onControlDragEnter (C++ function), 395
GuiControl::onControlDragExit (C++ function), 395
GuiControl::onControlDragged (C++ function), 395
GuiControl::onControlDropped (C++ function), 395
GuiControl::onDialogPop (C++ function), 395
GuiControl::onDialogPush (C++ function), 395
GuiControl::onGainFirstResponder (C++ function), 395
GuiControl::onLoseFirstResponder (C++ function), 395
GuiControl::onRemove (C++ function), 395
GuiControl::onSleep (C++ function), 396
GuiControl::onVisible (C++ function), 396
GuiControl::onWake (C++ function), 396
GuiControl::pointInControl (C++ function), 396
GuiControl::position (C++ member), 398
GuiControl::profile (C++ member), 398
GuiControl::resize (C++ function), 396
GuiControl::setActive (C++ function), 396
GuiControl::setCenter (C++ function), 396
GuiControl::setExtent (C++ function), 396
GuiControl::setFirstResponder (C++ function), 396

GuiControl::setFirstResponder (C++ member), 398
GuiControl::setPosition (C++ function), 396
GuiControl::setPositionGlobal (C++ function), 397
GuiControl::setProfile (C++ function), 397
GuiControl::setValue (C++ function), 397
GuiControl::setVisible (C++ function), 397
GuiControl::tooltip (C++ member), 398
GuiControl::tooltipProfile (C++ member), 398
GuiControl::variable (C++ member), 398
GuiControl::vertSizing (C++ member), 398
GuiControl::visible (C++ member), 398
GuiControlArrayControl::colCount (C++ member), 452
GuiControlArrayControl::colSizes (C++ member), 452
GuiControlArrayControl::colSpacing (C++ member), 452
GuiControlArrayControl::rowSize (C++ member), 452
GuiControlArrayControl::rowSpacing (C++ member), 452
GuiControlProfile::autoSizeHeight (C++ member), 398
GuiControlProfile::autoSizeWidth (C++ member), 398
GuiControlProfile::bevelColorHL (C++ member), 398
GuiControlProfile::bevelColorLL (C++ member), 398
GuiControlProfile::bitmap (C++ member), 398
GuiControlProfile::border (C++ member), 398
GuiControlProfile::borderColor (C++ member), 398
GuiControlProfile::borderColorHL (C++ member), 398
GuiControlProfile::borderColorNA (C++ member), 398
GuiControlProfile::borderThickness (C++ member), 398
GuiControlProfile::canKeyFocus (C++ member), 399
GuiControlProfile::category (C++ member), 399
GuiControlProfile::cursorColor (C++ member), 399
GuiControlProfile::fillColor (C++ member), 399
GuiControlProfile::fillColorHL (C++ member), 399
GuiControlProfile::fillColorNA (C++ member), 399
GuiControlProfile::fillColorSEL (C++ member), 399
GuiControlProfile::fontCharset (C++ member), 399
GuiControlProfile::fontColor (C++ member), 399
GuiControlProfile::fontColorHL (C++ member), 399
GuiControlProfile::fontColorLink (C++ member), 399
GuiControlProfile::fontColorLinkHL (C++ member), 399
GuiControlProfile::fontColorNA (C++ member), 399
GuiControlProfile::fontColors (C++ member), 399
GuiControlProfile::fontColorSEL (C++ member), 399
GuiControlProfile::fontSize (C++ member), 399
GuiControlProfile::fontType (C++ member), 399
GuiControlProfile::getStringWidth (C++ function), 398
GuiControlProfile::hasBitmapArray (C++ member), 399
GuiControlProfile::justify (C++ member), 399
GuiControlProfile::modal (C++ member), 399
GuiControlProfile::mouseOverSelected (C++ member), 399
GuiControlProfile::numbersOnly (C++ member), 399
GuiControlProfile::opaque (C++ member), 399
GuiControlProfile::profileForChildren (C++ member), 399
GuiControlProfile::returnTab (C++ member), 399
GuiControlProfile::soundButtonDown (C++ member), 400
GuiControlProfile::soundButtonOver (C++ member), 400
GuiControlProfile::tab (C++ member), 400
GuiControlProfile::textOffset (C++ member), 400
GuiCrossHairHud::damageFillColor (C++ member), 492
GuiCrossHairHud::damageFrameColor (C++ member), 492
GuiCrossHairHud::damageOffset (C++ member), 492
GuiCrossHairHud::damageRect (C++ member), 492
GuiCursor::bitmapName (C++ member), 400
GuiCursor::hotSpot (C++ member), 400
GuiCursor::renderOffset (C++ member), 400
GuiDirectoryFileListCtrl::fileFilter (C++ member), 436
GuiDirectoryFileListCtrl::filePath (C++ member), 436
GuiDirectoryFileListCtrl::getSelectedFile (C++ function), 436
GuiDirectoryFileListCtrl::getSelectedFiles (C++ function), 436
GuiDirectoryFileListCtrl::reload (C++ function), 436
GuiDirectoryFileListCtrl::setFilter (C++ function), 436
GuiDirectoryFileListCtrl::setPath (C++ function), 436
GuiDragAndDropControl::deleteOnMouseUp (C++ member), 483
GuiDragAndDropControl::startDragging (C++ function), 483
GuiDynamicCtrlArrayControl::autoCellSize (C++ member), 453
GuiDynamicCtrlArrayControl::colCount (C++ member), 453
GuiDynamicCtrlArrayControl::colSize (C++ member), 453
GuiDynamicCtrlArrayControl::colSpacing (C++ member), 453
GuiDynamicCtrlArrayControl::dynamicSize (C++ member), 453
GuiDynamicCtrlArrayControl::fillRowFirst (C++ member), 453
GuiDynamicCtrlArrayControl::frozen (C++ member), 453
GuiDynamicCtrlArrayControl::padding (C++ member), 453
GuiDynamicCtrlArrayControl::refresh (C++ function), 453
GuiDynamicCtrlArrayControl::rowCount (C++ member), 453
GuiDynamicCtrlArrayControl::rowSize (C++ member), 453
GuiDynamicCtrlArrayControl::rowSpacing (C++ member), 453
GuiFadeinBitmapCtrl::click (C++ function), 401

GuiFadeinBitmapCtrl::done (C++ member), 401
GuiFadeinBitmapCtrl::fadeColor (C++ member), 401
GuiFadeinBitmapCtrl::fadeInEase (C++ member), 401
GuiFadeinBitmapCtrl::fadeInTime (C++ member), 401
GuiFadeinBitmapCtrl::fadeOutEase (C++ member), 401
GuiFadeinBitmapCtrl::fadeOutTime (C++ member), 401
GuiFadeinBitmapCtrl::onDone (C++ function), 401
GuiFadeinBitmapCtrl::waitTime (C++ member), 401
GuiFrameSetCtrl::addColumn (C++ function), 454
GuiFrameSetCtrl::addRow (C++ function), 454
GuiFrameSetCtrl::autoBalance (C++ member), 456
GuiFrameSetCtrl::borderColor (C++ member), 456
GuiFrameSetCtrl::borderEnable (C++ member), 456
GuiFrameSetCtrl::borderMovable (C++ member), 456
GuiFrameSetCtrl::borderWidth (C++ member), 456
GuiFrameSetCtrl::columns (C++ member), 456
GuiFrameSetCtrl::frameBorder (C++ function), 454
GuiFrameSetCtrl::frameMinExtent (C++ function), 454
GuiFrameSetCtrl::frameMovable (C++ function), 454
GuiFrameSetCtrl::framePadding (C++ function), 454
GuiFrameSetCtrl::fudgeFactor (C++ member), 456
GuiFrameSetCtrl::getColumnCount (C++ function), 455
GuiFrameSetCtrl::getColumnOffset (C++ function), 455
GuiFrameSetCtrl::getFramePadding (C++ function), 455
GuiFrameSetCtrl::getRowCount (C++ function), 455
GuiFrameSetCtrl::getRowOffset (C++ function), 455
GuiFrameSetCtrl::removeColumn (C++ function), 455
GuiFrameSetCtrl::removeRow (C++ function), 455
GuiFrameSetCtrl::rows (C++ member), 456
GuiFrameSetCtrl::setColumnOffset (C++ function), 455
GuiFrameSetCtrl::setRowOffset (C++ function), 455
GuiFrameSetCtrl::updateSizes (C++ function), 455
GuiGameListMenuCtrl::activateRow (C++ function), 492
GuiGameListMenuCtrl::addRow (C++ function), 492
GuiGameListMenuCtrl::callbackOnA (C++ member), 493
GuiGameListMenuCtrl::callbackOnB (C++ member), 494
GuiGameListMenuCtrl::callbackOnX (C++ member), 494
GuiGameListMenuCtrl::callbackOnY (C++ member), 494
GuiGameListMenuCtrl::debugRender (C++ member), 494
GuiGameListMenuCtrl::getRowCount (C++ function), 493
GuiGameListMenuCtrl::getRowLabel (C++ function), 493
GuiGameListMenuCtrl::getSelectedRow (C++ function), 493
GuiGameListMenuCtrl::isRowEnabled (C++ function), 493
GuiGameListMenuCtrl::onChange (C++ function), 493
GuiGameListMenuCtrl::setRowEnabled (C++ function), 493
GuiGameListMenuCtrl::setRowLabel (C++ function), 493
GuiGameListMenuCtrl::setSelected (C++ function), 493
GuiGameListMenuProfile::hitAreaLowerRight (C++ member), 494
GuiGameListMenuProfile::hitAreaUpperLeft (C++ member), 494
GuiGameListMenuProfile::iconOffset (C++ member), 494
GuiGameListMenuProfile::rowSize (C++ member), 494
GuiGameListOptionsCtrl::addRow (C++ function), 494
GuiGameListOptionsCtrl::getCurrentOption (C++ function), 495
GuiGameListOptionsCtrl::selectOption (C++ function), 495
GuiGameListOptionsCtrl::setOptions (C++ function), 495
GuiGameListOptionsProfile::columnSplit (C++ member), 496
GuiGameListOptionsProfile::rightPad (C++ member), 496
GuiGraphCtrl::addAutoPlot (C++ function), 477
GuiGraphCtrl::addDatum (C++ function), 477
GuiGraphCtrl::centerY (C++ member), 478
GuiGraphCtrl::getDatum (C++ function), 477
GuiGraphCtrl::matchScale (C++ function), 477
GuiGraphCtrl::plotColor (C++ member), 478
GuiGraphCtrl::plotInterval (C++ member), 478
GuiGraphCtrl::plotType (C++ member), 478
GuiGraphCtrl::plotVariable (C++ member), 478
GuiGraphCtrl::removeAutoPlot (C++ function), 477
GuiGraphCtrl::setGraphType (C++ function), 478
GuiHealthBarHud::damageFillColor (C++ member), 496
GuiHealthBarHud::displayEnergy (C++ member), 496
GuiHealthBarHud::fillColor (C++ member), 496
GuiHealthBarHud::frameColor (C++ member), 496
GuiHealthBarHud::pulseRate (C++ member), 496
GuiHealthBarHud::pulseThreshold (C++ member), 496
GuiHealthBarHud::showFill (C++ member), 496
GuiHealthBarHud::showFrame (C++ member), 496
GuiHealthTextHud::fillColor (C++ member), 497
GuiHealthTextHud::frameColor (C++ member), 497
GuiHealthTextHud::pulseRate (C++ member), 497
GuiHealthTextHud::pulseThreshold (C++ member), 497
GuiHealthTextHud::showEnergy (C++ member), 497
GuiHealthTextHud::showFill (C++ member), 497
GuiHealthTextHud::showFrame (C++ member), 497
GuiHealthTextHud::showTrueValue (C++ member), 497
GuiHealthTextHud::textColor (C++ member), 497
GuiHealthTextHud::warningColor (C++ member), 497
GuiHealthTextHud::warnThreshold (C++ member), 497
GuiIconButtonCtrl::autoSize (C++ member), 402

GuiIconButtonCtrl::buttonMargin (C++ member), 402
GuiIconButtonCtrl::iconBitmap (C++ member), 402
GuiIconButtonCtrl::iconLocation (C++ member), 402
GuiIconButtonCtrl::makeIconSquare (C++ member), 402
GuiIconButtonCtrl::setBitmap (C++ function), 402
GuiIconButtonCtrl::sizeIconToButton (C++ member), 402
GuiIconButtonCtrl::textLocation (C++ member), 402
GuiIconButtonCtrl::textMargin (C++ member), 402
GuiInputCtrl::onInputEvent (C++ function), 484
GuiListBoxCtrl::addFilteredItem (C++ function), 403
GuiListBoxCtrl::allowMultipleSelections (C++ member), 410
GuiListBoxCtrl::clearItemColor (C++ function), 403
GuiListBoxCtrl::clearItems (C++ function), 403
GuiListBoxCtrl::clearSelection (C++ function), 403
GuiListBoxCtrl::colorBullet (C++ member), 410
GuiListBoxCtrl::deleteItem (C++ function), 403
GuiListBoxCtrl::doMirror (C++ function), 404
GuiListBoxCtrl::findItemText (C++ function), 404
GuiListBoxCtrl::fitParentWidth (C++ member), 410
GuiListBoxCtrl::getItemCount (C++ function), 404
GuiListBoxCtrl::getItemObject (C++ function), 404
GuiListBoxCtrl::getItemText (C++ function), 404
GuiListBoxCtrl::getLastClickItem (C++ function), 405
GuiListBoxCtrl::getSelCount (C++ function), 405
GuiListBoxCtrl::getSelectedItem (C++ function), 405
GuiListBoxCtrl::getSelectedItems (C++ function), 405
GuiListBoxCtrl::insertItem (C++ function), 405
GuiListBoxCtrl::isObjectMirrored (C++ function), 406
GuiListBoxCtrl::makeNameCallback (C++ member), 410
GuiListBoxCtrl::mirrorSet (C++ member), 410
GuiListBoxCtrl::onClearSelection (C++ function), 406
GuiListBoxCtrl::onDeleteKey (C++ function), 406
GuiListBoxCtrl::onDoubleClick (C++ function), 406
GuiListBoxCtrl::onMouseDragged (C++ function), 406
GuiListBoxCtrl::onMouseUp (C++ function), 407
GuiListBoxCtrl::onSelect (C++ function), 407
GuiListBoxCtrl::onUnselect (C++ function), 407
GuiListBoxCtrl::removeFilteredItem (C++ function), 407
GuiListBoxCtrl::setCurSel (C++ function), 408
GuiListBoxCtrl::setCurSelRange (C++ function), 408
GuiListBoxCtrl::setItemColor (C++ function), 408
GuiListBoxCtrl::setItemText (C++ function), 408
GuiListBoxCtrl::setItemTooltip (C++ function), 409
GuiListBoxCtrl::setMultipleSelection (C++ function), 409
GuiListBoxCtrl::setSelected (C++ function), 409
GuiMenuBar::addMenu (C++ function), 410
GuiMenuBar::addMenuItem (C++ function), 411
GuiMenuBar::addSubmenuItem (C++ function), 411
GuiMenuBar::clearMenuItems (C++ function), 412
GuiMenuBar::clearMenus (C++ function), 412
GuiMenuBar::clearSubmenuItems (C++ function), 412
GuiMenuBar::onMenuItemSelect (C++ function), 412
GuiMenuBar::onMenuSelect (C++ function), 413
GuiMenuBar::onMouseInMenu (C++ function), 413
GuiMenuBar::onSubmenuSelect (C++ function), 413
GuiMenuBar::padding (C++ member), 418
GuiMenuBar::removeMenu (C++ function), 413
GuiMenuBar::removeMenuItem (C++ function), 414
GuiMenuBar::setCheckmarkBitmapIndex (C++ function), 414
GuiMenuBar::setMenuBitmapIndex (C++ function), 414
GuiMenuBar::setMenuItemBitmap (C++ function), 415
GuiMenuBar::setMenuItemChecked (C++ function), 415
GuiMenuBar::setMenuItemEnable (C++ function), 415
GuiMenuBar::setMenuItemSubmenuState (C++ function), 416
GuiMenuBar::setMenuItemText (C++ function), 416
GuiMenuBar::setMenuItemVisible (C++ function), 416
GuiMenuBar::setMenuMargins (C++ function), 417
GuiMenuBar::setMenuText (C++ function), 417
GuiMenuBar::setMenuVisible (C++ function), 418
GuiMenuBar::setSubmenuItemChecked (C++ function), 418
GuiMessageVectorCtrl::allowedMatches (C++ member), 485
GuiMessageVectorCtrl::attach (C++ function), 484
GuiMessageVectorCtrl::detach (C++ function), 485
GuiMessageVectorCtrl::lineContinuedIndex (C++ member), 485
GuiMessageVectorCtrl::lineSpacing (C++ member), 485
GuiMessageVectorCtrl::matchColor (C++ member), 485
GuiMessageVectorCtrl::maxColorIndex (C++ member), 485
GuiMLTextCtrl::addText (C++ function), 419
GuiMLTextCtrl::allowColorChars (C++ member), 421
GuiMLTextCtrl::deniedSound (C++ member), 421
GuiMLTextCtrl::forceReflow (C++ function), 419
GuiMLTextCtrl::getText (C++ function), 419
GuiMLTextCtrl::lineSpacing (C++ member), 421
GuiMLTextCtrl::maxChars (C++ member), 421
GuiMLTextCtrl::onResize (C++ function), 420
GuiMLTextCtrl::onURL (C++ function), 420
GuiMLTextCtrl::scrollToBottom (C++ function), 420
GuiMLTextCtrl::scrollToTag (C++ function), 420
GuiMLTextCtrl::scrollToTop (C++ function), 420
GuiMLTextCtrl::setAlpha (C++ function), 420
GuiMLTextCtrl::setCursorPosition (C++ function), 421
GuiMLTextCtrl::setText (C++ function), 421
GuiMLTextCtrl::text (C++ member), 421
GuiMLTextCtrl::useURLMouseCursor (C++ member), 421
GuiMLTextEditCtrl::escapeCommand (C++ member), 437
GuiMouseEventCtrl::lockMouse (C++ member), 425

