1 Introduction
   1.1 Briefing ................................................. 3
   1.2 A Deeper Look ......................................... 4
   1.3 Compatibility Notes ................................. 7

2 Tutorials ......................................................
   2.1 Getting Started ......................................... 9
   2.2 Introducing Python ................................. 20

3 Editors ..........................................................
   3.1 Properties Editor ................................. 23
   3.2 Logic Editor ........................................... 61

4 Datablocks ...................................................
   4.1 Armature ................................................. 65
   4.2 Camera .................................................. 65
   4.3 Empty ................................................... 65
   4.4 Group .................................................... 65
   4.5 Lamp ..................................................... 66
   4.6 Mesh ....................................................... 66
   4.7 Object .................................................... 66
   4.8 Image ...................................................... 68
   4.9 Library .................................................... 68
   4.10 Sound ..................................................... 68
   4.11 Action ..................................................... 68
   4.12 Material ................................................ 68
   4.13 Scene ..................................................... 68
   4.14 Texture .................................................. 68
   4.15 World ..................................................... 68

5 Logic ..........................................................
   5.1 Introduction ............................................. 69
   5.2 Sensors .................................................... 70
   5.3 Controllers .............................................. 86
   5.4 Actuators ................................................ 95
   5.5 Properties ............................................... 121
   5.6 States ..................................................... 122
Welcome to UPBGE’s Documentation! Here you will find definitions of the available tools and features in UPBGE, step-by-step tutorials to certain tasks and the Python API for game logic programming with detailed information (and examples in some cases).

Visit the UPBGE’s website or contribute to this manual.
This section gives an overview about UPBGE’s capabilities, features, history and some differences between it and BGE, but not directly comparing UPBGE to BGE.

1.1 Briefing

1.1.1 History

The Uchronia Project Blender Game Engine (UPBGE) is a Blender’s built-in tool for real-time projects, from architectural visualizations and simulations to games.

Originally a fork from the Blender Foundation’s Blender Game Engine, became independent with the Blender Foundation’s announcement of BGE’s removal and replacement with a new Interactive Engine. With this independency, the UPBGE’s developers have freedom to change and add features that could not be changed before (because the possibility of an official Blender merge, now discarded).

1.1.2 Features

UPBGE have lots of new features, improvements and bugs fixed that BGE haven’t. Some features that UPBGE supports are:

- Realtime advanced physics powered by Bullet, including rigid bodies, soft bodies, obstacle simulation and path finding.
- Fully integrated audio engine powered by OpenAL and SDL, supporting 3D sound and sound effects.
- Easy and straightforward visual logic system.
- Powerful Python language bindings, allowing support to even more libraries through the use of PyPI.
- Development process entirely inside Blender, without needing to import/export assets.
- Execution of game in Blender’s viewport (for fast previewing) or on a standalone executable.
• Blender’s Linked Libraries feature, allowing to organize projects in multiple blend files.
• OpenGL custom shaders for visual effects and post processing.
• Fake or realtime reflections setting directly through the interface.

1.1.3 Development

UPBGE is maintained by a group of developers in their spare time and its community. You can contribute to UPBGE if you code in C++ or Python: just open a pull request, submit your changes and wait for the reviewers. Also, even if you don’t code, you can contribute by submitting bug reports, feature requests and participating discussions on issues.

1.2 A Deeper Look

1.2.1 Use Cases

UPBGE allows you to create real-time interactive 3D applications or simulations. This allows you to create almost any type of interactive project, like architectural presentations, virtual prototypes for robotic projects, physics simulation projects, simple and complex games, and much more.

Much can be achieved through UPBGE by default, as it provides a blank canvas full of features to be used, and even more can be accomplished by extending these features according to your needs. For example, if you’re working with robotics and needs to send or receive commands through a USB port, you can install the PySerial Python module for use with UPBGE. Or if you need a graphical feature that can’t be accomplished through the default UPBGE’s capabilities, you can write your own OpenGL shaders. The list goes on, and there’s a big chance that the project you aim to can be brought to life with UPBGE, using its features and abilities to be extended.

1.2.2 Sample Games

Here are some examples of games made with BGE/UPBGE.

Fig. 1: Screenshot from “Engine Roar”, produced with UPBGE.
1.2. A Deeper Look

Fig. 2: Screenshot from “Cave Miner”, produced with UPBGE.

Fig. 3: Screenshot from “Dead Cyborg”, produced with Blender Game Engine.
1.2.3 Under The Hood

UPBGE oversees a game loop, which processes logic, sound, physics and rendering simulations in sequential order. The engine is written in C++.

By default, the user has access to a powerful and high-level visual logic programming interface. The visual programming in UPBGE provides deep interaction with the simulation, and its functionality can be extended through Python scripting. It is designed to abstract the complex engine features into a simple user interface, which does not require experience with Programming.

UPBGE is closely integrated with the existing code base of Blender, which permits quick transitions between the traditional modeling feature set and game-specific functionality provided by the program. In this sense, the UPBGE can be efficiently used in all areas of game design, from prototyping to final release.

UPBGE can simulate content within Blender, however it also includes the ability to export a binary run-time to Linux, macOS, and MS-Windows.

There are a number of powerful libraries the UPBGE takes advantage of:

- **Audaspace**: A sound library for control of audio. Uses OpenAL or SDL.
- **Bullet**: A physics engine featuring 3D collision detection, soft body dynamics and rigid body dynamics.
- **Detour**: A pathfinding and spatial reasoning toolkit.
- **Recast**: A state of the art navigation mesh construction tool set for games.

1.2.4 Project Development Process

When creating a game or simulation in UPBGE, there are four essential steps:

1. Create visual elements that can be rendered. Usually, 3D models.
2. Enable interaction within the scene using logic to enable custom behavior and determine how it is invoked.
3. Create one (or more) camera to give a frustum from which to render the scene, and modify the parameters to support the environment in which the game will be displayed, such as Stereo rendering.
4. Launch the game, using the internal player or exporting a runtime to the appropriate platform.

1.3 Compatibility Notes

1.3.1 UPBGE VS Blender

UPBGE is fully integrated into Blender environment, but it doesn’t mean it supports all the features that Blender provides. Here’s some compatibility info about various features present in Blender relative to UPBGE.

Object types supported

- Armature
- Camera
- Empty
- Lamp
- Mesh
- Text

Note: Any other object type (like Curve, Speaker, Force Field, etc) will not be rendered into game. A Curve Object can be converted to Mesh though.

Data-blocks supported

- Action
- Armature
- Camera
- Group
- Image
- Lamp
- Library
- Material
- Mesh
- Object
- Scene
- Shapekey (along with Action, otherwise unused)
- Sound
- Text (partially)
- Texture (only Image based, procedural not supported)
- World
Note: Any other data-block type (like Line Styles, Particles, Brushes, etc) have no use or will not be rendered into game.

Note: Text only work partially, with advanced formatting features currently not supported.

### 1.3.2 UPBGE VS BGE

BGE also have some incompatibilities with UPBGE. UPBGE can partially load and execute games made in BGE, but a game made in UPBGE can’t be executed in BGE, resulting in several issues like:

- Logic can’t run most of the times.
- Materials get messed up.
- UPBGE do not support Multitexture material mode anymore. Set to GLSL when in vanilla BGE.
- Sometimes physics simulation get messed up.

Along with this compatibility with BGE, UPBGE comes with features not supported by BGE, like Modifiers applied automatically at game start (instead of discarded, as in BGE).
2.1 Getting Started

This tutorial aims to show the basic concepts of UPBGE, from the game loop behavior to the most common operations used in game development. This will give you important notions on how UPBGE works and how are its procedures.

Keep in mind that the focus of this tutorial are UPBGE and its game engine behavior, so it will not explain basic Blender navigation (like selection, shortcuts, etc). This tutorial assumes you already know how Blender works, and it focuses only on GE capabilities.

2.1.1 Basic Concepts

Game Loop

The game loop is something that every game or general software has: it consists of several processing steps and then, repeating it. To make it clear, think of a video player processing steps:

• It reads a portion of the video file from disk.
• It decodes the read file portion into a image.
• It shows the decoded image in the screen as a single frame.
• It repeats everything from the start.

Nevertheless, this article will present you how the game loop is built in UPBGE. It is essential to know how it works: if you keep that in mind, it will explain a lot of mysterious behavior that you might discover.

Each cycle in the game loop represents a logical frame, also known as logic tick. It is the smallest time unit within your game. In each loop:

• The scenes are processed.
• The devices are checked for input.
• The final image is rendered.
It is important to know that the render part might be skipped if the last frame was spending too much processing time, resulting in lag. There is a limit on how much renders can be skipped (default = 5). If this limit exceeds, a render will take place regardless how long it takes. Such 'render' lags will result in 'logic' lags, making the game run slower than expected.

**Scene Loop**

This loop is a bit more complex. The scenes loop cycles through all active scenes performing the following steps for each scene:

- **Logic processing** This first step processes the logic of the game, be it visual or Python.
- **Physics update** The physics update will be done after the logic runs, but before the render is drawn.
- **Sound playing** The sound playing is put in the last step of the scene loop.

Once all these steps are taken for all active scenes, the main loop continues.

**Logic Ticks**

The logic processing works through frames called *logic ticks*. By default, a game running at 60 frames per second also can run 60 logic ticks per second. In practical examples:

- If I have a number 0 and increase it by 1 each frame, after 1 second its value will be 60.
- If I have an object and move it 0.1 meters each frame, after 1 second it should have been moved 6 meters.
In UPBGE you can control the logic tick intervals for each object by skipping a given amount of ticks, allowing you to run logic at exact custom time intervals or optimize logic which doesn’t need to run each frame. The following diagram shows how logic tick skipping work.

![Fig. 3: Logic tick skipping diagram.](image)

In the given diagram, where dark red is logic execution and grey is a interval:

1. No ticks skipped between logic ticks, the logic runs each game frame.
2. Skipping of 2 ticks between each logic execution.
3. Skipping of 6 ticks between each logic execution.

To understand how the game loop and logic triggering works is important for you to shape the way your logic will be made and understand internal behaviors that UPBGE may show.

### 2.1.2 Moving A Cube

In this short tutorial we will introduce two essential elements for logic in UPBGE: logic triggering and game properties. These elements allows the interactivity with the set up objects.

A **game property** (also known as variable) is a value that is kept inside a object, allowing multiple uses. A common use of game properties is the amount life of the player, the current song playing, or anything else that can be useful to track for later use.

The **logic triggering** is the act of triggering some kind of event in the game. There’s lots of ways to trigger events in UPBGE, from detecting if a key is pressed to detecting an object colliding with another, or even triggering events continuously without detecting anything. After the detection of an trigger, an event can be happen, like a object to move, a property be changed, etc.

### Setup Scene

First, we must add objects to compose our scene. We need three basic objects:

- **Camera** A camera will allow us to see our scene from a point of view.
- **Lamp** A lamp will illuminate the scene objects, allowing us to see them. In this example we’ll use a Hemi lamp, which is a directional lamp that illuminates all objects in the scene.
- **Cube** A Cube object will be our visual feedback of our logic. As we can’t see a camera or lamp, we’ll move the Cube.

Once all objects were added, place them somewhat like the picture below:

Our Cube is not centered in the screen on purpose: we’ll move it in the front direction (−Y), so it’s good to see it moving after certain point.
Adding Logic

After the scene is set up, follow these steps:

- Go to the Logic Bricks Area.
- Add a Keyboard Sensor through the dropdown menu named Add Sensor.
- Add a AND Controller through the dropdown menu named Add Controller.
- Add a Motion Actuator through the dropdown menu named Add Actuator.
- Connect each brick by dragging and dropping one insert into another.

Now we must fill some information on the bricks:

- On the Keyboard Sensor, click on the Key field and press a key to assign a key to it.
- On the Motion Actuator, insert the value $-0.05$ in the field $Y$ of Loc.

The Logic Bricks Area should look like this:

Now, start the game engine (by default, pressing $P$ while focusing the 3D Viewport). If you press the key you assigned to the Keyboard Sensor, the Cube will move in the $-Y$ direction, and if you release the key, the Cube will stop.

This behavior happens for several reasons:

- The Keyboard Sensor emits a positive signal when the selected key is pressed, and emits a negative signal when the key is released.
- The AND Controller receives the signals from all connected sensors, and if all signals are positive, the controller emits an activation signal to all connected actuators, or an deactivation signal if one or more incoming signals are negative.
The **Motion Actuator** receives the activation signal from the controller and perform the motion. When it receives a deactivation signal, it stops performing the motion.

This is the basic of visual logic when using UPBGE, pretty straightforward. However, according to what you want to achieve, it can get a lot more complex.

### Logic Depending On Properties

In games, the logic depends on statuses most of the time. An enemy dies when its life reaches 0, the player can shoot while its ammo is greater than 0, and so on. In UPBGE, you can do these conditions through the use of properties.

To continue, perform the following steps:

- **In Properties Panel**, add a property through **Add Game Property**, set its name to **fuel**, its type to **Integer** and its value to 200.
- **In Logic Bricks Area**, add a Property Sensor, set its evaluation type to **Greater Than**, the property to **fuel** and the value to 0.
- Connect the Property Sensor to the **AND Controller**, along with the Keyboard Sensor.

This makes our Cube move only if the value of **fuel** is greater than 0. You can set the property **fuel** to 0 and play the game, and you will see that the Cube will not move. However, it would be good if we decrease the value of **fuel** as our Cube moves, until it reaches 0. To do that, do the following steps:

- Add a Property Actuator and connect it to the **AND Controller**.
- Set the mode of Property Actuator to **Add**, its property to **fuel** and its value to −1.
- Enable the pulse mode on **Keyboard Sensor**.
There’s a new factor involved here: the pulse mode on Keyboard Sensor. By default, a sensor sends a single positive signal to the controller when active, and a single negative signal when inactive. The pulse mode makes the signal be sent each logical frame (default is 60 frames per second). This is useful for us now, because we need our fuel to be decreased while we press the key without the need of releasing and pressing it again.

Go ahead em play the game. The Cube will move and, after some time, it stops. It happens because the Property Actuator has decreased 1 unit of fuel each frame, according to the Keyboard Sensor pulse mode, and when fuel reaches 0, the logic of the Motion Actuator doesn’t respond anymore. It would be good, however, to see the value of fuel be decreased over time. You can do this by enabling the debug flag on the fuel property and the Show Debug Properties on menu Game > Show Debug > Show Debug Properties, or on the Render editor, as shown in the figure below.

### Conclusion

The goal of this basic tutorial is to show how to work with the visual logic and properties on UPBGE. There’s more to be discovered about visual logic and properties, like other Property Types, the use of States with logic bricks, etc, and those subjects can be better understood on their own pages.

### 2.1.3 Playing An Animation

In games, animations can be used in many places other than characters - a sliding menu, a opening door, coins rotating, etc. In UPBGE, an animation can be played through the use of an Action. This tutorial will show you how to play an animation in a object and the concept of animation layering, allowing you to play and blend multiple animations at once in a single object.
Before We Start

Before we start using animations through logic, we need some animations. As we said before, UPBGE animations work through the use of Action, so we need to set our default Dope Sheet editor from Dope Sheet mode to Action Editor mode. Now, assuming you already know how to make simple animations adding keyframes, we’ll make two simple actions on a cube of a total 50 frames each, with the following settings:

- Rotation action named rotate: No rotation at frame 0, 180° at Z axis on frame 25, more 180° at Z axis on frame 50 (that is, a full 360°).
- Scaling action named scale: No scale at frame 0, double scale at frame 25 and no scale at frame 50 again.

Note the transform channels: rotate action only have Rotation channels, and scale action only have Scaling channels. This is important to properly blend those actions later on, as having the same channels on both (even without an actual transform) may not have the expected result in the end.

The Logic

Now, go to the Logic Bricks Area and do the following setup:

- Add two Keyboard Sensor through the dropdown menu named Add Sensor.
- Add two AND Controller through the dropdown menu named Add Controller.
- Add two Action Actuator through the dropdown menu named Add Actuator.
- Connect each brick by dragging and dropping one insert into another.

Now we must fill some fields:

- On the Keyboard Sensor, rename both, respectively, to rotate and scale, and set the field Key to, respectively, A and S.
- On the Action Actuator, rename both, respectively, to rotate and scale, set the playback type to Loop Stop, the value to its respective actions, End Frame to 50 and Layer to, respectively, 0 and 1.

The setup should look somewhat like the figure below:

![Action Editor showing each action.](image)
Now, play the game by pressing $P$ while focusing the 3D Viewport. When you press $A$ the cube should rotate, and when you press $A$ the cube should scale. When pressing both buttons the cube will rotate and scale at the same time, blending both actions. This is only possible due to different animation layers being blended together. For a matter of testing, set the actuator `scale`’s Layer to 0, the same value from the actuator `rotate`. When you play the game, you can’t play both animations at the same time: the last triggered actuator overwrites the currently playing.

**Conclusion**

This is how you play and blend animations using visual logic in UPBGE. There’s more to be discovered, like playback modes, blending and more, and this can be learnt from the `Action Actuator` page.

### 2.1.4 Using Linked Libraries In Game

`Linked Library` is a Blender feature that allows referencing a datablock from another blend file into the current one, allowing easier manipulation of common assets and better project management, as you only need to edit the original file to update all the references. UPBGE supports this Blender feature, specially group instances, which ones are fairly well integrated through Python. This tutorial aims to show how to use linked libraries, group instances and manage the links.

**Before We Start**

To proceed with this tutorial, we need some base files first. On a directory, save three empty blend files: `game.blend` (which will contain the references from the libraries), `lib_character.blend` (which will contain the actual model of our character) and `lib_scenery.blend` (which will contain the actual model of our ground). We’ll edit the file `game.blend` later, first we must create the contents of our libraries.

- **lib_character.blend** Add an Monkey object, add it to a group (Ctrl-G) and rename the group to `player`.
- **lib_scenery.blend** Add an Plane, scale it by 5 (so it have the size of 10x10), add it to a group (Ctrl-G) and rename the group to `ground`.

Both files should be set up somewhat like this:
Fig. 10: Split view of the two files.
Linking The Groups

Now, we’ll create a blend file named `game.blend` on the same directory as our previous files. In this file, we’ll do the following procedures:

- Go to the File menu, click the option Link.
- Select the either `lib_character.blend` or `lib_scenery.blend`.
- Select the option Group.
- Select the corresponding group and double click it / click Link from Library.

If everything went right, you should see in the 3D Viewport the group instances added. They are not editable as they are only Empty objects referencing groups from the original files.

With the instances added, all you have to do is place them with the desired transformations, and if you need to edit all the placed instances, just edit the original file to update all the references. Pretty handy, isn’t it?

Managing the Links

If you rename or move any of your libraries to another folder, you will face a common problem: broken links. There are several ways to deal with this problem, and this section will present you the use of the Outliner to manage links (and Datablocks).
I have renamed the library files to \textit{LibCharacter} and \textit{LibScenery}, which are different names from the previous ones (with underlines and only lower case characters). When opening \textit{game.blend}, the editor will complain about the non existing libraries, and our instances will be shown only as \textit{Empty} objects.

To fix this, we must go to the \textbf{Outliner} and select the \textit{Blend File} mode. On this mode we’ll see all :ref:`datablock-index’s` on our blend file.

The important elements here are the references to our libraries at the bottom: they are being shown as cracked icons and their previous names. We have two choices to fix this issue:

\textbf{Renaming the references} You only need to double click the references and provide the new names. If they are in another folder, you must provide the relative path.
Relocating the references  You only need to right click the references, select Relocate and select the corresponding file.

After fixing the broken references, their icons will change back to normal and the objects will be automatically updated in the 3D Viewport.

Understanding how to use and manage linked libraries is important to maintain a complex and healthy project environment. Much more can be achieved through the use of `bge.logic.LibLoad`, loading and unloading libraries dynamically using Python, but for simpler projects, linked libraries should do the job.

### 2.2 Introducing Python

#### 2.2.1 Basic Concepts

TODO

#### 2.2.2 Moving A Cube

TODO

#### 2.2.3 Playing An Animation

TODO
2.2. Introducing Python

Fig. 15: Broken references fix modes on the *Outliner*.

Fig. 16: Fixed references and objects updated on the *3D Viewport*. 
2.2.4 Dealing With Group Instances

TODO

2.2.5 Setting Up A Python Component

TODO
3.1 Properties Editor

Fig. 1: The Properties editor header.

The properties editor is an essential part of the development process. There, you can change almost all properties of the selected object, scene, camera, etc, like:

- The screen resolution;
- Frames per second;
- Background color;
- Name and position of selected object;
- Constraints and modifiers;
- Materials and textures properties;
- Physics properties of selected object;
- Much more.

In this section you have a detailed description about each tab of the Properties editor.

**Note:** The *Render layers* and *Particles* tabs don’t apply to UPBGE, so they won’t be explained here.

**Note:** This section will explain properties belonging to UPBGE only (Blender Game renderer) or, at most, relevant properties for game development. For other Blender properties, see the [official Blender manual](https://www.blender.org/manual/).
### 3.1.1 Render

The **Render** tab in *Properties Editor* exposes options related to game screen rendering.

Some of the options showed here may behave differently according to the conditions, as there are two separate game “players” for previewing the game during development. Note that while UPBGE is running in either player, the computer’s mouse and keyboard are captured by the game and by default, the mouse cursor is not visible (this can be changed in the *Display panel* of this tab). To exit the game, press the `Esc` key.

**Note:** Make sure that the render engine is set to **UPBGE** when attempting to set these controls, otherwise this description will not apply to what you see!

In **Render** tab, there are several panels available, as shown. Each one can be expanded or contracted using the usual triangle button. The features in each panel will be described in details below.

#### Embedded Player

![Embedded Player panel](image)

This panel provides information for the **Embedded Player** which allows games to be run inside the Blender **3D View**. The **Embedded Player** renders onto the **3D View** editor in the Blender GUI using the current perspective and zoom level of the **3D View**.

Note that the **Resolution** settings are independent of the size of the viewport preview pane. In fact, the **Resolution** controls seem to have no effect at all. The resolution and aspect ratio of the embedded preview are always fixed to the **3D View** editor, which behaves much like the **Extend** framing mode for the **Standalone Player**.

**Start**  Starts UPBGE inside the current Blender **3D View**. Shortcut `F` while mouse hovers the desired **3D View**.

**Resolution X/Y**  Sets the internal X/Y rendering resolution.

#### Standalone Player

This panel provides information for the **Standalone Player** which allows games to be run without Blender. See **Standalone Player** for further details.

The **Standalone Player** renders the scene from the perspective of the active scene camera and either creates a new desktop window or switches into fullscreen rendering mode.

The semantics of the **Standalone Player**’s **Resolution** controls differ for Windowed and Fullscreen modes. In Windowed mode (**Fullscreen** checkbox unchecked), the **Resolution** controls set the initial dimensions of the desktop window. The user may resize the window at any time, causing the rendering resolution to change accordingly. In Fullscreen mode (**Fullscreen** checkbox checked), the **Resolution** controls set the internal rendering resolution.
The actual display resolution will be a best fit depending on the user’s hardware. In either mode, the aspect ratio/cropping/scaling are determined by the Framing selection under the Display panel.

Regarding Fullscreen mode, it is important to remember that the Resolution settings in Fullscreen mode are only hints to the operating system. Each display and monitor combination will have a different set of resolutions that they are capable of displaying; so there can be little confidence that all end-users will actually get the resolution you suggest; unless you choose one of the most standard resolutions (e.g. 800x600 or 1024x768). If you insist on using higher resolutions, then you may want to state clearly in your documentation that only certain resolutions are supported. In most other cases, the user’s machine may select a resolution that is close to the one suggested; but the results can be unpredictable, especially in Letterbox framing mode.

Note that the Desktop checkbox has no effect in Windowed mode.

**Start** Launches the current blend file with the Standalone Player.

**Resolution**
- **X** Sets the X window size or fullscreen display resolution.
- **Y** Sets the Y window size or fullscreen display resolution.

**Fullscreen**
- **Off** Opens standalone game as a new window.
- **On** Opens standalone game in fullscreen.

**Desktop**
- **Off** Attempts to obey the Resolution specified above when in Fullscreen mode.
- **On** Keeps the current desktop resolution when in Fullscreen mode.

**Quality**
- **AA Samples** The number of AA samples to use for MSAA.
- **Bit Depth** Number of bits used to represent color of each pixel in fullscreen display.
**Refresh Rate**  Number of frames per second of fullscreen display.

**Stereo**

![Stereo panel](image)

Fig. 4: Stereo panel.

Toggle if use an stereo mode and, if use, select a stereo mode that will be used to capture stereo images of the game (and also, by implication, that stereo displays will use to render images in the *Standalone Player*).

- **None**  Render single images with no stereo.
- **Stereo**  Render dual images for stereo viewing using appropriate equipment.  See *Stereo Camera* for full details of available options.

**Shading**

![Shading panel](image)

Fig. 5: Shading panel.

Specifies each single visual components that will be rendered in the game.

- **Lights**  Toggles lights rendering.
- **Shaders**  Toggles GLSL shaders.
- **Shadows**  Toggles realtime shadows from lamps.
- **Environment Lighting**  Toggles environment lighting from *World tab*.
- **Ramps**  Toggles material ramps.
- **Nodes**  Toggles material nodes.
- **Extra Textures**  Toggles extra textures, like normal or specular maps.

**System**

The **System** panel at the Render tab lets the game developer specify options about the system performance regarding to frame discard and restrictions about frame rendering, the key to stop UPBGE, etc.
Use Frame Rate  Respect the frame rate rather than rendering as many frames as possible. When unchecked, this will inform Blender to run freely without frame rate restrictions. The frame rate is specified at the Display panel, also in the Render tab. For more information about frame rates, see Display panel.

Deprecation Warnings  Every time when the game developer uses a deprecated functionality (which in some cases are outdated or crippled OpenGL Graphic cards functions), the system will emit warnings about the deprecated function on the console.

Vsync  Change Vsync settings.

AA Samples  Set how many samples use in anti-aliasing.

HDR  The precision of the screen display (between 8, 16 and 32 bits).

Exit Key  This button specifies which key-press will exit the game.

Animations

Fig. 7: Animations panel in the Render tab.

Specifies animations settings of game, like frame rate.

Animation Frame Rate  This number button/slider specify the maximum frame rate at which the game will run. Minimum is 1, maximum is 120.

Restrict Animation Updates  Restrict number of animation updates to the animation FPS. This is better for performance, but can cause issues with smooth playback. When checked, this will force UPBGE to discard frames (even at the middle of redrawing, sometimes causing tearing artifacts) if the rate of frames rendered by the GPU is greater than the specified on Display panel.

Display

The Display panel in the Render tab lets the game developer specify whether the mouse cursor is shown during the game execution, and options to specify the framing style of the game to fit the window with the specified resolution.
Mouse Cursor Whether to show or not the mouse cursor when the game is running.

Framing Selects how the scene is to be fitted onto the display window or screen. There are three types of framing available:

- **Letterbox**
  - **In Windowed mode**: Maintains a 4:3 aspect ratio by scaling to fit the current window dimensions without cropping, covering any portions of the display that lie outside of the aspect ratio with color bars.
  - **In Fullscreen mode**: The behavior of this combination seems to be heavily dependent on the user’s hardware. The result can be quite unpredictable, especially when the resolution and aspect ratio differ too much from the machine’s capabilities. For this reason, *Extend* mode should be preferred for Fullscreen applications.

- **Extend** This mode behaves much like *Letterbox* mode, maintaining a 4:3 aspect ratio by scaling whenever possible; except that the camera frustum is expanded or contracted wherever necessary to fill any portions of the display that lie outside of the aspect ratio, instead of covering those portions of the scene with color bars, as with *Letterbox* mode, or distorting them scene, as with *Scale* mode.

- **Scale** In this mode, no attempt is made to maintain a particular aspect ratio. The scene and objects within will be stretched or squashed to fit the display exactly.

Color Bar This will let the game developer choose the bar colors when using the *Letterbox* Framing mode.

### Debug

The **Debug** panel at the **Render** tab toggles various specific debug helpers on UPBGE, from frame rate being showed on the screen to detailed physics visualization of specific elements, like armatures and camera frustum.

- **Framerate and Profile** When checked, this will show values for each of the calculations Blender is doing while the game is running on the top left of the screen.

- **Render Queries** Shows render queries information while the game runs.
**Properties**  When checked, the values of any properties which are selected to be debugged in the objects are shown on the top left side of the screen.

**Physics Visualization**  Shows a visualization of physics bounds and interactions (like hulls and collision shapes), and their interaction.

The following remaining options are dropdown menus which allows the following options:  - Disable: Disables the debug of the current option.  - Allow: Allow debugging from individual settings of the current option.  - Force: Allow debugging of the current option.

**Bounding Box**  Shows bounding volume boxes of objects while the game is running.

**Armatures**  Shows armatures while the game is running.

**Camera Frustum**  Shows camera limits visualization according to the current viewport dimensions while the game is running.

**Shadow Frustum**  Shows lamp’s shadows bounds while the game is running.

**Bake**

The **Bake** panel in the **Render** tab is very similar to its **Blender Render** counterpart and serves much the same purpose. See **Render Baking** for further details.