GuiMouseEventCtrl::onMouseDown (C++ function), 422
GuiMouseEventCtrl::onMouseDragged (C++ function), 422
GuiMouseEventCtrl::onMouseEnter (C++ function), 422
GuiMouseEventCtrl::onMouseLeave (C++ function), 423
GuiMouseEventCtrl::onMouseMove (C++ function), 423
GuiMouseEventCtrl::onMouseUp (C++ function), 423
GuiMouseEventCtrl::onRightMouseDown (C++ function), 424
GuiMouseEventCtrl::onRightMouseDragged (C++ function), 424
GuiMouseEventCtrl::onRightMouseUp (C++ function), 424
GuiObjectView::animSequence (C++ member), 383
GuiObjectView::cameraRotation (C++ member), 383
GuiObjectView::cameraSpeed (C++ member), 383
GuiObjectView::getCameraSpeed (C++ function), 379
GuiObjectView::getModel (C++ function), 379
GuiObjectView::getMountedModel (C++ function), 380
GuiObjectView::getMountSkin (C++ function), 380
GuiObjectView::getOrbitDistance (C++ function), 380
GuiObjectView::getSkin (C++ function), 380
GuiObjectView::lightAmbient (C++ member), 383
GuiObjectView::lightColor (C++ member), 383
GuiObjectView::lightDirection (C++ member), 383
GuiObjectView::maxOrbitDiststance (C++ member), 383
GuiObjectView::minOrbitDiststance (C++ member), 383
GuiObjectView::mountedNode (C++ member), 383
GuiObjectView::mountedShapeFile (C++ member), 383
GuiObjectView::mountedSkin (C++ member), 384
GuiObjectView::onMouseEnter (C++ function), 380
GuiObjectView::onMouseLeave (C++ function), 380
GuiObjectView::orbitDiststance (C++ member), 384
GuiObjectView::setCameraSpeed (C++ function), 381
GuiObjectView::setLightAmbient (C++ function), 381
GuiObjectView::setLightColor (C++ function), 381
GuiObjectView::setLightDirection (C++ function), 381
GuiObjectView::setModel (C++ function), 381
GuiObjectView::setMount (C++ function), 382
GuiObjectView::setMountedModel (C++ function), 382
GuiObjectView::setMountSkin (C++ function), 382
GuiObjectView::setOrbitDistance (C++ function), 382
GuiObjectView::setSeq (C++ function), 383
GuiObjectView::setSkin (C++ function), 383
GuiObjectView::shapeFile (C++ member), 384
GuiObjectView::skin (C++ member), 384
GuiPaneControl::barBehindText (C++ member), 456
GuiPaneControl::caption (C++ member), 456
GuiPaneControl::captionID (C++ member), 457
GuiPaneControl::collapsable (C++ member), 457
GuiPaneControl::setCollapsed (C++ function), 456
GuiPopUpMenuCtrl::add (C++ function), 437
GuiPopUpMenuCtrl::addScheme (C++ function), 437
GuiPopUpMenuCtrl::bitmap (C++ member), 438
GuiPopUpMenuCtrl::bitmapBounds (C++ member), 438
GuiPopUpMenuCtrl::changeTextById (C++ function), 437
GuiPopUpMenuCtrl::clear (C++ member), 438
GuiPopUpMenuCtrl::clearEntry (C++ function), 437
GuiPopUpMenuCtrl::findText (C++ function), 437
GuiPopUpMenuCtrl::forceClose (C++ member), 438
GuiPopUpMenuCtrl::forceOnAction (C++ member), 438
GuiPopUpMenuCtrl::getSelected (C++ member), 438
GuiPopUpMenuCtrl::getText (C++ member), 438
GuiPopUpMenuCtrl::getTextById (C++ function), 437
GuiPopUpMenuCtrl::maxPopupHeight (C++ member), 438
GuiPopUpMenuCtrl::replaceText (C++ function), 437
GuiPopUpMenuCtrl::reverseTextList (C++ member), 438
GuiPopUpMenuCtrl::sbUsesNAColor (C++ member), 438
GuiPopUpMenuCtrl::setEnumContent (C++ function), 437
GuiPopUpMenuCtrl::setFirstSelected (C++ function), 438
GuiPopUpMenuCtrl::setNoneSelected (C++ member), 438
GuiPopUpMenuCtrl::setSelected (C++ function), 438
GuiPopUpMenuCtrl::size (C++ member), 438
GuiPopUpMenuCtrl::sort (C++ member), 438
GuiPopUpMenuCtrl::sortID (C++ member), 438
GuiPopUpMenuCtrlEx::add (C++ function), 439
GuiPopUpMenuCtrlEx::addCategory (C++ function), 439
GuiPopUpMenuCtrlEx::addScheme (C++ function), 439
GuiPopUpMenuCtrlEx::bitmap (C++ member), 440
GuiPopUpMenuCtrlEx::bitmapBounds (C++ member), 440
GuiPopUpMenuCtrlEx::clear (C++ function), 439
GuiPopUpMenuCtrlEx::clearEntry (C++ function), 439
GuiPopUpMenuCtrlEx::findText (C++ function), 439
GuiPopUpMenuCtrlEx::forceClose (C++ function), 439
GuiPopUpMenuCtrlEx::forceOnAction (C++ function), 439
GuiPopUpMenuCtrlEx::getColorById (C++ member), 440
GuiPopUpMenuCtrlEx::getSelected (C++ function), 439
GuiPopUpMenuCtrlEx::getText (C++ function), 439
GuiPopUpMenuCtrlEx::getTextById (C++ function), 440
GuiPopUpMenuCtrlEx::hotTrackCallback (C++ member), 440
GuiPopUpMenuCtrlEx::maxPopupHeight (C++ member), 440
GuiPopUpMenuCtrlEx::replaceText (C++ member), 440
GuiPopUpMenuCtrlEx::reverseTextList (C++ member), 440
GuiPopUpMenuCtrlEx::sbUsesNAColor (C++ member), 441

GuiPopUpMenuCtrlEx::setEnumContent (C++ member), 441
GuiPopUpMenuCtrlEx::setNoneSelected (C++ function), 440
GuiPopUpMenuCtrlEx::setSelected (C++ function), 440
GuiPopUpMenuCtrlEx::setText (C++ function), 440
GuiPopUpMenuCtrlEx::size (C++ member), 441
GuiPopUpMenuCtrlEx::sort (C++ function), 440
GuiPopUpMenuCtrlEx::sortID (C++ function), 440
GuiProgressBitmapCtrl::bitmap (C++ member), 479
GuiProgressBitmapCtrl::setBitmap (C++ function), 479
GuiRolloutCtrl::autoCollapseSiblings (C++ member), 458
GuiRolloutCtrl::caption (C++ member), 458
GuiRolloutCtrl::clickCollapse (C++ member), 458
GuiRolloutCtrl::collapse (C++ function), 457
GuiRolloutCtrl::defaultHeight (C++ member), 458
GuiRolloutCtrl::expand (C++ function), 457
GuiRolloutCtrl::expanded (C++ member), 458
GuiRolloutCtrl::hideHeader (C++ member), 458
GuiRolloutCtrl::instantCollapse (C++ function), 457
GuiRolloutCtrl::instantExpand (C++ function), 458
GuiRolloutCtrl::isExpanded (C++ function), 458
GuiRolloutCtrl::margin (C++ member), 458
GuiRolloutCtrl::onCollapsed (C++ function), 458
GuiRolloutCtrl::onExpanded (C++ function), 458
GuiRolloutCtrl::onHeaderRightClick (C++ function), 458
GuiRolloutCtrl::sizeToContents (C++ function), 458
GuiRolloutCtrl::toggleCollapse (C++ function), 458
GuiRolloutCtrl::toggleExpanded (C++ function), 458
GuiScriptNotifyCtrl::onChildAdded (C++ function), 485
GuiScriptNotifyCtrl::onChildAdded (C++ member), 486
GuiScriptNotifyCtrl::onChildRemoved (C++ function), 486
GuiScriptNotifyCtrl::onChildRemoved (C++ member), 486
GuiScriptNotifyCtrl::onChildResized (C++ function), 486
GuiScriptNotifyCtrl::onChildResized (C++ member), 486
GuiScriptNotifyCtrl::onGainFirstResponder (C++ function), 486
GuiScriptNotifyCtrl::onGainFirstResponder (C++ member), 486
GuiScriptNotifyCtrl::onLoseFirstResponder (C++ function), 486
GuiScriptNotifyCtrl::onLoseFirstResponder (C++ member), 486
GuiScriptNotifyCtrl::onParentResized (C++ function), 486
GuiScriptNotifyCtrl::onParentResized (C++ member), 486
GuiScriptNotifyCtrl::onResize (C++ function), 486
GuiScriptNotifyCtrl::onResize (C++ member), 486
GuiScrollCtrl::childMargin (C++ member), 459
GuiScrollCtrl::computeSizes (C++ function), 459
GuiScrollCtrl::constantThumbHeight (C++ member), 459
GuiScrollCtrl::getScrollPosition (C++ function), 459
GuiScrollCtrl::getScrollPositionX (C++ function), 459
GuiScrollCtrl::getScrollPositionY (C++ function), 459
GuiScrollCtrl::hScrollBar (C++ member), 459
GuiScrollCtrl::lockHorizScroll (C++ member), 459
GuiScrollCtrl::lockVertScroll (C++ member), 459
GuiScrollCtrl::mouseWheelScrollSpeed (C++ member), 460
GuiScrollCtrl::onScroll (C++ function), 459
GuiScrollCtrl::scrollToBottom (C++ function), 459
GuiScrollCtrl::scrollToObject (C++ function), 459
GuiScrollCtrl::scrollToTop (C++ function), 459
GuiScrollCtrl::setScrollPosition (C++ function), 459
GuiScrollCtrl::vScrollBar (C++ member), 460
GuiScrollCtrl::willFirstRespond (C++ member), 460
GuiSeparatorCtrl::borderMargin (C++ member), 441
GuiSeparatorCtrl::caption (C++ member), 441
GuiSeparatorCtrl::invisible (C++ member), 441
GuiSeparatorCtrl::leftMargin (C++ member), 441
GuiSeparatorCtrl::type (C++ member), 441
GuiShapeNameHud::distanceFade (C++ member), 498
GuiShapeNameHud::fillColor (C++ member), 498
GuiShapeNameHud::frameColor (C++ member), 498
GuiShapeNameHud::labelFillColor (C++ member), 498
GuiShapeNameHud::labelFrameColor (C++ member), 498
GuiShapeNameHud::labelPadding (C++ member), 498
GuiShapeNameHud::showFill (C++ member), 498
GuiShapeNameHud::showFrame (C++ member), 498
GuiShapeNameHud::showLabelFill (C++ member), 498
GuiShapeNameHud::showLabelFrame (C++ member), 498
GuiShapeNameHud::textColor (C++ member), 498
GuiShapeNameHud::verticalOffset (C++ member), 498
GuiSliderCtrl::getValue (C++ function), 480
GuiSliderCtrl::isThumbBeingDragged (C++ function), 480
GuiSliderCtrl::onMouseDragged (C++ function), 480
GuiSliderCtrl::range (C++ member), 481
GuiSliderCtrl::setValue (C++ function), 481
GuiSliderCtrl::snap (C++ member), 481
GuiSliderCtrl::ticks (C++ member), 481
GuiSliderCtrl::value (C++ member), 481
GuiSpeedometerHud::center (C++ member), 460
GuiSpeedometerHud::color (C++ member), 460
GuiSpeedometerHud::length (C++ member), 460
GuiSpeedometerHud::maxAngle (C++ member), 460
GuiSpeedometerHud::maxSpeed (C++ member), 460
GuiSpeedometerHud::minAngle (C++ member), 460

GuiSpeedometerHud::tail (C++ member), 460
GuiSpeedometerHud::width (C++ member), 460
GuiSplitContainer::fixedPanel (C++ member), 461
GuiSplitContainer::fixedSize (C++ member), 461
GuiSplitContainer::orientation (C++ member), 461
GuiSplitContainer::splitPoint (C++ member), 461
GuiSplitContainer::splitterSize (C++ member), 461
GuiStackControl::changeChildPosition (C++ member), 462
GuiStackControl::changeChildSizeToFit (C++ member), 462
GuiStackControl::dynamicNonStackExtent (C++ member), 462
GuiStackControl::dynamicPos (C++ member), 462
GuiStackControl::dynamicSize (C++ member), 463
GuiStackControl::freeze (C++ function), 462
GuiStackControl::horizStacking (C++ member), 463
GuiStackControl::isFrozen (C++ function), 462
GuiStackControl::padding (C++ member), 463
GuiStackControl::stackingType (C++ member), 463
GuiStackControl::updateStack (C++ function), 462
GuiStackControl::vertStacking (C++ member), 463
GuiSwatchButtonCtrl::color (C++ member), 434
GuiSwatchButtonCtrl::gridBitmap (C++ member), 434
GuiSwatchButtonCtrl::setColor (C++ function), 434
GuiTabBookCtrl::addPage (C++ function), 463
GuiTabBookCtrl::allowReorder (C++ member), 464
GuiTabBookCtrl::defaultPage (C++ member), 464
GuiTabBookCtrl::frontTabPadding (C++ member), 464
GuiTabBookCtrl::getSelectedPage (C++ function), 463
GuiTabBookCtrl::minTabWidth (C++ member), 464
GuiTabBookCtrl::onTabRightClick (C++ function), 463
GuiTabBookCtrl::onTabSelected (C++ function), 463
GuiTabBookCtrl::selectedPage (C++ member), 464
GuiTabBookCtrl::selectPage (C++ function), 463
GuiTabBookCtrl::tabHeight (C++ member), 464
GuiTabBookCtrl::tabMargin (C++ member), 464
GuiTabBookCtrl::tabPosition (C++ member), 464
GuiTabPageCtrl::fitBook (C++ member), 464
GuiTabPageCtrl::select (C++ function), 464
GuiTextCtrl::maxLength (C++ member), 425
GuiTextCtrl::setText (C++ function), 425
GuiTextCtrl::setTextID (C++ function), 425
GuiTextCtrl::text (C++ member), 425
GuiTextCtrl::textID (C++ member), 426
GuiTextEditCtrl::clearSelectedText (C++ function), 442
GuiTextEditCtrl::deniedSound (C++ member), 444
GuiTextEditCtrl::escapeCommand (C++ member), 444
GuiTextEditCtrl::forceValidateText (C++ function), 442
GuiTextEditCtrl::getCursorPos (C++ function), 442
GuiTextEditCtrl::getText (C++ function), 442
GuiTextEditCtrl::historySize (C++ member), 444
GuiTextEditCtrl::isAllTextSelected (C++ function), 442
GuiTextEditCtrl::onReturn (C++ function), 443
GuiTextEditCtrl::onTabComplete (C++ function), 443
GuiTextEditCtrl::onValidate (C++ function), 443
GuiTextEditCtrl::password (C++ member), 444
GuiTextEditCtrl::passwordMask (C++ member), 444
GuiTextEditCtrl::selectAllText (C++ function), 443
GuiTextEditCtrl::setCursorPos (C++ function), 443
GuiTextEditCtrl::setText (C++ function), 443
GuiTextEditCtrl::sinkAllKeyEvents (C++ member), 444
GuiTextEditCtrl::tabComplete (C++ member), 444
GuiTextEditCtrl::validate (C++ member), 444
GuiTextEditSliderBitmapCtrl::bitmap (C++ member), 426
GuiTextEditSliderBitmapCtrl::focusOnMouseWheel (C++ member), 426
GuiTextEditSliderBitmapCtrl::format (C++ member), 426
GuiTextEditSliderBitmapCtrl::increment (C++ member), 426
GuiTextEditSliderBitmapCtrl::range (C++ member), 426
GuiTextEditSliderCtrl::focusOnMouseWheel (C++ member), 427
GuiTextEditSliderCtrl::format (C++ member), 427
GuiTextEditSliderCtrl::increment (C++ member), 427
GuiTextEditSliderCtrl::range (C++ member), 427
GuiTextListCtrl::addRow (C++ function), 444
GuiTextListCtrl::clear (C++ function), 445
GuiTextListCtrl::clearSelection (C++ function), 445
GuiTextListCtrl::clipColumnText (C++ member), 449
GuiTextListCtrl::columns (C++ member), 450
GuiTextListCtrl::findTextIndex (C++ function), 445
GuiTextListCtrl::fitParentWidth (C++ member), 450
GuiTextListCtrl::getRowId (C++ function), 445
GuiTextListCtrl::getRowNumById (C++ function), 445
GuiTextListCtrl::getRowText (C++ function), 446
GuiTextListCtrl::getRowTextById (C++ function), 446
GuiTextListCtrl::getSelectedId (C++ function), 446
GuiTextListCtrl::getSelectedRow (C++ function), 446
GuiTextListCtrl::isRowActive (C++ function), 446
GuiTextListCtrl::onDeleteKey (C++ function), 447
GuiTextListCtrl::onSelect (C++ function), 447
GuiTextListCtrl::removeRow (C++ function), 447
GuiTextListCtrl::removeRowById (C++ function), 447
GuiTextListCtrl::rowCount (C++ function), 447
GuiTextListCtrl::scrollVisible (C++ function), 448
GuiTextListCtrl::setRowActive (C++ function), 448
GuiTextListCtrl::setRowById (C++ function), 448
GuiTextListCtrl::setSelectedById (C++ function), 448
GuiTextListCtrl::setSelectedRow (C++ function), 449
GuiTextListCtrl::sort (C++ function), 449
GuiTextListCtrl::sortNumerical (C++ function), 449
GuiTheoraCtrl::backgroundColor (C++ member), 475
GuiTheoraCtrl::getCurrentTime (C++ function), 475
GuiTheoraCtrl::isPlaybackDone (C++ function), 475
GuiTheoraCtrl::matchVideoSize (C++ member), 475

GuiTheoraCtrl::pause (C++ function), 475
GuiTheoraCtrl::play (C++ function), 475
GuiTheoraCtrl::playOnWake (C++ member), 475
GuiTheoraCtrl::renderDebugInfo (C++ member), 475
GuiTheoraCtrl::setFile (C++ function), 475
GuiTheoraCtrl::stop (C++ function), 475
GuiTheoraCtrl::stopOnSleep (C++ member), 475
GuiTheoraCtrl::theoraFile (C++ member), 475
GuiTheoraCtrl::transcoder (C++ member), 475
GuiTickCtrl::setProcessTicks (C++ function), 487
GuiTreeViewCtrl::addChildSelectionByValue (C++ member), 469
GuiTreeViewCtrl::addSelection (C++ function), 465
GuiTreeViewCtrl::buildIconTable (C++ function), 465
GuiTreeViewCtrl::buildVisibleTree (C++ member), 469
GuiTreeViewCtrl::cancelRename (C++ member), 469
GuiTreeViewCtrl::canRenameObject (C++ function), 465
GuiTreeViewCtrl::canRenameObjects (C++ member), 469
GuiTreeViewCtrl::clear (C++ function), 465
GuiTreeViewCtrl::clearAllOnSingleSelection (C++ member), 469
GuiTreeViewCtrl::clearFilterText (C++ function), 465
GuiTreeViewCtrl::clearSelection (C++ function), 465
GuiTreeViewCtrl::compareToObjectID (C++ member), 469
GuiTreeViewCtrl::deleteObjectAllowed (C++ member), 469
GuiTreeViewCtrl::deleteSelection (C++ function), 465
GuiTreeViewCtrl::destroyTreeOnSleep (C++ member), 469
GuiTreeViewCtrl::dragToItemAllowed (C++ member), 469
GuiTreeViewCtrl::editItem (C++ function), 466
GuiTreeViewCtrl::expandItem (C++ function), 466
GuiTreeViewCtrl::findChildItemByName (C++ function), 466
GuiTreeViewCtrl::findItemByName (C++ function), 466
GuiTreeViewCtrl::findItemByObjectId (C++ function), 466
GuiTreeViewCtrl::findItemByValue (C++ function), 466
GuiTreeViewCtrl::fullRowSelect (C++ member), 469
GuiTreeViewCtrl::getChild (C++ function), 466
GuiTreeViewCtrl::getFilterText (C++ function), 466
GuiTreeViewCtrl::getFirstRootItem (C++ member), 469
GuiTreeViewCtrl::getItemCount (C++ member), 469
GuiTreeViewCtrl::getItemText (C++ function), 466
GuiTreeViewCtrl::getItemValue (C++ function), 466
GuiTreeViewCtrl::getNextSibling (C++ function), 466
GuiTreeViewCtrl::getParent (C++ function), 466
GuiTreeViewCtrl::getPrevSibling (C++ function), 466
GuiTreeViewCtrl::getSelectedItem (C++ function), 466
GuiTreeViewCtrl::getSelectedItemList (C++ member), 469
GuiTreeViewCtrl::getSelectedItemsCount (C++ member), 469
GuiTreeViewCtrl::getSelectedObject (C++ function), 466
GuiTreeViewCtrl::getSelectedObjectList (C++ member), 469
GuiTreeViewCtrl::getTextToRoot (C++ function), 466
GuiTreeViewCtrl::handleRenameObject (C++ function), 466
GuiTreeViewCtrl::hideSelection (C++ function), 466
GuiTreeViewCtrl::insertItem (C++ function), 467
GuiTreeViewCtrl::isItemSelected (C++ function), 467
GuiTreeViewCtrl::isParentItem (C++ function), 467
GuiTreeViewCtrl::isValidDragTarget (C++ function), 467
GuiTreeViewCtrl::itemHeight (C++ member), 469
GuiTreeViewCtrl::lockSelection (C++ function), 467
GuiTreeViewCtrl::markItem (C++ function), 467
GuiTreeViewCtrl::mouseDragging (C++ member), 469
GuiTreeViewCtrl::moveItemDown (C++ function), 467
GuiTreeViewCtrl::moveItemUp (C++ function), 467
GuiTreeViewCtrl::multipleSelections (C++ member), 469
GuiTreeViewCtrl::onAddGroupSelected (C++ function), 467
GuiTreeViewCtrl::onAddMultipleSelectionBegin (C++ function), 467
GuiTreeViewCtrl::onAddMultipleSelectionEnd (C++ function), 467
GuiTreeViewCtrl::onAddSelection (C++ function), 467
GuiTreeViewCtrl::onBeginReparenting (C++ function), 467
GuiTreeViewCtrl::onClearSelection (C++ function), 467
GuiTreeViewCtrl::onDefineIcons (C++ function), 467
GuiTreeViewCtrl::onDeleteObject (C++ function), 467
GuiTreeViewCtrl::onDeleteSelection (C++ function), 467
GuiTreeViewCtrl::onDragDropped (C++ function), 467
GuiTreeViewCtrl::onEndReparenting (C++ function), 467
GuiTreeViewCtrl::onInspect (C++ function), 468
GuiTreeViewCtrl::onKeyDown (C++ function), 468
GuiTreeViewCtrl::onMouseDragged (C++ function), 468
GuiTreeViewCtrl::onMouseUp (C++ function), 468
GuiTreeViewCtrl::onObjectDeleteCompleted (C++ function), 468
GuiTreeViewCtrl::onRemoveSelection (C++ function), 468
GuiTreeViewCtrl::onRenameValidate (C++ member), 469
GuiTreeViewCtrl::onReparent (C++ function), 468
GuiTreeViewCtrl::onRightMouseDown (C++ function), 468
GuiTreeViewCtrl::onRightMouseUp (C++ function), 468
GuiTreeViewCtrl::onSelect (C++ function), 468
GuiTreeViewCtrl::onUnselect (C++ function), 468
GuiTreeViewCtrl::open (C++ function), 468

GuiTreeViewCtrl::removeAllChildren (C++ member), 469
GuiTreeViewCtrl::removeChildSelectionByValue (C++ member), 469
GuiTreeViewCtrl::removeItem (C++ function), 468
GuiTreeViewCtrl::removeSelection (C++ function), 468
GuiTreeViewCtrl::renameInternal (C++ member), 469
GuiTreeViewCtrl::scrollVisible (C++ function), 468
GuiTreeViewCtrl::scrollVisibleByObjectId (C++ function), 468
GuiTreeViewCtrl::selectItem (C++ function), 468
GuiTreeViewCtrl::setDebug (C++ function), 468
GuiTreeViewCtrl::setFilterText (C++ function), 468
GuiTreeViewCtrl::setItemImages (C++ function), 468
GuiTreeViewCtrl::setItemTooltip (C++ function), 468
GuiTreeViewCtrl::showClassNameForUnnamedObjects (C++ member), 469
GuiTreeViewCtrl::showClassNames (C++ member), 469
GuiTreeViewCtrl::showInternalNames (C++ member), 469
GuiTreeViewCtrl::showItemRenameCtrl (C++ function), 468
GuiTreeViewCtrl::showObjectIds (C++ member), 469
GuiTreeViewCtrl::showObjectNames (C++ member), 470
GuiTreeViewCtrl::showRoot (C++ member), 470
GuiTreeViewCtrl::sort (C++ function), 468
GuiTreeViewCtrl::tabSize (C++ member), 470
GuiTreeViewCtrl::textOffset (C++ member), 470
GuiTreeViewCtrl::toggleHideSelection (C++ function), 468
GuiTreeViewCtrl::toggleLockSelection (C++ function), 468
GuiTreeViewCtrl::tooltipOnWidthOnly (C++ member), 470
GuiTreeViewCtrl::useInspectorTooltips (C++ member), 470
GuiTSCtrl::calculateViewDistance (C++ function), 384
GuiTSCtrl::cameraZRot (C++ member), 384
GuiTSCtrl::forceFOV (C++ member), 384
GuiTSCtrl::getWorldToScreenScale (C++ function), 384
GuiTSCtrl::project (C++ function), 384
GuiTSCtrl::reflectPriority (C++ member), 384
GuiTSCtrl::renderStyle (C++ member), 385
GuiTSCtrl::unproject (C++ function), 384
GuiWindowCtrl::attach (C++ function), 470
GuiWindowCtrl::attachTo (C++ function), 470
GuiWindowCtrl::canClose (C++ member), 471
GuiWindowCtrl::canCollapse (C++ member), 471
GuiWindowCtrl::canMaximize (C++ member), 471
GuiWindowCtrl::canMinimize (C++ member), 471
GuiWindowCtrl::canMove (C++ member), 471
GuiWindowCtrl::closeCommand (C++ member), 471
GuiWindowCtrl::edgeSnap (C++ member), 471
GuiWindowCtrl::onClose (C++ function), 470
GuiWindowCtrl::onCollapse (C++ function), 470
GuiWindowCtrl::onMaximize (C++ function), 470
GuiWindowCtrl::onMinimize (C++ function), 470
GuiWindowCtrl::onRestore (C++ function), 470
GuiWindowCtrl::resizeHeight (C++ member), 471
GuiWindowCtrl::resizeWidth (C++ member), 471
GuiWindowCtrl::selectWindow (C++ function), 470
GuiWindowCtrl::setCollapseGroup (C++ function), 471
GuiWindowCtrl::text (C++ member), 471
GuiWindowCtrl::toggleCollapseGroup (C++ function), 471

H

HoverVehicleData::brakingActivationSpeed (C++ member), 672
HoverVehicleData::brakingForce (C++ member), 672
HoverVehicleData::dragForce (C++ member), 672
HoverVehicleData::dustTrailEmitter (C++ member), 672
HoverVehicleData::dustTrailFreqMod (C++ member), 672
HoverVehicleData::dustTrailOffset (C++ member), 672
HoverVehicleData::engineSound (C++ member), 672
HoverVehicleData::floatingGravMag (C++ member), 672
HoverVehicleData::floatingThrustFactor (C++ member), 672
HoverVehicleData::floatSound (C++ member), 672
HoverVehicleData::forwardJetEmitter (C++ member), 672
HoverVehicleData::gyroDrag (C++ member), 672
HoverVehicleData::jetSound (C++ member), 672
HoverVehicleData::mainThrustForce (C++ member), 672
HoverVehicleData::normalForce (C++ member), 672
HoverVehicleData::pitchForce (C++ member), 672
HoverVehicleData::restorativeForce (C++ member), 673
HoverVehicleData::reverseThrustForce (C++ member), 673
HoverVehicleData::rollForce (C++ member), 673
HoverVehicleData::stabDampingConstant (C++ member), 673
HoverVehicleData::stabLenMax (C++ member), 673
HoverVehicleData::stabLenMin (C++ member), 673
HoverVehicleData::stabSpringConstant (C++ member), 673
HoverVehicleData::steeringForce (C++ member), 673
HoverVehicleData::strafeThrustForce (C++ member), 673
HoverVehicleData::triggerTrailHeight (C++ member), 673
HoverVehicleData::turboFactor (C++ member), 673
HoverVehicleData::vertFactor (C++ member), 673
HTTPObject::get (C++ function), 758
HTTPObject::post (C++ function), 758

I

importCachedFont (C++ function), 829
initContainerRadiusSearch (C++ function), 516
initContainerTypeSearch (C++ function), 516
initDisplayDeviceInfo (C++ function), 827
isalnum (C++ function), 366
isClass (C++ function), 306
isDebugBuild (C++ function), 777
isDefined (C++ function), 325
IsDirectory (C++ function), 352
isFile (C++ function), 352
isFunction (C++ function), 326
isJoystickEnabled (C++ function), 746
isLeapMotionActive (C++ function), 516
isMemberOfClass (C++ function), 306
isMethod (C++ function), 326
isOculusVRDeviceActive (C++ function), 516
isOVRHMDSimulated (C++ function), 516
isPackage (C++ function), 317
isQueueRegistered (C++ function), 316
isRazerHydraActive (C++ function), 516
isRazerHydraControllerDocked (C++ function), 517
isShippingBuild (C++ function), 777
isspace (C++ function), 366
isToolBuild (C++ function), 777
isValidObjectName (C++ function), 306
isWriteableFileName (C++ function), 352
isXInputConnected (C++ function), 746
Item::getLastStickyNormal (C++ function), 535
Item::getLastStickyPos (C++ function), 535
Item::isAtRest (C++ function), 535
Item::isRotating (C++ function), 535
Item::isStatic (C++ function), 535
Item::maxWarpTicks (C++ member), 536
Item::minWarpTicks (C++ member), 536
Item::onEnterLiquid (C++ function), 536
Item::onLeaveLiquid (C++ function), 536
Item::onStickyCollision (C++ function), 536
Item::rotate (C++ member), 536
Item::setCollisionTimeout (C++ function), 536
ItemData::elasticity (C++ member), 537
ItemData::friction (C++ member), 537
ItemData::gravityMod (C++ member), 537
ItemData::lightColor (C++ member), 537
ItemData::lightOnlyStatic (C++ member), 537
ItemData::lightRadius (C++ member), 537
ItemData::lightTime (C++ member), 537
ItemData::lightType (C++ member), 538
ItemData::maxVelocity (C++ member), 538
ItemData::simpleServerCollision (C++ member), 538
ItemData::sticky (C++ member), 538

L

LangTable::addLanguage (C++ function), 779
LangTable::getCurrentLanguage (C++ function), 779
LangTable::getLangName (C++ function), 779
LangTable::getNumLang (C++ function), 779
LangTable::getString (C++ function), 779
LangTable::setCurrentLanguage (C++ function), 779
LangTable::setDefaultLanguage (C++ function), 779
LeapMotionFrame::getFrameInternalId (C++ function), 738
LeapMotionFrame::getFrameRealTime (C++ function), 738
LeapMotionFrame::getFrameSimTime (C++ function), 738
LeapMotionFrame::getHandCount (C++ function), 738
LeapMotionFrame::getHandId (C++ function), 739
LeapMotionFrame::getHandPointablesCount (C++ function), 739
LeapMotionFrame::getHandPos (C++ function), 739
LeapMotionFrame::getHandRawPos (C++ function), 739
LeapMotionFrame::getHandRawTransform (C++ function), 739
LeapMotionFrame::getHandRot (C++ function), 739
LeapMotionFrame::getHandRotAxis (C++ function), 739
LeapMotionFrame::getHandTransform (C++ function), 739
LeapMotionFrame::getHandValid (C++ function), 739
LeapMotionFrame::getPointableHandIndex (C++ function), 740
LeapMotionFrame::getPointableId (C++ function), 740
LeapMotionFrame::getPointableLength (C++ function), 740
LeapMotionFrame::getPointablePos (C++ function), 740
LeapMotionFrame::getPointableRawPos (C++ function), 740
LeapMotionFrame::getPointableRawTransform (C++ function), 740
LeapMotionFrame::getPointableRot (C++ function), 740
LeapMotionFrame::getPointablesCount (C++ function), 740
LeapMotionFrame::getPointableTransform (C++ function), 740
LeapMotionFrame::getPointableType (C++ function), 741
LeapMotionFrame::getPointableValid (C++ function), 741
LeapMotionFrame::getPointableWidth (C++ function), 741
LeapMotionFrame::isFrameValid (C++ function), 741
LevelInfo::advancedLightmapSupport (C++ member), 708
LevelInfo::ambientLightBlendCurve (C++ member), 708
LevelInfo::ambientLightBlendPhase (C++ member), 708
LevelInfo::canvasClearColor (C++ member), 708
LevelInfo::decalBias (C++ member), 708
LevelInfo::fogAtmosphereHeight (C++ member), 709

LevelInfo::fogColor (C++ member), 709
LevelInfo::fogDensity (C++ member), 709
LevelInfo::fogDensityOffset (C++ member), 709
LevelInfo::nearClip (C++ member), 709
LevelInfo::soundAmbience (C++ member), 709
LevelInfo::soundDistanceModel (C++ member), 709
LevelInfo::visibleDistance (C++ member), 709
LightAnimData::brightnessA (C++ member), 635
LightAnimData::brightnessKeys (C++ member), 635
LightAnimData::brightnessPeriod (C++ member), 635
LightAnimData::brightnessSmooth (C++ member), 635
LightAnimData::brightnessZ (C++ member), 635
LightAnimData::colorA (C++ member), 635
LightAnimData::colorKeys (C++ member), 635
LightAnimData::colorPeriod (C++ member), 635
LightAnimData::colorSmooth (C++ member), 635
LightAnimData::colorZ (C++ member), 635
LightAnimData::offsetA (C++ member), 635
LightAnimData::offsetKeys (C++ member), 635
LightAnimData::offsetPeriod (C++ member), 635
LightAnimData::offsetSmooth (C++ member), 635
LightAnimData::OffsetZ (C++ member), 635
LightAnimData::rotA (C++ member), 635
LightAnimData::rotKeys (C++ member), 636
LightAnimData::rotPeriod (C++ member), 636
LightAnimData::rotSmooth (C++ member), 636
LightAnimData::rotZ (C++ member), 636
LightBase::animate (C++ member), 856
LightBase::animationPeriod (C++ member), 856
LightBase::animationPhase (C++ member), 856
LightBase::animationType (C++ member), 856
LightBase::attenuationRatio (C++ member), 856
LightBase::brightness (C++ member), 856
LightBase::castShadows (C++ member), 856
LightBase::color (C++ member), 856
LightBase::cookie (C++ member), 856
LightBase::fadeStartDistance (C++ member), 856
LightBase::flareScale (C++ member), 856
LightBase::flareType (C++ member), 857
LightBase::includeLightmappedGeometryInShadow (C++ member), 857
LightBase::isEnabled (C++ member), 857
LightBase::lastSplitTerrainOnly (C++ member), 857
LightBase::logWeight (C++ member), 857
LightBase::numSplits (C++ member), 857
LightBase::overDarkFactor (C++ member), 857
LightBase::pauseAnimation (C++ member), 857
LightBase::playAnimation (C++ function), 856
LightBase::priority (C++ member), 857
LightBase::representedInLightmap (C++ member), 857
LightBase::setLightEnabled (C++ function), 856
LightBase::shadowDarkenColor (C++ member), 857
LightBase::shadowDistance (C++ member), 857
LightBase::shadowSoftness (C++ member), 857
LightBase::shadowType (C++ member), 857
LightBase::texSize (C++ member), 857
LightDescription::animationPeriod (C++ member), 858
LightDescription::animationPhase (C++ member), 858
LightDescription::animationType (C++ member), 858
LightDescription::apply (C++ function), 858
LightDescription::attenuationRatio (C++ member), 858
LightDescription::brightness (C++ member), 858
LightDescription::castShadows (C++ member), 858
LightDescription::color (C++ member), 858
LightDescription::cookie (C++ member), 859
LightDescription::fadeStartDistance (C++ member), 859
LightDescription::flareScale (C++ member), 859
LightDescription::flareType (C++ member), 859
LightDescription::includeLightmappedGeometryInShadow (C++ member), 859
LightDescription::lastSplitTerrainOnly (C++ member), 859
LightDescription::logWeight (C++ member), 859
LightDescription::numSplits (C++ member), 859
LightDescription::overDarkFactor (C++ member), 859
LightDescription::range (C++ member), 859
LightDescription::representedInLightmap (C++ member), 859
LightDescription::shadowDarkenColor (C++ member), 859
LightDescription::shadowDistance (C++ member), 859
LightDescription::shadowSoftness (C++ member), 859
LightDescription::shadowType (C++ member), 859
LightDescription::texSize (C++ member), 859
LightFlareData::apply (C++ function), 860
LightFlareData::elementDist (C++ member), 860
LightFlareData::elementRect (C++ member), 860
LightFlareData::elementRotate (C++ member), 860
LightFlareData::elementScale (C++ member), 860
LightFlareData::elementTint (C++ member), 860
LightFlareData::elementUseLightColor (C++ member), 860
LightFlareData::flareEnabled (C++ member), 860
LightFlareData::flareTexture (C++ member), 860
LightFlareData::occlusionRadius (C++ member), 860
LightFlareData::overallScale (C++ member), 861
LightFlareData::renderReflectPass (C++ member), 861
Lightning::applyDamage (C++ function), 636
Lightning::boltStartRadius (C++ member), 637
Lightning::chanceToHitTarget (C++ member), 637
Lightning::color (C++ member), 637
Lightning::fadeColor (C++ member), 637
Lightning::strikeObject (C++ function), 636
Lightning::strikeRadius (C++ member), 637
Lightning::strikeRandomPoint (C++ function), 636
Lightning::strikesPerMinute (C++ member), 637
Lightning::strikeWidth (C++ member), 637
Lightning::useFog (C++ member), 637

Lightning::warningFlashes (C++ function), 636
LightningData::strikeSound (C++ member), 637
LightningData::strikeTextures (C++ member), 637
LightningData::thunderSounds (C++ member), 637
lightScene (C++ function), 862
listGFXResources (C++ function), 851
loadObject (C++ function), 306
lockMouse (C++ function), 746
log (C++ function), 311
logError (C++ function), 311
logWarning (C++ function), 311
ltrim (C++ function), 367

M

m2Pi (C++ function), 356
mAbs (C++ function), 356
mAcos (C++ function), 356
makeFullPath (C++ function), 352
makeRelativePath (C++ function), 352
Marker::msToNext (C++ member), 709
Marker::seqNum (C++ member), 709
Marker::smoothingType (C++ member), 709
Marker::type (C++ member), 710
mAsin (C++ function), 356
mAtan (C++ function), 356
Material::alphaRef (C++ member), 838
Material::alphaTest (C++ member), 838
Material::animFlags (C++ member), 838
Material::baseTex (C++ member), 838
Material::bumpAtlas (C++ member), 839
Material::bumpTex (C++ member), 839
Material::castShadows (C++ member), 839
Material::cellIndex (C++ member), 839
Material::cellLayout (C++ member), 839
Material::cellSize (C++ member), 839
Material::colorMultiply (C++ member), 839
Material::cubemap (C++ member), 839
Material::customFootstepSound (C++ member), 839
Material::customImpactSound (C++ member), 839
Material::detailMap (C++ member), 839
Material::detailNormalMap (C++ member), 839
Material::detailNormalMapStrength (C++ member), 839
Material::detailScale (C++ member), 839
Material::detailTex (C++ member), 839
Material::diffuseColor (C++ member), 839
Material::diffuseMap (C++ member), 839
Material::doubleSided (C++ member), 839
Material::dumpInstances (C++ member), 839
Material::dynamicCubemap (C++ member), 840
Material::effectColor (C++ member), 840
Material::emissive (C++ member), 840
Material::envMap (C++ member), 840
Material::envTex (C++ member), 840
Material::flush (C++ member), 840
Material::footstepSoundId (C++ member), 840
Material::getAnimFlags (C++ member), 841
Material::getFilename (C++ member), 841
Material::glow (C++ member), 841
Material::impactSoundId (C++ member), 841
Material::isAutoGenerated (C++ member), 841
Material::lightMap (C++ member), 841
Material::mapTo (C++ member), 841
Material::minnaertConstant (C++ member), 841
Material::normalMap (C++ member), 841
Material::overlayMap (C++ member), 841
Material::overlayTex (C++ member), 841
Material::parallaxScale (C++ member), 841
Material::pixelSpecular (C++ member), 841
Material::planarReflection (C++ member), 842
Material::reload (C++ member), 842
Material::rotPivotOffset (C++ member), 842
Material::rotSpeed (C++ member), 842
Material::scrollDir (C++ member), 842
Material::scrollSpeed (C++ member), 842
Material::sequenceFramePerSec (C++ member), 842
Material::sequenceSegmentSize (C++ member), 842
Material::setAutoGenerated (C++ member), 842
Material::showDust (C++ member), 842
Material::showFootprints (C++ member), 842
Material::specular (C++ member), 842
Material::specularMap (C++ member), 842
Material::specularPower (C++ member), 842
Material::specularStrength (C++ member), 842
Material::subSurface (C++ member), 842
Material::subSurfaceColor (C++ member), 842
Material::subSurfaceRolloff (C++ member), 842
Material::toneMap (C++ member), 842
Material::translucent (C++ member), 842
Material::translucentBlendOp (C++ member), 843
Material::translucentZWrite (C++ member), 843
Material::useAnisotropic (C++ member), 843
Material::vertColor (C++ member), 843
Material::vertLit (C++ member), 843
Material::waveAmp (C++ member), 843
Material::waveFreq (C++ member), 843
Material::waveType (C++ member), 843
mathInit (C++ function), 356
MatrixCreate (C++ function), 363
MatrixCreateFromEuler (C++ function), 364
MatrixMulPoint (C++ function), 364
MatrixMultiply (C++ function), 364
MatrixMulVector (C++ function), 364
mCeil (C++ function), 357
mClamp (C++ function), 357
mCos (C++ function), 357
mDegToRad (C++ function), 357
MeshRoad::bottomMaterial (C++ member), 695
MeshRoad::breakAngle (C++ member), 695