**Bake Mode**

- **Full Render**  Bakes all materials, textures, and lighting except specularity and SSS.
- **Ambient Occlusion**  Bakes ambient occlusion as specified in the World panels. Ignores all lights in the scene.
- **Shadows**  Bakes shadows and lighting.
- **Normals**  Bakes tangent and camera-space normals (among many others) to an RGB image.
- **Textures**  Bakes colors of materials and textures only, without shading.
- **Displacement**  Similar to baking normal maps, displacement maps can also be baked from a high-res object to an unwrapped low-res object, using the Selected to Active option.
- **Derivative**  Bake derivative map.
- **Vertex Colors**  Bake vertex colors.
- **Emissions**  Bakes Emit, or the Glow color of a material.
- **Alpha**  Bakes Alpha values, or transparency of a material.
- **Mirror Intensity**  Bake mirror intensity values.
Mirror Colors  Bake mirror colors.
Specular Intensity  Bake specular intensity values.
Specular Colors  Bake specular colors.

Bake from Multiresolution  Bake directly from a multi-resolution object.

Normalized

In Displacement Mode:  Normalize to the distance.
In Ambient Occlusion Mode:  Normalize without using material’s settings.

Normal Space  Normals can be baked in different spaces:

Camera space  Default method.
World space  Normals in world coordinates, dependent on object transformation and deformation.
Object space  Normals in object coordinates, independent of object transformation, but dependent on deformation.
Tangent space  Normals in tangent space coordinates, independent of object transformation and deformation. This is the new default, and the right choice in most cases, since then the normal map can be used for animated objects too.

Bake to Vertex Color  Bake to vertex colors instead of to a UV-mapped image.

Clear  If selected, clears the image to selected background color (default is black) before render.

Margin  Baked result is extended this many pixels beyond the border of each UV “island”, to soften seams in the texture.

Selected to Active  Bake shading on the surface of selected objects to the active object.

Distance  Maximum distance in blender units from active object to other object.
Bias  Bias in blender units toward faces further away from the object.

Split  The method used to split a quad into two triangles for baking.

Fixed  Split quads predictably (0,1,2)(0,2,3).
Fixed Alternate  Split quads predictably (1,2,3)(1,3,0).
Automatic  Split quads to give the least distortion while baking.

User Scale  Apply a custom scale to the derivative map instead of normalizing to the default (0.1).

Stereo Camera

Stereo rendering allow you to generate images that appear three-dimensional when wearing special glasses. This is achieved by rendering two separate images from cameras that are a small distance apart from each other, simulating how our own eyes see. When viewing a stereo image, one eye is limited to seeing one of the images, and the other eye sees the second image. Our brain is able to merge these together, making it appear that we are looking at a 3D object rather than a flat image. See Stereoscopy for more information on different stereoscopic viewing methods.

Stereo Settings

None  Disable stereo rendering.
Stereo Mode  Specifies the way in which the left-eye image and the right-eye image pixels are put together during rendering. This must be selected according to the type of apparatus available to display the appropriate images to the viewer’s eyes.

Anaglyph  One frame is displayed with both images color encoded with red-blue filters. This mode only requires glasses with color filters, there are no special requirements for the display screen and GPU.

Quad Buffer  Uses double buffering with a buffer for each eye, totaling four buffers (Left Front, Left Back, Right Front and Right Back), allowing to swap the buffers for both eyes in sync. See Quad Buffering for more information.

Side by Side  Lines are displayed one after the other, so providing the two images in two frames side-by-side.

Above-Below  Frames are displayed one after the other, so providing the two images in two frames, one above the other.

Interlaced  One frame is displayed with the two images on alternate lines of the display.

Vinterlaced  One frame is displayed with both images displayed on alternate columns of the display. This works with some 'autostereo displays'.

3D TV Top-Bottom  One frame displays the left image above and the right image below. The images are squashed vertically to fit. This mode is designed for passive 3D TV.

Eye Separation  This value is extremely important. It determines how far apart the two image-capturing cameras are, and thus how “deep” the scene appears. Too small a value and the image appears flat; too high a value can result in headaches and eye strain. The ideal value mimics the separation of the viewer’s two eyes.

3.1.2 Scene

The Scene tab in Properties Editor exposes options related to the current scene.

Physics

The Physics panel located in the Scene tab determine the type of physical rules that govern the current UPBGE scene, the gravity value to be used and some other options.

Physics Engine  Set the type of physics engine to use.

Bullet  The default physics engine, in active development. It handles movement and collision detection. The things that collide transfer momentum to the collided object.

None  No physics in use. Things are not affected by gravity and can fly about in a virtual space. Objects in motion stay in that motion.

Physics Solver  The physics constraints solver to use.

Sequential  Sequential physics solver, default solver.
**NNGC**  NNGC physics solver.

**MLCP Dantzig**  MLCP Dantzig physics solver.

**MLCP Lemke**  MLCP Lemke physics solver.

**Gravity**  The gravitational acceleration, m.s\(^{-2}\) (in units of meters per squared second), of this world. Each object that is an actor has a mass and size slider. In conjunction with the frame rate, Blender uses this info to calculate how fast the object should accelerate downward.

**Physics Steps**

**Max**  Sets the maximum number of physics steps per game frame if graphics slow down the game. Higher value allows physics to keep up with real-time.

**Substeps**  Sets the number of simulation sub-steps per physics time step. Higher value give better physics precision.

**FPS**  Set the nominal number of game frames per second. Physics fixed timestep = 1/fps, independently of actual frame rate.

**Time Scale**  Time scale to slow down or speed up animations and physics in game.

**Logic Steps**

**Max**  Sets the maximum number of logic frame per game frame if graphics slows down the game, higher value allows better synchronization with physics.

**Physics Deactivation**  These settings control the threshold at which physics is deactivated. These settings help reducing the processing spent on Physics simulation during the game.

**Linear Threshold**  The speed limit under which a rigid body will go to sleep (stop moving) if it stays below the limits for a time equal or longer than the deactivation time (sleeping is disabled when deactivation time is set to 0).
Angular Threshold  Same as linear threshold, but for rotation limit (in rad/s).

Time  The amount of time in which the object must have motion below the thresholds for physics to be disabled (0.0 disables physics deactivation).

Culling

Occlusion Culling  Use optimized Bullet DBVT tree for view frustum and occlusion culling (more efficient, but it can waste unnecessary CPU if the scene doesn’t have occluder objects).

Resolution  The size of the occlusion culling buffer in pixel, use higher value for better precision (slower). The optimized Bullet DBVT for view frustum and occlusion culling is activated internally by default.

Object Activity

Activity Culling  Enable object activity culling in this scene. The culling options can be set individually by object on Activity Culling panel on Object tab.

Obstacle Simulation

Fig. 13: Scene tab’s Obstacle Simulation panel.

Simulation used for obstacle avoidance in UPBGE, based on the RVO (Reciprocal Velocity Obstacles) principle. The aim is to prevent one or more actors colliding with obstacles.

Type

None  Obstacle simulation is disabled, actors are not able to avoid obstacles.

RVO (cells)  Obstacle simulation is based on the RVO method with cell sampling.

RVO (rays)  Obstacle simulation is based on the RVO method with ray sampling.

Level height  Max difference in heights of obstacles to enable their interaction. Used to define minimum margin between obstacles by height, when they are treated as those which are situated one above the other i.e. they do not influence to each other.

Visualization  Enable debug visualization for obstacle simulation.

Navigation Mesh

Rasterization

Cell size  Rasterized cell size.

Cell height  Rasterized cell height.

Agent

Height  Minimum height where the agent can still walk.

Radius  Radius of the agent.
Fig. 14: Scene tab’s Navigation Mesh panel.

**Max climb**  Maximum height between grid cells the agent can climb.

**Max slope**  Maximum walkable slope angle in degrees.

**Region**

**Min Region Size**  Minimum regions size. Smaller regions will be deleted.

**Merged Region Size**  Minimum regions size. Smaller regions will be merged.

**Partitioning**

**Watershed**  Classic Recast partitioning method generating the nicest tessellation.

**Monotone**  The fastest navmesh generation method, but may cause long thin polygons.

**Layers**  A reasonably fast method that produces better triangles than monotone partitioning.

**Polygonization**

**Max Edge Length**  Maximum contour edge length.

**Max Edge Error**  Maximum distance error from contour to cells.

**Vertices Per Poly**  Max number of vertices per polygon.

**Detail Mesh**

**Sample Distance**  Detail mesh sample spacing.

**Max Sample Error**  Detail mesh simplification max sample error.
Level of Detail

**Hysteresis** Use LoD hysteresis settings for the current scene.

*Slider from 0% to 100%* Minimum distance change required to transition to the previous level of detail.

Python Console

Enabling the panel’s checkbox allows to trigger an interactive Python console when the game is running through the specified shortcut.

**Keys** Set the keys to be pressed in order to activate the Python console in game.

Scene

**Camera** Used to select which camera is used as the active camera. You can also set the active camera in the 3D View with Ctrl-0.

**Background** Allows you to use a scene as a background, this is typically useful when you want to focus on animating the foreground for example, without background elements getting in the way.

This scene can have its own animation, physics simulations, etc, but you will have to select it from the Scene data-block menu, if you want to edit any of its contents.

Sets can themselves have a background set (they’re recursively included). So you can always make additions to existing scenes by using them as a background to a newly created scene where your additions are made.

**Tip:** This can also be used in combination with Linking to a Scene, where one blend-file contains the environment, which can be reused in many places.

---

3.1. Properties Editor
Camera  Active camera, used for rendering the scene.

Background  Background set scene.

Units

Fig. 18: Scene tab’s Units panel.

Unit Presets  Common unit scales to use.

Length

None  Uses Blender Units.

Metric, Imperial  Standard unit of measurement for lengths.

Angle  Standard unit for angular measurement.

Degrees, Radians

Tip:  When you are using Degrees, the radian value is also displayed in the tooltip.

Unit Scale  Scale factor to use when converting between Blender Units and Metric/Imperial.

Tip:  Usually you will want to use the Length presets to change to scale factor, as this does not require looking up values to use for conversion.

Separate Units  When Metric or Imperial display units as multiple values, for example, “2.285m” will become “2m 28.5cm”.

Table 1: Imperial Units

<table>
<thead>
<tr>
<th>Full Name</th>
<th>Short Name(s)</th>
<th>Scale of a Meter</th>
</tr>
</thead>
<tbody>
<tr>
<td>thou</td>
<td>mil</td>
<td>0.0000254</td>
</tr>
<tr>
<td>inch</td>
<td>”, in</td>
<td>0.0254</td>
</tr>
<tr>
<td>foot, feet</td>
<td>’, ft</td>
<td>0.3048</td>
</tr>
<tr>
<td>yard</td>
<td>yd</td>
<td>0.9144</td>
</tr>
<tr>
<td>chain</td>
<td>ch</td>
<td>20.1168</td>
</tr>
<tr>
<td>furlong</td>
<td>fur</td>
<td>201.168</td>
</tr>
<tr>
<td>mile</td>
<td>mi, m</td>
<td>1609.344</td>
</tr>
</tbody>
</table>
### 3.1.3 World

**World** settings enable you to set some basic effects which affect all scenes throughout your game, so giving it a feeling of unity and continuity. These include ambient light, depth effects (mist), etc.

**World**

![World tab’s World panel.](image)

These color settings allow you to set some general lighting effects for your game.

- **Paper Sky** Flatten blend or texture coordinates.
- **Blend Sky** Render background with natural progression from horizon to zenith.
- **Real Sky** Render background with a real horizon, relative to the camera angle.
- **Horizon Color** The RGB color at the horizon; i.e. the color and intensity of any areas in the scene which are not filled explicitly.
- **Zenith Color** The RGB color at the zenith.

---

### Table 2: Metric Units

<table>
<thead>
<tr>
<th>Full Name</th>
<th>Short Name(s)</th>
<th>Scale of a Meter</th>
</tr>
</thead>
<tbody>
<tr>
<td>micrometer</td>
<td>um</td>
<td>0.000001</td>
</tr>
<tr>
<td>millimeter</td>
<td>mm</td>
<td>0.001</td>
</tr>
<tr>
<td>centimeter</td>
<td>cm</td>
<td>0.01</td>
</tr>
<tr>
<td>decimeter</td>
<td>dm</td>
<td>0.1</td>
</tr>
<tr>
<td>meter</td>
<td>m</td>
<td>1.0</td>
</tr>
<tr>
<td>dekameter</td>
<td>dam</td>
<td>10.0</td>
</tr>
<tr>
<td>hectometer</td>
<td>hm</td>
<td>100.0</td>
</tr>
<tr>
<td>kilometer</td>
<td>km</td>
<td>1000.0</td>
</tr>
</tbody>
</table>

**Note:** The **Audio** panel settings in **Scene** tab don’t have effect in UPBGE. For audio settings, see [User Preferences](#).
**Ambient Color**  Ambient light mimics an overall background illumination obtained from diffusing surfaces. Its general color and intensity are set by these controls.

**Exposure**  Ammount of exponential color correction for light.

**Range**  The color range that will be mapped to 0-1.

---

**Environment Lighting**

![Environment Lighting Panel](image)

*Fig. 20: World tab’s Environment Lighting panel.*

**Environment Light** provides light coming from all directions.

Light is calculated with a ray-traced method which is the same as that used by Ambient Occlusion. The difference is that Environment lighting takes into account the “ambient” parameter of the material shading settings, which indicates the amount of ambient light/color that that material receives.

Also, you can choose the environment color source (white, sky color, sky texture) and the light energy.

**Energy**  Defines the strength of environment light.

**Environment Color**  Defines where the color of the environment light comes from.

Using both settings simultaneously produces better global lighting. It is good for mimicking the sky in outdoor lighting.

---

**Mist**

![Mist Panel](image)

*Fig. 21: World tab’s Mist panel.*

*Mist* can greatly enhance the illusion of depth in your rendering. To create *Mist*, UPBGE makes objects farther away more transparent (decreasing their Alpha value) so that they mix more of the background color with the object color. With *Mist* enabled, the further the object is away from the camera the less its alpha value will be.

**Mist**  Toggles mist on and off.

**Falloff**  Sets the shape of the falloff of the mist.

**Start**  The starting distance of the mist effect, measured from the camera. No misting will take place for objects closer than this distance.

**Depth**  The depth at which the opacity of objects falls to zero.

**Minimum intensity**  Overall minimum intensity of the mist effect.
3.1.4 Object

The Object tab from Properties Editor exposes settings and properties related to the current object. These properties and settings can be position, rotation, level of detail, group options and so on. For more detailed info about this tab, see Object properties in Blender Manual.

Name

The object’s name will identify the object in most cases, specially in fields inside the Blender editors. In UPBGE, if two identical objects are added in game (for example, with an Add Object logic or instanced through a group), both objects will have the same name, differently from Blender which appends a number in a repeating object, so be aware that retrieving an object through its name may not work sometimes. Prefer using Game Properties, instead.

Culling Bounding Volume

Predefined Bound Predefined mesh bounding volume used when Auto Update Bound is disabled.

Activity Culling

UPBGE’s Activity Culling feature allows to disable object’s physics and logic based on the distance to the nearest camera. This allows better performance by not processing objects which are too far away.

Physics Suspend physics of this object by its distance to nearest camera.

Physics Radius Distance to begin suspend physics of this object

Logic Suspend logic and animation of this object by its distance to nearest camera.

Logic Radius Distance to begin suspend logic and animation of this object

Levels of Detail

When creating visual assets it is often desirable to have a high amount of detail in the asset for up close viewing. However, this high amount of detail is wasted if the object is viewed from a distance, and brings down the scene’s
performance. To solve this, the asset can be swapped out at certain viewing distances. This is commonly referred to as a level of detail system. Each visual step of the asset is known as a level of detail. Levels of detail are most appropriate to use when you have a large scene where certain objects can be viewed both up close and from a distance.

**Distance Factor** The factor applied to distance computed in LoD.

**Add** Add a level of detail to this object

**Tools**

![Tools dropdown menu in Levels of Detail panel.](image)

Some tools for making levels of detail easier to manage and create can be found from the select menu next to the add button in the Levels of Detail panel.

**Set By Name** Searches the scene for specifically named objects and attempts to set them up as levels of detail on the currently selected object. The selected object must be the base level of detail (e.g. LOD0). This can be useful to quickly setup levels of detail on imported assets. In order to make use of this tool, your naming must be consistent, and each level must be prefixed or suffixed with “lodx” where x is the level that object is intended for. The case on “lod” must be consistent across all objects. Below are some example names that the tool will recognize.

- LOD0_Box, LOD1_Box, LOD2_Box
- Box.lod0, Box.lod1, Box.lod2
- LoD0box, LoD1box, LoD2box

**Generate** This tool generates and sets up levels of details based on the selected object. Generation is done using the Decimate Modifier. Generation does not apply the modifier to allow further changing the settings. Generated objects are automatically named based on the level they are generated for. Below are some settings for the operator.

![Generate Levels of Detail](image)

**Count** The number of levels desired after generation. This operator creates Count-1 new objects.

**Target Size** The ratio setting for the Decimate Modifier on the last level of detail. The ratio settings for the other levels are determined by linear interpolation.
**Package into Group**  With this setting enabled the operator performs some extra tasks to make the asset ready for easy linking into a new file. The base object and all of its levels of detail are placed into a group based on the base object’s name. Levels other than the base are hidden for both the viewport and rendering. This simplifies the appearance of the system and does not affect the appearance of the base object. Finally, all levels are parented to the base object to remove clutter from the Outliner.

**Clear All**  Clears the level of detail settings from the current object.

**Settings**

![Levels of Detail](image)

Level of detail settings can be found in the Object settings when the renderer is set to Blender Game. In the Levels of Detail panel is a button to add a new level of detail to the current object. The settings for each level of detail are displayed in its own box. The exception to this is the base level of detail. This is automatically setup as the current object with a distance setting of 0. To remove a level of detail, click on the X button in the top right corner of the box of the level to be removed.

**Object**  The object to use for this level of detail.

**Distance**  The distance at which this level of detail becomes visible.

**Use Mesh**  When this option is enabled, the mesh from the level of detail object is used until a lower level of detail overrides it.

**Use Material**  When this option is enabled, the material from the level of detail object is used until a lower level of detail overrides it.

**Transform and Delta Transform**

The **Transform** panel exposes the position, rotation and scale properties of the object, and the **Delta Transform** panel increments additional transformation values to **Transform** values. Note that these properties behave according to the object’s parent transform properties. However, this explanation is just a base to understand how the transformation values work in UPBGE. More info about the **Transform** panel can be found at Transform Properties.

In UPBGE there are two types of transformation values for the object: the **World** and the **Local** properties. The **World** values are the transformation values relative to the center of the world, and the **Local** values are the transformation values relative to the object’s parent object. For example:

- An object with a **World Position** of \((0, 0, 0)\) is literally at the center of the world.
Fig. 25: Object’s transform panels in Object tab.

- An object with a Local Position of \((0, 0, 0)\) is at the same position of its parent, even if its parent is not at the center of the world.

Be aware that, if the object doesn’t have a parent, the Local values behave the same as the World values. Technically, with the given information, the Transform panel works the same as the Local transform values, and the Delta Transform panel values are added to the World values of the object at game start.

**Relations**

Exposes values of relations of current object to other objects, scene, etc. For detailed info about object relations, see Object Relations.

**Layers**  The layers which the object is on the scene, multiple can be selected. The behavior is similar to Blender’s layers, as layers can keep the object hidden / shown or some actions can be applied only to objects in a specific layer (as lamps and shadows). Also, only objects in hidden layers can be added through logic. Detailed info about layers can be found at Layers.

**Parent**  The parent object of the current one. While the current object have a parent, its transformation values will be inherited from the parent. A parent may have multiple children, but the reverse is not true. The parenting behavior changes according to the selected mode in dropdown menu. Detailed info about parenting can be found at Parenting Objects.

**Relations Extras**

Exposes some extra settings about object’s relationship. Detailed info about relations extras can be found at Relations Extras.

**Slow Parent**  Creates a delay in parent relationship. Useful to easily smooth movement for character cameras, for example.

**Offset**  The amount of delay in Slow Parent.
Groups

Groups have multiple uses in Blender, but in UPBGE its main use is to allow creating maintainable libraries for games through the use of *dupli group instances*. Once one or several objects are added to a group, instances of this group can be added to the scene, and editing the original objects edits all the instances automatically in Blender. Detailed info about groups can be found at [Groups](#).

Display

These settings (except for *Object Color*) don’t affect the current object in UPBGE, only does in 3D Viewport. The exception, *Object Color*, can be used in game as value in material nodes, Python and material’s *Object Color* option. Detailed info about the Display panel can be found at [Display](#).

Duplication

Exposes several duplication modes, but the useful one in UPBGE is *Group*. When a group is selected in the dropdown menu, a group is instanced in the current object. By default, only empties are used in order to instance groups, but any kind of object can do it as well. More about dupli group instances uses in UPBGE on [Groups](#). Detailed info about the dupli group feature can be found at [DupliGroup](#).
3.1.5 Constraints

The Constraints tab in Properties editor exposes constraints setups for current object. Currently in UPBGE only the Rigid Body Joint constraint can be set through the user interface.

Rigid Body Joint Constraint

The Rigid Body Joint constraint is very special, it is used by the physics part of UPBGE to simulate a joint between its owner and its target. It offers four joint types: hinge type, ball-and-socket type, cone-twist type, and generic six-DoF (Degrees of Freedom) type.

The joint point and axes are defined and fixed relative to the owner. The target moves as if it were stuck to the center point of a stick, the other end of the stick rotating around the joint/pivot point.

Note: In order for this constraint to work properly, both objects (so the owner and the target object) need to have Collision Bounds enabled.

Options

Target Object used to select the constraints target, and is not functional (red state) when it has none.

Pivot Type

Ball Works like an ideal ball-and-socket joint, i.e. allows rotations around all axes like a shoulder joint.

Fig. 26: Rigid Body Joint panel.
**Hinge** Works in one plane, like an elbow: the owner and target can only rotate around the X axis of the pivot (joint point).

**Limits**

![Limits](image)

Angular limits for the X axis.

**Cone Twist** Similar to Ball, this is a point-to-point joint with limits added for the cone and twist axis.

**Limits**

![Limits](image)

Angular limits.

**Generic 6DOF** Works like the Ball option, but the target is no longer constrained at a fixed distance from the pivot point, by default (hence the six degrees of freedom: rotation and translation around/along the three axes). In fact, there is no longer a joint by default, with this option, but it enables additional settings which allow you to restrict some of these DoF:

**Limits**

![Limits](image)

Linear and angular limits for a given axis (of the pivot) in Blender Units and degrees respectively.

**Child Object** Normally, leave this blank. You can reset it to blank by right-clicking and selecting Reset to Default Value.

**Linked Collision** When enabled, this will disable the collision detection between the owner and the target (in the physical engine of the BGE).

**Display Pivot** When enabled, this will draw the pivot of the joint in the 3D Views. The most useful, especially with the Generic 6DOF joint type!

**Use Breaking** Allow breaking of constraint on high impulse.

**Breaking Impulse Threshold** Break constraint on impulse greater than threshold.

**Pivot** These three numeric fields allow you to relocate the pivot point, *in the owner’s space*. 

---

**3.1. Properties Editor**
Axis  These three numeric fields allow you to rotate the pivot point, in the owner’s space.

3.1.6 Texture

The Texture tab in Properties editor exposes textures and its attributes.

3.1.7 Physics

Fig. 27: Physics tab.

Physics

Options in Physics panel change according to its selected Physics Type. See below:

No Collision Physics

“No Collision” objects are completely unaffected by the simulation and do not affects other objects. They are useful as pure display objects, such as the child of a Custom Collision Hull (Collision Bounds).

For more documentation, see the general physics page.

Options

The only option available on No Collision types is:

Invisible  Does not display, the same as setting the object to unrendered (such as unchecking the camera icon in the Outliner).

Static Physics

Static objects participates in the simulation, affecting other objects, but are not affected by it. Meaning they do not react to physics, including gravity and collisions and this way will remain unresponsive in terms of location, rotation, or deformation.

It will, however, give collision reactions. Objects will bounce off of Static Objects, and rotational inertia will transfer to objects capable of rotating (that is, rigid body objects will spin in response, though Dynamic Objects will not).

Note that none of this prevents you from transforming the Static Objects with Logic Bricks or Python code. The visual objects will correctly move and their physics representation will update in the engine as well.

Another important note is that the default Collision Bounds is a Triangle Mesh, meaning it is higher in computational requirements but also in detail. This in turn means the “Radius” option has no effect by default.

For more documentation, see the general physics page.
Options

Note: bpy Access

Note that, most of these properties are accessible through the non-BGE scripting API via `bpy.data.objects["ObjectName"].game`, which is of type `bpy.types.GameObjectSetting`. This is useful, for example, to set a range of objects to have gradated values via a for-loop.

Actor  Enables detection by Near and Radar Sensors.

Ghost  Disables collisions completely, similar to No Collision.

Invisible  Does not display, the same as setting the object to unrendered (such as unchecking the camera icon in the Outliner).

Radius  See rigid body.

Anisotropic Friction  Isotropic friction is identical at all angles. Anisotropic is directionally-dependent. Here you can vary the coefficients for the three axes individually, or disable friction entirely. Python properties: `obj.game.use_anisotropic_friction` (boolean) and `obj.game.friction_coefficients` (a 3-element array).

Collision Bounds

The Static type differs from the others in that it defaults to a Triangle Mesh bounds, instead of a simple sphere. See rigid body.

Create Obstacle

Todo

Dynamic Physics

Dynamic objects give or receive collisions, but when they do so they themselves do not rotate in response. So, a Dynamic ball will hit a ramp and slide down, while a rigid body ball would begin rotating.

If you do not need the rotational response the Dynamic type can save the extra computation.

Note that these objects can still be rotated with Logic Bricks or Python code. Their physics meshes will update when you do these rotations – so collisions will be based on the new orientations.

For more documentation, see the general physics page.

Options

See Rigid Body Physics.

Rigid Body Physics

It will give/receive collisions and react with a change in its velocity and its rotation. A rigid body ball would begin rotating and roll down (where a Dynamic ball would only hit and slide down the ramp).
Fig. 28: Rigid Body collision type in Physics tab.
The idea behind rigid body dynamics is that the mesh does not deform. If you need deformation you will need to either go to Soft Body or else fake it with animated Actions.

For more documentation, see the general physics page.

### Options

**Actor** Enables detection by Near and Radar Sensors.

**Ghost** Disables collisions completely, similar to No collision.

**Invisible** Does not display, the same as setting the object to unrendered (such as unchecking the camera icon in the Outliner).

**Use Force Field** Materials can have physics settings on them as well: Friction, Elasticity, Force Field (positive or negative force), and also Dampening to other materials. When you turn on this checkbox, you are enabling the Material to exhibit this spring force.

**Rotate From Normal** Todo.

**No Sleeping** Prevents simulation meshes from sleeping. When an object has a linear velocity or angular velocity, it is in motion. It will detect collisions, receive gravity, etc. Once these thresholds are close to zero, it will cease these calculations – until another object interacts with it to wake it up.

### Attributes

**Mass** Affects the reaction due to collision between objects – more massive objects have more inertia. Will also affect material force fields. Will also change behaviors if you are using the suspension and steering portions of Bullet physics.

**Radius** If you have the “Collision Bounds: Sphere” set explicitly (or implicitly through having the Collision Bounds subpanel unchecked), this will multiply with the Object’s (unapplied) Scale. Note that none of the other bounds types are affected. Also note that in the 3D View the display will show this for all types, even though it is only actually used with Sphere.

<table>
<thead>
<tr>
<th>Basic</th>
<th>Radius= 1.5</th>
<th>Unapplied Scale</th>
<th>Applied Scale</th>
<th>Collision Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rolls, radius of 1 BU</td>
<td>Rolls, radius of 1.5 BU (after &quot;popping&quot; upward)</td>
<td>Rolls, radius of 1.5 BU</td>
<td>Rolls, radius of 1 BU (!)</td>
<td>Default (which is Sphere)</td>
</tr>
<tr>
<td>Slides, extent of 1 BU</td>
<td>Slides, extent of 1 BU</td>
<td>Slides, extent of 1 BU</td>
<td>Slides, extent of 1 BU</td>
<td>Box</td>
</tr>
<tr>
<td>Slides, extent of 1 BU (but with more friction than above)</td>
<td>Slides, extent of 1 BU (but with more friction than above)</td>
<td>Acts insane</td>
<td>Slides extent of 1.5 BU</td>
<td>Convex Hull</td>
</tr>
<tr>
<td>Slides, extent of 1 BU</td>
<td>Acts insane</td>
<td>Slides extent of 1.5 BU</td>
<td>Triangle Mesh</td>
<td></td>
</tr>
</tbody>
</table>

**Form Factor** For affecting the Inertia Tensor. The higher the value, the greater the rotational inertia, and thus the more resistant to torque. You might think this is strange, considering Dynamic types do not have torque in response to collisions – but you can still see this value’s effects when you manually apply Torque.

**Elasticity** The elasticity of collisions determines how much of the kinetic energy is retained after the collision. A value of 1 will result in a collision where objects bounce off each other, and the kinetic energy after the collision is the same as before. A value of 0 will result in a collision where the objects stick together after the collision, as all energy will have been converted to heat (or other energy forms that Blender also does not model).
In macroscopic nature (so bigger than atomic particles) an elasticity of 1 is never seen, as at least some energy is converted to heat, sound, etc. An elastic (elasticity=high) collision occurs when two metal balls collide. An inelastic (elasticity=low) collision is seen when two half-inflated beach balls collide.

**Anisotropic Friction** Isotropic friction is identical at all angles. Anisotropic is directionally-dependent. Here you can vary the coefficients for the three axes individually, or disable friction entirely.

**Linear Velocity** Limit the linear speed of an object.
- **Minimum** The object is allowed to be at complete rest, but as soon as it accelerates it will immediately jump to the minimum speed.
- **Maximum** Top speed of the object.