MeshRoad::EditorOpen (C++ member), 695
MeshRoad::Node (C++ member), 695
MeshRoad::postApply (C++ function), 695
MeshRoad::regenerate (C++ function), 695
MeshRoad::setNodeDepth (C++ function), 695
MeshRoad::showBatches (C++ member), 695
MeshRoad::showRoad (C++ member), 695
MeshRoad::showSpline (C++ member), 695
MeshRoad::sideMaterial (C++ member), 695
MeshRoad::textureLength (C++ member), 695
MeshRoad::topMaterial (C++ member), 695
MeshRoad::widthSubdivisions (C++ member), 695
MeshRoad::wireframe (C++ member), 695
Message::addReference (C++ function), 314
Message::freeReference (C++ function), 314
Message::getType (C++ function), 314
Message::onAdd (C++ function), 314
Message::onRemove (C++ function), 314
messageBox (C++ function), 777
MessageForwarder::toQueue (C++ member), 314
MessageVector::clear (C++ function), 488
MessageVector::deleteLine (C++ function), 488
MessageVector::dump (C++ function), 488
MessageVector::getLineIndexByTag (C++ function), 488
MessageVector::getLineTag (C++ function), 488
MessageVector::getLineText (C++ function), 489
MessageVector::getLineTextByTag (C++ function), 489
MessageVector::getNumLines (C++ function), 489
MessageVector::insertLine (C++ function), 489
MessageVector::popBackLine (C++ function), 489
MessageVector::popFrontLine (C++ function), 490
MessageVector::pushBackLine (C++ function), 490
MessageVector::pushFrontLine (C++ function), 490
mFloatLength (C++ function), 357
mFloor (C++ function), 357
mFMod (C++ function), 357
mIsPow2 (C++ function), 357
MissionArea::area (C++ member), 710
MissionArea::flightCeiling (C++ member), 710
MissionArea::flightCeilingRange (C++ member), 710
MissionArea::getArea (C++ function), 710
MissionArea::postApply (C++ function), 710
MissionArea::setArea (C++ function), 710
mLerp (C++ function), 358
mLog (C++ function), 358
mPi (C++ function), 358
mPow (C++ function), 358
mRadToDeg (C++ function), 358
mRound (C++ function), 358
mSaturate (C++ function), 358
mSin (C++ function), 358
mSolveCubic (C++ function), 358
mSolveQuadratic (C++ function), 359
mSolveQuartic (C++ function), 359
mSqrt (C++ function), 359
mTan (C++ function), 359

N

NavMesh::actorClimb (C++ member), 654
NavMesh::actorHeight (C++ member), 654
NavMesh::actorRadius (C++ member), 654
NavMesh::alwaysRender (C++ member), 654
NavMesh::borderSize (C++ member), 654
NavMesh::build (C++ function), 654
NavMesh::buildTiles (C++ function), 654
NavMesh::cancelBuild (C++ function), 654
NavMesh::cellHeight (C++ member), 654
NavMesh::cellSize (C++ member), 654
NavMesh::detailSampleDist (C++ member), 654
NavMesh::detailSampleError (C++ member), 654
NavMesh::fileName (C++ member), 654
NavMesh::load (C++ function), 654
NavMesh::maxEdgeLen (C++ member), 654
NavMesh::maxPolysPerTile (C++ member), 654
NavMesh::mergeRegionArea (C++ member), 655
NavMesh::minRegionArea (C++ member), 655
NavMesh::save (C++ function), 654
NavMesh::simplificationError (C++ member), 655
NavMesh::tileSize (C++ member), 655
NavMesh::walkableSlope (C++ member), 655
NavPath::alwaysRender (C++ member), 655
NavPath::from (C++ member), 655
NavPath::getCount (C++ function), 655
NavPath::getLength (C++ function), 655
NavPath::getNode (C++ function), 655
NavPath::isLooping (C++ member), 655
NavPath::mesh (C++ member), 655
NavPath::replan (C++ function), 655
NavPath::to (C++ member), 655
NavPath::waypoints (C++ member), 655
NavPath::xray (C++ member), 655
NetConnection::checkMaxRate (C++ function), 759
NetConnection::clearPaths (C++ function), 760
NetConnection::connect (C++ function), 760
NetConnection::connectLocal (C++ function), 760
NetConnection::getAddress (C++ function), 760
NetConnection::getGhostID (C++ function), 760
NetConnection::getGhostsActive (C++ function), 760
NetConnection::getPacketLoss (C++ function), 760
NetConnection::getPing (C++ function), 761
NetConnection::resolveGhostID (C++ function), 761
NetConnection::resolveObjectFromGhostIndex (C++ function), 761
NetConnection::setSimulatedNetParams (C++ function), 761
NetConnection::transmitPaths (C++ function), 761
NetObject::clearScopeToClient (C++ function), 764
NetObject::getClientObject (C++ function), 764

NetObject::getGhostID (C++ function), 765
NetObject::getServerObject (C++ function), 765
NetObject::isClientObject (C++ function), 765
NetObject::isServerObject (C++ function), 765
NetObject::scopeToClient (C++ function), 765
NetObject::setScopeAlways (C++ function), 765
nextToken (C++ function), 367

O

OcclusionVolume::edge (C++ member), 711
OcclusionVolume::plane (C++ member), 711
OcclusionVolume::point (C++ member), 711
onDataBlockObjectReceived (C++ function), 774
onLightManagerActivate (C++ function), 774, 863
onLightManagerDeactivate (C++ function), 774, 863
openFile (C++ function), 352
OpenFileDialog::MultipleFiles (C++ member), 336
OpenFileDialog::MustExist (C++ member), 336
openFolder (C++ function), 352
OpenFolderDialog::fileMustExist (C++ member), 336
ovrResetAllSensors (C++ function), 517

P

ParticleData::animateTexture (C++ member), 638
ParticleData::animTexFrames (C++ member), 639
ParticleData::animTexName (C++ member), 639
ParticleData::animTexTiling (C++ member), 639
ParticleData::colors (C++ member), 639
ParticleData::constantAcceleration (C++ member), 639
ParticleData::dragCoefficient (C++ member), 639
ParticleData::framesPerSec (C++ member), 639
ParticleData::gravityCoefficient (C++ member), 639
ParticleData::inheritedVelFactor (C++ member), 639
ParticleData::lifetimeMS (C++ member), 639
ParticleData::lifetimeVarianceMS (C++ member), 639
ParticleData::reload (C++ function), 638
ParticleData::sizes (C++ member), 639
ParticleData::spinRandomMax (C++ member), 639
ParticleData::spinRandomMin (C++ member), 639
ParticleData::spinSpeed (C++ member), 639
ParticleData::textureCoords (C++ member), 639
ParticleData::textureName (C++ member), 639
ParticleData::times (C++ member), 640
ParticleData::useInvAlpha (C++ member), 640
ParticleData::windCoefficient (C++ member), 640
ParticleEmitterData::alignDirection (C++ member), 641
ParticleEmitterData::alignParticles (C++ member), 641
ParticleEmitterData::ambientFactor (C++ member), 641
ParticleEmitterData::blendStyle (C++ member), 641
ParticleEmitterData::ejectionOffset (C++ member), 641
ParticleEmitterData::ejectionOffsetVariance (C++ member), 641
ParticleEmitterData::ejectionPeriodMS (C++ member), 641
ParticleEmitterData::ejectionVelocity (C++ member), 641
ParticleEmitterData::highResOnly (C++ member), 641
ParticleEmitterData::lifetimeMS (C++ member), 641
ParticleEmitterData::lifetimeVarianceMS (C++ member), 641
ParticleEmitterData::orientOnVelocity (C++ member), 641
ParticleEmitterData::orientParticles (C++ member), 641
ParticleEmitterData::overrideAdvance (C++ member), 641
ParticleEmitterData::particles (C++ member), 641
ParticleEmitterData::periodVarianceMS (C++ member), 642
ParticleEmitterData::phiReferenceVel (C++ member), 642
ParticleEmitterData::phiVariance (C++ member), 642
ParticleEmitterData::reload (C++ function), 640
ParticleEmitterData::renderReflection (C++ member), 642
ParticleEmitterData::reverseOrder (C++ member), 642
ParticleEmitterData::softnessDistance (C++ member), 642
ParticleEmitterData::sortParticles (C++ member), 642
ParticleEmitterData::textureName (C++ member), 642
ParticleEmitterData::thetaMax (C++ member), 642
ParticleEmitterData::thetaMin (C++ member), 642
ParticleEmitterData::useEmitterColors (C++ member), 642
ParticleEmitterData::useEmitterSizes (C++ member), 642
ParticleEmitterData::velocityVariance (C++ member), 642
ParticleEmitterNode::active (C++ member), 643
ParticleEmitterNode::emitter (C++ member), 643
ParticleEmitterNode::setActive (C++ function), 643
ParticleEmitterNode::setEmitterDataBlock (C++ function), 643
ParticleEmitterNode::velocity (C++ member), 643
ParticleEmitterNodeData::timeMultiple (C++ member), 643
Path::getPathId (C++ function), 712
Path::isLooping (C++ member), 712
PathCamera::onNode (C++ function), 727
PathCamera::popFront (C++ function), 727
PathCamera::pushBack (C++ function), 727
PathCamera::pushFront (C++ function), 727
PathCamera::reset (C++ function), 728
PathCamera::setPosition (C++ function), 728
PathCamera::setState (C++ function), 728
PathCamera::setTarget (C++ function), 729
pathConcat (C++ function), 352
pathCopy (C++ function), 353
pathOnMissionLoadDone (C++ function), 775

PfxVis::clear (C++ function), 843
PfxVis::hide (C++ function), 843
PfxVis::onWindowClosed (C++ function), 843
PfxVis::open (C++ function), 844
PfxVis::show (C++ function), 844
PhysicalZone::activate (C++ function), 713
PhysicalZone::appliedForce (C++ member), 713
PhysicalZone::deactivate (C++ function), 713
PhysicalZone::gravityMod (C++ member), 713
PhysicalZone::polyhedron (C++ member), 713
PhysicalZone::renderZones (C++ member), 713
PhysicalZone::velocityMod (C++ member), 713
PhysicsDebrisData::angularDamping (C++ member), 657
PhysicsDebrisData::angularSleepThreshold (C++ member), 657
PhysicsDebrisData::buoyancyDensity (C++ member), 657
PhysicsDebrisData::castShadows (C++ member), 657
PhysicsDebrisData::friction (C++ member), 657
PhysicsDebrisData::lifetime (C++ member), 657
PhysicsDebrisData::lifetimeVariance (C++ member), 657
PhysicsDebrisData::linearDamping (C++ member), 657
PhysicsDebrisData::linearSleepThreshold (C++ member), 657
PhysicsDebrisData::mass (C++ member), 657
PhysicsDebrisData::preload (C++ member), 657
PhysicsDebrisData::restitution (C++ member), 657
PhysicsDebrisData::shapeFile (C++ member), 657
PhysicsDebrisData::staticFriction (C++ member), 657
PhysicsDebrisData::waterDampingScale (C++ member), 657
PhysicsForce::attach (C++ function), 658
PhysicsForce::detach (C++ function), 658
PhysicsForce::isAttached (C++ function), 658
physicsPluginPresent (C++ member), 668
PhysicsShape::destroy (C++ function), 658
PhysicsShape::isDestroyed (C++ function), 658
PhysicsShape::playAmbient (C++ member), 658
PhysicsShape::restore (C++ function), 658
PhysicsShapeData::angularDamping (C++ member), 659
PhysicsShapeData::angularSleepThreshold (C++ member), 659
PhysicsShapeData::buoyancyDensity (C++ member), 659
PhysicsShapeData::Debris (C++ member), 659
PhysicsShapeData::destroyedShape (C++ member), 659
PhysicsShapeData::Explosion (C++ member), 659
PhysicsShapeData::friction (C++ member), 659
PhysicsShapeData::linearDamping (C++ member), 659
PhysicsShapeData::linearSleepThreshold (C++ member), 659
PhysicsShapeData::mass (C++ member), 659
PhysicsShapeData::restitution (C++ member), 659
PhysicsShapeData::shapeName (C++ member), 659
PhysicsShapeData::simType (C++ member), 659
PhysicsShapeData::staticFriction (C++ member), 659
PhysicsShapeData::waterDampingScale (C++ member), 659
Player::allowAllPoses (C++ function), 547
Player::allowCrouching (C++ function), 547
Player::allowJetJumping (C++ function), 547
Player::allowJumping (C++ function), 547
Player::allowProne (C++ function), 547
Player::allowSprinting (C++ function), 547
Player::allowSwimming (C++ function), 547
Player::checkDismountPoint (C++ function), 547
Player::clearControlObject (C++ function), 548
Player::crouchTrigger (C++ member), 550
Player::extendedMoveHeadPosRotIndex (C++ member), 550
Player::getControlObject (C++ function), 548
Player::getDamageLocation (C++ function), 548
Player::getNumDeathAnimations (C++ function), 548
Player::getPose (C++ function), 548
Player::getState (C++ function), 548
Player::imageTrigger0 (C++ member), 550
Player::imageTrigger1 (C++ member), 550
Player::jumpJetTrigger (C++ member), 550
Player::jumpTrigger (C++ member), 550
Player::maxImpulseVelocity (C++ member), 550
Player::maxPredictionTicks (C++ member), 550
Player::maxWarpTicks (C++ member), 550
Player::minWarpTicks (C++ member), 550
Player::proneTrigger (C++ member), 551
Player::renderCollision (C++ member), 551
Player::renderMyItems (C++ member), 551
Player::renderMyPlayer (C++ member), 551
Player::setActionThread (C++ function), 548
Player::setArmThread (C++ function), 550
Player::setControlObject (C++ function), 550
Player::sprintTrigger (C++ member), 551
Player::vehicleDismountTrigger (C++ member), 551
PlayerData::airControl (C++ member), 552
PlayerData::allowImageStateAnimation (C++ member), 552
PlayerData::animationDone (C++ function), 551
PlayerData::boundingBox (C++ member), 552
PlayerData::boxHeadBackPercentage (C++ member), 552
PlayerData::boxHeadFrontPercentage (C++ member), 552
PlayerData::boxHeadLeftPercentage (C++ member), 552
PlayerData::boxHeadPercentage (C++ member), 552
PlayerData::boxHeadRightPercentage (C++ member), 553
PlayerData::boxTorsoPercentage (C++ member), 553
PlayerData::bubbleEmitTime (C++ member), 553
PlayerData::crouchBoundingBox (C++ member), 553
PlayerData::crouchForce (C++ member), 553

PlayerData::DecalData (C++ member), 553
PlayerData::decalOffset (C++ member), 553
PlayerData::doDismount (C++ function), 551
PlayerData::dustEmitter (C++ member), 553
PlayerData::exitingWater (C++ member), 553
PlayerData::exitSplashSoundVelocity (C++ member), 553
PlayerData::fallingSpeedThreshold (C++ member), 553
PlayerData::firstPersonShadows (C++ member), 553
PlayerData::FootBubblesSound (C++ member), 553
PlayerData::FootHardSound (C++ member), 553
PlayerData::FootMetalSound (C++ member), 553
PlayerData::footPuffEmitter (C++ member), 553
PlayerData::footPuffNumParts (C++ member), 553
PlayerData::footPuffRadius (C++ member), 553
PlayerData::FootShallowSound (C++ member), 553
PlayerData::FootSnowSound (C++ member), 554
PlayerData::FootSoftSound (C++ member), 554
PlayerData::footstepSplashHeight (C++ member), 554
PlayerData::FootUnderwaterSound (C++ member), 554
PlayerData::FootWadingSound (C++ member), 554
PlayerData::groundImpactMinSpeed (C++ member), 554
PlayerData::groundImpactShakeAmp (C++ member), 554
PlayerData::groundImpactShakeDuration (C++ member), 554
PlayerData::groundImpactShakeFalloff (C++ member), 554
PlayerData::groundImpactShakeFreq (C++ member), 554
PlayerData::hardSplashSoundVelocity (C++ member), 554
PlayerData::horizMaxSpeed (C++ member), 554
PlayerData::horizResistFactor (C++ member), 554
PlayerData::horizResistSpeed (C++ member), 554
PlayerData::imageAnimPrefix (C++ member), 554
PlayerData::imageAnimPrefixFP (C++ member), 554
PlayerData::impactHardSound (C++ member), 554
PlayerData::impactMetalSound (C++ member), 554
PlayerData::impactSnowSound (C++ member), 554
PlayerData::impactSoftSound (C++ member), 555
PlayerData::impactWaterEasy (C++ member), 555
PlayerData::impactWaterHard (C++ member), 555
PlayerData::impactWaterMedium (C++ member), 555
PlayerData::jetJumpEnergyDrain (C++ member), 555
PlayerData::jetJumpForce (C++ member), 555
PlayerData::jetJumpSurfaceAngle (C++ member), 555
PlayerData::jetMaxJumpSpeed (C++ member), 555
PlayerData::jetMinJumpEnergy (C++ member), 555
PlayerData::jetMinJumpSpeed (C++ member), 555
PlayerData::jumpDelay (C++ member), 555
PlayerData::jumpEnergyDrain (C++ member), 555
PlayerData::jumpForce (C++ member), 555
PlayerData::jumpSurfaceAngle (C++ member), 555
PlayerData::jumpTowardsNormal (C++ member), 555
PlayerData::landSequenceTime (C++ member), 555
PlayerData::maxBackwardSpeed (C++ member), 555
PlayerData::maxCrouchBackwardSpeed (C++ member), 555
PlayerData::maxCrouchForwardSpeed (C++ member), 555
PlayerData::maxCrouchSideSpeed (C++ member), 556
PlayerData::maxForwardSpeed (C++ member), 556
PlayerData::maxFreelookAngle (C++ member), 556
PlayerData::maxJumpSpeed (C++ member), 556
PlayerData::maxLookAngle (C++ member), 556
PlayerData::maxProneBackwardSpeed (C++ member), 556
PlayerData::maxProneForwardSpeed (C++ member), 556
PlayerData::maxProneSideSpeed (C++ member), 556
PlayerData::maxSideSpeed (C++ member), 556
PlayerData::maxSprintBackwardSpeed (C++ member), 556
PlayerData::maxSprintForwardSpeed (C++ member), 556
PlayerData::maxSprintSideSpeed (C++ member), 556
PlayerData::maxStepHeight (C++ member), 556
PlayerData::maxTimeScale (C++ member), 556
PlayerData::maxUnderwaterBackwardSpeed (C++ member), 556
PlayerData::maxUnderwaterForwardSpeed (C++ member), 556
PlayerData::maxUnderwaterSideSpeed (C++ member), 556
PlayerData::mediumSplashSoundVelocity (C++ member), 556
PlayerData::minImpactSpeed (C++ member), 556
PlayerData::minJumpEnergy (C++ member), 557
PlayerData::minJumpSpeed (C++ member), 557
PlayerData::minLateralImpactSpeed (C++ member), 557
PlayerData::minLookAngle (C++ member), 557
PlayerData::minRunEnergy (C++ member), 557
PlayerData::minSprintEnergy (C++ member), 557
PlayerData::movingBubblesSound (C++ member), 557
PlayerData::onEnterLiquid (C++ function), 551
PlayerData::onEnterMissionArea (C++ function), 551
PlayerData::onLeaveLiquid (C++ function), 551
PlayerData::onLeaveMissionArea (C++ function), 552
PlayerData::onPoseChange (C++ function), 552
PlayerData::onStartSprintMotion (C++ function), 552
PlayerData::onStartSwim (C++ function), 552
PlayerData::onStopSprintMotion (C++ function), 552
PlayerData::onStopSwim (C++ function), 552
PlayerData::physicsPlayerType (C++ member), 557
PlayerData::pickupRadius (C++ member), 557
PlayerData::proneBoundingBox (C++ member), 557
PlayerData::proneForce (C++ member), 557
PlayerData::recoverDelay (C++ member), 557
PlayerData::recoverRunForceScale (C++ member), 557