**Angular Velocity** Limit the angular speed of an object.
- **Minimum** Clamp angular velocity to this minimum speed (except when totally still), in angle per second.
- **Maximum** Clamp angular velocity to this maximum speed, in angle per second.

**Damping** Increase the “sluggishness” of the object.
- **Translation** Resist movement (0 - 1). At 1 the object is completely immobile.
- **Rotation** Resist rotation, but not the kind of rotation that comes from a collision. For example, if a Motion Controller applies Torque to an object, this damping will be a factor.

**Lock Translation** Seize the object in the world along one or more axes. Note that this is global coordinates, not local or otherwise.

**Lock Rotation** Same as translation, but for rotation (also with respect to the global coordinates).

**Friction** Coulomb friction coefficient when inside the physics distance area.

**Force Field** Controls force field settings.
- **Force** Upward spring force when inside the physics distance area.
- **Distance** Distance of physics area.
- **Damping** Damping of the spring force when inside the physics distance area.
- **Align to Normal** Align dynamic game objects along the surface normal when inside the physics distance area.

**Soft Body Physics**

The most advanced type of object in the Game Engine. Also, it is the most finicky. If you are used to the fun experimentation that comes from playing around with the non-BGE soft body simulations (such as Cloth), you will probably find a frustrating lack of options and exciting results. Do not despair, we are here to help you get some reasonable settings.

Your setup will involve making sure you have sufficient geometry in the soft body’s mesh to support the deformation, as well as tweaking the options.

**Options**

**Actor** Enables detection by Near and Radar Sensors.

**Ghost** Disables collisions completely, similar to No Collision.

** Invisible** Does not display, the same as setting the object to unrendered (such as unchecking the camera icon in the Outliner).
Mass  Affects the reaction due to collision between objects – more massive objects have more inertia. Will also affect material force fields. Will also change behaviors if you are using the suspension and steering portions of Bullet physics.

Shape Match  Upon starting the Game Engine this will record the starting shape of the mesh as the “lowest energy” state. This means that the edges will have tension whenever they are flexed to some other form. This is set to on by default, and in this configuration turns the object into more of a thin sheet of metal rather than a cloth.

Threshold  Linearly scales the pose match.
- A threshold of 1.0 makes it behave like Shape Match on with a Linear Stiffness of 1.0.
- A threshold of 0.0 makes it behave like Shape Match off with a Linear Stiffness of 0.0.

Welding  TODO.

Position Iteration  Increase the accuracy at a linearly-increasing expense of time. The effect is visible especially with soft bodies that fall on sharp corners, though this can slow down even very simple scenes.

Linear Stiffness  Linear stiffness of the soft body links. This is most evident when you have Shape Match off, but it is also evident with it on.

Friction  Dynamic friction coefficient.

Margin  Small value makes the algorithm unstable.

Bending Constraint  Enable Bending Constraints.

Cluster Collision  Affects Collision sensors as well as physics.
- Rigid to Soft Body  Enable cluster collisions between rigid and soft bodies.
- Soft to Soft Body  Enable cluster collisions among soft bodies.

Iterations  Number of cluster iterations.

Hints

- A very important configurable in the case of soft body interactions is World Properties → Physics → Physics Steps → Substeps.
- Surprisingly, the more vertices you have in your hit object, the less likely the soft body is to react with it. If you try letting it hit a Plane, it might stop, but a subdivided Grid might fail.

Note:  Soft bodies do not work with the Collision, Touch, Near, and Radar logic brick sensors.

Warning:  A common practice within the non-BGE Cloth simulator is to employ Force Fields to animate the cloth. These do not work in the BGE, so you will have to figure out a way to use Python (or perhaps plain Logic Bricks) to apply forces to the soft body objects.

Goal Weights

TODO. See Python API.
Occlude Object Physics

If an Occlude type object is between the camera and another object, that other object will not be rasterized (calculated for rendering). It is culled because it is occluded.

There is a demo blend-file to exemplify some concepts: BGE-Physics-Objects-Occluder.blend

- A messed-up, subdivided Cube named “Cube”.
- Another one behind a “Physics Type: Occlude” plane, named “Cube.BG”.
- Another one outside the view Frustum, named “Cube.OffCamera”.

Now observe what happens to the profiling stats for each of the following (in order):

1. Hit P as the scene is. It hums along at a fairly slow rate. On my system the Rasterizer step takes 130ms. The frame rate will finally jump up once the “Cube” object has completely moved out of the view frustum. It is as if the Occluder does not do anything while the Cube is behind it.
2. Delete the “Cube.OffCamera” object above, and notice that there is no improvement in speed. This is the view frustum culling working for you – it does not matter if that object exists or not.
3. Hit Z to view wireframe. Notice that in the 3D View you can see “Cube.BG”, but once you press P, it is not there.
4. Make the “Occluder” object take up the whole camera’s view with S X 5. You will see a huge leap in frame rate, since almost nothing is being Rasterized. On my system the Rasterizer step drops to 5ms.
5. Try a run with World properties → Physics → Occlusion Culling disabled. It will be slow again.
6. Reenable World properties → Physics → Occlusion Culling and run it one more time to prove to yourself that your speed is back.
7. Change the Occluder to “Physics Type: Static”. Notice that it is back to the original slowness.
8. Change it back to “Physics Type: Occlude”.
9. Now make the “Occluder” invisible. The frame rate is back down to its original, slow rate.

Details

As far as Physics is concerned, this type is equivalent to Rigid Object “No collision”. The reason why the Occluder mode is mutually exclusive with other physics mode is to emphasize the fact that occluders should be specifically designed for that purpose and not every mesh should be an occluder. However, you can enable the Occlusion capability on physics objects using Python and Logic bricks. See (Link- TODO).

When an occluder object enters the view frustum, the BGE builds a Z-depth buffer from the faces of that object. Whether the faces are one-side or two-side is important: only the front faces and two-side faces are used to build the Z-depth buffer. If multiple occluders are in the view frustum, the BGE combines them and keeps the most foreground faces.

The resolution of the Z-depth buffer is controllable in the World settings with the “Occlusion Culling Resolution” button:

By default the resolution is 128 pixels for the largest dimension of the viewport while the resolution of the other dimension is set proportionally. Although 128 is a very low resolution, it is sufficient for the purpose of culling. The resolution can be increased to maximum 1024 but at great CPU expense.

The BGE traverses the DBVT (Dynamic Bounding Volume Tree) and for each node checks if it is entirely hidden by the occluders and if so, culls the node (and all the objects it contains).
To further optimize the feature, the BGE builds and uses the Z-depth buffer only when at least one occluder is in the view frustum. Until then, there is no performance decrease compared to regular view frustum culling.

**Recommendations**

Occlusion culling is most useful when the occluders are large objects (buildings, mountains...) that hide many complex objects in an unpredictable way. However, do not be too concerned about performance: even if you use it inappropriately, the performance decrease will be limited due to the structure of the algorithm.

There are situations where occlusion culling will not bring any benefit:

- If the occluders are small and do not hide many objects.
  In that case, occlusion culling is just dragging your CPU down.
- If the occluders are large but hides simple objects.
  In that case, you are better off sending the objects to the GPU.
- If the occluders are large and hides many complex objects but in a very predictable way.
  Example: a house full of complex objects. Although occlusion culling will perform well in this case, you will get better performance by implementing a specific logic that hides/unhides the objects; for instance making the objects visible only when the camera enters the house.
- Occluders can be visible graphic objects but beware that too many faces will make the Z-depth buffer creation slow.
  For example, a terrain is not a good candidate for occlusion: too many faces and too many overlap. Occluder can be invisible objects placed inside more complex objects (e.g. “in the walls” of a building with complex architecture). Occluders can have “holes” through which you will see objects.

**Sensor Physics**

The object detects static and dynamic objects but not other collisions sensors objects. The Sensor is similar to the physics objects that underlie the Near and Radar sensors. Like the Near and Radar object it is:

- Static and ghost.
- Invisible by default.
- Always active to ensure correct collision detection.
- Capable of detecting both static and dynamic objects.
- Ignoring collision with their parent.
- Capable of broadphase filtering based on:
  - Actor option: the colliding object must have the Actor flag set to be detected.
  - Property/material: as specified in the collision sensors attached to it.

Broadphase filtering is important for performance reason: the collision points will be computed only for the objects that pass the broadphase filter.

- Automatically removed from the simulation when no collision sensor is active on it.

Unlike the Near and Radar object it can:

- Take any shape, including triangle mesh.
- Be made visible for debugging (just use the Visible actuator).
• Have multiple collision sensors using it.

Other than that, the sensor objects are ordinary objects. You can move them freely or parent them. When parented to a dynamic object, they can provide advanced collision control to this object.

The type of collision capability depends on the shape:

• Box, sphere, cylinder, cone, convex hull provide volume detection.
• Triangle mesh provides surface detection but you can give some volume to the surface by increasing the margin in the Advanced Settings panel. The margin applies on both sides of the surface.

**Performance tip**

• Sensor objects perform better than Near and Radar: they do less synchronizations because of the Scenegraph optimizations and they can have multiple collision sensors on them (with different property filtering for example).
• Always prefer simple shape (box, sphere) to complex shape whenever possible.
• Always use broadphase filtering (avoid collision sensor with empty property/material).
• Use collision sensor only when you need them. When no collision sensor is active on the sensor object, it is removed from the simulation and consume no CPU.

**Known limitations**

• When running Blender in debug mode, you will see one warning line of the console:

```
warning btCollisionDispatcher::needsCollision: static-static collision!
In release mode this message is not printed.
```

• Collision margin has no effect on sphere, cone and cylinder shape.

**Settings**

**Invisible**  See *Here*.

**Collision Bounds**

See *Here*.

**Navigation Mesh Physics**

Pathfinding in Blender is based on the concept of navigation meshes. With this you can create navigation mesh for your level geometry directly in Blender and use it in Blender Game Engine (BGE) to get your actors to find path to the target and move along it. It is useful for creating an artificial intelligence. Besides path following, there are also a few other steering behaviors which can be assigned to the actor: *seek* and *flee*.

Pathfinding with a navigation mesh is most effective for big static obstacles. To enable actors to avoid small dynamic objects during their movement local obstacle avoidance can be used. If the obstacle simulation is enabled the actor will try to choose direction which is free of collision with obstacles on each frame during execution one of the steering behaviors.
Usage

To setup a navigation mesh access: *Physics* buttons and set the *Physics Type* to *Navigation Mesh*.

Now the following tools will be accessible.

**NavMesh Copy Face Index** Copies the navigation polygon index from the active face to selected faces.

**NavMesh New Face Index** Adds a new navigation polygon index to selected faces.

**NavMesh Reset Index Values** Assigns a new index to every faces.

**NavMesh Clear Data** Removes the navigation data from the mesh.

Character Physics

The character physics type is used for player-controlled characters, for which the other physics types often result unexpected results (bouncing off walls, sliding, etc.) and for which simple kinematics offer much more precision.

Properties

**Step Height** The maximum height of steps the character can run over.

**Jump Force** Upward velocity applied to character when jumping.

**Fall Speed Max** Maximum speed at which the character will fall.

**Max Jumps** Maximum number of jumps the character can make before it hits the ground.

**Note:** Obstacle traversal (e.g. step climbing) is governed by (in order of importance):
- The velocity of the character object
- The shape and margin of the collision bounds (character and obstacle)
- The *Step Height* parameter
- The leading slope of the obstacle

Character fall speed is governed by (in order of importance):
- The *Fall Speed Max* parameter
- The *Step Height* parameter (more *Step Height* -> more fall speed)

Example

Vehicle Controller Physics

Introduction

The Vehicle Controller is a special type of physics object that the Physics Engine (bullet) recognizes.

It is composed of a *rigid body* representing the chassis and a set of wheels that are set to *no collision*. Emphasizing the distinction between a Game Engine, Logical or Render object and its representation for the Physics Engine is important.
To simulate a vehicle as a true rigid body, on top of also rigid body wheels, with a real suspension system made with joints, would be far too complicated and unstable. Cars and other vehicles are complicated mechanical devices and most often we do not want to simulate that, only that it ‘acts as expected’. The Vehicle Controller exists to provide a dedicated way of simulating a vehicle behavior without having to simulate all the physics that would actually happen in the real world. It abstracts the complexity away by providing a simple interface with tweakable parameters such as suspension force, damping and compression.

**How It Works**

Bullet’s approach to a vehicle controller is called a “Raycast Vehicle”. Collision detection for the wheels is approximated by ray casts and the tire friction is an anisotropic friction model.

A raycast vehicle works by casting a ray for each wheel. Using the ray’s intersection point, we can calculate the suspension length and hence the suspension force that is then applied to the chassis, keeping it from hitting the ground. In effect, the vehicle chassis ‘floats’ along on the rays.

The friction force is calculated for each wheel where the ray contacts the ground. This is applied as a sideways and forwards force.

You can check Kester Maddock’s approach to vehicle simulation here. It includes some common problems, workarounds and tips and tricks.

**How to Use**

Currently the Vehicle Controller can only be used as a constraint via Python. There are plans to add it to the interface.

**Setup**

You should have a body acting as the chassis, set it as a ‘Rigid Body’.

The wheels should be separate objects set to ‘No Collision’. The vehicle controller will calculate the collisions for you as rays so, if you set it to something else, it will calculate it twice in different ways and produce weird results.

**Collisions**

A cylinder is typically a good collision shape for the wheels. For the chassis, the shape should be rough, like a box. If the vehicle is very complicated, you should split it into simpler objects and parent those (with their collision shapes) to the vehicle controller so that they will follow it. If your vehicle even has moving bits (weapons, wrecking balls, trolleys, etc.) they should also be simulated separately and connected to the vehicle as a joint.

**Python**

**Assembling the Vehicle**

The overall steps are:

1. Create a constraint for the vehicle and save its ID for future reference.
2. Attach the wheels.
4. Init variables.
You can see an example in the file below.

## Controlling the Vehicle

This is done in two parts and it should be modeled according to the desired behavior. You should think of your gameplay and research appropriate functions for the input. For instance, can the vehicle reverse? jump? drift? does it turn slowly? How much time does it take to brake or get to full speed? The first part is response to keys. Whenever the player presses a key, you should set a value accordingly, such as increase acceleration. Example:

```python
if key[0] == events.UPARROWKEY:
    logic.car["force"] = -15.0
elif key[0] == events.RIGHTARROWKEY:
    logic.car["steer"] -= 0.05
```

The second part is to compute the movement according to your functions:

```python
## apply engine force ##
for i in range(0, totalWheels):
    vehicle.applyEngineForce(logic.car["force"],i)
...
## slowly ease off gas and center steering ##
logic.car["steer"] *= 0.6
logic.car["force"] *= 0.9
```

Both should be run each frame.

### Example ###

The following demo file has a minimal drivable car and was taken from the book: “Game Development with Blender” by Dalai Felinto and Mike Pan.

vehicle_controller_demo.zip (last update 9 September 2014, tested with Blender 2.71)

### Collision Bounds ###

The first thing you must understand is the idea of the 3D Bounding Box. If you run through all the vertices of a mesh and record the lowest and highest x values, you have found the \( x_{\text{min/max}} \) the complete boundary for all x values within the mesh. Do this again for y and z, then make a rectangular prism out of these values, and you have a Bounding Box. This box could be oriented relative globally to the world or locally to the object’s rotation.

The \( x_{\text{extent}} \), then, is half of the distance between the x min/max.
Fig. 30: Demonstration of a Local Bounding Box (left) and a Global Bounding Box (right).

Throughout all of this you must be cognizant of the Object Origin. For the Game engine, the default \texttt{Shift-Ctrl-Alt-C} or \texttt{Set Origin $\rightarrow$ Origin to Geometry} is unlikely to get the desired placement of the Collision Bounds that you want. Instead, you should generally set the origin by looking at the Tool Shelf after you do the \texttt{Set Origin}, and changing the \texttt{Center} from \textit{Median Center} to \textit{Bounds Center}. Blender will remember this change for future \texttt{Shift-Ctrl-Alt-C} executions.

All Collision Bounds are centered on this origin. All boxes are oriented locally, so object rotation matters.

Fig. 31: Setting the origin to \textit{Bounds Center} instead of \textit{Median Center}.

A final introductory comment: When you set the Collision Bounds on an object, Blender will attempt to display a visualization of the bounds in the form of a dotted outline. Currently, there is a bug: \textit{The 3D View} does not display this bounds preview where it actually will be during the game. To see it, go to \texttt{Game $\rightarrow$ Show Physics Visualization} and look for the white (or green, if sleeping) geometry.

Now we can explain the various options for the \textit{Collision Bounds} settings:

- **Default** For Dynamic and Static objects, it is a Triangle Mesh (see below). For everything else, it is a Sphere (see below).

- **Capsule** Which is a cylinder with hemispherical caps, like a pill. Radius of the hemispheres is the greater of the X or Y extent. Height is the Z bounds.

- **Box** The X, Y, Z bounding box, as defined above.

- **Sphere** Radius is defined by the object’s scale (visible in the N properties panel) times the physics radius (can be found in \textit{Physics $\rightarrow$ Attributes $\rightarrow$ Radius}). Note: This is the only bounds that respects the Radius option.

- **Cylinder** Radius is the greater of the x or y extent. Height is the z bounds.

- **Convex Hull** Forms a shrink-wrapped, simplified geometry around the object.

- **Triangle mesh** Most expensive, but most precise. Collision will happen with all of triangulated polygons, instead of using a virtual mesh to approximate that collision.

- **By Hand** This is not an option in the Physics tab’s Collision Bounds settings, but a different approach, entirely. You create a second mesh, which is invisible, to be the physics representation. This becomes the parent for your display object. Then, your display object is set to ghost so it does not fight with the parent object. This method allows you to strike a balance between the accuracy of \textit{Triangle Mesh} with the efficiency of some of the others. See the demo of this in the dune buggy to the right.
Fig. 32: A convex hull sketch.

Fig. 33: Another way to create Collision Bounds – By hand.
Options

There are only two options in the Collision Bounds subpanel.

**Margin**  “Add extra margin around object for collision detection, small amount required for stability.” If you find your objects are getting stuck in places they should not, try increasing this to, say, 0.06.

Sometimes 0.06 is the default (such as on the Default Cube), but sometimes it is not. You have to keep an eye on the setting, or else learn the symptoms so you can respond when it gives you trouble. If you are lazy/paranoid/unsure/diligent/bored, you can always run this on the Python Console to bump all 0.0 margins to 0.06: for `obj in bpy.data.objects: obj.game.collision_margin = obj.game.collision_margin or 0.06`

**Compound**  “Add children to form compound collision object.” Basically, if you have a child object and do not have this enabled, the child’s collisions will not have an effect on that object “family” (though it will still push other objects around). If you do have it checked, the parent’s physics will respond to the child’s collision (thus updating the whole family).

Create Obstacle

Todo

3.1.8 Materials

Game Settings

![Game Settings panel.](image)

This panel contains properties that control how the object surfaces that use the material are rendered in real-time by the Blender Game Engine.

**Backface Cull**  Hide the back-faces of objects rendered with this material. If “Off”, both sides of the surface are visible (at the expense of lower rendering speed). Note that this setting is applied per material and not per face; e.g. if the material is applied to a cube, only the back and front faces of the cube are visible, and not both sides of each face.

**Invisible**  Hide all faces of objects rendered with this material.

**Text**  Use material as Text object in the Game Engine.

**Alpha Blend**  Controls how the alpha channel is used to create a transparent texture in the rendered image.

  **Alpha Sort**  Orders the sequence in which transparent objects are drawn on top of each other, so that ones in front receive more light than ones behind.

  **Alpha Blend**  Uses the alpha values present in the bitmap image sourced in the Image slot.
Alpha Clip  Uses the alpha channel as a simple mask.
Add  Render face transparent and add color of face.
Opaque (default)  All alpha values are ignored; the scene is completely non-transparent.

Face Orientation

Provides options regarding the orientation (i.e. rotation transformation) of faces to which the material is applied.

Shadow  Faces are used for shadow.
Billboard  Billboard with Z-axis constraint.
Halo  Screen-aligned billboard.
Normal (default)  No transformation.

3.1.9 Data

TODO

3.2 Logic Editor

Fig. 35: The Logic editor header.

TODO

3.2.1 Properties Panel

Fig. 36: The Properties panel.

TODO

3.2.2 Logic Bricks Area

TODO
3.2.3 Components Panel

TODO
Fig. 38: The Components panel.
4.1 Armature

See also:
See the Python reference of this object in \texttt{KX\_GameObject} or \texttt{Bl\_ArmatureObject}.

TODO

4.2 Camera

See also:
See the Python reference of this object in \texttt{KX\_GameObject} or \texttt{KX\_CameraObject}.

TODO

4.3 Empty

See also:
See the Python reference of this object in \texttt{KX\_GameObject}.

TODO

4.4 Group

TODO
4.5 Lamp

See also:
See the Python reference of this object in `KX_GameObject` or `KX_LightObject`.

TODO

4.6 Mesh

See also:
See the Python reference of this object in `KX_GameObject` or `KX_NavMeshObject`.

![Add menu with Mesh primitives](image)

Fig. 1: Default Mesh primitives in Add menu.

Mesh objects, along with Text objects, are the only ones which can be seen at runtime. Its uses are many, from visual meshes, invisible physics meshes, etc.

4.7 Object

See also:
See the Python reference of common objects in `KX_GameObject`.

UPBGE supports several object types from Blender, as listed below. The unsupported object types (like Curve, Metaball, Lattice, etc) can be present at Blender scene, but will be removed automatically at game start, so they should not be taken in account in game development process.
Note: Definitions present in this section are related to object’s behavior at game runtime. For definitions about object editing inside Blender editor, refer to Properties Editor.

Each object type will be explained at its respective page, but all of them have common attributes and features which will be explained in this page. Those attributes and features are:

**Ability to contain logic and game properties** This means that logic and property changing is not bound to specific object types: any object can run logic. This should be used with caution, as scattering logic and properties across multiple objects can turn any project into a mess.

**Ability to have a physics type** This means that any object can react to physics, even if this object can’t be visible at runtime (as any object other than Mesh objects). However, only Mesh objects can benefit from certain collision bounds, like Triangle Mesh and Convex Hull.

**World and local transformation values** Even non-visible objects fit a point in scene space. These transformation values are stored in object in a form of world and local transforms (like position, scale and orientation). Note that local transforms are relative to object’s parent (or world center, if it lacks a parent).

**Ability to have animations playing on it** Although Mesh and Armature objects benefits more from this feature (with Shape Keys and Armature actions, respectively), any object can be animated at runtime, from transformations to some specific object attributes.

**General object values** These general values varies from object color, slow parent’s time offset, etc.

Those attributes and features are present in every object type, with no exception. Other attributes depends on specific object types, like Lamps or Cameras.

---

Fig. 2: Supported object types in Add menu.
4.7.1 Text

See also:
See the Python reference of this object in \texttt{KX\_GameObject} or \texttt{KX\_FontObject}.

TODO

4.8 Image

TODO

4.9 Library

TODO

4.10 Sound

TODO

4.11 Action

TODO

4.12 Material

TODO

4.13 Scene

TODO

4.14 Texture

TODO

4.15 World

TODO
5.1 Introduction

*Game Logic* is the default scripting layer in the Game Engine. Each *Game Object* in the game may store a collection of logical components (Logic Bricks) which control its behavior within the scene. Logic bricks can be combined to perform user-defined actions that determine the progression of the simulation.

5.1.1 Logic Bricks

The main part of game logic can be set up through a graphical interface the Logic Editor, and therefore does not require detailed programming knowledge. Logic is set up as blocks (or “bricks”) which represent preprogrammed functions; these can be tweaked and combined to create the game/application. There are three types of logic brick: *Sensors*, *Controllers* and *Actuators*. Sensors are primitive event listeners, which are triggered by specific events, such as a collision, a key press or mouse movement. Controllers carry out logic operations on sensor output, and trigger connected actuators when their operating conditions are met. Actuators interact with the simulation directly, and are the only components in the game which are able to do so (other than the Python controller, and other simulation components such as Physics).

5.1.2 Properties

*Properties* are like variables in other programming languages. They are used to save and access data values either for the whole game (e.g. scores), or for particular objects/players (e.g. names). However, in the Blender Game Engine, a property is associated with an object. Properties can be of different types, and are set up in a special area of the Logic Editor.

5.1.3 States

Another useful feature is object *States*. At any time while the simulation is running, the object will process any logic which belongs to the current state of the object. States can be used to define groups of behavior – e.g. an actor object
may be “sleeping”, “awake” or “dead”, and its logic behavior may be different in each of these three states. The states of an object are set up, displayed and edited in the Controller logic bricks for the object.

## 5.2 Sensors

### 5.2.1 Introduction

Sensors are the logic bricks that cause the logic to do anything. Sensors give an output when something happens, e.g. a trigger event such as a collision between two objects, a key pressed on the keyboard, or a timer for a timed event going off. When a sensor is triggered, a positive pulse is sent to all controllers that are linked to it.

The logic blocks for all types of sensor may be constructed and changed using the Logic Editor; details of this process are given in the Sensor Editing page.

### Common Options

![Common Sensor options.](image)

All sensors have a set of common buttons, fields and menus. They are organized as follows:

**Triangle button**  Collapses the sensor information to a single line (toggle).

**Sensor type menu**  Specifies the type of the sensor.

**Sensor name**  The name of the sensor. This can be selected by the user. It is used to access sensors with Python; it needs to be unique among the selected objects.

**Pin button**  Display the sensor even when it is not linked to a visible states controller.

**Checkbox button**  Sets active state of the sensor.

**X button**  Deletes the sensor.

**Note:**  Triggers

If a controller does not get triggered by any connected sensor (regardless of the sensors’ state) it will not be activated at all.
A sensor triggers the connected controllers on state change. When the sensor changes its state from negative to positive or positive to negative, the sensor triggers the connected controllers. A sensor triggers a connected controller as well when the sensor changes from deactivation to activation.

The following parameters specify how the sensor triggers connected controllers:

**True level triggering** If this is set, the connected controllers will be triggered as long as the sensor’s state is positive. The sensor will trigger with the delay (see parameter: frequency) of the sensor.

**False level triggering** If this is set, the connected controllers will be triggered as long as the sensor’s state is negative. The sensor will trigger with the delay (see parameter: frequency) of the sensor.

**Freq** Despite its name “Frequency”, this parameter sets the delay between repeated triggers, measured in frames (also known as logic ticks). The default value is 0 and it means no delay. It is only used if at least one of the level triggering parameters are enabled.

Raising the value of freq is a good way for saving performance costs by avoiding to execute controllers or activate actuators more often than necessary.

Examples: (assuming the default frame rate with a frequency of 60 Hz (60 frames per second)).

<table>
<thead>
<tr>
<th>Freq</th>
<th>Meaning</th>
<th>Frames with trigger</th>
<th>Frames without trigger</th>
<th>Period in frames</th>
<th>Frequency in frames/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The sensor triggers the next frame.</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td>1</td>
<td>The sensor triggers at one frame and waits another one until it triggers again. It results in half speed.</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>29</td>
<td>The sensor triggers at one frame and waits 29 frames until it triggers again.</td>
<td>1</td>
<td>29</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td>59</td>
<td>The sensor triggers at one frame and waits 59 frames until it triggers again.</td>
<td>1</td>
<td>59</td>
<td>30</td>
<td>1</td>
</tr>
</tbody>
</table>

**Level Button** Triggers connected controllers when state (of the build-in state machine) changes (for more information see States).

The following parameters specify how the sensor’s status gets evaluated:

**Tap Button** Changes the sensor’s state to negative one frame after changing to positive even if the sensor evaluation remains positive. As this is a state change it triggers the connected controllers as well. Only one of Tap or Level can be activated. If the TRUE level triggering is set, the sensor state will consecutive change from True to False until the sensor evaluates False. The FALSE level triggering will be ignored when the Tap parameter is set.

**Invert Button** This inverts the sensor output. If this is set, the sensor’s state will be inverted. This means the sensor’s state changes to positive when evaluating False and changes to False when evaluating True. If the Tap parameter is set, the sensor triggers the controller based on the inverted sensor state.

### 5.2.2 Sensor Editing

Blender sensors can be set up and edited in the left-hand column of the Logic Panel. This page describes the general column controls, and also those parameters which are common to all individual sensor types.

The image shows a typical sensor column with a single example sensor. At the top of this column, the column heading includes menus and buttons to control which of all the sensors in the current Game Logic are displayed.
The column headings contain controls to set which sensors, and the level of detail given, in the sensor column. This is very useful for hiding unnecessary sensors so that the necessary ones are visible and easier to reach. Both these can be controlled individually.

**Sensors**

- **Show Objects** Expands all objects.
- **Hide Objects** Collapses all objects to just a bar with their name.
- **Show Sensors** Expands all sensors.
- **Hide Sensors** Collapses all sensors to bars with their names.

It is also possible to filter which sensors are viewed using the four heading buttons:

- **Sel** Shows all sensors for selected objects.
- **Act** Shows only sensors belonging to the active object.
- **Link** Shows sensors which have a link to a controller.
- **State** Only sensors connected to a controller with active states are shown.

**Object Heading**

In the column list, sensors are grouped by object. By default, sensors for every selected object appear in the list, but this may be modified by the column heading filters.
At the head of each displayed object sensor list, two entries appear:

**Name**  The name of the object.

**Add Sensor**  When clicked, a menu appears with the available sensor types. Selecting an entry adds a new sensor to the object. See *Sensors* for a list of available sensor types.

### 5.2.3 Sensor Types

**Actuator Sensor**

See also:

See the Python reference of this logic brick in *SCA_ActuatorSensor*.