PlayerData::renderFirstPerson (C++ member), 557
PlayerData::runEnergyDrain (C++ member), 557
PlayerData::runForce (C++ member), 557
PlayerData::runSurfaceAngle (C++ member), 557
PlayerData::shapeNameFP (C++ member), 558
PlayerData::Splash (C++ member), 558
PlayerData::splashAngle (C++ member), 558
PlayerData::splashEmitter (C++ member), 558
PlayerData::splashFreqMod (C++ member), 558
PlayerData::splashVelEpsilon (C++ member), 558
PlayerData::splashVelocity (C++ member), 558
PlayerData::sprintCanJump (C++ member), 558
PlayerData::sprintEnergyDrain (C++ member), 558
PlayerData::sprintForce (C++ member), 558
PlayerData::sprintPitchScale (C++ member), 558
PlayerData::sprintStrafeScale (C++ member), 558
PlayerData::sprintYawScale (C++ member), 558
PlayerData::swimBoundingBox (C++ member), 558
PlayerData::swimForce (C++ member), 558
PlayerData::transitionToLand (C++ member), 558
PlayerData::upMaxSpeed (C++ member), 558
PlayerData::upResistFactor (C++ member), 558
PlayerData::upResistSpeed (C++ member), 558
PlayerData::waterBreathSound (C++ member), 558
playJournal (C++ function), 778
playJournalToVideo (C++ function), 827
PointLight::radius (C++ member), 861
populateAllFontCacheRange (C++ function), 829
populateAllFontCacheString (C++ function), 829
populateFontCacheRange (C++ function), 829
populateFontCacheString (C++ function), 829
Portal::backSidePassable (C++ member), 714
Portal::frontSidePassable (C++ member), 714
Portal::isExteriorPortal (C++ function), 714
Portal::isInteriorPortal (C++ function), 714
PostEffect::allowReflectPass (C++ member), 824
PostEffect::clearShaderMacros (C++ function), 822
PostEffect::disable (C++ function), 822
PostEffect::dumpShaderDisassembly (C++ function), 822
PostEffect::enable (C++ function), 822
PostEffect::getAspectRatio (C++ function), 822
PostEffect::isEnabled (C++ function), 822
PostEffect::isEnabled (C++ member), 824
PostEffect::onAdd (C++ function), 822
PostEffect::onDisabled (C++ function), 822
PostEffect::oneFrameOnly (C++ member), 824
PostEffect::onEnabled (C++ function), 822
PostEffect::onThisFrame (C++ member), 824
PostEffect::preProcess (C++ function), 822
PostEffect::reload (C++ function), 823
PostEffect::removeShaderMacro (C++ function), 823
PostEffect::renderBin (C++ member), 824
PostEffect::renderPriority (C++ member), 824
PostEffect::renderTime (C++ member), 824
PostEffect::setShaderConst (C++ function), 823
PostEffect::setShaderConsts (C++ function), 823
PostEffect::setShaderMacro (C++ function), 823
PostEffect::setTexture (C++ function), 824
PostEffect::shader (C++ member), 824
PostEffect::skip (C++ member), 824
PostEffect::stateBlock (C++ member), 824
PostEffect::target (C++ member), 824
PostEffect::targetClear (C++ member), 824
PostEffect::targetClearColor (C++ member), 824
PostEffect::targetDepthStencil (C++ member), 824
PostEffect::targetFormat (C++ member), 824
PostEffect::targetScale (C++ member), 825
PostEffect::targetSize (C++ member), 825
PostEffect::targetViewport (C++ member), 825
PostEffect::texture (C++ member), 825
PostEffect::toggle (C++ function), 824
Precipitation::animateSplashes (C++ member), 645
Precipitation::boxHeight (C++ member), 645
Precipitation::boxWidth (C++ member), 645
Precipitation::doCollision (C++ member), 645
Precipitation::dropAnimateMS (C++ member), 645
Precipitation::dropSize (C++ member), 645
Precipitation::fadeDist (C++ member), 645
Precipitation::fadeDistEnd (C++ member), 645
Precipitation::followCam (C++ member), 645
Precipitation::glowIntensity (C++ member), 646
Precipitation::hitPlayers (C++ member), 646
Precipitation::hitVehicles (C++ member), 646
Precipitation::maxMass (C++ member), 646
Precipitation::maxSpeed (C++ member), 646
Precipitation::maxTurbulence (C++ member), 646
Precipitation::minMass (C++ member), 646
Precipitation::minSpeed (C++ member), 646
Precipitation::modifyStorm (C++ function), 644
Precipitation::numDrops (C++ member), 646
Precipitation::reflect (C++ member), 646
Precipitation::rotateWithCamVel (C++ member), 646
Precipitation::setPercentage (C++ function), 645
Precipitation::setTurbulence (C++ function), 645
Precipitation::splashMS (C++ member), 646
Precipitation::splashSize (C++ member), 646
Precipitation::turbulenceSpeed (C++ member), 646
Precipitation::useLighting (C++ member), 646
Precipitation::useTrueBillboards (C++ member), 646
Precipitation::useTurbulence (C++ member), 646
Precipitation::useWind (C++ member), 646
PrecipitationData::dropShader (C++ member), 647
PrecipitationData::dropsPerSide (C++ member), 647
PrecipitationData::dropTexture (C++ member), 647
PrecipitationData::soundProfile (C++ member), 647
PrecipitationData::splashesPerSide (C++ member), 647
PrecipitationData::splashShader (C++ member), 647

PrecipitationData::splashTexture (C++ member), 647
Prefab::fileName (C++ member), 715
Prefab::onLoad (C++ function), 715
profilerDump (C++ function), 308
profilerDumpToFile (C++ function), 308
profilerEnable (C++ function), 308
profilerMarkerEnable (C++ function), 308
profilerReset (C++ function), 308
Projectile::initialPosition (C++ member), 559
Projectile::initialVelocity (C++ member), 559
Projectile::presimulate (C++ function), 559
Projectile::sourceObject (C++ member), 559
Projectile::sourceSlot (C++ member), 559
ProjectileData::armingDelay (C++ member), 560
ProjectileData::bounceElasticity (C++ member), 560
ProjectileData::bounceFriction (C++ member), 560
ProjectileData::decal (C++ member), 560
ProjectileData::Explosion (C++ member), 560
ProjectileData::fadeDelay (C++ member), 560
ProjectileData::gravityMod (C++ member), 561
ProjectileData::impactForce (C++ member), 561
ProjectileData::isBallistic (C++ member), 561
ProjectileData::lifetime (C++ member), 561
ProjectileData::lightDesc (C++ member), 561
ProjectileData::muzzleVelocity (C++ member), 561
ProjectileData::onCollision (C++ function), 560
ProjectileData::onExplode (C++ function), 560
ProjectileData::ParticleEmitter (C++ member), 561
ProjectileData::particleWaterEmitter (C++ member), 561
ProjectileData::projectileShapeName (C++ member), 561
ProjectileData::scale (C++ member), 561
ProjectileData::sound (C++ member), 561
ProjectileData::Splash (C++ member), 561
ProjectileData::velInheritFactor (C++ member), 561
ProjectileData::waterExplosion (C++ member), 561
ProximityMine::explode (C++ function), 562
ProximityMineData::armingDelay (C++ member), 563
ProximityMineData::armingSound (C++ member), 563
ProximityMineData::autoTriggerDelay (C++ member), 563
ProximityMineData::explosionOffset (C++ member), 563
ProximityMineData::onExplode (C++ function), 563
ProximityMineData::onTriggered (C++ function), 563
ProximityMineData::triggerDelay (C++ member), 563
ProximityMineData::triggerOnOwner (C++ member), 563
ProximityMineData::triggerRadius (C++ member), 563
ProximityMineData::triggerSound (C++ member), 563
ProximityMineData::triggerSpeed (C++ member), 563
PxCloth::attachments (C++ member), 660
PxCloth::bending (C++ member), 660
PxCloth::bendingStiffness (C++ member), 660
PxCloth::damping (C++ member), 660
PxCloth::dampingCoefficient (C++ member), 660
PxCloth::density (C++ member), 660
PxCloth::friction (C++ member), 660
PxCloth::Material (C++ member), 660
PxCloth::samples (C++ member), 660
PxCloth::selfCollision (C++ member), 660
PxCloth::size (C++ member), 660
PxCloth::thickness (C++ member), 660
PxCloth::triangleCollision (C++ member), 660
PxMaterial::dynamicFriction (C++ member), 661
PxMaterial::restitution (C++ member), 661
PxMaterial::staticFriction (C++ member), 661
PxMultiActor::broken (C++ member), 662
PxMultiActor::debugRender (C++ member), 662
PxMultiActor::setAllHidden (C++ function), 661
PxMultiActor::setBroken (C++ function), 661
PxMultiActor::setMeshHidden (C++ function), 662
PxMultiActorData::angularDrag (C++ member), 662
PxMultiActorData::breakForce (C++ member), 662
PxMultiActorData::buoyancyDensity (C++ member), 662
PxMultiActorData::clientOnly (C++ member), 662
PxMultiActorData::dumpModel (C++ member), 662
PxMultiActorData::linearDrag (C++ member), 662
PxMultiActorData::Material (C++ member), 662
PxMultiActorData::noCorrection (C++ member), 662
PxMultiActorData::physXStream (C++ member), 662
PxMultiActorData::reload (C++ member), 662
PxMultiActorData::shapeName (C++ member), 662
PxMultiActorData::singlePlayerOnly (C++ member), 662
PxMultiActorData::string (C++ member), 662
PxMultiActorData::waterDragScale (C++ member), 663

Q

quit (C++ function), 778
quitWithErrorMessage (C++ function), 778

R

RadialImpulseEvent::send (C++ function), 663
RazerHydraFrame::getControllerButton1 (C++ function), 741
RazerHydraFrame::getControllerButton2 (C++ function), 741
RazerHydraFrame::getControllerButton3 (C++ function), 741
RazerHydraFrame::getControllerButton4 (C++ function), 741
RazerHydraFrame::getControllerCount (C++ function), 741
RazerHydraFrame::getControllerDocked (C++ function), 742
RazerHydraFrame::getControllerEnabled (C++ function), 742
RazerHydraFrame::getControllerPos (C++ function), 742

RazerHydraFrame::getControllerRawPos (C++ function), 742
RazerHydraFrame::getControllerRawTransform (C++ function), 742
RazerHydraFrame::getControllerRot (C++ function), 742
RazerHydraFrame::getControllerRotAxis (C++ function), 742
RazerHydraFrame::getControllerSequenceNum (C++ function), 742
RazerHydraFrame::getControllerShoulderButton (C++ function), 743
RazerHydraFrame::getControllerStartButton (C++ function), 743
RazerHydraFrame::getControllerThumbButton (C++ function), 743
RazerHydraFrame::getControllerThumbStick (C++ function), 743
RazerHydraFrame::getControllerTransform (C++ function), 743
RazerHydraFrame::getControllerTrigger (C++ function), 743
RazerHydraFrame::getFrameInternalId (C++ function), 743
RazerHydraFrame::getFrameRealTime (C++ function), 743
RazerHydraFrame::getFrameSimTime (C++ function), 743
RazerHydraFrame::isFrameValid (C++ function), 743
ReflectorDesc::detailAdjust (C++ member), 715
ReflectorDesc::farDist (C++ member), 715
ReflectorDesc::maxRateMs (C++ member), 715
ReflectorDesc::nearDist (C++ member), 715
ReflectorDesc::objectTypeMask (C++ member), 716
ReflectorDesc::priority (C++ member), 716
ReflectorDesc::texSize (C++ member), 716
ReflectorDesc::useOcclusionQuery (C++ member), 716
registerMessageListener (C++ function), 316
registerMessageQueue (C++ function), 316
reInitMaterials (C++ member), 854
reloadTextures (C++ function), 851
removeField (C++ function), 376
removeGlobalShaderMacro (C++ function), 827
removeRecord (C++ function), 376
removeTaggedString (C++ function), 775
removeWord (C++ function), 376
RenderBinManager::binType (C++ member), 865
RenderBinManager::getBinType (C++ function), 865
RenderBinManager::processAddOrder (C++ member), 865
RenderBinManager::renderOrder (C++ member), 865
RenderFormatToken::aaLevel (C++ member), 845
RenderFormatToken::copyEffect (C++ member), 845
RenderFormatToken::depthFormat (C++ member), 845
RenderFormatToken::format (C++ member), 845
RenderFormatToken::resolveEffect (C++ member), 845
RenderMeshExample::Material (C++ member), 869
RenderMeshExample::postApply (C++ function), 869
RenderOcclusionMgr::debugRender (C++ member), 869
RenderPassManager::addManager (C++ function), 866
RenderPassManager::getManager (C++ function), 866
RenderPassManager::getManagerCount (C++ function), 866
RenderPassManager::removeManager (C++ function), 866
RenderPassStateBin::stateToken (C++ member), 867
RenderPassStateToken::disable (C++ function), 867
RenderPassStateToken::enable (C++ function), 867
RenderPassStateToken::enabled (C++ member), 867
RenderPassStateToken::toggle (C++ function), 867
RenderShapeExample::shapeFile (C++ member), 870
RenderTerrainMgr::renderWireframe (C++ member), 869
resetFPSTracker (C++ function), 517
resetLightManager (C++ function), 863
resetXInput (C++ function), 746
restWords (C++ function), 376
RigidShape::forceClientTransform (C++ function), 664
RigidShape::freezeSim (C++ function), 664
RigidShape::onEnterLiquid (C++ function), 664
RigidShape::onLeaveLiquid (C++ function), 665
RigidShape::reset (C++ function), 665
RigidShapeData::bodyFriction (C++ member), 666
RigidShapeData::bodyRestitution (C++ member), 666
RigidShapeData::cameraDecay (C++ member), 666
RigidShapeData::cameraLag (C++ member), 666
RigidShapeData::cameraOffset (C++ member), 666
RigidShapeData::cameraRoll (C++ member), 666
RigidShapeData::collisionTol (C++ member), 666
RigidShapeData::contactTol (C++ member), 666
RigidShapeData::dragForce (C++ member), 666
RigidShapeData::dustEmitter (C++ member), 666
RigidShapeData::dustHeight (C++ member), 666
RigidShapeData::dustTrailEmitter (C++ member), 666
RigidShapeData::exitingWater (C++ member), 666
RigidShapeData::exitSplashSoundVelocity (C++ member), 666
RigidShapeData::hardImpactSound (C++ member), 666
RigidShapeData::hardImpactSpeed (C++ member), 666
RigidShapeData::hardSplashSoundVelocity (C++ member), 666
RigidShapeData::impactWaterEasy (C++ member), 667
RigidShapeData::impactWaterHard (C++ member), 667
RigidShapeData::impactWaterMedium (C++ member), 667
RigidShapeData::integration (C++ member), 667
RigidShapeData::massBox (C++ member), 667
RigidShapeData::massCenter (C++ member), 667
RigidShapeData::maxDrag (C++ member), 667

RigidShapeData::mediumSplashSoundVelocity (C++ member), 667
RigidShapeData::minDrag (C++ member), 667
RigidShapeData::minImpactSpeed (C++ member), 667
RigidShapeData::minRollSpeed (C++ member), 667
RigidShapeData::softImpactSound (C++ member), 667
RigidShapeData::softImpactSpeed (C++ member), 667
RigidShapeData::softSplashSoundVelocity (C++ member), 667
RigidShapeData::splashEmitter (C++ member), 667
RigidShapeData::splashFreqMod (C++ member), 667
RigidShapeData::splashVelEpsilon (C++ member), 667
RigidShapeData::triggerDustHeight (C++ member), 667
RigidShapeData::vertFactor (C++ member), 667
RigidShapeData::waterWakeSound (C++ member), 667
River::EditorOpen (C++ member), 689
River::FlowMagnitude (C++ member), 689
River::LowLODDistance (C++ member), 689
River::Node (C++ member), 689
River::regenerate (C++ function), 689
River::SegmentLength (C++ member), 689
River::setBatchSize (C++ function), 689
River::setMaxDivisionSize (C++ function), 689
River::setMetersPerSegment (C++ function), 689
River::setNodeDepth (C++ function), 689
River::showNodes (C++ member), 689
River::showRiver (C++ member), 689
River::showSpline (C++ member), 689
River::showWalls (C++ member), 689
River::showWireframe (C++ member), 689
River::SubdivideLength (C++ member), 689
rtrim (C++ function), 367
rumble (C++ function), 746

S

SaveFileDialog::OverwritePrompt (C++ member), 337
saveJournal (C++ function), 778
saveObject (C++ function), 306
ScatterSky::ambientScale (C++ member), 685
ScatterSky::applyChanges (C++ function), 684
ScatterSky::attenuationRatio (C++ member), 685
ScatterSky::azimuth (C++ member), 685
ScatterSky::brightness (C++ member), 685
ScatterSky::castShadows (C++ member), 685
ScatterSky::colorize (C++ member), 685
ScatterSky::colorizeAmount (C++ member), 685
ScatterSky::cookie (C++ member), 685
ScatterSky::elevation (C++ member), 685
ScatterSky::exposure (C++ member), 685
ScatterSky::fadeStartDistance (C++ member), 685
ScatterSky::flareScale (C++ member), 685
ScatterSky::flareType (C++ member), 685
ScatterSky::fogScale (C++ member), 685
ScatterSky::includeLightmappedGeometryInShadow (C++ member), 685
ScatterSky::lastSplitTerrainOnly (C++ member), 685
ScatterSky::logWeight (C++ member), 685
ScatterSky::moonAzimuth (C++ member), 685
ScatterSky::moonElevation (C++ member), 685
ScatterSky::moonEnabled (C++ member), 685
ScatterSky::moonLightColor (C++ member), 685
ScatterSky::moonMat (C++ member), 686
ScatterSky::moonScale (C++ member), 686
ScatterSky::nightColor (C++ member), 686
ScatterSky::nightCubemap (C++ member), 686
ScatterSky::nightFogColor (C++ member), 686
ScatterSky::numSplits (C++ member), 686
ScatterSky::overDarkFactor (C++ member), 686
ScatterSky::rayleighScattering (C++ member), 686
ScatterSky::representedInLightmap (C++ member), 686
ScatterSky::shadowDarkenColor (C++ member), 686
ScatterSky::shadowDistance (C++ member), 686
ScatterSky::shadowSoftness (C++ member), 686
ScatterSky::shadowType (C++ member), 686
ScatterSky::skyBrightness (C++ member), 686
ScatterSky::sunScale (C++ member), 686
ScatterSky::sunSize (C++ member), 686
ScatterSky::texSize (C++ member), 686
ScatterSky::useNightCubemap (C++ member), 686
sceneDumpZoneStates (C++ function), 517
sceneGetZoneOwner (C++ function), 517
SceneObject::getEulerRotation (C++ function), 564
SceneObject::getForwardVector (C++ function), 564
SceneObject::getInverseTransform (C++ function), 564
SceneObject::getMountedObject (C++ function), 564
SceneObject::getMountedObjectCount (C++ function), 564
SceneObject::getMountedObjectNode (C++ function), 564
SceneObject::getMountNodeObject (C++ function), 564
SceneObject::getObjectBox (C++ function), 565
SceneObject::getObjectMount (C++ function), 565
SceneObject::getPosition (C++ function), 565
SceneObject::getRightVector (C++ function), 565
SceneObject::getScale (C++ function), 565
SceneObject::getTransform (C++ function), 565
SceneObject::getType (C++ function), 565
SceneObject::getUpVector (C++ function), 565
SceneObject::getWorldBox (C++ function), 565
SceneObject::getWorldBoxCenter (C++ function), 565
SceneObject::isGlobalBounds (C++ function), 565
SceneObject::isMounted (C++ function), 565
SceneObject::isRenderEnabled (C++ member), 566
SceneObject::isSelectionEnabled (C++ member), 566
SceneObject::mountNode (C++ member), 566
SceneObject::mountObject (C++ function), 565
SceneObject::mountPID (C++ member), 566