The *Actuator Sensor* detects when a particular actuator receives an activation pulse. It sends a *TRUE* pulse when the specified actuator is activated. The sensor also sends a *FALSE* pulse when the specified actuator is deactivated.

**Properties**

See *Sensor Common Options* for common options.

**Actuator**  Name of actuator (n.b. this must be owned by the same object).
See the Python reference of this logic brick in `SCA_AlwaysSensor`.

The *Always Sensor* gives a continuous output signal at regular intervals. It is used for things that need to be done every logic tick, or at every \( x \) logic tick (with non-null \( f \)), or at start-up (with *Tap*).

![Always sensor](image)

Fig. 6: Always sensor.

**Properties**

See *Sensor Common Options* for common options.

This sensor does not have any special options.

**Example**

**Armature Sensor**

See also:

See the Python reference of this logic brick in `KX_ArmatureSensor`.

The *Armature Sensor* is used to detect changes in values of an IK solver.

![Armature sensor](image)

Fig. 7: Armature Sensor.

**Note:** The *Armature Sensor* only is available for armature objects.

**Properties**

**Bone Name** The bone to check for changes in value.

**Constraint Name** The bone constraint to check for changes in value.
Test  How the sensor checks for changes in the bone.

  State Changed  Any changes will invoke the sensor.
  Lin error below/above  TODO.
  Rot error below/above  TODO.

Value  Some tests will take a value, this value is used in the comparison when detecting changes.

Example

Collision Sensor

See also:
See the Python reference of this logic brick in *KX_CollisionSensor*.

A *Collision Sensor* works like a “touch sensor” but can also filter by property or material. Only objects with the property/material with that name will generate a positive pulse upon collision. Leave blank for collision detection with any object.

Note:  Soft Bodies

The *Collision* sensor cannot detect collisions with soft bodies. This is a limitation in Bullet, the physics library used by the Game Engine.

![Collision Sensor](image)

Fig. 8: Collision sensor.

Properties

See *Sensor Common Options* for common options.

Pulse button  Makes it sensible to other collisions even if it is still in touch with the object that triggered the last positive pulse.

M/P button  Toggles between material and property filtering.

Example

Delay Sensor

See also:
See the Python reference of this logic brick in *SCA_DelaySensor*. 

5.2. Sensors  75
The *Delay Sensor* is designed for delaying reactions a number of logic ticks. This is useful if another action has to be done first or to time events.

![Delay Sensor](image)

Fig. 9: Delay sensor.

**Properties**

See *Sensor Common Options* for common options.

**Delay**  The number of logic ticks the sensor waits before sending a positive pulse.

**Duration**  The number of logic ticks the sensor waits before sending the negative pulse.

**Repeat Button**  Makes the sensor restart after the delay and duration time is up.

**Example**

**Joystick Sensor**

See also:

See the Python reference of this logic brick in *SCA_JoystickSensor*.

The *Joystick Sensor* triggers whenever the joystick moves. It also detects events on a range of ancillary controls on the joystick device (hat, buttons, etc.). More than one joystick may be used (see “Index”). The exact layout of the joystick controls will depend on the make and model of joystick used.

![Joystick Sensor](image)

Fig. 10: Joystick sensor.
Properties

See Sensor Common Options for common options.

**Event Type**  A menu to select which joystick event to use, each is described later.

**Index**  Specifies which joystick to use.

**All Events**  Sensor triggers for all events on this joystick’s current type.

Single Axis

Detect movement in a single joystick Axis.

![Joystick Single Axis](image)

**Fig. 11: Joystick Single Axis.**

**Axis Number**  Axis to detect a change.

- 1 = Horizontal axis (left/right)
- 2 = Vertical axis (forward/back)
- 3 = Paddle axis up/down
- 4 = Joystick axis twist left/right

**Axis Threshold**  Threshold at which joystick fires.

Hat

Dectects movement of a specific hat control on the joystick.

**Hat number**  Specifies which hat to use (max. 2).

**Hat Direction**  Specifies the direction to use: up, down, left, right, up/right, up/left, down/right, down/left.

Axis

**Axis Number**  Specifies the axis (1 or 2).

**Axis Threshold**  Threshold at which joystick fires.

**Axis Direction**  Specifies the direction to use:
Fig. 12: Joystick Hat.

Fig. 13: Joystick Axis.
• (Axis Number = 1) Joystick Left, Right, Up, Down
• (Axis Number = 2) Paddle upper (Left); paddle Lower (Right);
• Joystick twist left (Up) Joystick twist right (Down)

**Button**

![Joystick Button](image)

Fig. 14: Joystick Button.

Specifies the *button number* to use.

**Example**

**Keyboard Sensor**

See also:

See the Python reference of this logic brick in *SCA_KeyboardSensor*.

The *Keyboard* sensor is for detecting keyboard input. It can also save keyboard input to a *String property*.

**Properties**

See *Sensor Common Options* for common options.

- **Key** This field detects presses on a named key. Press the button with no label and a key to assign that key to the sensor. This is the active key, which will trigger the *TRUE* pulse. Click the button and then click outside of the button to deassign the key. A *FALSE* pulse is given when the key is released.

- **All keys button** Sends a *TRUE* pulse when any key is pressed. This is useful for custom key maps with a *Python controller*.

- **First Modifier, Second Modifier** Specifies additional key(s), all of which must be held down while the active key is pressed in order for the sensor to give a *TRUE* pulse. These are selected in the same way as Key. This is useful if you wish to use key combinations, for example *Ctrl-R* or *Shift-Alt-Esc* to do a specific action.

- **Log Toggle** Assigns a *Bool* property which determines if the keystroke will or will not be logged in the target *String*. This property needs to be *TRUE* if you wish to log your keystrokes.
**Target**  The name of property to which the keystrokes are saved. This property must be of type *String*. Together with a *Property* sensor this can be used for example to enter passwords.

**Example**

**Message Sensor**

*See also:*  
See the Python reference of this logic brick in *KX_NetworkMessageSensor*.  
The *Message Sensor* can be used to detect either text messages or property values. The sensor sends a positive pulse once an appropriate message is sent from anywhere in the engine. It can be set up to only send a pulse upon a message with a specific subject.

---

**Note:**  See *Message Actuator* for how to send messages.

---

**Properties**

See *Sensor Common Options* for common options.
Subject  Specifies the message that must be received to trigger the sensor (this can be left blank).

Example

Mouse Sensor

See also:
See the Python reference of this logic brick in SCA_MouseSensor.

The Mouse Sensor detects mouse events.

![Mouse Sensor](image)

Fig. 17: Mouse sensor.

Properties

See Sensor Common Options for common options.

The controller consist only of a list of types of mouse events. A FALSE pulse is given when any of these conditions ends.

Mouse over any  Gives a TRUE pulse if the mouse moves over any game object.

Mouse over  Gives a TRUE pulse if the mouse moves over the owner object.

Movement  Any movement with the mouse causes a stream of TRUE pulses.

Wheel Down  Causes a stream of TRUE pulses as the scroll wheel of the mouse moves down.

Wheel Up  Causes a stream of TRUE pulses as the scroll wheel of the mouse moves up.

Right button  Gives a TRUE pulse.

Middle button  Gives a TRUE pulse.

Left button  Gives a TRUE pulse.

Note:  There is no logic brick for specific mouse movement and reactions (such as first person camera), these have to be coded in Python.

Near Sensor

See also:
See the Python reference of this logic brick in KX_NearSensor.
A *Near Sensor* detects objects that move to within a specific distance of themselves. It can filter objects with properties, like the *Collision sensor*.

**Note:**
1. The Near sensor can detect objects “through” other objects (walls, etc.).
2. Objects must have “Actor” enabled to be detected.

**Note:** The *Near* sensor cannot detect soft bodies. This is a limitation in Bullet, the physics library used by the Game Engine.

![Fig. 18: Near sensor.](image)

**Properties**

See *Sensor Common Options* for common options.

- **Property**  This field can be used to limit the sensor to look for only those objects with this property.
- **Distance**  The number of Blender units it will detect objects within.
- **Reset**  The distance the object needs to be to reset the sensor (send a *FALSE* pulse).

**Example**

**Property Sensor**

See also:

See the Python reference of this logic brick in *SCA_PropertySensor*.

The *Property Sensor* detects changes in the properties of its owner object.

**Properties**

See *Sensor Common Options* for common options.

- **Evaluation Type**  Specifies how the property will be evaluated against the value(s).
- **Greater Than**  Sends a *TRUE* pulse when the property value is greater than the *Value* in the sensor.
**Less Than** Sends a **TRUE** pulse when the property value is less than the **Value** in the sensor.

**Changed** Sends a **TRUE** pulse as soon as the property value changes.

**Interval** Sends a **TRUE** pulse when the **Value** of the property is between the **Min** and **Max** values of the sensor.

**Not Equal** Sends a **TRUE** pulse when the property value differs from the **Value** in the sensor.

**Equal** Sends a **TRUE** pulse when the property value matches the **Value** in the sensor.

**Note:** The names of other properties can also be entered to compare properties.

---

**Example**

**Radar Sensor**

See also:

See the Python reference of this logic brick in *KX_RadarSensor*.

The **Radar Sensor** works much like the **Near sensor**, but only within an angle from an axis, forming an invisible cone with the top in the objects’ center and base at a distance on an axis. This sensor is useful for giving bots sight only in front of them, for example.

**Note:** Soft Bodies

The **Radar** sensor cannot detect soft bodies. This is a limitation in Bullet, the physics library used by the Game Engine.

**Note:**

1. The Radar sensor can detect objects “through” other objects (walls, etc.).

2. Objects must have “Actor” enabled to be detected.
Properties

See Sensor Common Options for common options.

**Property**  This field can be used to limit the sensor to look for only those objects with this property.

**Axis**  This menu determines the direction of the radar cone. The ± signs is whether it is on the axis direction (+), or the opposite (-).

**Angle**  Determines the angle of the cone.

**Distance**  Determines the length of the cone. (Blender units).

Example

**Random Sensor**

See also:

See the Python reference of this logic brick in SCA_RandomSensor.

The Random Sensor generates random pulses.

Properties

See Sensor Common Options for common options.
Seed  This field to enter the initial seed for the random number algorithm.

Note:  0 is not random, but is useful for testing and debugging purposes.

Note:  If you run several times with the same Seed, the sequence of intervals you get will be the same for each run, although the intervals will be randomly distributed.

Example

Ray Sensor

See also:
See the Python reference of this logic brick in `KX_RaySensor`.

The Ray Sensor shoots a ray in the direction of an axis and sends a positive pulse once it hits something. It can be filtered to only detect objects with a given material or property.

![Ray sensor](fig22.png)

Fig. 22: Ray sensor.

Properties

See *Sensor Common Options* for common options.

**Property**  This field can be used to limit the sensor to look for only those objects with this property.

Note:
1. Unless the Property field is set, the Ray sensor can detect objects “through” other objects (walls, etc.).
2. Objects must have “Actor” enabled to be detected.

**Axis**  This menu determines the direction of the ray. The ± signs is whether it is on the axis direction (+), or the opposite (-).

**Range**  Determines the length of the ray (in Blender units).

**X-Ray Mode button**  Makes it x-ray, so that it sees through objects that do not have the property or material specified in the filter field.
Example

5.3 Controllers

5.3.1 Introduction

The controllers are the bricks that collect data sent by the sensors, and also specify the state for which they operate. After performing the specified logic operations, they send out pulse signals to drive the actuators to which they are connected.

When a sensor is activated, it sends out a positive pulse, and when it is deactivated, it sends out a negative pulse. The controllers’ job is to check and combine these pulses to trigger the proper response.

The logic blocks for all types of controller may be constructed and changed using the Logic Editor; details of this process are given in the Controller Editing page.

Controller Types

There are eight types of controller logic brick to carry out the logic process on the input signal(s). This table gives a quick overview of the logic operations performed by the logical controller types. The first column, input, represents the number of positive pulses sent from the connected sensors. The following columns represent each controller’s response to those pulses. True means the conditions of the controller are fulfilled, and the actuators it is connected to will be activated; false means the controller’s conditions are not met and nothing will happen. Please consult the individual controller pages for a more detailed description of each controller.

<table>
<thead>
<tr>
<th>Positive sensors</th>
<th>Controllers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>AND</strong></td>
</tr>
<tr>
<td>None</td>
<td>False</td>
</tr>
<tr>
<td>One</td>
<td>False</td>
</tr>
<tr>
<td>Multiple, not all</td>
<td>False</td>
</tr>
<tr>
<td>All</td>
<td>True</td>
</tr>
</tbody>
</table>

Note: It is assumed that more than one sensor is connected to the controller. For only one sensor, consult the “All” line.

5.3.2 Controller Editing

Blender controllers can be set up and edited in the central column of the Logic Panel. This page describes the general column controls, those parameters which are common to all individual controller types, and how different states for the objects in the logic system can be set up and edited.
The image shows a typical controller column with a single controller. At the top of this column, and for sensors and actuators, the column heading includes menus and buttons to control which of all the controllers in the current Game Logic are displayed.

**Column Heading**

![Controller Column Headings](image)

Fig. 24: Controller Column headings.

The column headings contain controls to set which controllers appear, and the level of detail given, in the controller column. This is very useful for hiding unnecessary controllers so that the necessary ones are visible and easier to reach. Both these can be controlled individually.

**Controllers**

- **Show Objects** Expands all objects.
- **Hide Objects** Collapses all objects to just a bar with their name.
- **Show Controllers** Expands all Controllers.
- **Hide Controllers** Collapses all Controllers to bars with their names.

It is also possible to filter which controllers are viewed using the three heading buttons:

- **Sel** Shows all controllers for selected objects.
- **Act** Shows only controllers belonging to the active object.
- **Link** Shows controllers which have a link to actuators/sensors.

**Object Heading**

![Object Column Headings](image)

In the column list, controllers are grouped by object. By default, controllers for every selected object appear in the list, but this may be modified by the column heading filters.

5.3. **Controllers**
At the head of each displayed object controller list, three entries appear: Used States Button Shows which states are in use for the object. Detailed description of the marked panel is given in States.

Name  The name of the object.

Add Controller  When clicked, a menu appears with the available controller types. Selecting an entry adds a new controller to the object. See Controllers for a list of available controller types.

Standard Controller Parts

The controller heading is standard to every controller.

Controller Type menu  Specifies the type of the controller.

Controller Name  The name of the controller. This can be selected by the user. It is used to access controllers with Python; it needs to be unique among the selected objects.

State Index  Sets the designated state for which this controller will operate.

Preference Button  If on, this controller will operate before all other non-preference controllers (useful for start-up scripts).

Active Checkbox  When unchecked the controller is deactivated, no pulses will be sent to the connect actuators.

X Button  Deletes the sensor.
5.3.3 Controller Types

AND Controller

See also:

See the Python reference of this logic brick in `SCA_ANDController`.

This controller gives a positive (TRUE) output when All its inputs are TRUE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.

Options

![Fig. 25: AND Controller.](image)

See standard controller parts for descriptions of the remaining options.

OR Controller

See also:

See the Python reference of this logic brick in `SCA_ORController`.

This controller gives a positive (TRUE) output when Any one or more of its inputs is TRUE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.

Options

![Fig. 26: OR Controller.](image)

See standard controller parts for descriptions of the remaining options.

NAND Controller

See also:

See the Python reference of this logic brick in `SCA_NANDController`.
This controller *activates* all connected actuators if:

- The game object is in the designated state.
- At least one connected sensor triggers the controller.
- At least one connected sensor evaluated False.

This controller *deactivates* all connected actuators if:

- The game object is in the designated state.
- At least one connected sensor triggers the controller.
- *All* connected sensor evaluated True.

### Options

![NAND Controller](image1.png)

Fig. 27: NAND Controller.

See *standard controller parts* for descriptions of the remaining options.

### NOR Controller

**See also:**

See the Python reference of this logic brick in *SCA_NORController*.

This controller gives a positive (*TRUE*) output when None of its inputs is *TRUE*, and The object is in the designated State. For all other conditions the controller gives a negative (*FALSE*) output.

### Options

![NOR Controller](image2.png)

Fig. 28: NOR Controller.

See *standard controller parts* for descriptions of the remaining options.
XOR Controller

See also:
See the Python reference of this logic brick in `SCA_XORController`.
This controller gives a positive (TRUE) output when One (and only one) of its inputs are TRUE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.

Options

![XOR Controller](image)

Fig. 29: XOR Controller.

See standard controller parts for descriptions of the remaining options.

XNOR Controller

See also:
See the Python reference of this logic brick in `SCA_XNORController`.
This controller gives a positive (TRUE) output when One (and only one) of its inputs is FALSE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.

Options

![XNOR Controller](image)

Fig. 30: XNOR Controller.

See standard controller parts for descriptions of the remaining options.

Expression Controller

See also:
See the Python reference of this logic brick in `SCA_IController`.

5.3. Controllers
This controller evaluates a user written expression, and gives a positive (TRUE) output when The result of the expression is TRUE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.

Fig. 31: Expression Controller.

Expression

The expression, which is written in the field, can consist of variables, constants and operators. These must follow the rules laid out below.

Variables

You can use:
- sensors names,
- properties: assign a game property to an object and use it in a controller expression.

These cannot contain blank spaces.

Operations

Mathematical Operations

Operators: *, /, +, -
Returns: a number
Examples: 3 + 2, 35 / 5

Logical Operations

- Comparison operators: <, >, >=, <=, ==, !=
- Booleans operators: AND, OR, NOT

Returns: True or False.
Examples: 3 > 2 (True), 1 AND 0 (False)
Conditional Statement (if)

Use:

```python
if( expression, pulse_if_expression_is_true, pulse_if_expression_is_false )
```

If the controller evaluates `expression` to True:

- if `pulse_if_expression_is_true` is True, the controller sends a positive pulse to the connected actuators.
- if `pulse_if_expression_is_true` is False, the controller sends a negative pulse to the connected actuators.

If the controller evaluates `expression` to False:

- if `pulse_if_expression_is_false` is True, the controller sends a positive pulse to the connected actuators.
- if `pulse_if_expression_is_false` is False, the controller sends a negative pulse to the connected actuators.

Examples

Given the object has a property `coins` equal to 30:

```python
coins > 20
```
returns True (the controller sends a positive pulse to the connected actuators).

Given the object has:

- A sensor called `Key_Inserted` equal to True.
- A property named `Fuel` equal to False.

```python
Key_Inserted AND Fuel.
```
Will return False (the controller sends a negative pulse to the connected actuators).

This is the same as doing:

```python
if (Key_Inserted AND Fuel, True, False)
```

Instead, you could do:

```python
if (Key_Inserted AND Fuel, False, True)
```
to return a positive pulse when `Key_Inserted AND Fuel` returns False.

You can also do:

```python
if ((Key_Inserted AND Fuel) OR (coins > 20), True, False)
```
This expression returns True, hence in this case the controller sends a positive pulse to the connected actuators.

Parts of the Expression Controller

See `standard controller parts` for descriptions of the remaining options.
Python Controller

See also:
See the Python reference of this logic brick in *SCA_PythonController*.

The Python controller runs a Python script when a sensor triggers the controller. This Python script can interact with the scene or logic bricks through *Blender's API*.

A Python script can either run as an entire file or a single module. A file must be added in the Text editor, and is identified simply by its name, not its path. Names are case sensitive. Modules are identified by the file name *without* the extension followed by a . and then the name of the module. For example:

A file *myscript.py* contains:

```python
def myModule ():
    print("Go Open Source!");
```

The function can be accessed as *myscript.myModule*, which will run `print("Go Open Source!");` every time the controller is triggered.

The entire file can be run by setting the type to *Script* and setting the name to *myscript.py*.

Parts of the Python Controller

Type  Specifies whether it is a module or entire file.
Name  The name of the file to be loaded.
D (Use Debug)  Continuously reloads the file.
See *standard controller parts* for descriptions of the remaining options.

See also:

For more information on the Python API, see:

- The API docs.
- *This chapter for more Game Engine related API.*

## 5.4 Actuators

### 5.4.1 Introduction

Actuators perform actions, such as move, create objects, play a sound. The actuators initiate their functions when they get a positive pulse from one (or more) of their controllers.

The logic blocks for all types of actuator may be constructed and changed using the Logic Editor; details of this process are given in the *Actuator Editing* page.

### Common Options

![Common Actuator options](image)

Fig. 34: Common Actuator options.

All actuators have a set of common buttons, fields and menus. They are organized as follows:

- **Triangle button**  Collapses the sensor information to a single line (toggle).
- **Actuator type menu**  Specifies the type of the sensor.
- **Actuator name**  The name of the actuator. This can be selected by the user. It is used to access actuators with Python; it needs to be unique among the selected objects.
- **X Button**  Deletes the actuator.

### 5.4.2 Actuator Editing

Blender actuators can be set up and edited in the right-hand column of the Logic Panel. This page describes the general column controls, and also those parameters which are common to all individual actuator types.
Fig. 35: Actuator Column with a typical actuator.

The image shows a typical actuator column with a single example actuator. At the top of this column, the column heading includes menus and buttons to control which of all the actuators in the current Game Logic are displayed.

**Column Heading**

Fig. 36: Actuator Column heading.

The column headings contain controls to set which actuators, and the level of detail given, in the actuator column. This is very useful for hiding unnecessary actuators, so that the necessary ones are visible and easier to reach. Both these can be controlled individually.

**Actuators**

- **Show Objects** Expands all objects.
- **Hide Objects** Collapses all objects to just a bar with their name.
- **Show Actuators** Expands all actuators.
- **Hide Actuators** Collapses all actuators to bars with their names.

It is also possible to filter which actuators are viewed using the four heading buttons:
Sel  Shows all actuators for selected objects.
Act  Shows only actuators belonging to the active object.
Link Shows actuators which have a link to a controller.
State Only actuators connected to a controller with active states are shown.

**Object Heading**

![Actuator Object heading](image)

In the column list, actuators are grouped by object. By default, actuators for every selected object appear in the list, but this may be modified by the column heading filters.

At the head of each displayed object sensor list, two entries appear:

- **Name**  The name of the object.
- **Add** When clicked, a menu appears with the available actuator types. Selecting an entry adds a new actuator to the object. See *Actuators* for list of available actuator types.

### 5.4.3 Actuators Types

**Armature Actuator**

See also:

See the Python reference of this logic brick in *BL_ArmatureActuator*.

The *Armature Actuator* is used to modify bone constraints.

![Armature Actuator](image)
Note: The Armature Actuator only is available for armature objects.

**Properties**

**Constraint Type** Action to perform on the bone constraint.
- **Run Armature** Enables an armature to be allowed to move.
- **Enable/Disable** Used to disable a constraint by selecting the constraint via the Data ID.
- **Set Target** Used to change the constraint’s Target by selecting the new target via the Data ID.
- **Set Weight** Used to change the weight of a constraint by selecting a new weight with the *Weight* field.
- **Set Influence** Used to change the Influence of a constraint by selecting a new weight with the *Influence* field.

**Constraint** Several of the *Constraint Types* need you to select a constraint to use, this is done via this Data ID.

**Example**

**Action Actuator**

See also:

See the Python reference of this logic brick in *BL_ActionActuator*.

The Action Actuator controls animation actions, and sets the playback method. It is only visible when an armature is selected, because the actions are stored in the armature.

![Action Actuator](image)

Fig. 39: Action Actuator.

**Properties**

**Action Playback Type**

- **Play** Play F-Curve once from start to end when a TRUE pulse is received.
**Ping Pong**  Play F-Curve once from start to end when a **TRUE** pulse is received. When the end is reached play F-Curve once from end to start when a **TRUE** pulse is received.

**Flipper**  Play F-Curve once from start to end when a **TRUE** pulse is received. (Plays backwards when a **FALSE** pulse is received).

**Loop End**  Play F-Curve continuously from end to start when a **TRUE** pulse is received.

**Loop Start**  Play F-Curve continuously from start to end when a **TRUE** pulse is received.

**Property**  Uses a property to define what frame is displayed.

**Action**  Select the action to use.

**Continue**  Restore last frame when switching on/off, otherwise play from the start each time.

**Start Frame**  Set the start frame of the action.

**End Frame**  Set the end frame of the action.

**Child Button**  Update action on all children objects as well.

**Blending**  Number of frames of motion blending.

**Priority**  Execution priority – lower numbers will override actions with higher numbers. With 2 or more actions at once, the overriding channels must be lower in the stack.

**Frame Property**  Assign the action’s current frame number to this property.

**Property**  Use this property to define the Action position. Only for Property playback type.

**Layer**  The animation layer to play the action on.

**Layer Weight**  How much of the previous layer to blend into this one.

**Example**

**Camera Actuator**

See also:

See the Python reference of this logic brick in **KX_CameraActuator**.

The *Camera Actuator* makes the camera follow or track an object.

![Camera Actuator](image)

Fig. 40: Camera Actuator.

**Properties**

**Camera Object**  Name of the Game Object that the camera follows/tracks.
**Height**  Height the camera tries to stay above the Game Object’s object center.

**Axis**  Axis in which the Camera follows (X or Y).

**Min**  Minimum distance for the camera to follow the Game Object.

**Max**  Maximum distance for the camera to follow the Game Object.

**Damping**  Strength of the constraint that drives the camera behind the target. The higher the parameter, the quicker the camera will adjust to be inside the constrained range (of min, max and height).

### Example

**Constraints Actuator**

See also:

See the Python reference of this logic brick in `KXConstraintActuator`

The *Constraints Actuator* adds a constraint to the location or orientation of an object. This is useful for controlling the physics of the object in-game.

### Properties

**Constraint Mode**  Constraint type to use. Each is described below.

#### Force Field Constraint

Create a force field buffer zone along one axis of the object.

![Fig. 41: Force Field Constraint.](image)

**Damping**  Damping factor of the Fh spring force.

**Distance**  Height of Fh area.

**Rot Fh**  Make game object axis parallel to the normal of trigger object.

**Direction**  Axis in which to create force field (can be + or -, or None).

**Force**  Force value to be used.

**N**  When on, use a horizontal spring force on slopes.

**M/P**  Trigger on another Object will be either Material (M) or Property (P).

**Property**  Property/Material that triggers the Force Field constraint (blank for all Properties/Materials).
**Per**  Persistence button When on, force field constraint always looks at Property/Material; when off, turns itself off if it cannot find the Property/Material.

**Time**  Number of frames for which constraint remains active.

**RotDamp**  Damping factor for rotation.

**Orientation Constraint**

Constrain the specified axis in the Game to a specified direction in the World axis.

![Orientation Constraint](image)

Fig. 42: Orientation Constraint.

**Direction**  Game axis to be modified (X, Y, Z or none).

**Damping**  Delay (frames) of the constraint response.

**Time**  Time (frames) for the constraint to remain active.

**Reference Direction**  Reference direction (global coordinates) for the specified game axis.

**Min Angle**  Minimum angle for the axis modification.

**Max Angle**  Maximum angle for the axis modification.

**Distance Constraint**

Maintain the distance the Game Object has to be from a surface.

![Distance Constraint](image)

Fig. 43: Distance Constraint.

**Direction**  Axis Direction (X, Y, Z, -X, -Y, -Z, or None).
L If on, use local axis (otherwise use World axis).

N If on, orient the Game Object axis with the mesh normal.

Range Maximum length of ray used to check for Material/Property on another game object.

**Force Distance** Distance to be maintained between object and the Material/Property that triggers the *Distance Constraint*.

**Damping** Delay (frames) of the constraint response.

**M/P** Trigger on another Object will be either Material (M) or Property (P).

**Property** Property/Material that triggers the Force Field constraint (blank for all Properties/Materials).

Per Persistence button: When on, force field constraint always looks at Property/Material; when off, turns itself off if it cannot find the Property/Material.

**Time** Number of frames for which constraint remains active.

**Rotation Damping** Damping factor for rotation.

**Location Constraint**

Limit the position of the Game Object within one World Axis direction. To limit movement within an area or volume, use two or three constraints.

![Location Constraint](image)

Fig. 44: Location Constraint.

**Limit** Axis in which to apply limits (LocX, LocY, LocZ or none).

**Min** Minimum limit in specified axis (Blender Units).

**Max** Maximum limit in specified axis (Blender Units).

**Damping** Delay (frames) of the constraint.

**Examples**

**Edit Object Actuator**

See also:

See the Python reference of this logic brick in *KX_SCA_AddObjectActuator, KX_SCA_DynamicActuator, KX_SCA_EndObjectActuator* and *KX_SCA_ReplaceMeshActuator*.

The *Edit Object Actuator* allows the user to edit settings of objects in-game. In example edits the object’s mesh, adds objects, or destroys them. It can also change the mesh of an object (and soon also recreate the collision mesh).
Properties

Edit Object  How the object is modified, each is described below.

Dynamics

Provides a menu of Dynamic Operations to set up dynamics options for object.

Set Mass  Enables the user to set the mass of the current object for Physics.

Disable Rigid Body  Disables the Rigid Body state of the object – disables collision.

Enable Rigid Body  Disables the Rigid Body state of the object – enables collision.

Suspend Dynamics  Suspends the object dynamics (object velocity).

Restore Dynamics  Resumes the object dynamics (object velocity).

Track To

Makes the object “look at” another object, in 2D or 3D. The Y axis is considered the front of the object.

Replace Mesh

Replace mesh with another. Both the mesh and/or its physics can be replaced, together or independently.
Mesh Name of mesh to replace the current mesh.

Gfx Button Replace visible mesh.

Phys Button Replace physics mesh (not compound shapes).

**End Object**

Destroy the current object (note, debug properties will display error Zombie Object in console).

**Add Object**

Adds an object at the center of the current object. The object that is added needs to be on another, hidden, layer.

Object The name of the object that is going to be added.