SceneObject::mountPos (C++ member), 566
SceneObject::mountRot (C++ member), 566
SceneObject::position (C++ member), 566
SceneObject::rotation (C++ member), 566
SceneObject::scale (C++ member), 566
SceneObject::setScale (C++ function), 566
SceneObject::setTransform (C++ function), 566
SceneObject::unmount (C++ function), 566
SceneObject::unmountObject (C++ function), 566
screenShot (C++ function), 851
ScriptGroup::onAdd (C++ function), 322
ScriptGroup::onRemove (C++ function), 323
ScriptMsgListener::onAdd (C++ function), 315
ScriptMsgListener::onAddToQueue (C++ function), 315
ScriptMsgListener::onMessageObjectReceived (C++ function), 315
ScriptMsgListener::onMessageReceived (C++ function), 315
ScriptMsgListener::onRemove (C++ function), 316
ScriptMsgListener::onRemoveFromQueue (C++ function), 316
ScriptObject::onAdd (C++ function), 323
ScriptObject::onRemove (C++ function), 323
ScriptTickObject::callOnAdvanceTime (C++ member), 324
ScriptTickObject::isProcessingTicks (C++ function), 323
ScriptTickObject::onAdvanceTime (C++ function), 323
ScriptTickObject::onInterpolateTick (C++ function), 323
ScriptTickObject::onProcessTick (C++ function), 324
ScriptTickObject::setProcessTicks (C++ function), 324
setAllSensorPredictionTime (C++ function), 517
setCoreLangTable (C++ function), 786
setCurrentDirectory (C++ function), 353
setDefaultFov (C++ function), 729
setField (C++ function), 376
setFov (C++ function), 729
setLightManager (C++ function), 863
setLogMode (C++ function), 311
setNetPort (C++ function), 775
setOVRHMDAsGameConnectionDisplayDevice (C++ function), 517
setOVRHMDCurrentIPD (C++ function), 517
setOVRSensorGravityCorrection (C++ function), 517
setOVRSensorYawCorrection (C++ function), 517
setPixelShaderVersion (C++ function), 852
setRandomSeed (C++ function), 365
setRecord (C++ function), 377
setReflectFormat (C++ function), 852
setSensorPredictionTime (C++ function), 518
setVariable (C++ function), 326
setWord (C++ function), 377
setZoomSpeed (C++ function), 729
SFXAmbience::dopplerFactor (C++ member), 787
SFXAmbience::environment (C++ member), 787
SFXAmbience::rolloffFactor (C++ member), 787
SFXAmbience::soundTrack (C++ member), 787
SFXAmbience::states (C++ member), 787
SFXController::getCurrentSlot (C++ function), 788
SFXController::setCurrentSlot (C++ function), 788
SFXController::trace (C++ member), 788
sfxCreateDevice (C++ function), 812
sfxCreateSource (C++ function), 812, 813
sfxDeleteDevice (C++ function), 813
sfxDeleteWhenStopped (C++ function), 814
SFXDescription::coneInsideAngle (C++ member), 788
SFXDescription::coneOutsideAngle (C++ member), 788
SFXDescription::coneOutsideVolume (C++ member), 789
SFXDescription::fadeInEase (C++ member), 789
SFXDescription::fadeInTime (C++ member), 789
SFXDescription::fadeLoops (C++ member), 789
SFXDescription::fadeOutEase (C++ member), 789
SFXDescription::fadeOutTime (C++ member), 789
SFXDescription::is3D (C++ member), 789
SFXDescription::isLooping (C++ member), 789
SFXDescription::isStreaming (C++ member), 789
SFXDescription::maxDistance (C++ member), 789
SFXDescription::parameters (C++ member), 789
SFXDescription::pitch (C++ member), 789
SFXDescription::priority (C++ member), 789
SFXDescription::referenceDistance (C++ member), 790
SFXDescription::REVERB_DIRECTHFAUTO (C++ member), 790
SFXDescription::REVERB_INSTANCE0 (C++ member), 790
SFXDescription::REVERB_INSTANCE1 (C++ member), 790
SFXDescription::REVERB_INSTANCE2 (C++ member), 790
SFXDescription::REVERB_INSTANCE3 (C++ member), 790
SFXDescription::REVERB_ROOMAUTO (C++ member), 790
SFXDescription::REVERB_ROOMHFAUTO (C++ member), 790
SFXDescription::reverbAirAbsorptionFactor (C++ member), 790
SFXDescription::reverbDirect (C++ member), 790
SFXDescription::reverbDirectHF (C++ member), 790
SFXDescription::reverbDopplerFactor (C++ member), 790
SFXDescription::reverbExclusion (C++ member), 790
SFXDescription::reverbExclusionLFRatio (C++ member), 790
SFXDescription::reverbFlags (C++ member), 790
SFXDescription::reverbObstruction (C++ member), 790
SFXDescription::reverbObstructionLFRatio (C++ member), 790

SFXDescription::reverbOcclusion (C++ member), 790
SFXDescription::reverbOcclusionDirectRatio (C++ member), 790
SFXDescription::reverbOcclusionLFRatio (C++ member), 791
SFXDescription::reverbOcclusionRoomRatio (C++ member), 791
SFXDescription::reverbOutsideVolumeHF (C++ member), 791
SFXDescription::reverbReverbRolloffFactor (C++ member), 791
SFXDescription::reverbRoom (C++ member), 791
SFXDescription::reverbRoomHF (C++ member), 791
SFXDescription::reverbRoomRolloffFactor (C++ member), 791
SFXDescription::rolloffFactor (C++ member), 791
SFXDescription::scatterDistance (C++ member), 791
SFXDescription::sourceGroup (C++ member), 791
SFXDescription::streamPacketSize (C++ member), 791
SFXDescription::streamReadAhead (C++ member), 791
SFXDescription::useCustomReverb (C++ member), 791
SFXDescription::useHardware (C++ member), 791
SFXDescription::volume (C++ member), 791
sfxDumpSources (C++ function), 814
sfxDumpSourcesToString (C++ function), 814
SFXEmitter::coneInsideAngle (C++ member), 792
SFXEmitter::coneOutsideAngle (C++ member), 792
SFXEmitter::coneOutsideVolume (C++ member), 792
SFXEmitter::fadeInTime (C++ member), 793
SFXEmitter::fadeOutTime (C++ member), 793
SFXEmitter::fileName (C++ member), 793
SFXEmitter::getSource (C++ function), 792
SFXEmitter::is3D (C++ member), 793
SFXEmitter::isLooping (C++ member), 793
SFXEmitter::isStreaming (C++ member), 793
SFXEmitter::maxDistance (C++ member), 793
SFXEmitter::pitch (C++ member), 793
SFXEmitter::play (C++ function), 792
SFXEmitter::playOnAdd (C++ member), 793
SFXEmitter::referenceDistance (C++ member), 793
SFXEmitter::renderColorInnerCone (C++ member), 818
SFXEmitter::renderColorOuterCone (C++ member), 818
SFXEmitter::renderColorOutsideVolume (C++ member), 818
SFXEmitter::renderColorPlayingInRange (C++ member), 818
SFXEmitter::renderColorPlayingOutOfRange (C++ member), 818
SFXEmitter::renderColorRangeSphere (C++ member), 818
SFXEmitter::renderColorStoppedInRange (C++ member), 818
SFXEmitter::renderColorStoppedOutOfRange (C++ member), 818
SFXEmitter::renderEmitters (C++ member), 818
SFXEmitter::renderPointDistance (C++ member), 818
SFXEmitter::renderRadialIncrements (C++ member), 818
SFXEmitter::renderSweepIncrements (C++ member), 818
SFXEmitter::scatterDistance (C++ member), 793
SFXEmitter::sourceGroup (C++ member), 793
SFXEmitter::stop (C++ function), 792
SFXEmitter::track (C++ member), 793
SFXEmitter::useTrackDescriptionOnly (C++ member), 793
SFXEmitter::volume (C++ member), 793
SFXEnvironment::airAbsorptionHF (C++ member), 794
SFXEnvironment::decayHFRatio (C++ member), 794
SFXEnvironment::decayLFRatio (C++ member), 794
SFXEnvironment::decayTime (C++ member), 794
SFXEnvironment::density (C++ member), 794
SFXEnvironment::diffusion (C++ member), 794
SFXEnvironment::echoDepth (C++ member), 794
SFXEnvironment::echoTime (C++ member), 794
SFXEnvironment::envDiffusion (C++ member), 794
SFXEnvironment::envSize (C++ member), 794
SFXEnvironment::flags (C++ member), 794
SFXEnvironment::HFReference (C++ member), 794
SFXEnvironment::LFReference (C++ member), 794
SFXEnvironment::modulationDepth (C++ member), 794
SFXEnvironment::modulationTime (C++ member), 794
SFXEnvironment::reflections (C++ member), 794
SFXEnvironment::reflectionsDelay (C++ member), 795
SFXEnvironment::reflectionsPan (C++ member), 795
SFXEnvironment::reverb (C++ member), 795
SFXEnvironment::REVERB_CORE0 (C++ member), 795
SFXEnvironment::REVERB_CORE1 (C++ member), 795
SFXEnvironment::REVERB_DECAYHFLIMIT (C++ member), 795
SFXEnvironment::REVERB_DECAYTIMESCALE (C++ member), 795
SFXEnvironment::REVERB_ECHOTIMESCALE (C++ member), 795
SFXEnvironment::REVERB_HIGHQUALITYDPL2REVERB (C++ member), 795
SFXEnvironment::REVERB_HIGHQUALITYREVERB (C++ member), 795
SFXEnvironment::REVERB_MODULATIONTIMESCALE (C++ member), 795
SFXEnvironment::REVERB_REFLECTIONSDELAYSCALE (C++ member), 795
SFXEnvironment::REVERB_REFLECTIONSSCALE (C++ member), 795
SFXEnvironment::REVERB_REVERBDELAYSCALE (C++ member), 795

SFXEnvironment::REVERB_REVERBSCALE (C++ member), 795
SFXEnvironment::reverbDelay (C++ member), 795
SFXEnvironment::reverbPan (C++ member), 795
SFXEnvironment::room (C++ member), 795
SFXEnvironment::roomHF (C++ member), 795
SFXEnvironment::roomLF (C++ member), 795
SFXEnvironment::roomRolloffFactor (C++ member), 795
SFXFMODEvent::fmodGroup (C++ member), 819
SFXFMODEvent::fmodName (C++ member), 819
SFXFMODEvent::fmodParameterRanges (C++ member), 819
SFXFMODEvent::fmodParameterValues (C++ member), 819
SFXFMODEventGroup::fmodGroup (C++ member), 820
SFXFMODEventGroup::fmodName (C++ member), 820
SFXFMODEventGroup::fmodProject (C++ member), 820
SFXFMODEventGroup::freeData (C++ function), 819
SFXFMODEventGroup::isDataLoaded (C++ function), 819
SFXFMODEventGroup::loadData (C++ function), 819
SFXFMODProject::fileName (C++ member), 820
SFXFMODProject::mediaPath (C++ member), 820
sfxGetActiveStates (C++ function), 814
sfxGetAvailableDevices (C++ function), 814
sfxGetDeviceInfo (C++ function), 814
sfxGetDistanceModel (C++ function), 814
sfxGetDopplerFactor (C++ function), 814
sfxGetRolloffFactor (C++ function), 814
SFXParameter::channel (C++ member), 797
SFXParameter::defaultValue (C++ member), 797
SFXParameter::description (C++ member), 797
SFXParameter::getParameterName (C++ function), 796
SFXParameter::onUpdate (C++ function), 797
SFXParameter::range (C++ member), 797
SFXParameter::reset (C++ function), 797
SFXParameter::value (C++ member), 797
sfxPlay (C++ function), 815
SFXPlayList::delayTimeIn (C++ member), 799
SFXPlayList::delayTimeInVariance (C++ member), 799
SFXPlayList::delayTimeOut (C++ member), 799
SFXPlayList::delayTimeOutVariance (C++ member), 799
SFXPlayList::fadeTimeIn (C++ member), 799
SFXPlayList::fadeTimeInVariance (C++ member), 799
SFXPlayList::fadeTimeOut (C++ member), 799
SFXPlayList::fadeTimeOutVariance (C++ member), 799
SFXPlayList::loopMode (C++ member), 799
SFXPlayList::maxDistance (C++ member), 799
SFXPlayList::maxDistanceVariance (C++ member), 799
SFXPlayList::numSlotsToPlay (C++ member), 799
SFXPlayList::pitchScale (C++ member), 799
SFXPlayList::pitchScaleVariance (C++ member), 799
SFXPlayList::random (C++ member), 799
SFXPlayList::referenceDistance (C++ member), 799
SFXPlayList::referenceDistanceVariance (C++ member), 799
SFXPlayList::repeatCount (C++ member), 799
SFXPlayList::replay (C++ member), 800
SFXPlayList::state (C++ member), 800
SFXPlayList::stateMode (C++ member), 800
SFXPlayList::trace (C++ member), 800
SFXPlayList::track (C++ member), 800
SFXPlayList::transitionIn (C++ member), 800
SFXPlayList::transitionOut (C++ member), 800
SFXPlayList::volumeScale (C++ member), 800
SFXPlayList::volumeScaleVariance (C++ member), 800
sfxPlayOnce (C++ function), 815, 816
SFXProfile::fileName (C++ member), 801
SFXProfile::getSoundDuration (C++ function), 801
SFXProfile::preload (C++ member), 801
sfxSetDistanceModel (C++ function), 816
sfxSetDopplerFactor (C++ function), 816
sfxSetRolloffFactor (C++ function), 817
SFXSound::getDuration (C++ function), 801
SFXSound::getPosition (C++ function), 801
SFXSound::isReady (C++ function), 801
SFXSound::setPosition (C++ function), 802
SFXSource::addMarker (C++ function), 804
SFXSource::addParameter (C++ function), 804
SFXSource::description (C++ member), 807
SFXSource::getAttenuatedVolume (C++ function), 804
SFXSource::getFadeInTime (C++ function), 804
SFXSource::getFadeOutTime (C++ function), 804
SFXSource::getParameter (C++ function), 804
SFXSource::getParameterCount (C++ function), 805
SFXSource::getPitch (C++ function), 805
SFXSource::getStatus (C++ function), 805
SFXSource::getVolume (C++ function), 805
SFXSource::isPaused (C++ function), 805
SFXSource::isPlaying (C++ function), 805
SFXSource::isStopped (C++ function), 805
SFXSource::onParameterValueChange (C++ function), 805
SFXSource::onStatusChange (C++ function), 805
SFXSource::pause (C++ function), 805
SFXSource::removeParameter (C++ function), 806
SFXSource::setCone (C++ function), 806
SFXSource::setFadeTimes (C++ function), 806
SFXSource::setPitch (C++ function), 806
SFXSource::setTransform (C++ function), 806
SFXSource::setVolume (C++ function), 806
SFXSource::statusCallback (C++ member), 807
SFXSource::stop (C++ function), 806
SFXSpace::edge (C++ member), 807
SFXSpace::plane (C++ member), 807

SFXSpace::point (C++ member), 807
SFXSpace::soundAmbience (C++ member), 807
SFXState::activate (C++ function), 808
SFXState::deactivate (C++ function), 808
SFXState::disable (C++ function), 808
SFXState::enable (C++ function), 808
SFXState::excludedStates (C++ member), 808
SFXState::includedStates (C++ member), 808
SFXState::isActive (C++ function), 808
SFXState::isDisabled (C++ function), 808
SFXState::onActivate (C++ function), 808
SFXState::onDeactivate (C++ function), 808
sfxStop (C++ function), 817
sfxStopAndDelete (C++ function), 817
SFXTrack::description (C++ member), 809
SFXTrack::parameters (C++ member), 809
ShaderData::defines (C++ member), 854
ShaderData::DXPixelShaderFile (C++ member), 855
ShaderData::DXVertexShaderFile (C++ member), 855
ShaderData::OGLPixelShaderFile (C++ member), 855
ShaderData::OGLVertexShaderFile (C++ member), 855
ShaderData::pixVersion (C++ member), 855
ShaderData::reload (C++ function), 854
ShaderData::useDevicePixVersion (C++ member), 855
ShapeBase::applyDamage (C++ function), 567
ShapeBase::applyImpulse (C++ function), 567
ShapeBase::applyRepair (C++ function), 567
ShapeBase::blowUp (C++ function), 567
ShapeBase::canCloak (C++ function), 568
ShapeBase::changeMaterial (C++ function), 568
ShapeBase::destroyThread (C++ function), 568
ShapeBase::dumpMeshVisibility (C++ function), 568
ShapeBase::getAIRepairPoint (C++ function), 568
ShapeBase::getCameraFov (C++ function), 568
ShapeBase::getControllingClient (C++ function), 568
ShapeBase::getControllingObject (C++ function), 568
ShapeBase::getDamageFlash (C++ function), 568
ShapeBase::getDamageLevel (C++ function), 568
ShapeBase::getDamagePercent (C++ function), 569
ShapeBase::getDamageState (C++ function), 569
ShapeBase::getDefaultCameraFov (C++ function), 569
ShapeBase::getEnergyLevel (C++ function), 569
ShapeBase::getEnergyPercent (C++ function), 569
ShapeBase::getEyePoint (C++ function), 569
ShapeBase::getEyeTransform (C++ function), 569
ShapeBase::getEyeVector (C++ function), 569
ShapeBase::getImageAltTrigger (C++ function), 569
ShapeBase::getImageAmmo (C++ function), 569
ShapeBase::getImageGenericTrigger (C++ function), 569
ShapeBase::getImageLoaded (C++ function), 570
ShapeBase::getImageScriptAnimPrefix (C++ function), 570
ShapeBase::getImageSkinTag (C++ function), 570
ShapeBase::getImageState (C++ function), 570
ShapeBase::getImageTarget (C++ function), 570
ShapeBase::getImageTrigger (C++ function), 570
ShapeBase::getLookAtPoint (C++ function), 570
ShapeBase::getMaxDamage (C++ function), 570
ShapeBase::getModelFile (C++ function), 570
ShapeBase::getMountedImage (C++ function), 571
ShapeBase::getMountSlot (C++ function), 571
ShapeBase::getMuzzlePoint (C++ function), 571
ShapeBase::getMuzzleVector (C++ function), 571
ShapeBase::getPendingImage (C++ function), 571
ShapeBase::getRechargeRate (C++ function), 571
ShapeBase::getRepairRate (C++ function), 571
ShapeBase::getShapeName (C++ function), 571
ShapeBase::getSkinName (C++ function), 571
ShapeBase::getSlotTransform (C++ function), 572
ShapeBase::getTargetCount (C++ function), 572
ShapeBase::getTargetName (C++ function), 572
ShapeBase::getVelocity (C++ function), 572
ShapeBase::getWhiteOut (C++ function), 572
ShapeBase::hasImageState (C++ function), 572
ShapeBase::isAIControlled (C++ member), 578
ShapeBase::isCloaked (C++ function), 572
ShapeBase::isDestroyed (C++ function), 572
ShapeBase::isDisabled (C++ function), 572
ShapeBase::isEnabled (C++ function), 572
ShapeBase::isHidden (C++ function), 572
ShapeBase::isImageFiring (C++ function), 572
ShapeBase::isImageMounted (C++ function), 573
ShapeBase::mountImage (C++ function), 573
ShapeBase::pauseThread (C++ function), 573
ShapeBase::playAudio (C++ function), 573
ShapeBase::playThread (C++ function), 573
ShapeBase::setAllMeshesHidden (C++ function), 574
ShapeBase::setCameraFov (C++ function), 574
ShapeBase::setCloaked (C++ function), 574
ShapeBase::setDamageFlash (C++ function), 574
ShapeBase::setDamageLevel (C++ function), 574
ShapeBase::setDamageState (C++ function), 574
ShapeBase::setDamageVector (C++ function), 574
ShapeBase::setEnergyLevel (C++ function), 574
ShapeBase::setHidden (C++ function), 574
ShapeBase::setImageAltTrigger (C++ function), 574
ShapeBase::setImageAmmo (C++ function), 575
ShapeBase::setImageGenericTrigger (C++ function), 575
ShapeBase::setImageLoaded (C++ function), 575
ShapeBase::setImageScriptAnimPrefix (C++ function), 575
ShapeBase::setImageTarget (C++ function), 575
ShapeBase::setImageTrigger (C++ function), 575
ShapeBase::setInvincibleMode (C++ function), 576
ShapeBase::setMeshHidden (C++ function), 576
ShapeBase::setRechargeRate (C++ function), 576
ShapeBase::setRepairRate (C++ function), 576
ShapeBase::setShapeName (C++ function), 576