Time The time (in frames) the object stays alive before it disappears. Zero makes it stay forever.
**Linear Velocity**  Linear Velocity, works like in the motion actuator but on the created object instead of the object itself. Useful for shooting objects, create them with an initial speed.

**Angular Velocity**  Angular velocity, works like in the motion actuator but on the created object instead of the object itself.

**Filter 2D Actuator**

See also:

See the Python reference of this logic brick in `SCA_2DFilterActuator`.

The Filter 2D Actuator adds image filters, that apply on final render of objects. There are several 2D filters each listed below. Most are self-explanatory, however, some are special and will be described in detail later.

- Custom Filter
- Invert
- Sepia
- Gray Scale
- Prewitt
- Sobel
- Laplacian
- Erosion
- Dilation
- Sharpen
- Blur
- Motion Blur
- Remove Filter
- Disable Filter
- Enable Filter

![Fig. 50: Edit Object actuator.](image)

**Properties**

**Filter 2D Type**  Selects the type of 2D filter to use. All 2D filters are rendered with fragment shaders so your hardware must support fragment shaders. Several of the filters are called “built-in filters”, these are: Blur, Sharpen,
Dilation, Erosion, Laplacian, Sobel, Prewitt, Gray Scale, Sepia, and Invert. There are however some filters that work differently from the ones above and are described separately.

**Pass Number**  The pass number for the filter to use.

Details of special filters are described below.

**Motion Blur**

*Motion Blur* is a 2D Filter that needs previous rendering information to produce motion effect on objects. Below you can see *Motion Blur* filter in Blender window, along with its logic bricks:

![Motion Blur filter in Blender window](image)

**Fig. 51: 2D Filters: Motion Blur.**

You can enable Motion Blur filter using a *Python* controller:

```python
from bge import render
render.enableMotionBlur(0.85)
```

And disable it:

```python
from bge import render
render.disableMotionBlur()
```
Enable/Disable Filters

There are two filters which can be used to either Enable or Disable other filters.

To enable/disable a filter on a specific pass:
1. Create appropriate sensor(s) and controller(s).
2. Create a 2D Filter actuator.
3. Select either Enable Filter or Disable Filter depending on what you want to do.
4. Set the pass number you want to disable the filter on it.

Removing Filters

The Remove Filter is used to remove other 2D filters.

To remove a filter on a specific pass:
1. Create appropriate sensor(s) and controller(s).
2. Create a 2D Filter actuator.
3. Select Remove Filter.
4. Set the pass number you want to remove the filter from it.

Custom Filters

Custom filters give you the ability to define your own 2D filter using GLSL. Its usage is the same as built-in filters, but you must select Custom Filter in 2D Filter actuator, then write shader program into the Text Editor, and then place shader script name on actuator.

Blue Sepia Example:

```glsl
uniform sampler2D bgl_RenderedTexture;
void main()
{
  vec4 texcolor = texture2D(bgl_RenderedTexture, gl_TexCoord[0].st);
  float gray = dot(texcolor.rgb, vec3(0.299, 0.587, 0.114));
  gl_FragColor = vec4(gray * vec3(0.8, 1.0, 1.2), texcolor.a);
}
```

Examples
Built-in Filters

Game Actuator

See also:

See the Python reference of this logic brick in `KX_GameActuator`.

The Game Actuator handles the entire game and this way allows the user to perform game-specific functions, such as restart, quit, and load.
Properties

Game

Load bge.logic.globalDict  Load bge.logic.globalDict from .bgeconf.

Save bge.logic.globalDict  Save bge.logic.globalDict to .bgeconf.

Quit Game  Once the actuator is activated, the blenderplayer exits the runtime.

Restart Game  Once the actuator is activated, the blenderplayer restarts the game (reloads from file).

Start Game From File  Once the actuator is activated, the blenderplayer starts the blend-file from the path specified.

   File  Path to the blend-file to load.

Note:  If you use the keyboard sensor as a hook for Esc, in the event that the quit game actuator fails, such as an error in a Python file, the game will be unable to close. Data may be recovered from quit.blend File → Recover Last Session

Example

Message Actuator

See also:

See the Python reference of this logic brick in KX_NetworkMessageActuator.

The Message Actuator allows the user to send data across a scene, and between scenes themselves, which can be received by other objects to activate them.

Properties

To  Object to broadcast to. Leave blank if broadcast to all (or sending to another scene).
**Subject** Subject of message. Useful if sending certain types of message, such as “end-game”, to a message sensor listening for “end game” and Quit Game actuator.

**Body** Body of message sent (only read by Python).
- **Text** User-specified text in body.
- **Property** User-specified property.

**Tip:** You can use the `Message Actuator` to send data, such as scores to other objects, or even across scenes! (alternatively use `bge.logic.globalDict`).

---

**Example**

**Mouse Actuator**

See also:

See the Python reference of this logic brick in `KX_MouseActuator`.

The `Mouse Actuator` allows two modes of operation, to show/hide the mouse cursor or to control object rotation with the mouse. The mouse rotation is flexible enough to allow any type of mouse look as well as banking for flight controls.

**Properties**

- **Mode** Determines the mouse mode.
  - **Visibility** Allows to show/hide the mouse cursor.
  - **Look** Controls the object rotation according to X and/or Y mouse movement. Moreover, the object rotation can be constrained using thresholds and capping angles.

**Visibility**

Mouse actuator for Visibility.

![Visibility](image)

**Visibility** Toggles the visibility of the mouse cursor.

**Look**

Servo control is a powerful way to achieve motion in way which mimics the movement of objects in the physical world. It consists in a servo controller that adjusts the force on the object in order to achieve a given speed.
Note: To make Mouse Look work in a smoother way, it is necessary that the width and height screen resolutions, in the render window, are set with even numbers (i.e. 1920x1080).

![Fig. 57: Mouse Actuator: Look.](image)

**Use X axis, Y axis**  Specifies the object which the actuator owner uses as a reference for movement, for moving platforms for example. If empty it will use world reference.

**Sensitivity**  The target linear velocity, in each of the three axes, which the object will try and achieve.

**Threshold**  Coordinates specified are Global (gray) or Local (white).

**Min**  Sets maximum and minimum limits for the force applied to the object. If disabled (i.e. X, Y or Z buttons are gray) the force applied is unlimited.

**Max**  Set the Proportional Coefficient. This controls the reaction to differences between the actual and target linear velocity.

**Object axis X, Y, Z**  Set the Integral Coefficient. This controls the reaction to the sum of errors so far in this move.

**Local**  Apply locally the rotation around the object axis selected.

**Reset**  Reset the cursor’s X/Y position to the center of the screen space after calculating.

**Example**

**Motion Actuator**

See also:

See the Python reference of this logic brick in *KX_ObjectActuator*.

The *Motion Actuator* sets an object into motion. There are two modes of operation, Simple or Servo, in which the object can either teleport and rotate, or dynamically move.

**Properties**

**Motion Type**  Determines the type of motion:

- **Simple Motion**  Applies a change in location and/or rotation directly.
**Servo Control**  Sets a target speed, and also how quickly it reaches that speed.

**Character Motion**  TODO.

### Simple Motion

*Simple Motion* gives control over position and velocity, but does this as an instant displacement; the object never passes any of the coordinates between the start and end positions. This can interfere with the physical simulation of other objects, and can cause an object to go through another object. The *Servo Control* actuator does not suffer from this, since it produces physically correct velocities, and leaves updating the position to the physics simulation.

![Fig. 58: Motion actuator for Simple Motion.](image)

- **Loc**  The object jumps the number of Blender units entered, each time a pulse is received.
- **Rot**  The object rotates by the specified amount, each time a pulse is received.
- **L**  Coordinates specified are Global (gray) or Local (white).

### Servo Control

The Servo Control actuator influences the velocity of a game object by applying forces, resulting in correct behavior when colliding with other objects controlled by the physics simulation. The amount of force necessary is determined by a *PID controller*, a type of controller that is often used in control systems. Only the positional velocity is influenced by this actuator; it does not control rotation at all, and it controls position only indirectly.

Controlling the position is not necessary in that respect; that is left to a player moving the object via direction-type controls (such as the WASD keys in a first person shooter). In such a scenario, each direction-key sensor should be attached to a different Servo Control actuator setting a different target velocity.

**Tip:** To use the Servo Control actuator, it is necessary to set the object’s Physics Type to “Dynamic” or “Rigid Body”, and to mark the object as “Actor” in the same panel. This actuator does not work with the Character physics type.

- **Reference Object**  Specifies the object which the actuator uses as a reference for the velocity. When set, it will use a velocity relative to that object instead of absolute (i.e. world-relative) velocity. Use this for a player object standing on a moving platform.
- **Linear Velocity**  The target linear velocity for the object.
- **L**  Determines whether the Linear Velocity specified are in Local (button depressed) or Global (button released) coordinates.
X, Y, Z force limits  Sets minimum and maximum limits for the force applied to the object. If disabled (i.e. X, Y or Z buttons are depressed) the force applied is unlimited.

The following three coefficients determine the response to the velocity error, which is the difference between the target velocity and the object’s actual velocity.

Proportional Coefficient  This controls the reaction proportional to the velocity error. Small values cause smooth (but possibly too slow) changes in velocity. Higher values cause rapid changes, but may cause overshooting.

Integral Coefficient  This controls the reaction to the sum of errors so far. Using only the Proportional component results in a systematic velocity error if there is friction: some velocity delta is necessary to produce the force that compensates the friction. Using the Integral component suppresses this effect (the target velocity is achieved on average) but can create oscillations; the control will speed to compensate the initial velocity error. To avoid the oscillation, the Proportional component must be used with the Integral component (the Proportional component damps the control) This is why the GUI sets the Proportional Coefficient systematically when you change the Integral Coefficient.

Derivative Coefficient  Set the Derivative Coefficient. This dampens the acceleration when the target velocity is almost reached.

Character Motion

TODO.

Parent Actuator

See also:
See the Python reference of this logic brick in KX_ParentActuator.

5.4. Actuators
The *Parent Actuator* enables the user to change the parent relationships of the current object.

![Fig. 60: Parent Actuator.](image)

### Properties

**Scene** The type of parenting operation.

- **Set Parent** Make this object to be current object’s parent.
- **Parent Object** Name of parent object.
- **Compound** Add this object shape to the parent shape (only if the parent shape is already compound).
- **Ghost** Make this object ghost while parented.

- **Remove Parent** Remove all parents of current object.

### Example

**Property Actuator**

See also:

See the Python reference of this logic brick in *SCA_PropertyActuator*.

The *Property Actuator* changes the value of a given property (like assigning, adding, or copying) once the actuator itself is activated.

![Fig. 61: Property Actuator.](image)

### Properties

**Mode**

- **Assign** The *Property* target property will become equal to the set *Value* once the actuator is activated.

- **Add** Adds *Value* to the value of the property *Property* once the actuator is activated (enter a negative value to decrease). For *Bool*, a value other than 0 (also negative) is counted as True.
Copy Copies a property from another Object to a property of the actuator owner once the actuator is activated.

Toggle Switches 0 to 1 and any other number than 0 to 0 once the actuator is activated. Useful for on/off switches.

Level TODO.

Property The target property that this actuator will change.

Value The value to be used to change the property.

Example

You have a character, it has a property named “hp” (hit/health points) to determine when he has taken enough damage to die. hp is an int with the start value of 100.

You set up two Collision sensors, one for enemy bullets, and one for picking up more health. The first one is connected (through an AND controller) to an Add Property actuator with the property hp and the value -10. Every time the player is hit by an enemy bullet he loses 10 HP. The other sensor is connected (through an AND controller) to an other Add Property actuator, this one with the value 50. So every time the player collides with a health item the HP increases by 50. Next you set up a Property sensor for an interval, greater than 100. This is connected (through an AND controller) to an Assign Property actuator which is set to 100. So if the players HP increases over 100 it is set to 100.

Random Actuator

See also:

See the Python reference of this logic brick in SCA_RandomActuator.

The Random Actuator creates a random value which can be stored in a property of the object.

Properties

Seed Starting seed for random generator.

Distribution

Distributions from which to select the random value. The default entry of Boolean Constant gives either True or False, which is useful for test purposes.
Each distribution has one common property called: Property. This can be either a float, integer, or a boolean depending on the distribution type.

**Float Neg. Exp.** Values drop off exponentially with the specified half-life time.
- **Half-Life Time** Half-life time.

**Float normal** Random numbers from a normal distribution.
- **Mean** Mean of normal distribution.
- **SD** Standard deviation of normal distribution.

**Float uniform** Random values selected uniformly between a minimum (Min) and maximum (Max) values.

**Float constant** Returns a constant value specified in the Value field.

**Int Poisson** Random numbers from a Poisson distribution. The mean of the equation is defined by the Mean value.

**Int uniform** Random values selected uniformly between a minimum (Min) and maximum (Max) values.

**Int constant** Returns a constant value specified by the Value field.

**Bool Bernoulli** Returns a random distribution using the Bernoulli distribution with specified ratio of TRUE pulses. This ratio is calculated by the Chance value.

**Bool uniform** A 50/50 chance of obtaining True/False.

**Bool constant** Returns a constant value specified in the value field, must be either True or False.

### Example

**Scene Actuator**

See also:

See the Python reference of this logic brick in *KX_SceneActuator*.

The *Scene Actuator* manages the scenes in the user’s blend-file, these can be used as levels or for UI and background.

![Fig. 63: Scene actuator.](image)

### Properties

The actuator has several modes described below:

- **Restart** Restarts the current scene, everything in the scene is reset.
- **Set Scene** Changes scene to selected one.
- **Set Camera** Changes which camera is used.
- **Add Overlay Scene** This adds an other scene, and draws it on top of the current scene. It is good for interfacing: keeping the health bar, ammo meter, speed meter in an overlay scene makes them always visible.
- **Add Background Scene** This is the opposite of an overlay scene, it is drawn behind the current scene.
- **Remove Scene** Removes a scene.
Suspend Scene  Pauses a scene.
Resume Scene  Resumes a paused scene.

Note: A scene that is paused cannot resume itself. You need an active scene to resume other scene that it is paused.

Example

Steering Actuator

See also:
See the Python reference of this logic brick in KX_SteeringActuator.

The Steering Actuator provides simple pathfinding for an object by moving it towards a target object, with options to seek, flee, or follow a path. This actuator will not actually try to avoid obstacles by deviating the objects course.

Fig. 64: Steering Actuator.

Properties

Behavior  Seek, Flee or Path following
Target Object  The game object to seek.
Navigation Mesh Object  The name of the navigation mesh object used by the Steering Actuator when in Path following behavior. The game object will use the Navigation Mesh to create a path to follow the Target Object.

Tip:  You can create your own mesh to use for navigation and make it a Navigation Mesh in: Properties editor → Physics → Physics panel → choosing Physics Type: Navigation Mesh Or you can let Blender create a Navigation Mesh, then select a mesh. (Floor, or ground, or etc.): Properties editor → Scene → Navigation mesh object panel → Build navigation mesh

Distance  The maximum distance for the game object approach the Target Object.

Velocity  The velocity used to seek the Target Object.

Acceleration  The maximum acceleration to use when seeking the Target Object.

Turn Speed  The maximum turning speed to use when seeking the Target Object.

Facing  Set a game object axis that always faces the Target Object.

Axis  The game object axis that always faces the Target Object. Options are: Positive (X, Y, Z) and Negative (-X, -Y, -Z).

Axis N  Use the Normal of the Navigation Mesh to align the up vector of the game object.

Self Terminated

Disabled  Stops moving toward the Target Object once it reaches the maximum distance to approach the Target Object. Will follow the Target Object if it moves further away than the maximum distance.

Enabled  Stops moving toward the Target Object once it reaches the maximum distance to approach the Target Object. Will not follow even if the Target Object moves further away than the maximum distance.

Visualize  This checkbox let the user specify whether to show or not the debug informations of the actuator. It is also necessary to enable Debug Properties in the Display menu of the Render tab.

Example

Sound Actuator

See also:

See the Python reference of this logic brick in KX_SoundActuator.

The Sound Actuator allows the user to play sound files in the game engine.

Properties

Sound File  Load a new sound file or select one from the list.

Play Mode  How the sound effect is played.

- **Play Stop**  The sound effect is played when activated. Stops instantly when deactivated.
- **Play End**  The sound effect is played when activated. When deactivated, stops after finishing playing the sound. The sound is not replayed if activated while still playing.
- **Loop Stop**  The sound is played as infinite loop when activated. Stops instantly when deactivated.
Loop End  The sound is played as infinite loop when activated. When deactivated, stops after finishing playing the sound.

Loop Bidirectional  The sound is played as infinite ping-pong loop. When deactivated, stops after finishing playing the sound.

Loop Bidirectional Stop  The sound is played as infinite ping-pong loop. Stops instantly when deactivated.

Volume  The volume at which the sound effect is played.

Pitch  The pitch at which the sound effect is played. 0 is default, 12 is one octave.

3D Sound

Fig. 65: Sound Actuator.

Fig. 66: The cones point in the direction of the objects negative Z axis.

If enabled, the sound is affected by distance, speed of the emitting object and various other things. The options below are only available if 3d Sound is enabled.
Note: 3D sound is influenced by the Audio panel in Scene Settings. A brief description of the different distance models can be found here.

Minimum Gain  The minimum gain of the sound, no matter how far it is away.

Maximum Gain  The maximum gain of the sound, no matter how near it is.

Reference Distance  The cones point in the direction of the objects negative Z axis. The distance at which the sound has a gain of 1.0.

Maximum Distance  The maximum distance at which the sound can be heard.

Rolloff  The influence factor on volume depending on distance. The higher, the more the sound will fade with distance.

Cone Outer Gain  The gain outside the outer cone. The gain inside the outer cone will be interpolated between this value and the normal gain inside the inner cone (Volume). Note that the cones always point in the direction of the objects local -Z axis (figure right).

Cone Outer Angle  The angle of the outer cone.

Cone Inner Angle  The angle of the inner cone.

Example

State Actuator

See also:

See the Python reference of this logic brick in `KX_StateActuator`.

The State Actuator allows the user to create complex logic, while retaining a clear user interface. It does this by having different states, and performing operations upon them.

Note: With the state actuator, you can create tiers of logic, without the need for hundreds of properties. Use it well, and you benefit greatly, but often problems may be circumvented by Python.

Fig. 67: State actuator.

Properties

Operation

  - **Change State**  Change from the current state to the state specified.
  - **Remove State**  Removes the specified states from the active states (deactivates them).
  - **Add State**  Adds the specified states to the active states (activates them).
  - **Set State**  Moves from the current state to the state specified, deactivating other added states.
Example

Visibility Actuator

See also:
See the Python reference of this logic brick in `KX_VisibilityActuator`.
The *Visibility Actuator* allows the user to change the visibility of objects during run-time.

**Note:** Using the visibility actuator will save on Rasterizer usage, however, not Physics, and so is limited in terms of Level of Detail (LOD). For LOD look at replace mesh, but be aware that the logic required can negate the effect of the LOD.

![Visibility actuator](image)

**Fig. 68:** Visibility actuator.

**Properties**

- **Visible** Toggle checkbox to toggle visibility.
- **Occlusion** Toggle checkbox to toggle occlusion. Must be initialized from the *Physics* tab.
- **Children** Toggle checkbox to toggle recursive setting – will set visibility / occlusion state to all child objects, children of children (recursively).

Example

5.5 Properties

Properties are the game logic equivalent to variables. They are stored with the object, and can be used to represent things about them such as ammo, health, name, and so on.

5.5.1 Property Types

There are five types of properties:

- **Timer** Starts at the property value and counts upwards as long as the object exists. It can for example be used if you want to know how long time it takes the player to complete a level.
- **Float** Uses decimal numbers as values, can range from -10000.000 to 10000.000. It is useful for precision values.
- **Integer** Uses integers (whole numbers) as values, between -10000 and 10000. Useful for counting things such as ammunition, where decimals are unnecessary.
- **String** Takes text as value. Can store 128 characters.
Boolean

Boolean variable, has two values: true or false. This is useful for things that have only two modes, like a light switch.

5.5.2 Using Properties

When a game is running, values of properties are set, manipulated, and evaluated using the Property Sensor and the Property Actuator.

Logic Properties are created and edited using the panel on the left of the Logic Editor panel. The top menu provides a list of the available property types.

Add Game Property button  This button adds a new property to the list, default is a Float property named prop, followed by a number if there already is one with this name.

Name field  Where you give your property its name, this is how you are going to access it through Python or expressions. The way to do so in Python is by dictionary style look-up (GameObject["propname"]). The name is case sensitive.

Type menu  This menu determines which type of property it is. The available options are in Property Types.

Value field  Sets the initial value of the property.

Information (i button)  Display property value in debug information. If debugging is turned on, the value of the property is given in the top left-hand corner of the screen while the game is running. To turn debugging on, tick the Show Debug Properties checkbox in the Game menu. All properties with debugging activated will then be presented with their object name, property name and value during gameplay. This is useful if you suspect something with your properties is causing problems.

5.6 States

In the BGE, an object can have different “states”. At any time while the game is playing, the current state of the object defines its behavior. For instance, a character in your game may have states representing awake, sleeping or dead. At any moment their behavior in response to a loud bang will be dependent on their current state; they may crouch down (awake); wake up (asleep) or do nothing (dead).

5.6.1 How States Operate

States are set up and used through controllers: note that only controllers, not actuators and sensors, are directly controlled by the state system. Each object has a number of states (up to 30; default = 1), and can only be in one state at any particular time. A controller must always specify the state for which it will operate – it will only give an output pulse if a) its logic conditions are met, and b) the object is currently in the specified State. States are set up and edited in the object’s Controller settings (for details see below).
Tip: State settings are automatic in simple games. By default, the number of states for each object is 1, and all controllers are set to use State 1. So, if a game does not need multiple states, everything will work without explicitly setting states – you do not need to bother about states at all.

One of the actuators, the State actuator, can set or unset the object’s State bits, and so allow the object’s reaction to a sensor signal to depend on its current state. So, in the above example, the actor will have a number of controllers connected to the “loud bang” sensor, for each of the “awake”, “asleep” or “dead” states. These will operate different actuators depending on the current state of the actor, and some of these actuators may switch the actor’s state under appropriate conditions.

5.6.2 Editing States

States are set up and edited using the Controller (center) column of the Game Logic Panel. To see the State panel, click on the State Panel Button shown. The panel shows two areas for each of the 30 available states; these show Visible states, and Initial states (see below). Setting up the State system for a game is performed by choosing the appropriate state for each controller in the object’s logic.

The display of an object’s state logic, and other housekeeping, is carried out using the State Panel for the object, which is switched on and off using the button shown. The panel is divided into two halves, Visible and Initial.
5.6.3 Visible States

In the Visible area, each of the 30 available states is represented by a light-gray square. This panel shows what logic is visible for the logic brick displayed for the object. At the right is the All button; if clicked, then all the object’s logic bricks are displayed (this is a toggle), and all State Panel squares are light gray. Otherwise, individual states can be clicked to make their logic visible. (Note that you can click more than one square). Clicking the square again deselects the state.

States for the object that are in use (i.e. the object has controllers which operate in that state) have dots in them, and squares are dark gray if these controllers are shown in the Game Logic display. The display of their connected sensors and actuators can also be controlled if the State buttons at the head of their columns are ticked.

![State Panel initial](image)

Fig. 72: State Panel initial.

5.6.4 Initial State

In the Initial area, each of the 30 available states is again represented by a light-gray square. One of these states may be clicked as the state in which the object starts when the game is run.

At the right is the button; if clicked, and the Game→Show Debug Properties is clicked, the current state of the object is shown in the top left-hand corner of the display while the game is running.
6.1 Licensing of Games

Blender and the UPBGE/BGE are licensed as GNU GPL, which means that your games (if they include Blender software) have to comply with that license as well. This only applies to the software, or the bundle if it has software in it, not to the artwork you make with Blender. All your Blender creations are your sole property.

GNU GPL – also called “Free Software” – is a license that aims at keeping the licensed software free, forever. GNU GPL does not allow you to add new restrictions or limitations on the software you received under that license. That works fine if you want your clients or your audience to have the same rights as you have (with Blender).

In summary, the software and source code are bound to the GNU GPL, but the blend-files (models, textures, sounds) are not.

6.1.1 Standalone Games

In case you save out your game as a single standalone (using addons for this purpose, for example), the blend-file gets included in the binary (the Blender Player). That requires the blend-file to be compatible with the GNU GPL license.

In this case, you could decide to load and run another blend-file game (using the Game Actuator logic brick). That file then is not part of the binary, so you can apply any license you wish on it.

6.1.2 More Information

More information you can find in the blender.org FAQ.

6.2 Performance Considerations

When developing games, game engineers, software and hardware developers uses some tools to fine-tune their games to specific platforms and operating systems, defining a basic usage scenario whereas the users would have the best
possible experience with the game.

Most of these tools, are software tools available for the specific Game Engines whereas the games were being developed and will run.

Blender Game Engine also comes with some visual tools to fine-tune the games being developed, so the game developers could test the best usage scenario and minimum software and hardware requirements to run the game.

In Blender, those tools are available at the System and Display panel of Render tab in the Properties editor. There are options for specific performance adjusts and measurements, ways to control the frame rate or the way the contents are rendered in Blender window (game viewport) while the game runs, as well as controls for maintaining geometry allocated in graphic cards memory.

**Blender Game Engine rendering system controls:** System – Controls for Scene rendering while the game is running.

**Blender Game Engine Performance measurements:** Display – Controls for showing specific data about performance while the game is running.

### 6.3 Standalone Player

The standalone player allows a Blender game to be run without having to load the Blender system. This allows games to be distributed to other users, without requiring them a detailed knowledge of Blender (and also without the possibility of unauthorized modification). Note that the Game Engine Save as Runtime is an add-on facility which must be pre-loaded before use.

The following procedure will give a standalone version of a working game.

1. File → User Preferences → Add-ons → Game Engine → Save As Game Engine Runtime enable the checkbox. (You can also Save User Settings, in which case the add-on will always be present whenever Blender is reloaded).
2. File → Export → Save As Game Engine Runtime (give appropriate directory/filename) confirm with Save as Game Engine Runtime.

The game can then be executed by running the appropriate .exe file. Note that all appropriate libraries are automatically loaded by the add-on.

If you are interested in licensing your game, read Licensing for a discussion of the issues involved.

**Tip:** Exporting...

If the game is to be exported to other computers, make a new empty directory for the game runtime and all its ancillary libraries, etc. Then make sure the whole directory is transferred to the target computer.

### 6.4 Release Procedure

TODO
7.1 Introduction to the UPBGE Manual

Our aim to provide a complete and concise reference manual.

The manual has as a goal to provide:

- Insight in Blender/UPBGE’s way of working, its internal (technical) design - in order to understand options and tools.
- Detailed functional description of all features, tools and options in UPBGE.

7.1.1 Conventions

**Keyboards**

Hotkey letters are shown in this manual like they appear on a keyboard; for example:

- \text{G} refers to the lowercase \text{g}.
- \text{Shift, Ctrl, Alt} are specified as modifier keys.
- \text{Ctrl-W, Shift-Alt-A,...} indicates that these keys should be pressed simultaneously.
- \text{Numpad0 to Numpad9, NumpadPlus} refer to the keys on the separate numeric keypad.

Other keys are referred to by their names, such as \text{Esc, Tab, F1 to F12}. Of special note are the arrow keys, \text{Left, Right} and so on.

**Mouse**

This manual refers to mouse buttons as:

- \text{LMB} Left Mouse Button
- \text{RMB} Right Mouse Button
7.1.2 Contribute

The UPBGE Manual is a community driven effort to which anyone can contribute. Either if you found a typo or if you want to improve the general quality of the documentation, there are several options for helping out. You can:

1. Fix problems, improve the documentation and write new sections. Make a Pull Request with the proposed changes in UPBGE Docs project.
2. Report problems in the documentation.
3. Get involved in discussions through the IRC channel at “upbgecoders” channel.

7.2 License

UPBGE itself is released under the GNU General Public License.

Except where otherwise noted, the content of the UPBGE Manual is available under a Creative Commons Attribution-ShareAlike 4.0 International License or any later version. Excluded from the CC-BY-SA are also the used logos, trademarks, icons, source code and Python scripts.

This UPBGE manual contains exclusive information from UPBGE Documentation Team but it is also base on others manuals and articles whose authors are exposed below:

The Blender Manual
by the Blender Documentation Team
is licensed under a CC-BY-SA v4.0.

The "Game Development with Blender" book
by the Mike Pan, Dalai Felinto and Other Authors
is licensed under a CC-BY-SA v4.0.

The UPBGE Manual
by the UPBGE Documentation Team
is licensed under a CC-BY-SA v4.0.

It means that when contributing to the manual you do not hold exclusive copyright to your text. You are, of course, acknowledged and appreciated for your contribution. However, others can change and improve your text in order to keep the manual consistent and up to date.

If you want to use the UPBGE manual in other sites or other formats, please attribute the different authors and include hyperlinks (online) or URLs (in print) to the different manuals as pointed above out.

7.3 What’s New

This page lists major changes and additions to the manual. The UPBGE release notes are located on the website.

8.1 Game Types (bge.types)

8.1.1 Introduction

This module contains the classes that appear as instances in the Game Engine. A script must interact with these classes if it is to affect the behaviour of objects in a game.

The following example would move an object (i.e. an instance of \texttt{KX\_GameObject}) one unit up.

```python
# bge.types.SCA\_PythonController
cnt = bge.logic.getCurrentController()

# bge.types.KX\_GameObject
obj = cnt.owner
obj.worldPosition.z += 1
```

To run the code, it could be placed in a Blender text block and executed with a \texttt{SCA\_PythonController} logic brick.