ShapeBase::setSkinName (C++ function), 576
ShapeBase::setThreadDir (C++ function), 576
ShapeBase::setThreadPosition (C++ function), 576
ShapeBase::setThreadTimeScale (C++ function), 576
ShapeBase::setVelocity (C++ function), 577
ShapeBase::setWhiteOut (C++ function), 577
ShapeBase::skin (C++ member), 578
ShapeBase::startFade (C++ function), 577
ShapeBase::stopAudio (C++ function), 577
ShapeBase::stopThread (C++ function), 577
ShapeBase::unmountImage (C++ function), 577
ShapeBase::validateCameraFov (C++ function), 577
ShapeBaseData::cameraCanBank (C++ member), 580
ShapeBaseData::cameraDefaultFov (C++ member), 580
ShapeBaseData::cameraMaxDist (C++ member), 580
ShapeBaseData::cameraMaxFov (C++ member), 580
ShapeBaseData::cameraMinDist (C++ member), 580
ShapeBaseData::cameraMinFov (C++ member), 580
ShapeBaseData::checkDeployPos (C++ function), 578
ShapeBaseData::computeCRC (C++ member), 580
ShapeBaseData::cubeReflectorDesc (C++ member), 580
ShapeBaseData::Debris (C++ member), 580
ShapeBaseData::debrisShapeName (C++ member), 580
ShapeBaseData::density (C++ member), 580
ShapeBaseData::destroyedLevel (C++ member), 580
ShapeBaseData::disabledLevel (C++ member), 580
ShapeBaseData::drag (C++ member), 580
ShapeBaseData::Explosion (C++ member), 580
ShapeBaseData::firstPersonOnly (C++ member), 580
ShapeBaseData::getDeployTransform (C++ function), 578
ShapeBaseData::inheritEnergyFromMount (C++ member), 581
ShapeBaseData::isInvincible (C++ member), 581
ShapeBaseData::mass (C++ member), 581
ShapeBaseData::maxDamage (C++ member), 581
ShapeBaseData::maxEnergy (C++ member), 581
ShapeBaseData::mountedImagesBank (C++ member), 581
ShapeBaseData::observeThroughObject (C++ member), 581
ShapeBaseData::onCollision (C++ function), 578
ShapeBaseData::onDamage (C++ function), 579
ShapeBaseData::onDestroyed (C++ function), 579
ShapeBaseData::onDisabled (C++ function), 579
ShapeBaseData::onEnabled (C++ function), 579
ShapeBaseData::onEndSequence (C++ function), 579
ShapeBaseData::onForceUncloak (C++ function), 579
ShapeBaseData::onImpact (C++ function), 579
ShapeBaseData::onTrigger (C++ function), 580
ShapeBaseData::renderWhenDestroyed (C++ member), 581
ShapeBaseData::repairRate (C++ member), 581
ShapeBaseData::shadowEnable (C++ member), 581
ShapeBaseData::shadowMaxVisibleDistance (C++ member), 581
ShapeBaseData::shadowProjectionDistance (C++ member), 581
ShapeBaseData::shadowSize (C++ member), 581
ShapeBaseData::shadowSphereAdjust (C++ member), 581
ShapeBaseData::shapeFile (C++ member), 581
ShapeBaseData::underwaterExplosion (C++ member), 581
ShapeBaseData::useEyePoint (C++ member), 581
ShapeBaseImageData::accuFire (C++ member), 586
ShapeBaseImageData::animateAllShapes (C++ member), 586
ShapeBaseImageData::animateOnServer (C++ member), 586
ShapeBaseImageData::camShakeAmp (C++ member), 587
ShapeBaseImageData::camShakeFreq (C++ member), 587
ShapeBaseImageData::casing (C++ member), 587
ShapeBaseImageData::cloakable (C++ member), 587
ShapeBaseImageData::computeCRC (C++ member), 587
ShapeBaseImageData::correctMuzzleVector (C++ member), 587
ShapeBaseImageData::correctMuzzleVectorTP (C++ member), 587
ShapeBaseImageData::emap (C++ member), 587
ShapeBaseImageData::eyeOffset (C++ member), 587
ShapeBaseImageData::eyeRotation (C++ member), 587
ShapeBaseImageData::firstPerson (C++ member), 587
ShapeBaseImageData::imageAnimPrefix (C++ member), 587
ShapeBaseImageData::imageAnimPrefixFP (C++ member), 587
ShapeBaseImageData::lightBrightness (C++ member), 587
ShapeBaseImageData::lightColor (C++ member), 587
ShapeBaseImageData::lightDuration (C++ member), 587
ShapeBaseImageData::lightRadius (C++ member), 587
ShapeBaseImageData::lightType (C++ member), 587
ShapeBaseImageData::mass (C++ member), 587
ShapeBaseImageData::maxConcurrentSounds (C++ member), 587
ShapeBaseImageData::minEnergy (C++ member), 588
ShapeBaseImageData::mountPoint (C++ member), 588
ShapeBaseImageData::offset (C++ member), 588
ShapeBaseImageData::onMount (C++ function), 586
ShapeBaseImageData::onUnmount (C++ function), 586
ShapeBaseImageData::Projectile (C++ member), 588
ShapeBaseImageData::rotation (C++ member), 588
ShapeBaseImageData::scriptAnimTransitionTime (C++ member), 588
ShapeBaseImageData::shakeCamera (C++ member), 588

ShapeBaseImageData::shapeFile (C++ member), 588
ShapeBaseImageData::shapeFileFP (C++ member), 588
ShapeBaseImageData::shellExitDir (C++ member), 588
ShapeBaseImageData::shellExitVariance (C++ member), 588
ShapeBaseImageData::shellVelocity (C++ member), 588
ShapeBaseImageData::stateAllowImageChange (C++ member), 588
ShapeBaseImageData::stateAlternateFire (C++ member), 588
ShapeBaseImageData::stateDirection (C++ member), 588
ShapeBaseImageData::stateEjectShell (C++ member), 588
ShapeBaseImageData::stateEmitter (C++ member), 589
ShapeBaseImageData::stateEmitterNode (C++ member), 589
ShapeBaseImageData::stateEmitterTime (C++ member), 589
ShapeBaseImageData::stateEnergyDrain (C++ member), 589
ShapeBaseImageData::stateFire (C++ member), 589
ShapeBaseImageData::stateIgnoreLoadedForReady (C++ member), 589
ShapeBaseImageData::stateLoadedFlag (C++ member), 589
ShapeBaseImageData::stateName (C++ member), 589
ShapeBaseImageData::stateRecoil (C++ member), 589
ShapeBaseImageData::stateReload (C++ member), 589
ShapeBaseImageData::stateScaleAnimation (C++ member), 589
ShapeBaseImageData::stateScaleAnimationFP (C++ member), 589
ShapeBaseImageData::stateScaleShapeSequence (C++ member), 589
ShapeBaseImageData::stateScript (C++ member), 589
ShapeBaseImageData::stateSequence (C++ member), 589
ShapeBaseImageData::stateSequenceNeverTransition (C++ member), 590
ShapeBaseImageData::stateSequenceRandomFlash (C++ member), 590
ShapeBaseImageData::stateSequenceTransitionIn (C++ member), 590
ShapeBaseImageData::stateSequenceTransitionOut (C++ member), 590
ShapeBaseImageData::stateSequenceTransitionTime (C++ member), 590
ShapeBaseImageData::stateShapeSequence (C++ member), 590
ShapeBaseImageData::stateSound (C++ member), 590
ShapeBaseImageData::stateSpinThread (C++ member), 590
ShapeBaseImageData::stateTimeoutValue (C++ member), 590
ShapeBaseImageData::stateTransitionGeneric0In (C++ member), 590
ShapeBaseImageData::stateTransitionGeneric0Out (C++ member), 590
ShapeBaseImageData::stateTransitionGeneric1In (C++ member), 590
ShapeBaseImageData::stateTransitionGeneric1Out (C++ member), 590
ShapeBaseImageData::stateTransitionGeneric2In (C++ member), 590
ShapeBaseImageData::stateTransitionGeneric2Out (C++ member), 590
ShapeBaseImageData::stateTransitionGeneric3In (C++ member), 590
ShapeBaseImageData::stateTransitionGeneric3Out (C++ member), 590
ShapeBaseImageData::stateTransitionOnAltTriggerDown (C++ member), 590
ShapeBaseImageData::stateTransitionOnAltTriggerUp (C++ member), 591
ShapeBaseImageData::stateTransitionOnAmmo (C++ member), 591
ShapeBaseImageData::stateTransitionOnLoaded (C++ member), 591
ShapeBaseImageData::stateTransitionOnMotion (C++ member), 591
ShapeBaseImageData::stateTransitionOnNoAmmo (C++ member), 591
ShapeBaseImageData::stateTransitionOnNoMotion (C++ member), 591
ShapeBaseImageData::stateTransitionOnNoTarget (C++ member), 591
ShapeBaseImageData::stateTransitionOnNotLoaded (C++ member), 591
ShapeBaseImageData::stateTransitionOnNotWet (C++ member), 591
ShapeBaseImageData::stateTransitionOnTarget (C++ member), 591
ShapeBaseImageData::stateTransitionOnTimeout (C++ member), 591
ShapeBaseImageData::stateTransitionOnTriggerDown (C++ member), 591
ShapeBaseImageData::stateTransitionOnTriggerUp (C++ member), 591
ShapeBaseImageData::stateTransitionOnWet (C++ member), 591
ShapeBaseImageData::stateWaitForTimeout (C++ member), 591
ShapeBaseImageData::useEyeNode (C++ member), 591
ShapeBaseImageData::useRemainderDT (C++ member), 591
ShapeBaseImageData::usesEnergy (C++ member), 591
shellExecute (C++ function), 778

SimDataBlock::reloadOnLocalClient (C++ function), 510
SimObject::assignFieldsFrom (C++ function), 502
SimObject::assignPersistentId (C++ function), 502
SimObject::call (C++ function), 502
SimObject::canSave (C++ member), 507
SimObject::canSaveDynamicFields (C++ member), 507
SimObject::className (C++ member), 507
SimObject::clone (C++ function), 502
SimObject::deepClone (C++ function), 502
SimObject::dump (C++ function), 502
SimObject::dumpClassHierarchy (C++ function), 502
SimObject::dumpGroupHierarchy (C++ function), 502
SimObject::dumpMethods (C++ function), 502
SimObject::getCanSave (C++ function), 503
SimObject::getClassName (C++ function), 503
SimObject::getClassNamespace (C++ function), 503
SimObject::getDebugInfo (C++ function), 503
SimObject::getDeclarationLine (C++ function), 503
SimObject::getDynamicField (C++ function), 503
SimObject::getDynamicFieldCount (C++ function), 503
SimObject::getField (C++ function), 503
SimObject::getFieldCount (C++ function), 503
SimObject::getFieldType (C++ function), 503
SimObject::getFieldValue (C++ function), 503
SimObject::getFilename (C++ function), 504
SimObject::getGroup (C++ function), 504
SimObject::getId (C++ function), 504
SimObject::getInternalName (C++ function), 504
SimObject::getName (C++ function), 504
SimObject::getSuperClassNamespace (C++ function), 504
SimObject::hidden (C++ member), 507
SimObject::internalName (C++ member), 507
SimObject::isChildOfGroup (C++ function), 504
SimObject::isEditorOnly (C++ function), 504
SimObject::isExpanded (C++ function), 504
SimObject::isField (C++ function), 504
SimObject::isInNamespaceHierarchy (C++ function), 504
SimObject::isMemberOfClass (C++ function), 505
SimObject::isMethod (C++ function), 505
SimObject::isNameChangeAllowed (C++ function), 505
SimObject::isSelected (C++ function), 505
SimObject::locked (C++ member), 507
SimObject::name (C++ member), 507
SimObject::parentGroup (C++ member), 507
SimObject::persistentId (C++ member), 507
SimObject::save (C++ function), 505
SimObject::schedule (C++ function), 505
SimObject::setCanSave (C++ function), 505
SimObject::setClassNamespace (C++ function), 505
SimObject::setEditorOnly (C++ function), 505
SimObject::setFieldType (C++ function), 505
SimObject::setFieldValue (C++ function), 506
SimObject::setFilename (C++ function), 506
SimObject::setHidden (C++ function), 506
SimObject::setInternalName (C++ function), 506
SimObject::setIsExpanded (C++ function), 506
SimObject::setIsSelected (C++ function), 506
SimObject::setLocked (C++ function), 506
SimObject::setName (C++ function), 506
SimObject::setNameChangeAllowed (C++ function), 506
SimObject::setSuperClassNamespace (C++ function), 506
SimObject::superClass (C++ member), 507
SimpleMessageEvent::msg (C++ function), 766
SimpleNetObject::setMessage (C++ function), 766
SimSet::acceptsAsChild (C++ function), 507
SimSet::add (C++ function), 508
SimSet::bringToFront (C++ function), 508
SimSet::callOnChildren (C++ function), 508
SimSet::callOnChildrenNoRecurse (C++ function), 508
SimSet::clear (C++ function), 508
SimSet::deleteAllObjects (C++ function), 508
SimSet::findObjectByInternalName (C++ function), 508
SimSet::getCount (C++ function), 508
SimSet::getFullCount (C++ function), 508
SimSet::getObject (C++ function), 508
SimSet::getObjectIndex (C++ function), 508
SimSet::getRandom (C++ function), 509
SimSet::isMember (C++ function), 509
SimSet::listObjects (C++ function), 509
SimSet::onObjectAdded (C++ function), 509
SimSet::onObjectRemoved (C++ function), 509
SimSet::pushToBack (C++ function), 509
SimSet::remove (C++ function), 509
SimSet::reorderChild (C++ function), 509
SimSet::sort (C++ function), 509
SimXMLDocument::addComment (C++ function), 339
SimXMLDocument::addData (C++ function), 339
SimXMLDocument::addHeader (C++ function), 339
SimXMLDocument::addNewElement (C++ function), 340
SimXMLDocument::addText (C++ function), 340
SimXMLDocument::attribute (C++ function), 340
SimXMLDocument::attributeExists (C++ function), 340
SimXMLDocument::attributeF32 (C++ function), 340
SimXMLDocument::attributeS32 (C++ function), 340
SimXMLDocument::clear (C++ function), 341
SimXMLDocument::clearError (C++ function), 341
SimXMLDocument::elementValue (C++ function), 341
SimXMLDocument::firstAttribute (C++ function), 341
SimXMLDocument::getData (C++ function), 341
SimXMLDocument::getErrorDesc (C++ function), 341
SimXMLDocument::getText (C++ function), 341
SimXMLDocument::lastAttribute (C++ function), 342
SimXMLDocument::loadFile (C++ function), 342

SimXMLDocument::nextAttribute (C++ function), 342
SimXMLDocument::nextSiblingElement (C++ function), 342
SimXMLDocument::parse (C++ function), 342
SimXMLDocument::popElement (C++ function), 342
SimXMLDocument::prevAttribute (C++ function), 342
SimXMLDocument::pushChildElement (C++ function), 342
SimXMLDocument::pushFirstChildElement (C++ function), 342
SimXMLDocument::pushNewElement (C++ function), 343
SimXMLDocument::readComment (C++ function), 343
SimXMLDocument::removeText (C++ function), 343
SimXMLDocument::reset (C++ function), 343
SimXMLDocument::saveFile (C++ function), 343
SimXMLDocument::setAttribute (C++ function), 343
SimXMLDocument::setObjectAttributes (C++ function), 343
SkyBox::drawBottom (C++ member), 687
SkyBox::fogBandHeight (C++ member), 687
SkyBox::Material (C++ member), 687
SkyBox::postApply (C++ member), 687
snapToggle (C++ function), 498
SpawnSphere::autoSpawn (C++ member), 593
SpawnSphere::indoorWeight (C++ member), 593
SpawnSphere::onAdd (C++ function), 592
SpawnSphere::outdoorWeight (C++ member), 593
SpawnSphere::radius (C++ member), 593
SpawnSphere::spawnClass (C++ member), 593
SpawnSphere::spawnDatablock (C++ member), 593
SpawnSphere::spawnObject (C++ function), 592
SpawnSphere::spawnProperties (C++ member), 593
SpawnSphere::spawnScript (C++ member), 593
SpawnSphere::spawnTransform (C++ member), 593
SpawnSphere::sphereWeight (C++ member), 593
SplashData::acceleration (C++ member), 648
SplashData::colors (C++ member), 648
SplashData::delayMS (C++ member), 648
SplashData::delayVariance (C++ member), 648
SplashData::ejectionAngle (C++ member), 648
SplashData::ejectionFreq (C++ member), 648
SplashData::emitter (C++ member), 648
SplashData::Explosion (C++ member), 648
SplashData::height (C++ member), 648
SplashData::lifetimeMS (C++ member), 648
SplashData::lifetimeVariance (C++ member), 648
SplashData::numSegments (C++ member), 648
SplashData::ringLifetime (C++ member), 648
SplashData::scale (C++ member), 648
SplashData::soundProfile (C++ member), 648
SplashData::startRadius (C++ member), 648
SplashData::texFactor (C++ member), 648
SplashData::texture (C++ member), 648
SplashData::texWrap (C++ member), 648
SplashData::times (C++ member), 648
SplashData::velocity (C++ member), 648
SplashData::width (C++ member), 649
SpotLight::innerAngle (C++ member), 862
SpotLight::outerAngle (C++ member), 862
SpotLight::range (C++ member), 862
StartClientReplication (C++ function), 707
startFileChangeNotifications (C++ function), 353
StartFoliageReplication (C++ function), 707
startsWith (C++ function), 367
startVideoCapture (C++ function), 827
StaticShapeData::dynamicType (C++ member), 594
StaticShapeData::noIndividualDamage (C++ member), 595
stopFileChangeNotifications (C++ function), 353
stopSampling (C++ function), 827
stopVideoCapture (C++ function), 827
strasc (C++ function), 368
strchr (C++ function), 368
strchrpos (C++ function), 368
strcmp (C++ function), 368
StreamObject::copyFrom (C++ function), 344
StreamObject::getPosition (C++ function), 344
StreamObject::getStatus (C++ function), 344
StreamObject::getStreamSize (C++ function), 345
StreamObject::isEOF (C++ function), 345
StreamObject::isEOS (C++ function), 346
StreamObject::readLine (C++ function), 346
StreamObject::readLongString (C++ function), 346
StreamObject::readString (C++ function), 347
StreamObject::readSTString (C++ function), 347
StreamObject::setPosition (C++ function), 347
StreamObject::writeLine (C++ function), 347
StreamObject::writeLongString (C++ function), 348
StreamObject::writeString (C++ function), 348
strformat (C++ function), 368
stricmp (C++ function), 369
strinatcmp (C++ function), 369
stripChars (C++ function), 369
StripMLControlChars (C++ function), 428
stripTrailingNumber (C++ function), 370
strIsMatchExpr (C++ function), 370
strIsMatchMultipleExpr (C++ function), 370
strlen (C++ function), 370
strlwr (C++ function), 370
strnatcmp (C++ function), 371
strpos (C++ function), 371
strrchr (C++ function), 371
strrchrpos (C++ function), 372
strrepeat (C++ function), 372
strreplace (C++ function), 372
strstr (C++ function), 372
strupr (C++ function), 373

Sun::ambient (C++ member), 687
Sun::animate (C++ member), 687
Sun::apply (C++ member), 687
Sun::attenuationRatio (C++ member), 687
Sun::azimuth (C++ member), 687
Sun::brightness (C++ member), 687
Sun::castShadows (C++ member), 687
Sun::color (C++ member), 687
Sun::cookie (C++ member), 687
Sun::coronaEnabled (C++ member), 687
Sun::coronaMaterial (C++ member), 687
Sun::coronaScale (C++ member), 687
Sun::coronaTint (C++ member), 687
Sun::coronaUseLightColor (C++ member), 688
Sun::elevation (C++ member), 688
Sun::fadeStartDistance (C++ member), 688
Sun::flareScale (C++ member), 688
Sun::flareType (C++ member), 688
Sun::includeLightmappedGeometryInShadow (C++ member), 688
Sun::lastSplitTerrainOnly (C++ member), 688
Sun::logWeight (C++ member), 688
Sun::numSplits (C++ member), 688
Sun::overDarkFactor (C++ member), 688
Sun::representedInLightmap (C++ member), 688
Sun::shadowDarkenColor (C++ member), 688
Sun::shadowDistance (C++ member), 688
Sun::shadowSoftness (C++ member), 688
Sun::shadowType (C++ member), 688
Sun::texSize (C++ member), 688

T

TCPObject::connect (C++ function), 768
TCPObject::disconnect (C++ function), 769
TCPObject::listen (C++ function), 769
TCPObject::onConnected (C++ function), 769
TCPObject::onConnectFailed (C++ function), 769
TCPObject::onConnectionRequest (C++ function), 769
TCPObject::onDisconnect (C++ function), 769
TCPObject::onDNSFailed (C++ function), 769
TCPObject::onDNSResolved (C++ function), 769
TCPObject::onLine (C++ function), 769
TCPObject::send (C++ function), 769
telnetSetParameters (C++ function), 308
TerrainBlock::baseTexSize (C++ member), 696
TerrainBlock::castShadows (C++ member), 696
TerrainBlock::createNew (C++ member), 696
TerrainBlock::debugRender (C++ member), 698
TerrainBlock::exportHeightMap (C++ function), 696
TerrainBlock::exportLayerMaps (C++ function), 696
TerrainBlock::import (C++ function), 696
TerrainBlock::lightMapSize (C++ member), 696
TerrainBlock::save (C++ function), 696
TerrainBlock::screenError (C++ member), 696
TerrainBlock::squareSize (C++ member), 696
TerrainBlock::terrainFile (C++ member), 697
TerrainMaterial::detailDistance (C++ member), 716
TerrainMaterial::detailMap (C++ member), 716
TerrainMaterial::detailSize (C++ member), 716
TerrainMaterial::detailStrength (C++ member), 716
TerrainMaterial::diffuseMap (C++ member), 716
TerrainMaterial::diffuseSize (C++ member), 716
TerrainMaterial::macroDistance (C++ member), 716
TerrainMaterial::macroMap (C++ member), 716
TerrainMaterial::macroSize (C++ member), 716
TerrainMaterial::macroStrength (C++ member), 716
TerrainMaterial::normalMap (C++ member), 717
TerrainMaterial::parallaxScale (C++ member), 717
TerrainMaterial::useSideProjection (C++ member), 717
TheoraTextureObject::loop (C++ member), 825
TheoraTextureObject::pause (C++ function), 825
TheoraTextureObject::play (C++ function), 825
TheoraTextureObject::SFXDescription (C++ member), 826
TheoraTextureObject::stop (C++ function), 825
TheoraTextureObject::texTargetName (C++ member), 826
TheoraTextureObject::theoraFile (C++ member), 826
TimeOfDay::addTimeOfDayEvent (C++ function), 717
TimeOfDay::animate (C++ function), 717
TimeOfDay::axisTilt (C++ member), 717
TimeOfDay::azimuthOverride (C++ member), 717
TimeOfDay::dayLength (C++ member), 717
TimeOfDay::dayScale (C++ member), 717
TimeOfDay::nightScale (C++ member), 718
TimeOfDay::play (C++ member), 718
TimeOfDay::setDayLength (C++ function), 717
TimeOfDay::setPlay (C++ function), 717
TimeOfDay::setTimeOfDay (C++ function), 717
TimeOfDay::startTime (C++ member), 718
TimeOfDay::time (C++ member), 718
trace (C++ function), 308
Trigger::enterCommand (C++ member), 595
Trigger::getNumObjects (C++ function), 595
Trigger::getObject (C++ function), 595
Trigger::leaveCommand (C++ member), 595
Trigger::onAdd (C++ function), 595
Trigger::onRemove (C++ function), 595
Trigger::polyhedron (C++ member), 595
Trigger::renderTriggers (C++ member), 625
Trigger::tickCommand (C++ member), 595
TriggerData::clientSide (C++ member), 596
TriggerData::onEnterTrigger (C++ function), 596
TriggerData::onLeaveTrigger (C++ function), 596
TriggerData::onTickTrigger (C++ function), 596
TriggerData::tickPeriodMS (C++ member), 596
trim (C++ function), 373

TSShapeConstructor::addCollisionDetail (C++ function), 598
TSShapeConstructor::addImposter (C++ function), 599
TSShapeConstructor::addMesh (C++ function), 599
TSShapeConstructor::addNode (C++ function), 600
TSShapeConstructor::addPrimitive (C++ function), 600
TSShapeConstructor::addSequence (C++ function), 601
TSShapeConstructor::addTrigger (C++ function), 601
TSShapeConstructor::adjustCenter (C++ member), 614
TSShapeConstructor::adjustFloor (C++ member), 614
TSShapeConstructor::alwaysImport (C++ member), 614
TSShapeConstructor::alwaysImportMesh (C++ member), 615
TSShapeConstructor::baseShape (C++ member), 615
TSShapeConstructor::dumpShape (C++ function), 602
TSShapeConstructor::forceUpdateMaterials (C++ member), 615
TSShapeConstructor::getBounds (C++ function), 602
TSShapeConstructor::getDetailLevelCount (C++ function), 602
TSShapeConstructor::getDetailLevelIndex (C++ function), 602
TSShapeConstructor::getDetailLevelName (C++ function), 602
TSShapeConstructor::getDetailLevelSize (C++ function), 602
TSShapeConstructor::getImposterDetailLevel (C++ function), 602
TSShapeConstructor::getImposterSettings (C++ function), 603
TSShapeConstructor::getMeshCount (C++ function), 603
TSShapeConstructor::getMeshMaterial (C++ function), 603
TSShapeConstructor::getMeshName (C++ function), 603
TSShapeConstructor::getMeshSize (C++ function), 603
TSShapeConstructor::getMeshType (C++ function), 604
TSShapeConstructor::getNodeChildCount (C++ function), 604
TSShapeConstructor::getNodeChildName (C++ function), 604
TSShapeConstructor::getNodeCount (C++ function), 605
TSShapeConstructor::getNodeIndex (C++ function), 605
TSShapeConstructor::getNodeName (C++ function), 605
TSShapeConstructor::getNodeObjectCount (C++ function), 605
TSShapeConstructor::getNodeObjectName (C++ function), 605
TSShapeConstructor::getNodeParentName (C++ function), 606
TSShapeConstructor::getNodeTransform (C++ function), 606
TSShapeConstructor::getObjectCount (C++ function), 606
TSShapeConstructor::getObjectIndex (C++ function), 606
TSShapeConstructor::getObjectName (C++ function), 606
TSShapeConstructor::getObjectNode (C++ function), 607
TSShapeConstructor::getSequenceBlend (C++ function), 607
TSShapeConstructor::getSequenceCount (C++ function), 607
TSShapeConstructor::getSequenceCyclic (C++ function), 607
TSShapeConstructor::getSequenceFrameCount (C++ function), 607
TSShapeConstructor::getSequenceGroundSpeed (C++ function), 607
TSShapeConstructor::getSequenceIndex (C++ function), 608
TSShapeConstructor::getSequenceName (C++ function), 608
TSShapeConstructor::getSequencePriority (C++ function), 608
TSShapeConstructor::getSequenceSource (C++ function), 608
TSShapeConstructor::getTargetCount (C++ function), 608
TSShapeConstructor::getTargetName (C++ function), 608
TSShapeConstructor::getTrigger (C++ function), 609
TSShapeConstructor::getTriggerCount (C++ function), 609
TSShapeConstructor::ignoreNodeScale (C++ member), 615
TSShapeConstructor::lodType (C++ member), 615
TSShapeConstructor::matNamePrefix (C++ member), 615
TSShapeConstructor::neverImport (C++ member), 615
TSShapeConstructor::neverImportMesh (C++ member), 615
TSShapeConstructor::notifyShapeChanged (C++ function), 609
TSShapeConstructor::onLoad (C++ function), 609
TSShapeConstructor::onUnload (C++ function), 609
TSShapeConstructor::removeDetailLevel (C++ function), 609
TSShapeConstructor::removeImposter (C++ function), 609
TSShapeConstructor::removeMesh (C++ function), 609
TSShapeConstructor::removeNode (C++ function), 610
TSShapeConstructor::removeObject (C++ function), 610
TSShapeConstructor::removeSequence (C++ function), 610
TSShapeConstructor::removeTrigger (C++ function), 610
TSShapeConstructor::renameDetailLevel (C++ function), 610

TSShapeConstructor::renameNode (C++ function), 611
TSShapeConstructor::renameObject (C++ function), 611
TSShapeConstructor::renameSequence (C++ function), 611
TSShapeConstructor::saveShape (C++ function), 611
TSShapeConstructor::sequence (C++ member), 615
TSShapeConstructor::setBounds (C++ function), 611
TSShapeConstructor::setDetailLevelSize (C++ function), 611
TSShapeConstructor::setMeshMaterial (C++ function), 612
TSShapeConstructor::setMeshSize (C++ function), 612
TSShapeConstructor::setMeshType (C++ function), 612
TSShapeConstructor::setNodeParent (C++ function), 612
TSShapeConstructor::setNodeTransform (C++ function), 613
TSShapeConstructor::setObjectNode (C++ function), 613
TSShapeConstructor::setSequenceBlend (C++ function), 613
TSShapeConstructor::setSequenceCyclic (C++ function), 613
TSShapeConstructor::setSequenceGroundSpeed (C++ function), 614
TSShapeConstructor::setSequencePriority (C++ function), 614
TSShapeConstructor::singleDetailSize (C++ member), 616
TSShapeConstructor::unit (C++ member), 616
TSShapeConstructor::upAxis (C++ member), 616
TSShapeConstructor::writeChangeSet (C++ function), 614
TSStatic::allowPlayerStep (C++ member), 617
TSStatic::changeMaterial (C++ function), 617
TSStatic::collisionType (C++ member), 617
TSStatic::decalType (C++ member), 617
TSStatic::forceDetail (C++ member), 617
TSStatic::getModelFile (C++ function), 617
TSStatic::getTargetCount (C++ function), 617
TSStatic::getTargetName (C++ function), 617
TSStatic::meshCulling (C++ member), 617
TSStatic::originSort (C++ member), 618
TSStatic::playAmbient (C++ member), 618
TSStatic::renderNormals (C++ member), 618
TSStatic::shapeName (C++ member), 618
TSStatic::skin (C++ member), 618
TurretShape::doRespawn (C++ function), 621
TurretShape::getAllowManualFire (C++ function), 621
TurretShape::getAllowManualRotation (C++ function), 621
TurretShape::getState (C++ function), 621
TurretShape::getTurretEulerRotation (C++ function), 621
TurretShape::respawn (C++ member), 622
TurretShape::setAllowManualFire (C++ function), 621
TurretShape::setAllowManualRotation (C++ function), 621
TurretShape::setTurretEulerRotation (C++ function), 621
TurretShapeData::cameraOffset (C++ member), 622
TurretShapeData::headingRate (C++ member), 622
TurretShapeData::maxHeading (C++ member), 622
TurretShapeData::maxPitch (C++ member), 622
TurretShapeData::minPitch (C++ member), 622
TurretShapeData::onMountObject (C++ function), 622
TurretShapeData::onStickyCollision (C++ function), 622
TurretShapeData::onUnmountObject (C++ function), 622
TurretShapeData::pitchRate (C++ member), 622
TurretShapeData::startLoaded (C++ member), 623
TurretShapeData::weaponLinkType (C++ member), 623
TurretShapeData::zRotOnly (C++ member), 623

U

unitTest_runTests (C++ function), 306
unregisterMessageListener (C++ function), 317
unregisterMessageQueue (C++ function), 317

V

validateMemory (C++ function), 308
VectorAdd (C++ function), 360
VectorCross (C++ function), 360
VectorDist (C++ function), 360
VectorDot (C++ function), 361
VectorLen (C++ function), 361
VectorLerp (C++ function), 361
VectorNormalize (C++ function), 362
VectorOrthoBasis (C++ function), 362
VectorScale (C++ function), 362
VectorSub (C++ function), 363
Vehicle::disableMove (C++ member), 673
Vehicle::workingQueryBoxSizeMultiplier (C++ member), 673
Vehicle::workingQueryBoxStaleThreshold (C++ member), 674
VehicleData::bodyFriction (C++ member), 675
VehicleData::bodyRestitution (C++ member), 675
VehicleData::cameraDecay (C++ member), 675
VehicleData::cameraLag (C++ member), 675
VehicleData::cameraOffset (C++ member), 675
VehicleData::cameraRoll (C++ member), 675
VehicleData::collDamageMultiplier (C++ member), 675
VehicleData::collDamageThresholdVel (C++ member), 675
VehicleData::collisionTol (C++ member), 675
VehicleData::contactTol (C++ member), 675
VehicleData::damageEmitter (C++ member), 675
VehicleData::damageEmitterOffset (C++ member), 675
VehicleData::damageLevelTolerance (C++ member), 675
VehicleData::dustEmitter (C++ member), 675
VehicleData::dustHeight (C++ member), 675

VehicleData::exitingWater (C++ member), 676
VehicleData::exitSplashSoundVelocity (C++ member), 676
VehicleData::hardImpactSound (C++ member), 676
VehicleData::hardImpactSpeed (C++ member), 676
VehicleData::hardSplashSoundVelocity (C++ member), 676
VehicleData::impactWaterEasy (C++ member), 676
VehicleData::impactWaterHard (C++ member), 676
VehicleData::impactWaterMedium (C++ member), 676
VehicleData::integration (C++ member), 676
VehicleData::jetEnergyDrain (C++ member), 676
VehicleData::jetForce (C++ member), 676
VehicleData::massBox (C++ member), 676
VehicleData::massCenter (C++ member), 676
VehicleData::maxDrag (C++ member), 676
VehicleData::maxSteeringAngle (C++ member), 676
VehicleData::mediumSplashSoundVelocity (C++ member), 676
VehicleData::minDrag (C++ member), 676
VehicleData::minImpactSpeed (C++ member), 676
VehicleData::minJetEnergy (C++ member), 676
VehicleData::minRollSpeed (C++ member), 677
VehicleData::numDmgEmitterAreas (C++ member), 677
VehicleData::onEnterLiquid (C++ function), 674
VehicleData::onLeaveLiquid (C++ function), 674
VehicleData::powerSteering (C++ member), 677
VehicleData::softImpactSound (C++ member), 677
VehicleData::softImpactSpeed (C++ member), 677
VehicleData::softSplashSoundVelocity (C++ member), 677
VehicleData::splashEmitter (C++ member), 677
VehicleData::splashFreqMod (C++ member), 677
VehicleData::splashVelEpsilon (C++ member), 677
VehicleData::steeringReturn (C++ member), 677
VehicleData::steeringReturnSpeedScale (C++ member), 677
VehicleData::triggerDustHeight (C++ member), 677
VehicleData::waterWakeSound (C++ member), 677

W

warn (C++ function), 311
WaterBlock::gridElementSize (C++ member), 690
WaterBlock::gridSize (C++ member), 690
WaterObject::baseColor (C++ member), 690
WaterObject::clarity (C++ member), 690
WaterObject::cubemap (C++ member), 690
WaterObject::density (C++ member), 690
WaterObject::depthGradientMax (C++ member), 690
WaterObject::depthGradientTex (C++ member), 690
WaterObject::distortEndDist (C++ member), 690
WaterObject::distortFullDepth (C++ member), 690
WaterObject::distortStartDist (C++ member), 690
WaterObject::emissive (C++ member), 690
WaterObject::foamAmbientLerp (C++ member), 691
WaterObject::foamDir (C++ member), 691
WaterObject::foamMaxDepth (C++ member), 691
WaterObject::foamOpacity (C++ member), 691
WaterObject::foamRippleInfluence (C++ member), 691
WaterObject::foamSpeed (C++ member), 691
WaterObject::foamTex (C++ member), 691
WaterObject::foamTexScale (C++ member), 691
WaterObject::fresnelBias (C++ member), 691
WaterObject::fresnelPower (C++ member), 691
WaterObject::fullReflect (C++ member), 691
WaterObject::liquidType (C++ member), 691
WaterObject::overallFoamOpacity (C++ member), 691
WaterObject::overallRippleMagnitude (C++ member), 691
WaterObject::overallWaveMagnitude (C++ member), 691
WaterObject::reflectDetailAdjust (C++ member), 691
WaterObject::reflectivity (C++ member), 691
WaterObject::reflectMaxRateMs (C++ member), 691
WaterObject::reflectNormalUp (C++ member), 691
WaterObject::reflectPriority (C++ member), 691
WaterObject::reflectTexSize (C++ member), 691
WaterObject::rippleDir (C++ member), 691
WaterObject::rippleMagnitude (C++ member), 691
WaterObject::rippleSpeed (C++ member), 691
WaterObject::rippleTex (C++ member), 692
WaterObject::rippleTexScale (C++ member), 692
WaterObject::soundAmbience (C++ member), 692
WaterObject::specularColor (C++ member), 692
WaterObject::specularPower (C++ member), 692
WaterObject::underwaterColor (C++ member), 692
WaterObject::useOcclusionQuery (C++ member), 692
WaterObject::viscosity (C++ member), 692
WaterObject::waterFogDensity (C++ member), 692
WaterObject::waterFogDensityOffset (C++ member), 692
WaterObject::waveDir (C++ member), 692
WaterObject::waveMagnitude (C++ member), 692
WaterObject::waveSpeed (C++ member), 692
WaterObject::wetDarkening (C++ member), 692
WaterObject::wetDepth (C++ member), 692
WaterObject::wireframe (C++ member), 693
WaterPlane::gridElementSize (C++ member), 693
WaterPlane::gridSize (C++ member), 693
WayPoint::markerName (C++ member), 718
WayPoint::team (C++ member), 718
WheeledVehicle::getWheelCount (C++ function), 679
WheeledVehicle::setWheelPowered (C++ function), 680
WheeledVehicle::setWheelSpring (C++ function), 680
WheeledVehicle::setWheelSteering (C++ function), 680
WheeledVehicle::setWheelTire (C++ function), 680
WheeledVehicleData::brakeTorque (C++ member), 681
WheeledVehicleData::engineBrake (C++ member), 681
WheeledVehicleData::engineSound (C++ member), 681

WheeledVehicleData::engineTorque (C++ member), 681
WheeledVehicleData::jetSound (C++ member), 681
WheeledVehicleData::maxWheelSpeed (C++ member), 681
WheeledVehicleData::squealSound (C++ member), 681
WheeledVehicleData::tireEmitter (C++ member), 681
WheeledVehicleData::WheelImpactSound (C++ member), 681
WheeledVehicleSpring::antiSwayForce (C++ member), 681
WheeledVehicleSpring::damping (C++ member), 681
WheeledVehicleSpring::force (C++ member), 681
WheeledVehicleSpring::length (C++ member), 681
WheeledVehicleTire::kineticFriction (C++ member), 682
WheeledVehicleTire::lateralDamping (C++ member), 682
WheeledVehicleTire::lateralForce (C++ member), 682
WheeledVehicleTire::lateralRelaxation (C++ member), 682
WheeledVehicleTire::longitudinalDamping (C++ member), 682
WheeledVehicleTire::longitudinalForce (C++ member), 682
WheeledVehicleTire::longitudinalRelaxation (C++ member), 682
WheeledVehicleTire::mass (C++ member), 682
WheeledVehicleTire::radius (C++ member), 682
WheeledVehicleTire::restitution (C++ member), 682
WheeledVehicleTire::shapeFile (C++ member), 682
WheeledVehicleTire::staticFriction (C++ member), 682
writeFontCache (C++ function), 829

Z

ZipObject::addFile (C++ function), 348
ZipObject::closeArchive (C++ function), 349
ZipObject::closeFile (C++ function), 349
ZipObject::deleteFile (C++ function), 349
ZipObject::extractFile (C++ function), 349
ZipObject::getFileEntry (C++ function), 349
ZipObject::getFileEntryCount (C++ function), 349
ZipObject::openArchive (C++ function), 349
ZipObject::openFileForRead (C++ function), 350
ZipObject::openFileForWrite (C++ function), 350
Zone::ambientLightColor (C++ member), 719
Zone::dumpZoneState (C++ function), 719
Zone::edge (C++ member), 719
Zone::getZoneId (C++ function), 719
Zone::plane (C++ member), 719
Zone::point (C++ member), 719
Zone::soundAmbience (C++ member), 719
Zone::useAmbientLightColor (C++ member), 719
Zone::zoneGroup (C++ member), 719