8.1.2 Categories

Objects

- \texttt{BL\_ArmatureObject}
- \texttt{KX\_Camera}
- \texttt{KX\_FontObject}
- \texttt{KX\_GameObject}
- \texttt{KX\_LightObject}
- \texttt{KX\_NavMeshObject}
Logic Bricks

- SCA_ILogicBrick

Sensors

- SCA_ISensor
- KX_ArmatureSensor
- KX_CollisionSensor
- KX_MouseFocusSensor
- KX_MovementSensor
- KX_NearSensor
- KX_NetworkMessageSensor
- KX_RadarSensor
- KX_RaySensor
- SCA_ActuatorSensor
- SCA_AlwaysSensor
- SCA_DelaySensor
- SCA_JoystickSensor
- SCA_KeyboardSensor
- SCA_MouseSensor
- SCA_PropertySensor
- SCA_RandomSensor

Controllers

- SCA_IController
- SCA_ANDController
- SCA_NANDController
- SCA_ORController
- SCA_NORController
- SCA_XORController
- SCA_XNORController
- SCA_PythonController

Actuators

- SCA_IActuator
- BL_ActionActuator
- BL_ArmatureActuator
- KX_CameraActuator
- KX_ConstraintActuator
- KX_GameActuator
- KX_MouseActuator
- KX_NetworkMessageActuator
- KX_ObjectActuator
- KX_ParentActuator
- KX_SCA_AddObjectActuator
- KX_SCA_DynamicActuator
- KX_SCA_EndObjectActuator
- KX_SCA_ReplaceMeshActuator
• KX_SceneActuator
• KX_SoundActuator
• KX_StateActuator
• KX_SteeringActuator
• KX_TrackToActuator
• KX.VisibilityActuator
• SCA_2DFilterActuator
• SCA_PropertyActuator
• SCA_RandomActuator
• SCA_VibrationActuator

Blender Data

• BL_ArmatureBone
• BL_ArmatureChannel
• BL_ArmatureConstraint
• BL_Texture
• KX_BlenderMaterial
• KX_Mesh
• KX_Scene
• KX_WorldInfo

UPBGE Data

• BL_Shader
• EXP_ListValue
• EXP_PropValue
• EXP_PyObjectPlus
• EXP_Value
• KX_2DFilterManager
• KX_2DFilterOffScreen
• KX_2DFilter
• KX_BatchGroup
• KX_BoundingBox
• KX_CharacterWrapper
• KX_CollisionContactPoint
• KX_ConstraintWrapper
• KX_CubeMap
• KX_LibLoadStatus
• KX_LodLevel
• KX_LodManager
• KX_PlanarMap
• KX_PolyProxy
• KX_PythonComponent
• KX_TextureRenderer
• KX_VehicleWrapper
• SCA_IObject
• SCA_InputEvent
• SCA_PythonJoystick
• SCA_PythonKeyboard
• SCA_PythonMouse
8.1.3 Index

**BL_ActionActuator(SCA_IActuator)**

Base class — *SCA_IActuator*

class **BL_ActionActuator** *(SCA_IActuator)*

Action Actuators apply an action to an actor.

- **action**
  The name of the action to set as the current action.
  
  Type string

- **frameStart**
  Specifies the starting frame of the animation.
  
  Type float

- **frameEnd**
  Specifies the ending frame of the animation.
  
  Type float

- **blendIn**
  Specifies the number of frames of animation to generate when making transitions between actions.
  
  Type float

- **priority**
  Sets the priority of this actuator. Actuators will lower priority numbers will override actuators with higher numbers.
  
  Type integer

- **frame**
  Sets the current frame for the animation.
  
  Type float

- **propName**
  Sets the property to be used in FromProp playback mode.
  
  Type string

- **mode**
  The operation mode of the actuator. Can be one of these constants.
  
  Type integer

- **useContinue**
  The actions continue option. True or False. When True, the action will always play from where last left off, otherwise negative events to this actuator will reset it to its start frame.
  
  Type boolean

- **framePropName**
  The name of the property that is set to the current frame number.
  
  Type string
BL_ArmatureActuator(SCA_IActuator)

base class — SCA_IActuator

class BL_ArmatureActuator(SCA_IActuator)
Armature Actuators change constraint condition on armatures.

- **type**
  The type of action that the actuator executes when it is active.
  
  Can be one of *these constants*
  
  Type integer

- **constraint**
  The constraint object this actuator is controlling.
  
  Type BL_ArmatureConstraint

- **target**
  The object that this actuator will set as primary target to the constraint it controls.
  
  Type KX_GameObject

- **subtarget**
  The object that this actuator will set as secondary target to the constraint it controls.
  
  Type KX_GameObject.

**Note:** Currently, the only secondary target is the pole target for IK constraint.

- **weight**
  The weight this actuator will set on the constraint it controls.
  
  Type float.

**Note:** Currently only the IK constraint has a weight. It must be a value between 0 and 1.

**Note:** A weight of 0 disables a constraint while still updating constraint runtime values (see BL_ArmatureConstraint)

- **influence**
  The influence this actuator will set on the constraint it controls.
  
  Type float.

BL_ArmatureBone(EXP_PyObjectPlus)

base class — EXP_PyObjectPlus

class BL_ArmatureBone(EXP_PyObjectPlus)
Proxy to Blender bone structure. All fields are read-only and comply to RNA names. All space attribute correspond to the rest pose.

- **name**
  bone name.
Type string

connected
true when the bone head is struck to the parent’s tail.

Type boolean

hinge
true when bone doesn’t inherit rotation or scale from parent bone.

Type boolean

inherit_scale
true when bone inherits scaling from parent bone.

Type boolean

bbone_segments
number of B-bone segments.

Type integer

roll
bone rotation around head-tail axis.

Type float

head
location of head end of the bone in parent bone space.

Type vector [x, y, z]

tail
location of head end of the bone in parent bone space.

Type vector [x, y, z]

length
bone length.

Type float

arm_head
location of head end of the bone in armature space.

Type vector [x, y, z]

arm_tail
location of tail end of the bone in armature space.

Type vector [x, y, z]

arm_mat
matrix of the bone head in armature space.

Type matrix [4][4]

Note: This matrix has no scale part.

bone_mat
rotation matrix of the bone in parent bone space.

Type matrix [3][3]
parent
  parent bone, or None for root bone.
  Type  `BL_ArmatureBone`

children
  list of bone’s children.
  Type  list of `BL_ArmatureBone`

**BL_ArmatureChannel**(EXP_PyObjectPlus)

class `BL_ArmatureChannel` *(EXP_PyObjectPlus)*
Proxy to armature pose channel. Allows to read and set armature pose. The attributes are identical to RNA attributes, but mostly in read-only mode.

name
  channel name (=bone name), read-only.
  Type  string

bone
  return the bone object corresponding to this pose channel, read-only.
  Type  `BL_ArmatureBone`

parent
  return the parent channel object, None if root channel, read-only.
  Type  `BL_ArmatureChannel`

has_ik
  true if the bone is part of an active IK chain, read-only. This flag is not set when an IK constraint is defined but not enabled (miss target information for example).
  Type  boolean

ik_dof_x
  true if the bone is free to rotation in the X axis, read-only.
  Type  boolean

ik_dof_y
  true if the bone is free to rotation in the Y axis, read-only.
  Type  boolean

ik_dof_z
  true if the bone is free to rotation in the Z axis, read-only.
  Type  boolean

ik_limit_x
  true if a limit is imposed on X rotation, read-only.
  Type  boolean

ik_limit_y
  true if a limit is imposed on Y rotation, read-only.
  Type  boolean

8.1. Game Types *(bge.types)*
**ik_limit_z**
true if a limit is imposed on Z rotation, read-only.

**Type** boolean

**ik_rot_control**
true if channel rotation should applied as IK constraint, read-only.

**Type** boolean

**ik_lin_control**
true if channel size should applied as IK constraint, read-only.

**Type** boolean

**location**
displacement of the bone head in armature local space, read-write.

**Type** vector [X, Y, Z]

**Note:** You can only move a bone if it is unconnected to its parent. An action playing on the armature may change the value. An IK chain does not update this value, see joint_rotation.

**Note:** Changing this field has no immediate effect, the pose is updated when the armature is updated during the graphic render (see `BL_ArmatureObject.update`).

**scale**
scale of the bone relative to its parent, read-write.

**Type** vector [sizeX, sizeY, sizeZ]

**Note:** An action playing on the armature may change the value. An IK chain does not update this value, see joint_rotation.

**Note:** Changing this field has no immediate effect, the pose is updated when the armature is updated during the graphic render (see `BL_ArmatureObject.update`).

**rotation_quaternion**
rotation of the bone relative to its parent expressed as a quaternion, read-write.

**Type** vector [qr, qi, qj, qk]

**Note:** This field is only used if rotation_mode is 0. An action playing on the armature may change the value. An IK chain does not update this value, see joint_rotation.

**Note:** Changing this field has no immediate effect, the pose is updated when the armature is updated during the graphic render (see `BL_ArmatureObject.update`).

**rotation_euler**
rotation of the bone relative to its parent expressed as a set of euler angles, read-write.
Type vector [X, Y, Z]

Note: This field is only used if rotation_mode is > 0. You must always pass the angles in [X, Y, Z] order; the order of applying the angles to the bone depends on rotation_mode. An action playing on the armature may change this field. An IK chain does not update this value, see joint_rotation.

Note: Changing this field has no immediate effect, the pose is updated when the armature is updated during the graphic render (see BL_ArmatureObject.update)

rotation_mode
Method of updating the bone rotation, read-write.
Type integer (one of these constants)

channel_matrix
pose matrix in bone space (deformation of the bone due to action, constraint, etc), Read-only. This field is updated after the graphic render, it represents the current pose.
Type matrix [4][4]

pose_matrix
pose matrix in armature space, read-only. This field is updated after the graphic render, it represents the current pose.
Type matrix [4][4]

pose_head
position of bone head in armature space, read-only.
Type vector [x, y, z]

pose_tail
position of bone tail in armature space, read-only.
Type vector [x, y, z]

ik_min_x
minimum value of X rotation in degree (<= 0) when X rotation is limited (see ik_limit_x), read-only.
Type float

ik_max_x
maximum value of X rotation in degree (>= 0) when X rotation is limited (see ik_limit_x), read-only.
Type float

ik_min_y
minimum value of Y rotation in degree (<= 0) when Y rotation is limited (see ik_limit_y), read-only.
Type float

ik_max_y
maximum value of Y rotation in degree (>= 0) when Y rotation is limited (see ik_limit_y), read-only.
Type float

ik_min_z
minimum value of Z rotation in degree (<= 0) when Z rotation is limited (see ik_limit_z), read-only.
Type float
**ik_max_z**
maximum value of Z rotation in degree (>= 0) when Z rotation is limited (see ik_limit_z), read-only.

**Type** float

**ik_stiffness_x**
bone rotation stiffness in X axis, read-only.

**Type** float between 0 and 1

**ik_stiffness_y**
bone rotation stiffness in Y axis, read-only.

**Type** float between 0 and 1

**ik_stiffness_z**
bone rotation stiffness in Z axis, read-only.

**Type** float between 0 and 1

**ik_stretch**
ratio of scale change that is allowed, 0=bone can’t change size, read-only.

**Type** float

**ik_rot_weight**
weight of rotation constraint when ik_rot_control is set, read-write.

**Type** float between 0 and 1

**ik_lin_weight**
weight of size constraint when ik_lin_control is set, read-write.

**Type** float between 0 and 1

**joint_rotation**
Control bone rotation in term of joint angle (for robotic applications), read-write.

When writing to this attribute, you pass a [x, y, z] vector and an appropriate set of euler angles or quaternion is calculated according to the rotation_mode.

When you read this attribute, the current pose matrix is converted into a [x, y, z] vector representing the joint angles.

The value and the meaning of the x, y, z depends on the ik_dof_x/ik_dof_y/ik_dof_z attributes:

- **1DoF joint X, Y or Z**: the corresponding x, y, or z value is used an a joint angle in radiant
- **2DoF joint X+Y or Z+Y**: treated as 2 successive 1DoF joints: first X or Z, then Y. The x or z value is used as a joint angle in radiant along the X or Z axis, followed by a rotation along the new Y axis of y radiants.
- **2DoF joint X+Z**: treated as a 2DoF joint with rotation axis on the X/Z plane. The x and z values are used as the coordinates of the rotation vector in the X/Z plane.
- **3DoF joint X+Y+Z**: treated as a revolute joint. The [x, y, z] vector represents the equivalent rotation vector to bring the joint from the rest pose to the new pose.

**Type** vector [x, y, z]

**Note:** The bone must be part of an IK chain if you want to set the ik_dof_x/ik_dof_y/ik_dof_z attributes via the UI, but this will interfere with this attribute since the IK solver will overwrite the pose. You can
stay in control of the armature if you create an IK constraint but do not finalize it (e.g. don’t set a target) the IK solver will not run but the IK panel will show up on the UI for each bone in the chain.

**Note:** [0, 0, 0] always corresponds to the rest pose.

**Note:** You must request the armature pose to update and wait for the next graphic frame to see the effect of setting this attribute (see `BL_ArmatureObject.update`).

**Note:** You can read the result of the calculation in rotation or euler_rotation attributes after setting this attribute.

### BL_ArmatureConstraint(EP_PyObjectPlus)

base class — `EXP_PyObjectPlus`

class `BL_ArmatureConstraint` (`EXP_PyObjectPlus`)

Proxy to Armature Constraint. Allows to change constraint on the fly. Obtained through `BL_ArmatureObject.constraints`.

**Note:** Not all armature constraints are supported in the GE.

**type**

Type of constraint, (read-only).

Use one of these constants.

| Type | integer, one of CONSTRAINT_TYPE_* constants |

**name**

Name of constraint constructed as `<bone_name>:<constraint_name>`. constraints list.

| Type | string |

This name is also the key subscript on `BL_ArmatureObject`.

**enforce**

fraction of constraint effect that is enforced. Between 0 and 1.

| Type | float |

**headtail**

Position of target between head and tail of the target bone: 0=head, 1=tail.

| Type | float |

**Note:** Only used if the target is a bone (i.e target object is an armature).

**lin_error**

runtime linear error (in Blender units) on constraint at the current frame.
This is a runtime value updated on each frame by the IK solver. Only available on IK constraint and iTaSC solver.

**Type** float

**rot_error**
Runtime rotation error (in radiant) on constraint at the current frame.

**Type** float.

This is a runtime value updated on each frame by the IK solver. Only available on IK constraint and iTaSC solver.

It is only set if the constraint has a rotation part, for example, a CopyPose+Rotation IK constraint.

**target**
Primary target object for the constraint. The position of this object in the GE will be used as target for the constraint.

**Type** **KX_GameObject**.

**subtarget**
Secondary target object for the constraint. The position of this object in the GE will be used as secondary target for the constraint.

**Type** **KX_GameObject**.

Currently this is only used for pole target on IK constraint.

**active**
True if the constraint is active.

**Type** boolean

---

**Note:** An inactive constraint does not update lin_error and rot_error.

---

**ik_weight**
Weight of the IK constraint between 0 and 1.

Only defined for IK constraint.

**Type** float

**ik_type**
Type of IK constraint, (read-only).

Use one of these constants.

**Type** integer.

**ik_flag**
Combination of IK constraint option flags, read-only.

Use one of these constants.

**Type** integer

**ik_dist**
Distance the constraint is trying to maintain with target, only used when ik_type=CONSTRAINT_IK_DISTANCE.

**Type** float
**ik_mode**

Use one of these constants.

Additional mode for IK constraint. Currently only used for Distance constraint:

Type integer

**BL_ArmatureObject(KX_GameObject)**

class BL_ArmatureObject (KX_GameObject)

An armature object.

**constraints**

The list of armature constraint defined on this armature. Elements of the list can be accessed by index or string. The key format for string access is `<bone_name>:<constraint_name>`.

Type list of BL_ArmatureConstraint

**channels**

The list of armature channels. Elements of the list can be accessed by index or name the bone.

Type list of BL_ArmatureChannel

**update()**

Ensures that the armature will be updated on next graphic frame.

This action is unnecessary if a KX_ArmatureActuator with mode run is active or if an action is playing. Use this function in other cases. It must be called on each frame to ensure that the armature is updated continously.

**draw()**

Draw lines that represent armature to view it in real time.

**BL_Shader(EXP_PyObjectPlus)**

class BL_Shader (EXP_PyObjectPlus)

BL_Shader is a class used to compile and use custom shaders scripts. It supports vertex, fragment and geometry shader scripts. The shader is compiled with a generated header for the three shaders scripts. This header set the #version directive, so the user must not define his own #version.

**enabled**

Set shader enabled to use.

Type boolean

**objectCallbacks**

The list of python callbacks executed when the shader is used to render an object. All the functions can expect as argument the object currently rendered.

```python
def callback(object):
    print("render object \%r \% object.name")
```

Type list of functions and/or methods
bindCallbacks
The list of python callbacks executed when the shader is begin used to render.

    Type  list of functions and/or methods

setUniformfv (name, fList)
Set a uniform with a list of float values

    Parameters
    • name (string) – the uniform name
    • fList (list[float]) – a list (2, 3 or 4 elements) of float values

delSource()
Clear the shader. Use this method before the source is changed with setSource.

getFragmentProg ()
Returns the fragment program.

    Returns  The fragment program.
    Return type  string

getVertexProg ()
Get the vertex program.

    Returns  The vertex program.
    Return type  string

isValid ()
Check if the shader is valid.

    Returns  True if the shader is valid
    Return type  boolean

setAttrib (enum)
Set attribute location. (The parameter is ignored a.t.m. and the value of “tangent” is always used.)

    Parameters  enum (integer) – attribute location value

setSampler (name, index)
Set uniform texture sample index.

    Parameters
    • name (string) – Uniform name
    • index (integer) – Texture sample index.

setSource (vertexProgram, fragmentProgram, apply)
Set the vertex and fragment programs

    Parameters
    • vertexProgram (string) – Vertex program
    • fragmentProgram (string) – Fragment program
    • apply (boolean) – Enable the shader.

setSourceList (sources, apply)
Set the vertex, fragment and geometry shader programs.

    Parameters
• **sources** *(dict)* – Dictionary of all programs. The keys *vertex*, *fragment* and *geometry* represent shader programs of the same name. *geometry* is an optional program. This dictionary can be similar to:

```python
sources = {
    "vertex" : vertexProgram,
    "fragment" : fragmentProgram,
    "geometry" : geometryProgram
}
```

• **apply** *(boolean)* – Enable the shader.

```python
setUniform1f(name, fx)
```

Set a uniform with 1 float value.

**Parameters**

- **name** *(string)* – the uniform name
- **fx** *(float)* – Uniform value

```python
setUniform1i(name, ix)
```

Set a uniform with an integer value.

**Parameters**

- **name** *(string)* – the uniform name
- **ix** *(integer)* – the uniform value

```python
setUniform2f(name, fx, fy)
```

Set a uniform with 2 float values.

**Parameters**

- **name** *(string)* – the uniform name
- **fx** *(float)* – first float value
- **fy** *(float)* – second float value

```python
setUniform2i(name, ix, iy)
```

Set a uniform with 2 integer values.

**Parameters**

- **name** *(string)* – the uniform name
- **ix** *(integer)* – first integer value
- **iy** *(integer)* – second integer value

```python
setUniform3f(name, fx, fy, fz)
```

Set a uniform with 3 float values.

**Parameters**

- **name** *(string)* – the uniform name
- **fx** *(float)* – first float value
- **fy** *(float)* – second float value
- **fz** *(float)* – third float value

```python
setUniform3i(name, ix, iy, iz)
```

Set a uniform with 3 integer values
Parameters

- `name (string)` – the uniform name
- `ix (integer)` – first integer value
- `iy (integer)` – second integer value
- `iz (integer)` – third integer value

`ssetUniform4f(name, fx, fy, fz, fw)`
Set a uniform with 4 float values.

Parameters

- `name (string)` – the uniform name
- `fx (float)` – first float value
- `fy (float)` – second float value
- `fz (float)` – third float value
- `fw (float)` – fourth float value

`ssetUniform4i(name, ix, iy, iz, iw)`
Set a uniform with 4 integer values

Parameters

- `name (string)` – the uniform name
- `ix (integer)` – first integer value
- `iy (integer)` – second integer value
- `iz (integer)` – third integer value
- `iw (integer)` – fourth integer value

`ssetUniformDef(name, type)`
Define a new uniform

Parameters

- `name (string)` – the uniform name
- `type (integer)` – uniform type, one of these constants

`ssetUniformMatrix3(name, mat, transpose)`
Set a uniform with a 3x3 matrix value

Parameters

- `name (string)` – the uniform name
- `mat (3x3 matrix)` – A 3x3 matrix `[[f, f, f], [f, f, f], [f, f, f]]`
- `transpose (boolean)` – set to True to transpose the matrix

`ssetUniformMatrix4(name, mat, transpose)`
Set a uniform with a 4x4 matrix value

Parameters

- `name (string)` – the uniform name
- `mat (4x4 matrix)` – A 4x4 matrix `[[f, f, f, f], [f, f, f, f], [f, f, f, f], [f, f, f, f]]`
- `transpose (boolean)` – set to True to transpose the matrix
setUniformiv(name, iList)
Set a uniform with a list of integer values

Parameters
- name (string) – the uniform name
- iList (list [integer]) – a list (2, 3 or 4 elements) of integer values

setUniformEyeF(name)
Set a uniform with a float value that reflects the eye being render in stereo mode: 0.0 for the left eye, 0.5 for the right eye. In non stereo mode, the value of the uniform is fixed to 0.0. The typical use of this uniform is in stereo mode to sample stereo textures containing the left and right eye images in a top-bottom order.

Parameters
- name (string) – the uniform name

validate()
Validate the shader object.

BL_Texture(EXP_Value)

base class — EXP_Value

class BL_Texture(EXP_Value)
A texture object that contains attributes of a material texture.

diffuseIntensity
Amount texture affects diffuse reflectivity.

Type float
diffuseFactor
Amount texture affects diffuse color.

Type float
alpha
Amount texture affects alpha.

Type float
specularIntensity
Amount texture affects specular reflectivity.

Type float
specularFactor
Amount texture affects specular color.

Type float
hardness
Amount texture affects hardness.

Type float
emit
Amount texture affects emission.

Type float
mirror
Amount texture affects mirror color.

Type float

8.1. Game Types (bge.types)
normal
   Amount texture affects normal values.
   Type float
parallaxBump
   Height of parallax occlusion mapping.
   Type float
parallaxStep
   Number of steps to achieve parallax effect.
   Type float
lodBias
   Amount bias on mipmapping.
   Type float
bindCode
   Texture bind code/Id/number.
   Type integer
renderer
   Texture renderer of this texture.
   Type KX_CubeMap, KX_PlanarMap or None
ior
   Index Of Refraction used to compute refraction.
   Type float (1.0 to 50.0)
refractionRatio
   Amount refraction mixed with reflection.
   Type float (0.0 to 1.0)
uvOffset
   Offset applied to texture UV coordinates (mainly translation on U and V axis).
   Type mathutils.Vector
uvSize
   Scale applied to texture UV coordinates.
   Type mathutils.Vector
uvRotation
   Rotation applied to texture UV coordinates.
   Type float (radians)

EXP_ListValue(EXP_PropValue)

base class — EXP_PropValue
class EXP_ListValue (EXP_PropValue)
   This is a list like object used in the game engine internally that behaves similar to a python list in most ways.
   As well as the normal index lookup (val = clist[i]), EXP_ListValue supports string lookups (val = scene.objects["Cube"])
Other operations such as `len(clist), list(clist), clist[0:10]` are also supported.

**append**(val)
Add an item to the list (like python's `append`)

**Warning:** Appending values to the list can cause crashes when the list is used internally by the game engine.

**count**(val)
Count the number of instances of a value in the list.

Returns number of instances

Return type integer

**index**(val)
Return the index of a value in the list.

Returns The index of the value in the list.

Return type integer

**reverse**()
Reverse the order of the list.

**get**(key, default=None)
Return the value matching key, or the default value if it's not found.

Returns The key value or a default.

**filter**(name, prop)
Return a list of items with name matching `name` regex and with a property matching `prop` regex. If `name` is empty every items are checked, if `prop` is empty no property check is proceeded.

Returns The list of matching items.

**from_id**(id)
This is a function especially for the game engine to return a value with a specific id.

Since object names are not always unique, the id of an object can be used to get an object from the EXP_ValueList.

Example:

```python
myObID = id(gameObject)
ob = scene.objects.from_id(myObID)
```

Where `myObID` is an int or long from the `id` function.

This has the advantage that you can store the id in places you could not store a GameObject.

**Warning:** The id is derived from a memory location and will be different each time the game engine starts.

**Warning:** The id can't be stored as an integer in game object properties, as those only have a limited range that the id may not be contained in. Instead an id can be stored as a string game property and converted back to an integer for use in from_id lookups.
**EXP_PropValue(EXP_Value)**

base class — *EXP_Value*

class **EXP_PropValue**(*EXP_Value*)

This class has no python functions

**EXP_PyObjectPlus**

class **EXP_PyObjectPlus**

*EXP_PyObjectPlus* base class of most other types in the Game Engine.

invalid

Test if the object has been freed by the game engine and is no longer valid.

Normally this is not a problem but when storing game engine data in the GameLogic module, KX_Scenes or other KX_GameObjects its possible to hold a reference to invalid data. Calling an attribute or method on an invalid object will raise a SystemError.

The invalid attribute allows testing for this case without exception handling.

Type boolean

**EXP_Value(EXP_PyObjectPlus)**

base class — *EXP_PyObjectPlus*

class **EXP_Value**(*EXP_PyObjectPlus*)

This class is a basis for other classes.

name

The name of this EXP_Value derived object (read-only).

Type string

**KX_2DFilter(BL_Shader)**

base class — *BL_Shader*

class **KX_2DFilter**(*BL_Shader*)

2D filter shader object. Can be altered with *BL_Shader*'s functions.

**Warning:** The vertex shader must not apply modelview and projection transformation. It should be similar to:

```python
void main()
{
    gl_Position = gl_Vertex;
}
```

mipmap

Request mipmap generation of the render *bgl_RenderedTexture* texture. Accessing mipmapping level is similar to:

```python
```
uniform sampler2D bgl_RenderedTexture;

void main()
{
    float level = 2.0; //.mipmap level
    gl_FragColor = textureLod(bgl_RenderedTexture, gl_TexCoord[0].st, level);
}

Type boolean

offScreen
The custom off screen the filter render to (read-only).
Type bge.types.KX_2DFilterOffScreen or None

setTexture(index, bindCode, samplerName="")
Set specified texture bind code bindCode in specified slot index. Any call to setTexture should be followed by a call to BL_Shader.setSampler with the same index if samplerName is not specified.

Parameters
- index(integer) – The texture slot.
- bindCode(integer) – The texture bind code/Id.
- samplerName(string) – The shader sampler name set to index if samplerName is passed in the function. (optional)

setCubeMap(index, bindCode, samplerName="")
Set specified cube map texture bind code bindCode in specified slot index. Any call to setCubeMap should be followed by a call to BL_Shader.setSampler with the same index if samplerName is not specified.

Parameters
- index(integer) – The texture slot.
- bindCode(integer) – The cube map texture bind code/Id.
- samplerName(string) – The shader sampler name set to index if samplerName is passed in the function. (optional)

addOffScreen(slots, depth=False, width=-1, height=-1, hdr=bge.render.HDR_NONE, mipmap=False)
Register a custom off screen to render the filter to.

Parameters
- slots(integer) – The number of color texture attached to the off screen, between 0 and 8 excluded.
- depth(boolean) – True of the off screen use a depth texture (optional).
- width(integer) – The off screen width, -1 if it can be resized dynamically when the viewport dimensions changed (optional).
- height(integer) – The off screen height, -1 if it can be resized dynamically when the viewport dimensions changed (optional).
- hdr (one of these constants) – The image quality HDR of the color textures (optional).
• **mipmap** *(boolean)* – True if the color texture generate mipmap at the end of the filter rendering (optional).

**Note:** If the off screen is created using a dynamic size *(width and height to -1)* its bind codes will be unavailable before the next render of the filter and the it can change when the viewport is resized.

```python
removeOffScreen()
```
Unregister the custom off screen the filter render to.

### KX_2DFilterManager(EXP_PyObjectPlus)

**base class — EXP_PyObjectPlus**

**class KX_2DFilterManager(EXP_PyObjectPlus)**

2D filter manager used to add, remove and find filters in a scene.

**addFilter** *(index, type, fragmentProgram)*

Add a filter to the pass index `index`, type `type` and fragment program if custom filter.

**Parameters**

- **index** *(integer)* – The filter pass index.
- **type** *(integer)* – The filter type, one of:
  - `bge.logic.RAS_2DFILTER_BLUR`
  - `bge.logic.RAS_2DFILTER_DILATION`
  - `bge.logic.RAS_2DFILTER_EROSION`
  - `bge.logic.RAS_2DFILTER_SHARPEN`
  - `bge.logic.RAS_2DFILTER_LAPLACIAN`
  - `bge.logic.RAS_2DFILTER_PREWITT`
  - `bge.logic.RAS_2DFILTER_SOBEL`
  - `bge.logic.RAS_2DFILTER_GRAYSCALE`
  - `bge.logic.RAS_2DFILTER_SEPIA`
  - `bge.logic.RAS_2DFILTER_CUSTOMFILTER`
- **fragmentProgram** *(string)* – The filter shader fragment program. Used only if type is `bge.logic.RAS_2DFILTER_CUSTOMFILTER`, if empty or not specified the filter is created without shader, waiting call to `BL_Shader.setSourceList`. (optional)

**Returns** The 2D Filter.

**Return type** KX_2DFilter

**removeFilter** *(index)*

Remove filter to the pass index `index`.

**Parameters** `index` *(integer)* – The filter pass index.

**getFilter** *(index)*

Return filter to the pass index `index`.
Warning If the 2D Filter is added with a SCA_2DFilterActuator, the filter will be available only after the 2D Filter program is linked. The python script to get the filter has to be executed one frame later. A delay sensor can be used.

Parameters index (integer) – The filter pass index.

Returns The filter in the specified pass index or None.

Return type KX_2DFilter or None

KX_2DFilterOffScreen(Exp_Value)

base class — Exp_Value
class KX_2DFilterOffScreen(Exp_Value)
  2D filter custom off screen.

width
  The off screen width, -1 if the off screen can be resized dynamically (read-only).

  Type integer

height
  The off screen height, -1 if the off screen can be resized dynamically (read-only).

  Type integer

colorBindCodes
  The bind code of the color textures attached to the off screen (read-only).

Warning: If the off screen can be resized dynamically (width of height equal to -1), the bind codes may change.

  Type list of 8 integers

depthBindCode
  The bind code of the depth texture attached to the off screen (read-only).

Warning: If the off screen can be resized dynamically (width of height equal to -1), the bind code may change.

  Type integer

KX_ArmatureSensor(SCA_I Sensor)

base class — SCA_I Sensor
class KX_ArmatureSensor(SCA_I Sensor)
  Armature sensor detect conditions on armatures.

type
  The type of measurement that the sensor make when it is active.

  Can be one of these constants

  Type integer.
constraint
The constraint object this sensor is watching.

Type BL_ArmatureConstraint

value
The threshold used in the comparison with the constraint error. The linear error is only updated on CopyPose/Distance IK constraint with iTaSC solver. The rotation error is only updated on CopyPose+rotation IK constraint with iTaSC solver. The linear error on CopyPose is always >= 0: it is the norm of the distance between the target and the bone. The rotation error on CopyPose is always >= 0: it is the norm of the equivalent rotation vector between the bone and the target orientations. The linear error on Distance can be positive if the distance between the bone and the target is greater than the desired distance, and negative if the distance is smaller.

Type float

KX_BatchGroup(EXP_Value)

base class — EXP_Value

class KX_BatchGroup (EXP_Value)
The batch group is class containing a mesh resulting of the merging of meshes used by objects. The meshes are merged with the transformation of the objects using it. An instance of this class is not owned by one object but shared between objects. In consideration an instance of KX_BatchGroup have to instanced with as argument a list of at least one object containing the meshes to merge. This can be done in a way similar to:

```python
import bge

scene = bge.logic.getCurrentScene()

batchGroup = types.KX_BatchGroup([scene.objects["Cube"], scene.objects["Plane"]])
```

objects
The list of the objects merged. (read only)

Type EXP_ListValue of KX_GameObject

merge (objects)
Merge meshes using the transformation of the objects using them.

Parameters
objects – The objects to merge.

split (objects)
Split the meshes of the objects using them and restore their original meshes.

Parameters
objects – The objects to unmerge.

destruct()
Destruct the batch group and restore all the objects to their original meshes.

KX_BlenderMaterial(EXP_PyObjectPlus)

base class — EXP_PyObjectPlus

class KX_BlenderMaterial (EXP_PyObjectPlus)
This is the interface to materials in the game engine.

Materials define the render state to be applied to mesh objects.
The example below shows a simple GLSL shader setup allowing to dynamically mix two texture channels in a material. All materials of the object executing this script should have two textures using separate UV maps in the two first texture channels.

The code works for both Multitexture and GLSL rendering modes.

```python
from bge import logic

vertex_shader = ""

void main(void)
{
    // simple projection of the vertex position to view space
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
    // coordinate of the 1st texture channel
    gl_TexCoord[0] = gl_MultiTexCoord0;
    // coordinate of the 2nd texture channel
    gl_TexCoord[1] = gl_MultiTexCoord1;
}
""

fragment_shader = ""

uniform sampler2D texture_0;
uniform sampler2D texture_1;
uniform float factor;

void main(void)
{
    vec4 color_0 = texture2D(texture_0, gl_TexCoord[0].st);
    vec4 color_1 = texture2D(texture_1, gl_TexCoord[1].st);
    gl_FragColor = mix(color_0, color_1, factor);
}
""

object = logic.getCurrentController().owner

for mesh in object.meshes:
    for material in mesh.materials:
        shader = material.getShader()
        if shader is not None:
            if not shader.isValid():
                shader.setSource(vertex_shader, fragment_shader, True)

                # get the first texture channel of the material
                shader.setSampler('texture_0', 0)
                # get the second texture channel of the material
                shader.setSampler('texture_1', 1)
                # pass another uniform to the shader
                shader.setUniform1f('factor', 0.3)
```

**shader**

The material’s shader.

Type *BL_Shader*

**blending**

Ints used for pixel blending, (src, dst), matching the `setBlending` method.

Type (integer, integer)
getShader()
Returns the material’s shader.

Returns the material’s shader
Return type BL_Shader

getTextureBindcode (textureslot)
Returns the material’s texture OpenGL bind code/id/number/name.

Deprecated since version use: bge.types.BL_Texture.bindCode()

Parameters textureslot (integer) – Specifies the texture slot number

Returns the material’s texture OpenGL bind code/id/number/name
Return type integer

alpha
The material’s alpha transparency.

Type float between 0.0 and 1.0 inclusive

hardness
How hard (sharp) the material’s specular reflection is.

Type integer between 1 and 511 inclusive

emit
Amount of light to emit.

Type float between 0.0 and 2.0 inclusive

ambient
Amount of ambient light on the material.

Type float between 0.0 and 1.0 inclusive

specularAlpha
Alpha transparency for specular areas.

Type float between 0.0 and 1.0 inclusive (alpha must be < 1.0)

specularIntensity
How intense (bright) the material’s specular reflection is.

Type float between 0.0 and 1.0 inclusive

diffuseIntensity
The material’s amount of diffuse reflection.

Type float between 0.0 and 1.0 inclusive

specularColor
The material’s specular color.

Type mathutils.Color

diffuseColor
The material’s diffuse color.

Type mathutils.Color

textures
List of all material’s textures.

Type List of BL_Texture (read only)
**setBlending** (*src, dest*)  
Set the pixel color arithmetic functions.

**Parameters**

- **src (int)** – Specifies how the red, green, blue, and alpha source blending factors are computed, one of...
  - GL_ZERO
  - GL_ONE
  - GL_SRC_COLOR
  - GL_ONE_MINUS_SRC_COLOR
  - GL_DST_COLOR
  - GL_ONE_MINUS_DST_COLOR
  - GL_SRC_ALPHA
  - GL_ONE_MINUS_SRC_ALPHA
  - GL_DST_ALPHA
  - GL_ONE_MINUS_DST_ALPHA
  - GL_SRC_ALPHA_SATURATE

- **dest (int)** – Specifies how the red, green, blue, and alpha destination blending factors are computed, one of...
  - GL_ZERO
  - GL_ONE
  - GL_SRC_COLOR
  - GL_ONE_MINUS_SRC_COLOR
  - GL_DST_COLOR
  - GL_ONE_MINUS_DST_COLOR
  - GL_SRC_ALPHA
  - GL_ONE_MINUS_SRC_ALPHA
  - GL_DST_ALPHA
  - GL_ONE_MINUS_DST_ALPHA
  - GL_SRC_ALPHA_SATURATE

**KX_BoundingBox(EXP_PyObjectPlus)**

**base class** — **EXP_PyObjectPlus**

**class KX_BoundingBox(EXP_PyObjectPlus)**

A bounding volume box of a game object. Used to get and alterate the volume box or AABB.

```python
from bge import logic
from mathutils import Vector
owner = logic.getCurrentController().owner
```

(continues on next page)
box = owner.cullingBox

# Disable auto update to allow modify the box.
box.autoUpdate = False

print(box)
# Increase the height of the box of 1.
box.max = box.max + Vector((0, 0, 1))

print(box)

min
The minimal point in x, y and z axis of the bounding box.
   Type mathutils.Vector

max
The maximal point in x, y and z axis of the bounding box.
   Type mathutils.Vector

center
   The center of the bounding box. (read only)
   Type mathutils.Vector

radius
   The radius of the bounding box. (read only)
   Type float

autoUpdate
   Allow to update the bounding box if the mesh is modified.
   Type boolean

KX_Camera(KX_GameObject)

base class — KX_GameObject
class KX_Camera(KX_GameObject)
   A Camera object.

   INSIDE
      See sphereInsideFrustum and boxInsideFrustum

   INTERSECT
      See sphereInsideFrustum and boxInsideFrustum

   OUTSIDE
      See sphereInsideFrustum and boxInsideFrustum

   lens
      The camera’s lens value.
      Type float

   lodDistanceFactor
      The factor to multiply distance to camera to adjust levels of detail. A float < 1.0f will make the distance to
      camera used to compute levels of detail decrease.
Type float

**fov**
The camera’s field of view value.
Type float

**ortho_scale**
The camera’s view scale when in orthographic mode.
Type float

**near**
The camera’s near clip distance.
Type float

**far**
The camera’s far clip distance.
Type float

**shift_x**
The camera’s horizontal shift.
Type float

**shift_y**
The camera’s vertical shift.
Type float

**perspective**
True if this camera has a perspective transform, False for an orthographic projection.
Type boolean

**frustum_culling**
True if this camera is frustum culling.
Type boolean

**activityCulling**
True if this camera is used to compute object distance for object activity culling.
Type boolean

**projection_matrix**
This camera’s 4x4 projection matrix.

**Note:** This is the identity matrix prior to rendering the first frame (any Python done on frame 1).

Type 4x4 Matrix [[float]]

**modelview_matrix**
This camera’s 4x4 model view matrix. (read-only).
Type 4x4 Matrix [[float]]

**Note:** This matrix is regenerated every frame from the camera’s position and orientation. Also, this is the identity matrix prior to rendering the first frame (any Python done on frame 1).
**camera_to_world**
This camera’s camera to world transform. (read-only).

- **Type**: 4x4 Matrix [[float]]

**Note**: This matrix is regenerated every frame from the camera’s position and orientation.

**world_to_camera**
This camera’s world to camera transform. (read-only).

- **Type**: 4x4 Matrix [[float]]

**Note**: Regenerated every frame from the camera’s position and orientation.

**Note**: This is camera_to_world inverted.

**useViewport**
True when the camera is used as a viewport, set True to enable a viewport for this camera.

- **Type**: boolean

**sphereInsideFrustum** (centre, radius)
Tests the given sphere against the view frustum.

- **Parameters**
  - centre (list [x, y, z]) – The centre of the sphere (in world coordinates.)
  - radius (float) – the radius of the sphere

- **Returns**: INSIDE, OUTSIDE or INTERSECT

- **Return type**: integer

**Note**: When the camera is first initialized the result will be invalid because the projection matrix has not been set.

```python
from bge import logic
cont = logic.getCurrentController()
cam = cont.owner

# A sphere of radius 4.0 located at [x, y, z] = [1.0, 1.0, 1.0]
if cam.sphereInsideFrustum([1.0, 1.0, 1.0], 4) != cam.OUTSIDE):
    # Sphere is inside frustum !
    # Do something useful !
else:
    # Sphere is outside frustum
```

**boxInsideFrustum** (box)
Tests the given box against the view frustum.

- **Parameters**
  - box (list of lists) – Eight (8) corner points of the box (in world coordinates.)

- **Returns**: INSIDE, OUTSIDE or INTERSECT
Note: When the camera is first initialized the result will be invalid because the projection matrix has not been set.

```python
from bge import logic
cont = logic.getCurrentController()
cam = cont.owner

# Box to test...
box = []
box.append([-1.0, -1.0, -1.0])
box.append([-1.0, -1.0, 1.0])
box.append([-1.0, 1.0, -1.0])
box.append([-1.0, 1.0, 1.0])
box.append([ 1.0, -1.0, -1.0])
box.append([ 1.0, -1.0, 1.0])
box.append([ 1.0, 1.0, -1.0])
box.append([ 1.0, 1.0, 1.0])

if (cam.boxInsideFrustum(box) != cam.OUTSIDE):
    # Box is inside/intersects frustum!
    # Do something useful!
else:
    # Box is outside the frustum!
```

pointInsideFrustum(point)
Tests the given point against the view frustum.

**Parameters**
- **point** (*3D Vector*) – The point to test (in world coordinates.)

**Returns**
- True if the given point is inside this camera’s viewing frustum.

**Return type**
- boolean

---

Note: When the camera is first initialized the result will be invalid because the projection matrix has not been set.

```python
from bge import logic
cont = logic.getCurrentController()
cam = cont.owner

# Test point [0.0, 0.0, 0.0]
if (cam.pointInsideFrustum([0.0, 0.0, 0.0])):
    # Point is inside frustum!
    # Do something useful!
else:
    # Box is outside the frustum!
```

getchCameraToWorld()
Returns the camera-to-world transform.

**Returns**
- the camera-to-world transform matrix.

**Return type**
- matrix (4x4 list)

getWorldToCamera()
Returns the world-to-camera transform.

---

8.1. Game Types (bge.types)
This returns the inverse matrix of getCameraToWorld().

**Returns**  the world-to-camera transform matrix.

**Return type** matrix (4x4 list)

---

**setOnTop()**
Set this camera’s viewport on top of all other viewports.

---

**setViewport** (left, bottom, right, top)
Sets the region of this viewport on the screen in pixels.

Use `bge.render.getWindowHeight` and `bge.render.getWindowWidth` to calculate values relative to the entire display.

**Parameters**
- **left** (integer)  – left pixel coordinate of this viewport
- **bottom** (integer)  – bottom pixel coordinate of this viewport
- **right** (integer)  – right pixel coordinate of this viewport
- **top** (integer)  – top pixel coordinate of this viewport

---

**getScreenPosition** (object)
Gets the position of an object projected on screen space.

```python
# For an object in the middle of the screen, coord = [0.5, 0.5]
coord = camera.getScreenPosition(object)
```

**Parameters**  object (KX_GameObject or 3D Vector)  – object name or list [x, y, z]

**Returns**  the object’s position in screen coordinates.

**Return type** list [x, y]

---

**getScreenVect** (x, y)
Gets the vector from the camera position in the screen coordinate direction.

**Parameters**
- **x** (float)  – X Axis
- **y** (float)  – Y Axis

**Return type** 3D Vector

**Returns**  The vector from screen coordinate.

```python
# Gets the vector of the camera front direction:
m_vect = camera.getScreenVect(0.5, 0.5)
```

---

**getScreenRay** (x, y, dist=inf, property=None)
Look towards a screen coordinate (x, y) and find first object hit within dist that matches prop. The ray is similar to KX_GameObject->rayCastTo.

**Parameters**
- **x** (float)  – X Axis
- **y** (float)  – Y Axis
- **dist** (float)  – max distance to look (can be negative => look behind); 0 or omitted => detect up to other

• **property** *(string)* – property name that object must have; can be omitted => detect any object

**Return type** *KX_GameObject*

**Returns** the first object hit or None if no object or object does not match prop

```python
# Gets an object with a property "wall" in front of the camera within a distance of 100:
target = camera.getScreenRay(0.5, 0.5, 100, "wall")
```

---

**KX_CameraActuator** *(SCA_IActuator)*

base class — *SCA_IActuator*

class **KX_CameraActuator**( *SCA_IActuator*)

Applies changes to a camera.

- **damping**
  strength of of the camera following movement.
  
  **Type** float

- **axis**
  The camera axis (0, 1, 2) for positive XYZ, (3, 4, 5) for negative XYZ.
  
  **Type** int

- **min**
  minimum distance to the target object maintained by the actuator.
  
  **Type** float

- **max**
  maximum distance to stay from the target object.
  
  **Type** float

- **height**
  height to stay above the target object.
  
  **Type** float

- **object**
  the object this actuator tracks.
  
  **Type** *KX_GameObject* or None

---

**KX_CharacterWrapper** *(EXP_PyObjectPlus)*

base class — *EXP_PyObjectPlus*

class **KX_CharacterWrapper**( *EXP_PyObjectPlus*)

A wrapper to expose character physics options.

- **onGround**
  Whether or not the character is on the ground. (read-only)
  
  **Type** boolean

- **gravity**
  The gravity value used for the character.
Type float

**fallSpeed**
The character falling speed.

Type float

**maxJumps**
The maximum number of jumps a character can perform before having to touch the ground. By default this is set to 1. 2 allows for a double jump, etc.

Type int in [0, 255], default 1

**jumpCount**
The current jump count. This can be used to have different logic for a single jump versus a double jump. For example, a different animation for the second jump.

Type int

**jumpSpeed**
The character jumping speed.

Type float

**maxSlope**
The maximum slope which the character can climb.

Type float

**walkDirection**
The speed and direction the character is traveling in using world coordinates. This should be used instead of applyMovement() to properly move the character.

Type Vector((x, y, z))

jump()
The character jumps based on it’s jump speed.

setVelocity(velocity, time, local=False)
Sets the character’s linear velocity for a given period.

This method sets character’s velocity through it’s center of mass during a period.

**Parameters**
- **velocity** (3D Vector) – Linear velocity vector.
- **time** (float) – Period while applying linear velocity.
- **local** (boolean) –
  - False: you get the “global” velocity ie: relative to world orientation.
  - True: you get the “local” velocity ie: relative to object orientation.

reset()
Resets the character velocity and walk direction.

**KX_CollisionContactPoint(EXP_Value)**

base class — **EXP_Value**

class KX_CollisionContactPoint (EXP_Value)
A collision contact point passed to the collision callbacks.
import bge

def oncollision(object, point, normal, points):
    print("Hit by", object)
    for point in points:
        print(point.localPointA)
        print(point.localPointB)
        print(point.worldPoint)
        print(point.normal)
        print(point.combinedFriction)
        print(point.combinedRestitution)
        print(point.appliedImpulse)

cont = bge.logic.getCurrentController()
own = cont.owner
own.collisionCallbacks = [oncollision]

localPointA
The contact point in the owner object space.
    Type mathutils.Vector

localPointB
The contact point in the collider object space.
    Type mathutils.Vector

worldPoint
The contact point in world space.
    Type mathutils.Vector

normal
The contact normal in owner object space.
    Type mathutils.Vector

combinedFriction
The combined friction of the owner and collider object.
    Type float

combinedRollingFriction
The combined rolling friction of the owner and collider object.
    Type float

combinedRestitution
The combined restitution of the owner and collider object.
    Type float

appliedImpulse
The applied impulse to the owner object.
    Type float

KX_CollisionSensor(SCA_ISensor)

base class — SCA_ISensor

8.1. Game Types (bge.types)  163
class KX_CollisionSensor (SCA_ISensor)
Collision sensor detects collisions between objects.

    propName
    The property or material to collide with.
    Type string

    useMaterial
    Determines if the sensor is looking for a property or material. KX_True = Find material; KX_False = Find property.
    Type boolean

    usePulseCollision
    When enabled, changes to the set of colliding objects generate a pulse.
    Type boolean

hitObject
The last collided object. (read-only).
    Type KX_GameObject or None

hitObjectList
A list of colliding objects. (read-only).
    Type EXP_ListValue of KX_GameObject

hitMaterial
The material of the object in the face hit by the ray. (read-only).
    Type string

KX_ConstraintActuator(SCA_IActuator)
base class — SCA_IActuator
class KX_ConstraintActuator (SCA_IActuator)
A constraint actuator limits the position, rotation, distance or orientation of an object.

    damp
    Time constant of the constraint expressed in frame (not use by Force field constraint).
    Type integer

    rotDamp
    Time constant for the rotation expressed in frame (only for the distance constraint), 0 = use damp for rotation as well.
    Type integer

    direction
    The reference direction in world coordinate for the orientation constraint.
    Type 3-tuple of float: (x, y, z)

    option
    Binary combination of these constants
    Type integer
time
activation time of the actuator. The actuator disables itself after this many frame. If set to 0, the actuator is not limited in time.

Type integer

propName
the name of the property or material for the ray detection of the distance constraint.

Type string

min
The lower bound of the constraint. For the rotation and orientation constraint, it represents radiant.

Type float

distance
the target distance of the distance constraint.

Type float

max
the upper bound of the constraint. For rotation and orientation constraints, it represents radiant.

Type float

rayLength
the length of the ray of the distance constraint.

Type float

limit
type of constraint. Use one of the these constants

Type integer.

KX_ConstraintWrapper(EXP_PyObjectPlus)

base class — EXP_PyObjectPlus

class KX_ConstraintWrapper (EXP_PyObjectPlus)

KX_ConstraintWrapper

getConstraintId(val)
Returns the contraint ID

Returns the constraint ID

Return type integer

setParameter(axis, value0, value1)
Set the contraint limits

Parameters axis (integer) –

Note:

• Lowerlimit == Upperlimit -&gt; axis is locked
• Lowerlimit &gt; Upperlimit -&gt; axis is free
• Lowerlimit &lt; Upperlimit -&gt; axis it limited in that range
For PHY_LINEHINGE_CONSTRAINT = 2 or PHY_ANGULAR_CONSTRAINT = 3:
axis = 3 is a constraint limit, with low/high limit value
• 3: X axis angle

Parameters
• (min) \((value_0)\) – Set the minimum limit of the axis
• (max) \((value_1)\) – Set the maximum limit of the axis

For PHY_CONE_TWIST_CONSTRAINT = 4:
axis = 3..5 are constraint limits, high limit values
• 3: X axis angle
• 4: Y axis angle
• 5: Z axis angle

Parameters
• (min) \((value_0)\) – Set the minimum limit of the axis
• (max) \((value_1)\) – Set the maximum limit of the axis

For PHY_GENERIC_6DOF_CONSTRAINT = 12:
axis = 0..2 are constraint limits, with low/high limit value
• 0: X axis position
• 1: Y axis position
• 2: Z axis position

axis = 3..5 are relative constraint (Euler) angles in radians
• 3: X axis angle
• 4: Y axis angle
• 5: Z axis angle

Parameters
• (min) \((value_0)\) – Set the minimum limit of the axis
• (max) \((value_1)\) – Set the maximum limit of the axis

axis = 6..8 are translational motors, with value0=target velocity, value1 = max motor force
• 6: X axis position
• 7: Y axis position
• 8: Z axis position

axis = 9..11 are rotational motors, with value0=target velocity, value1 = max motor force
• 9: X axis angle
• 10: Y axis angle
• 11: Z axis angle

Parameters
  • (speed) (value0) – Set the linear velocity of the axis
  • (force) (value1) – Set the maximum force limit of the axis

axis = 12..14 are for linear springs on each of the position of freedom
  • 12: X axis position
  • 13: Y axis position
  • 14: Z axis position

axis = 15..17 are for angular springs on each of the angle of freedom in radians
  • 15: X axis angle
  • 16: Y axis angle
  • 17: Z axis angle

Parameters
  • (stiffness) (value0) – Set the stiffness of the spring
  • (damping) (value1) – Tendency of the spring to return to it’s original position

getParam(axis)
Get the contraint position or euler angle of a generic 6DOF constraint

Parameters  axis (integer) –

axis = 0..2 are linear constraint values
  • 0: X axis position
  • 1: Y axis position
  • 2: Z axis position

Returns  position
Return type  float

axis = 3..5 are relative constraint (Euler) angles in radians
  • 3: X axis angle
  • 4: Y axis angle
  • 5: Z axis angle

Returns  angle
Return type  float

constraint_id
Returns the contraint ID (read only)

Type  integer

8.1. Game Types (bge.types)
constraint_type
Returns the constraint type (read only)

Type  integer
  •  POINTTOPOINT_CONSTRAINT
  •  LINEHINGE_CONSTRAN
  •  ANGULAR_CONSTRAINT
  •  CONETWIST_CONSTRAINT
  •  VEHICLE_CONSTRAINT
  •  GENERIC_6DOF_CONSTRAINT

breakingThreshold
The impulse threshold breaking the constraint, if the constraint is broken enabled is set to False.

Type  float greater or equal to 0

enabled
The status of the constraint. Set to True to restore a constraint after breaking.

Type  boolean

KX_CubeMap(KX_TextureRenderer)

base class  —  KX_TextureRenderer
class KX_CubeMap(KX_TextureRenderer)
Python API for realtime cube map textures.

```python
import bge

scene = bge.logic.getCurrentScene()
# The object using a realtime cube map in its material.
obj = scene.objects['Suzanne']
mat = obj.meshes[0].materials[0]
# Obtain the realtime cube map from the material texture.
cubemap = mat.textures[0].renderer

# Set the render position to the "Cube" object position.
cubemap.viewpointObject = scene.objects['Cube']

# Change the culling clip start and clip end.
cubemap.clipStart = 5.0
cubemap.clipEnd = 20.0

# Disable render on third layer.
cubemap.ignoreLayers = (1 << 2)

# Disable per frame update.
cubemap.autoUpdate = False
# Ask to update for this frame only.
cubemap.update()
```
KX_FontObject(KX_GameObject)

base class — KX_GameObject

class KX_FontObject (KX_GameObject)
A Font object.

```python
# Display a message about the exit key using a Font object.
import bge
co = bge.logic.getCurrentController()
font = co.owner
exit_key = bge.events.EventToString(bge.logic.getExitKey())
if exit_key.endswith("KEY"):
    exit_key = exit_key[:-3]
font.text = "Press key '%%s' to quit the game." % exit_key
```

text
The text displayed by this Font object.

Type string

resolution
The resolution of the font police.

Warning: High resolution can use a lot of memory and may crash.

Type float (0.1 to 50.0)

size
The size (scale factor) of the font object, scaled from font object origin (affects text resolution).

Warning: High size can use a lot of memory and may crash.

Type float (0.0001 to 40.0)

dimensions
The size (width and height) of the current text in Blender Units.

Type mathutils.Vector

KX_GameActuator(SCA_IActuator)

base class — SCA_IActuator

class KX_GameActuator (SCA_IActuator)
The game actuator loads a new .blend file, restarts the current .blend file or quits the game.

fileName
the new .blend file to load.

Type string
**mode**

The mode of this actuator. Can be one of these constants

Type: Int

---

**KX_GameObject(SCA_IObject)**

base class — SCA_IObject

class KX_GameObject (SCA_IObject)

All game objects are derived from this class.

Properties assigned to game objects are accessible as attributes of this class.

---

Note: Calling ANY method or attribute on an object that has been removed from a scene will raise a SystemError, if an object may have been removed since last accessing it use the invalid attribute to check.

---

KX_GameObject can be subclassed to extend functionality. For example:

```python
import bge

class CustomGameObject(bge.types.KX_GameObject):
    RATE = 0.05

    def __init__(self, old_owner):
        # "old_owner" can just be ignored. At this point, "self" is
        # already the object in the scene, and "old_owner" has been
        # destroyed.

        # New attributes can be defined - but we could also use a game
        # property, like "self['rate']".
        self.rate = CustomGameObject.RATE

    def update(self):
        self.worldPosition.z += self.rate

        # switch direction
        if self.worldPosition.z > 1.0:
            self.rate = -CustomGameObject.RATE
        elif self.worldPosition.z < 0.0:
            self.rate = CustomGameObject.RATE

    # Called first
    def mutate(cont):
        old_object = cont.owner
        mutated_object = CustomGameObject(cont.owner)

        # After calling the constructor above, references to the old object
        # should not be used.
        assert(old_object is not mutated_object)
        assert(old_object.invalid)
        assert(mutated_object is cont.owner)

    # Called later - note we are now working with the mutated object.
    def update(cont):
        cont.owner.update()
```

---
When subclassing objects other than empties and meshes, the specific type should be used - e.g. inherit from `BL_ArmatureObject` when the object to mutate is an armature.

name

The object’s name.

Type: string

mass

The object’s mass

Type: float

Note: The object must have a physics controller for the mass to be applied, otherwise the mass value will be returned as 0.0.

isSuspendDynamics

The object’s dynamic state (read-only).

Type: boolean

See also:

- `suspendDynamics()` and `restoreDynamics()` allow you to change the state.

linearDamping

The object’s linear damping, also known as translational damping. Can be set simultaneously with angular damping using the `setDamping()` method.

Type: float between 0 and 1 inclusive.

Note: The object must have a physics controller for the linear damping to be applied, otherwise the value will be returned as 0.0.

angularDamping

The object’s angular damping, also known as rotationation damping. Can be set simultaneously with linear damping using the `setDamping()` method.

Type: float between 0 and 1 inclusive.

Note: The object must have a physics controller for the angular damping to be applied, otherwise the value will be returned as 0.0.

linVelocityMin

Enforces the object keeps moving at a minimum velocity.

Type: float

Note: Applies to dynamic and rigid body objects only.

Note: A value of 0.0 disables this option.
Note: While objects are stationary the minimum velocity will not be applied.

**linVelocityMax**
Clamp the maximum linear velocity to prevent objects moving beyond a set speed.

*Type* float

Note: Applies to dynamic and rigid body objects only.

Note: A value of 0.0 disables this option (rather than setting it stationary).

**angularVelocityMin**
Enforces the object keeps rotating at a minimum velocity. A value of 0.0 disables this.

*Type* non-negative float

Note: Applies to dynamic and rigid body objects only. While objects are stationary the minimum velocity will not be applied.

**angularVelocityMax**
Clamp the maximum angular velocity to prevent objects rotating beyond a set speed. A value of 0.0 disables clamping; it does not stop rotation.

*Type* non-negative float

Note: Applies to dynamic and rigid body objects only.

**localInertia**
the object’s inertia vector in local coordinates. Read only.

*Type* Vector((ix, iy, iz))

**parent**
The object’s parent object. (read-only).

*Type* KX_GameObject or None

**groupMembers**
Returns the list of group members if the object is a group object (dupli group instance), otherwise None is returned.

*Type* EXP_ListValue of KX_GameObject or None

**groupObject**
Returns the group object (dupli group instance) that the object belongs to or None if the object is not part of a group.

*Type* KX_GameObject or None

**collisionGroup**
The object’s collision group.

*Type* bitfield
collisionMask
The object’s collision mask.

Type bitfield

collisionCallbacks
A list of functions to be called when a collision occurs.

Type list of functions and/or methods

Callbacks should either accept one argument (object), or four arguments (object, point, normal, points). For simplicity, per colliding object the first collision point is reported in second and third argument.

```python
# Function form
def callback_four(object, point, normal, points):
    print('Hit by %r with %i contacts points' % (object.name, len(points)))
def callback_three(object, point, normal):
    print('Hit by %r at %s with normal %s' % (object.name, point, normal))
def callback_one(object):
    print('Hit by %r' % object.name)

def register_callback(controller):
    controller.owner.collisionCallbacks.append(callback_four)
    controller.owner.collisionCallbacks.append(callback_three)
    controller.owner.collisionCallbacks.append(callback_one)

# Method form
class YourGameEntity(bge.types.KX_GameObject):
    def __init__(self, old_owner):
        self.collisionCallbacks.append(self.on_collision_four)
        self.collisionCallbacks.append(self.on_collision_three)
        self.collisionCallbacks.append(self.on_collision_one)

    def on_collision_four(self, object, point, normal, points):
        print('Hit by %r with %i contacts points' % (object.name, len(points)))
    def on_collision_three(self, object, point, normal):
        print('Hit by %r at %s with normal %s' % (object.name, point, normal))
    def on_collision_one(self, object):
        print('Hit by %r' % object.name)
```

Note: For backward compatibility, a callback with variable number of arguments (using *args) will be passed only the object argument. Only when there is more than one fixed argument (not counting self for methods) will the four-argument form be used.

scene
The object’s scene. (read-only).

Type KX_Scene or None

visible
visibility flag.

Type boolean
Note: Game logic will still run for invisible objects.

layer
The layer mask used for shadow and real-time cube map render.
Type bitfield
cullingBox
The object’s bounding volume box used for culling.
Type KX_BoundingBox
culled
Returns True if the object is culled, else False.

Warning: This variable returns an invalid value if it is called outside the scene’s callbacks KX_Scene.pre_draw and KX_Scene.post_draw.

Type boolean (read only)

color
The object color of the object. [r, g, b, a]
Type mathutils.Vector
occlusion
occlusion capability flag.
Type boolean

physicsCulling
True if the object suspends its physics depending on its nearest distance to any camera.
Type boolean

logicCulling
True if the object suspends its logic and animation depending on its nearest distance to any camera.
Type boolean

physicsCullingRadius
Suspend object’s physics if this radius is smaller than its nearest distance to any camera and physicsCulling set to True.
Type float

logicCullingRadius
Suspend object’s logic and animation if this radius is smaller than its nearest distance to any camera and logicCulling set to True.
Type float

position
The object’s position. [x, y, z] On write: local position, on read: world position
Deprecated since version use: localPosition and worldPosition.
Type mathutils.Vector
orientation
The object’s orientation. 3x3 Matrix. You can also write a Quaternion or Euler vector. On write: local orientation, on read: world orientation

Deprecated since version use: localOrientation and worldOrientation.
Type mathutils.Matrix

scaling
The object’s scaling factor. [sx, sy, sz] On write: local scaling, on read: world scaling

Deprecated since version use: localScale and worldScale.
Type mathutils.Vector

localOrientation
The object’s local orientation. 3x3 Matrix. You can also write a Quaternion or Euler vector.
Type mathutils.Matrix

worldOrientation
The object’s world orientation. 3x3 Matrix.
Type mathutils.Matrix

localScale
The object’s local scaling factor. [sx, sy, sz]
Type mathutils.Vector

worldScale
The object’s world scaling factor. [sx, sy, sz]
Type mathutils.Vector

localPosition
The object’s local position. [x, y, z]
Type mathutils.Vector

worldPosition
The object’s world position. [x, y, z]
Type mathutils.Vector

localTransform
The object’s local space transform matrix. 4x4 Matrix.
Type mathutils.Matrix

worldTransform
The object’s world space transform matrix. 4x4 Matrix.
Type mathutils.Matrix

localLinearVelocity
The object’s local linear velocity. [x, y, z]
Type mathutils.Vector

worldLinearVelocity
The object’s world linear velocity. [x, y, z]
Type mathutils.Vector

localAngularVelocity
The object’s local angular velocity. [x, y, z]
worldAngularVelocity
The object’s world angular velocity. [x, y, z]
Type \textit{mathutils.Vector}

gravity
The object’s gravity. [x, y, z]
Type \textit{mathutils.Vector}

timeOffset
adjust the slowparent delay at runtime.
Type float

state
the game object’s state bitmask, using the first 30 bits, one bit must always be set.
Type int

meshes
a list meshes for this object.
Type list of \textit{KX.Mesh}

Note: Most objects use only 1 mesh.

Note: Changes to this list will not update the KX.GameObject.

batchGroup
The object batch group containing the batched mesh.
Type \textit{KX.BatchGroup}

sensors
a sequence of \textit{SCA.ISensor} objects with string/index lookups and iterator support.
Type list

Note: This attribute is experimental and may be removed (but probably won’t be).

Note: Changes to this list will not update the KX.GameObject.

controllers
a sequence of \textit{SCA.IController} objects with string/index lookups and iterator support.
Type list of \textit{SCA.ISensor}

Note: This attribute is experimental and may be removed (but probably won’t be).

Note: Changes to this list will not update the KX.GameObject.
**actuators**

A list of `SCA_IActuator` with string/index lookups and iterator support.

*Type* list

---

**Note:** This attribute is experimental and may be removed (but probably won’t be).

---

**Note:** Changes to this list will not update the KX_GameObject.

---

**attrDict**

Get the object’s internal python attribute dictionary for direct (faster) access.

*Type* dict

---

**components**

All python components.

*Type* `EXP_ListValue` of `KX_PYPythonComponent`’s

---

**children**

Direct children of this object, (read-only).

*Type* `EXP_ListValue` of `KX_GameObject`’s

---

**childrenRecursive**

All children of this object including children’s children, (read-only).

*Type* `EXP_ListValue` of `KX_GameObject`’s

---

**life**

The number of frames until the object ends, assumes one frame is 1/50 second (read-only).

*Type* float

---

**debug**

If true, the object’s debug properties will be displayed on screen.

*Type* boolean

---

**debugRecursive**

If true, the object’s and children’s debug properties will be displayed on screen.

*Type* boolean

---

**currentLodLevel**

The index of the level of detail (LOD) currently used by this object (read-only).

*Type* int

---

**lodManager**

Return the lod manager of this object. Needed to access to lod manager to set attributes of levels of detail of this object. The lod manager is shared between instance objects and can be changed to use the lod levels of an other object. If the lod manager is set to `None` the object’s mesh backs to the mesh of the previous first lod level.

*Type* `KX_LodManager`

---

**endObject()**

Delete this object, can be used in place of the EndObject Actuator.

The actual removal of the object from the scene is delayed.
replaceMesh (mesh, useDisplayMesh=True, usePhysicsMesh=False)
Replace the mesh of this object with a new mesh. This works the same way as the actuator.

Parameters
- mesh (MeshProxy or string) – mesh to replace or the mesh’s name.
- useDisplayMesh (boolean) – when enabled the display mesh will be replaced (optional argument).
- usePhysicsMesh (boolean) – when enabled the physics mesh will be replaced (optional argument).

setVisible (visible[, recursive])
Sets the game object’s visible flag.

Parameters
- visible (boolean) – the visible state to set.
- recursive (boolean) – optional argument to set all children’s visibility flag too, defaults to False if no value passed.

setOcclusion (occlusion[, recursive])
Sets the game object’s occlusion capability.

Parameters
- occlusion (boolean) – the state to set the occlusion to.
- recursive (boolean) – optional argument to set all children’s occlusion flag too, defaults to False if no value passed.

alignAxisToVect (vect, axis=2, factor=1.0)
Aligns any of the game object’s axes along the given vector.

Parameters
- vect (3D vector) – a vector to align the axis.
- axis (integer) – The axis you want to align
  - 0: X axis
  - 1: Y axis
  - 2: Z axis
- factor (float) – Only rotate a fraction of the distance to the target vector (0.0 - 1.0)

getAxisVect (vect)
Returns the axis vector rotates by the object’s world space orientation. This is the equivalent of multiplying the vector by the orientation matrix.

Parameters vect (3D Vector) – a vector to align the axis.

Returns The vector in relation to the object’s rotation.

Return type 3d vector.

applyMovement (movement[, local])
Sets the game object’s movement.

Parameters
- movement (3D Vector) – movement vector.
- local –
– False: get the “global” movement ie: relative to world orientation.
– True: get the “local” movement ie: relative to object orientation.

Default to False if not passed.

• **local** – boolean

**applyRotation** \((rotation[, \text{local}])\)

Sets the game object’s rotation.

**Parameters**

• **rotation** \((3D \ Vector)\) – rotation vector.

• **local** –

  – False: get the “global” rotation ie: relative to world orientation.
  – True: get the “local” rotation ie: relative to object orientation.

Default to False if not passed.

• **local** – boolean

**applyForce** \((force[, \text{local}])\)

Sets the game object’s force.

This requires a dynamic object.

**Parameters**

• **force** \((3D \ Vector)\) – force vector.

• **local** \((boolean)\) –

  – False: get the “global” force ie: relative to world orientation.
  – True: get the “local” force ie: relative to object orientation.

Default to False if not passed.

**applyTorque** \((torque[, \text{local}])\)

Sets the game object’s torque.

This requires a dynamic object.

**Parameters**

• **torque** \((3D \ Vector)\) – torque vector.

• **local** \((boolean)\) –

  – False: get the “global” torque ie: relative to world orientation.
  – True: get the “local” torque ie: relative to object orientation.

Default to False if not passed.

**getLinearVelocity** \([\text{local}]\)

Gets the game object’s linear velocity.

This method returns the game object’s velocity through it’s center of mass, ie no angular velocity component.

**Parameters**

**local** \((boolean)\) –

• False: get the “global” velocity ie: relative to world orientation.

• True: get the “local” velocity ie: relative to object orientation.
Default to False if not passed.

**Returns**  the object’s linear velocity.

**Return type**  \textit{Vector(}vx, vy, vz\text{)}

\textbf{setLinearVelocity} (velocity\text{, local \text{)}}

Sets the game object’s linear velocity.

This method sets game object’s velocity through it’s center of mass, ie no angular velocity component.

This requires a dynamic object.

**Parameters**

\begin{itemize}
  \item \textbf{velocity} (3D \textit{Vector}) – linear velocity vector.
  \item \textbf{local} (\textit{boolean}) –
    \begin{itemize}
      \item False: get the “global” velocity ie: relative to world orientation.
      \item True: get the “local” velocity ie: relative to object orientation.
    \end{itemize}
\end{itemize}

Default to False if not passed.

\textbf{getAngularVelocity} (\text{local \text{)}}

Gets the game object’s angular velocity.

**Parameters**  \textbf{local} (\textit{boolean}) –

\begin{itemize}
  \item False: get the “global” velocity ie: relative to world orientation.
  \item True: get the “local” velocity ie: relative to object orientation.
\end{itemize}

Default to False if not passed.

**Returns**  the object’s angular velocity.

**Return type**  \textit{Vector(}vx, vy, vz\text{)}

\textbf{setAngularVelocity} (velocity\text{, local \text{)}}

Sets the game object’s angular velocity.

This requires a dynamic object.

**Parameters**

\begin{itemize}
  \item \textbf{velocity} (\textit{boolean}) – angular velocity vector.
  \item \textbf{local} –
    \begin{itemize}
      \item False: get the “global” velocity ie: relative to world orientation.
      \item True: get the “local” velocity ie: relative to object orientation.
    \end{itemize}
\end{itemize}

Default to False if not passed.

\textbf{getVelocity} (\text{point \text{)}}

Gets the game object’s velocity at the specified point.

Gets the game object’s velocity at the specified point, including angular components.

**Parameters**  \textbf{point} (3D \textit{Vector}) – optional point to return the velocity for, in local coordinates, defaults to (0, 0, 0) if no value passed.

**Returns**  the velocity at the specified point.

**Return type**  \textit{Vector(}vx, vy, vz\text{)}}
getReactionForce()
Gets the game object’s reaction force.

The reaction force is the force applied to this object over the last simulation timestep. This also includes impulses, eg from collisions.

Returns the reaction force of this object.

Return type Vector((fx, fy, fz))

Note: This is not implemented at the moment.

applyImpulse(point, impulse, local)
Applies an impulse to the game object.

This will apply the specified impulse to the game object at the specified point. If point != position, applyImpulse will also change the object’s angular momentum. Otherwise, only linear momentum will change.

Parameters

• point (point [ix, iy, iz] the point to apply the impulse to (in world or local coordinates)) – the point to apply the impulse to (in world or local coordinates)

• impulse (3D Vector) – impulse vector.

• local (boolean) –
  – False: get the “global” impulse ie: relative to world coordinates with world orientation.
  – True: get the “local” impulse ie: relative to local coordinates with object orientation.

Default to False if not passed.

setDamping(linear_damping, angular_damping)
Sets both the linearDamping and angularDamping simultaneously. This is more efficient than setting both properties individually.

Parameters

• linear_damping (float [0, 1]) – Linear (“translational”) damping factor.

• angular_damping (float [0, 1]) – Angular (“rotational”) damping factor.

suspendPhysics([freeConstraints])
Suspends physics for this object.

Parameters freeConstraints (bool) – When set to True physics constraints used by the object are deleted. Else when False (the default) constraints are restored when restoring physics.

restorePhysics()
Resumes physics for this object. Also reinstates collisions.

suspendDynamics([ghost])
Suspends dynamics physics for this object.

Parameters ghost (bool) – When set to True, collisions with the object will be ignored, similar to the “ghost” checkbox in Blender. When False (the default), the object becomes static but still collide with other objects.

See also:

isSuspendDynamics allows you to inspect whether the object is in a suspended state.
**restoreDynamics()**
Resumes dynamics physics for this object. Also reinstates collisions; the object will no longer be a ghost.

*Note:* The object’s linear velocity will be applied from when the dynamics were suspended.

**enableRigidBody()**
Enables rigid body physics for this object.
Rigid body physics allows the object to roll on collisions.

**disableRigidBody()**
Disables rigid body physics for this object.

**setParent(parent, compound=True, ghost=True)**
Sets this object’s parent. Control the shape status with the optional compound and ghost parameters:

In that case you can control if it should be ghost or not:

**Parameters**
- **parent** *(KX_GameObject)* – new parent object.
- **compound** *(boolean)* – whether the shape should be added to the parent compound shape.
  - True: the object shape should be added to the parent compound shape.
  - False: the object should keep its individual shape.
- **ghost** *(boolean)* – whether the object should be ghost while parented.
  - True: if the object should be made ghost while parented.
  - False: if the object should be solid while parented.

*Note:* If the object type is sensor, it stays ghost regardless of ghost parameter

**removeParent()**
Removes this object’s parent.

**getPhysicsId()**
Returns the user data object associated with this game object’s physics controller.

**getPropertyNames()**
Gets a list of all property names.

*Returns* All property names for this object.

*Return type* list

**getDistanceTo(other)**

*Parameters* other *(KX_GameObject or list [x, y, z])* – a point or another KX_GameObject to measure the distance to.

*Returns* distance to another object or point.

*Return type* float

**getVectTo(other)**

Returns the vector and the distance to another object or point. The vector is normalized unless the distance is 0, in which a zero length vector is returned.
Parameters `other` (`KX_GameObject` or list `[x, y, z]`) – a point or another `KX_GameObject`

to get the vector and distance to.

Returns (distance, globalVector(3), localVector(3))

Return type 3-tuple `(float, 3-tuple (x, y, z), 3-tuple (x, y, z))`

`rayCastTo(other, dist=0, prop="")`

Look towards another point/object and find first object hit within dist that matches prop.

The ray is always casted from the center of the object, ignoring the object itself. The ray is casted towards the center of another object or an explicit `[x, y, z]` point. Use `rayCast()` if you need to retrieve the hit point

Parameters

- `other` (`KX_GameObject` or 3-tuple) – `[x, y, z]` or object towards which the ray is casted
- `dist` (float) – max distance to look (can be negative => look behind); 0 or omitted => detect up to other
- `prop` (string) – property name that object must have; can be omitted => detect any object

Returns the first object hit or None if no object or object does not match prop

Return type `KX_GameObject`

`rayCast(objto, objfrom=None, dist=0, prop="", face=False, xray=False, poly=0, mask=0xFFFF)`

Look from a point/object to another point/object and find first object hit within dist that matches prop. if poly is 0, returns a 3-tuple with object reference, hit point and hit normal or (None, None, None) if no hit. if poly is 1, returns a 4-tuple with in addition a `KX_PolyProxy` as 4th element. if poly is 2, returns a 5-tuple with in addition a 2D vector with the UV mapping of the hit point as 5th element.

```python
# shoot along the axis gun-gunAim (gunAim should be collision-free)
obj, point, normal = gun.rayCast(gunAim, None, 50)
if obj:
    # do something
    pass
```

The face parameter determines the orientation of the normal.

- 0 => hit normal is always oriented towards the ray origin (as if you casted the ray from outside)
- 1 => hit normal is the real face normal (only for mesh object, otherwise face has no effect)

The ray has X-Ray capability if xray parameter is 1, otherwise the first object hit (other than self object) stops the ray. The prop and xray parameters interact as follow.

- prop off, xray off: return closest hit or no hit if there is no object on the full extend of the ray.
- prop off, xray on : idem.
- prop on, xray off: return closest hit if it matches prop, no hit otherwise.
- prop on, xray on : return closest hit matching prop or no hit if there is no object matching prop on the full extend of the ray.

The `KX_PolyProxy` 4th element of the return tuple when poly=1 allows to retrieve information on the polygon hit by the ray. If there is no hit or the hit object is not a static mesh, None is returned as 4th element.

The ray ignores collision-free objects and faces that dont have the collision flag enabled, you can however use ghost objects.

Parameters
• `objto` *(KX_GameObject or 3-tuple)* – [x, y, z] or object to which the ray is casted
• `objfrom` *(KX_GameObject or 3-tuple or None)* – [x, y, z] or object from which the ray is casted; None or omitted => use self object center
• `dist` *(float)* – max distance to look (can be negative => look behind); 0 or omitted => detect up to to
• `prop` *(string)* – property name that object must have; can be omitted or “” => detect any object
• `face` *(integer)* – normal option: 1=>return face normal; 0 or omitted => normal is oriented towards origin
• `xray` *(integer)* – X-ray option: 1=>skip objects that don’t match prop; 0 or omitted => stop on first object
• `poly` *(integer)* – polygon option: 0, 1 or 2 to return a 3-, 4- or 5-tuple with information on the face hit.
  – 0 or omitted: return value is a 3-tuple (object, hitpoint, hitnormal) or (None, None, None) if no hit
  – 1: return value is a 4-tuple and the 4th element is a `KX_PolyProxy` or None if no hit or the object doesn’t use a mesh collision shape.
  – 2: return value is a 5-tuple and the 5th element is a 2-tuple (u, v) with the UV mapping of the hit point or None if no hit, or the object doesn’t use a mesh collision shape, or doesn’t have a UV mapping.
• `mask` *(bitfield)* – collision mask: The collision mask (16 layers mapped to a 16-bit integer) is combined with each object’s collision group, to hit only a subset of the objects in the scene. Only those objects for which `collisionGroup & mask` is true can be hit.

**Returns**

*(object, hitpoint, hitnormal)* or *(object, hitpoint, hitnormal, polygon)* or *(object, hitpoint, hitnormal, polygon, hituv)*.

• object, hitpoint and hitnormal are None if no hit.
• polygon is valid only if the object is valid and is a static object, a dynamic object using mesh collision shape or a soft body object, otherwise it is None
• hituv is valid only if polygon is valid and the object has a UV mapping, otherwise it is None

**Return type**

• 3-tuple *(KX_GameObject, 3-tuple (x, y, z), 3-tuple (nx, ny, nz))*
• or 4-tuple *(KX_GameObject, 3-tuple (x, y, z), 3-tuple (nx, ny, nz), KX_PolyProxy)*
• or 5-tuple *(KX_GameObject, 3-tuple (x, y, z), 3-tuple (nx, ny, nz), KX_PolyProxy, 2-tuple (u, v))*

**Note:** The ray ignores the object on which the method is called. It is casted from/to object center or explicit [x, y, z] points.

collide *(obj)*

Test if this object collides object *obj*. 
Parameters `obj` (string or `KX_GameObject`) – the object to test collision with

Returns

- collide, True if this object collides object `obj`
- points, contact point data of the collision or None

Return type 2-tuple (boolean, list of `KX_CollisionContactPoint` or None)

`setCollisionMargin` *(margin)*
Set the objects collision margin.

Parameters `margin` *(float)* – the collision margin distance in blender units.

**Note:** If this object has no physics controller (a physics ID of zero), this function will raise RuntimeError.

`sendsMessage` *(subject, body="", to="")*
Sends a message.

Parameters

- `subject` *(string)* – The subject of the message
- `body` *(string)* – The body of the message (optional)
- `to` *(string)* – The name of the object to send the message to (optional)

`reinstancePhysicsMesh` *(gameObject, meshObject, dupli)*
Updates the physics system with the changed mesh.
If no arguments are given the physics mesh will be re-created from the first mesh assigned to the game object.

Parameters

- `gameObject` *(string, KX_GameObject or None)* – optional argument, set the physics shape from this gameObjets mesh.
- `meshObject` *(string, MeshProxy or None)* – optional argument, set the physics shape from this mesh.
- `dupli` *(boolean)* – optional argument, duplicate the physics shape.

Returns True if reinstance succeeded, False if it failed.

Return type boolean

**Note:** If this object has instances the other instances will be updated too.

**Note:** The gameObject argument has an advantage that it can convert from a mesh with modifiers applied (such as the Subdivision Surface modifier).

**Warning:** Only triangle mesh type objects are supported currently (not convex hull)

8.1. Game Types (`bge.types`) 185
Warning: If the object is a part of a compound object it will fail (parent or child)

Warning: Rebuilding the physics mesh can be slow, running many times per second will give a performance hit.

Warning: Duplicate the physics mesh can use much more memory, use this option only for duplicated meshes else use replacePhysicsShape().

replacePhysicsShape(gameObject)
Replace the current physics shape.

Parameters gameObject (string, KX_GameObject) – set the physics shape from this gameObjs.

get(key[, default])
Return the value matching key, or the default value if its not found. :arg key: the matching key :type key: string :arg default: optional default value is the key isn’t matching, defaults to None if no value passed. :return: The key value or a default.

playAction(name, start_frame, end_frame, layer=0, priority=0, blendin=0, play_mode=KX_ACTION_MODE_PLAY, layer_weight=0.0, ipo_flags=0, speed=1.0, blend_mode=KX_ACTION_BLEND_BLEND)
Plays an action.

Parameters

• name(string) – the name of the action
• start(float) – the start frame of the action
• end(float) – the end frame of the action
• layer(integer) – the layer the action will play in (actions in different layers are added/blended together)
• priority(integer) – only play this action if there isn’t an action currently playing in this layer with a higher (lower number) priority
• blendin(float) – the amount of blending between this animation and the previous one on this layer
• play_mode (one of these constants) – the play mode
• layer_weight(float) – how much of the previous layer to use for blending
• ipo_flags(int bitfield) – flags for the old IPO behaviors (force, etc)
• speed(float) – the playback speed of the action as a factor (1.0 = normal speed, 2.0 = 2x speed, etc)
• blend_mode (one of these constants) – how to blend this layer with previous layers

stopAction([layer])
Stop playing the action on the given layer.

Parameters layer(integer) – The layer to stop playing, defaults to 0 if no value passed.
**getActionFrame([layer])**

Gets the current frame of the action playing in the supplied layer.

**Parameters**  
* layer (*integer*) – The layer that you want to get the frame from, defaults to 0 if no value passed.

**Returns**  
The current frame of the action

**Return type**  
float

**getActionName([layer])**

Gets the name of the current action playing in the supplied layer.

**Parameters**  
* layer (*integer*) – The layer that you want to get the action name from, defaults to 0 if no value passed.

**Returns**  
The name of the current action

**Return type**  
string

**setActionFrame(frame[, layer])**

Set the current frame of the action playing in the supplied layer.

**Parameters**  
* layer (*integer*) – The layer where you want to set the frame, default to 0 if no value passed.
* frame (*float*) – The frame to set the action to

**isPlayingAction([layer])**

Checks to see if there is an action playing in the given layer.

**Parameters**  
* layer (*integer*) – The layer to check for a playing action, defaults to 0 if no value passed.

**Returns**  
Whether or not the action is playing

**Return type**  
boolean

**addDebugProperty(name[, debug])**

Adds a single debug property to the debug list.

**Parameters**  
* name (*string*) – name of the property that added to the debug list.
* debug (*boolean*) – the debug state, default to True if no value passed.

---

**KX_LibLoadStatus(EXP_PyObjectPlus)**

**base class**  
EXP_PyObjectPlus

**class KX_LibLoadStatus(EXP_PyObjectPlus)**

An object providing information about a LibLoad() operation.

```python
# Print a message when an async LibLoad is done
import bge

def finished_cb(status):
    print("Library (%s) loaded in %.2fms." % (status.libraryName, status.timeTaken))

bge.logic.LibLoad('myblend.blend', 'Scene', async=True).onFinish = finished_cb
```

---

8.1. Game Types (bge.types) 187
onFinish
A callback that gets called when the lib load is done.
  Type callable

finished
The current status of the lib load.
  Type boolean

progress
The current progress of the lib load as a normalized value from 0.0 to 1.0.
  Type float

libraryName
The name of the library being loaded (the first argument to LibLoad).
  Type string

timeTaken
The amount of time, in seconds, the lib load took (0 until the operation is complete).
  Type float

KX_LightObject(KX_GameObject)

base class — KX_GameObject

class KX_LightObject(KX_GameObject)
A Light object.

# Turn on a red alert light.
import bge
co = bge.logic.getCurrentController()
ligh = co.owner

light.energy = 1.0
light.color = [1.0, 0.0, 0.0]

SPOT
A spot light source. See attribute type

SUN
A point light source with no attenuation. See attribute type

NORMAL
A point light source. See attribute type

HEMI
A hemi light source. See attribute type

type
The type of light - must be SPOT, SUN or NORMAL

energy
The brightness of this light.
  Type float

shadowClipStart
The shadowmap clip start, below which objects will not generate shadows.
shadowClipEnd
The shadowmap clip end, beyond which objects will not generate shadows.

Type float (read only)

shadowFrustumSize
Size of the frustum used for creating the shadowmap.

Type float (read only)

shadowBindId
The OpenGL shadow texture bind number/id.

Type int (read only)

shadowMapType
The shadow map type (0 -> Simple; 1 -> Variance)

Type int (read only)

shadowBias
The shadow buffer sampling bias.

Type float (read only)

shadowBleedBias
The bias for reducing light-bleed on variance shadow maps.

Type float (read only)

useShadow
Returns True if the light has Shadow option activated, else returns False.

Type boolean (read only)

shadowColor
The color of this light shadows. Black = (0.0, 0.0, 0.0), White = (1.0, 1.0, 1.0).

Type mathutils.Color (read only)

shadowMatrix
Matrix that converts a vector in camera space to shadow buffer depth space.

Computed as: mat4_perspective_to_depth * mat4_lamp_to_perspective * mat4_world_to_lamp * mat4_cam_to_world.

mat4_perspective_to_depth is a fixed matrix defined as follow:
0.5 0.0 0.0 0.5 0.0 0.5 0.0 0.5 0.0 0.0 0.5 0.5 0.0 0.0 0.0 1.0

Type Matrix4x4 (read only)

distance
The maximum distance this light can illuminate. (SPOT and NORMAL lights only).

Type float

color
The color of this light. Black = [0.0, 0.0, 0.0], White = [1.0, 1.0, 1.0].

Type list [r, g, b]

lin_attenuation
The linear component of this light’s attenuation. (SPOT and NORMAL lights only).
Type float

**quad attenuation**
The quadratic component of this light’s attenuation (SPOT and NORMAL lights only).

  Type float

**spotsize**
The cone angle of the spot light, in degrees (SPOT lights only).

  Type float in [0 - 180]

**spotblend**
Specifies the intensity distribution of the spot light (SPOT lights only).

  Type float in [0 - 1]

---

**Note:** Higher values result in a more focused light source.

**staticShadow**
Enables static shadows. By default (staticShadow=False) the shadow cast by the lamp is recalculated every frame. When this is not needed, set staticShadow=True. In that case, call `updateShadow()` to request a shadow update.

  Type boolean.

**updateShadow()**
Set the shadow to be updated next frame if the lamp uses a static shadow, see `staticShadow`.

---

**KX_LodLevel(EXP_PyObjectPlus)**

base class — `EXP_PyObjectPlus`

class **KX_LodLevel** (`EXP_PyObjectPlus`)
A single lod level for a game object lod manager.

**mesh**
The mesh used for this lod level. (read only)

  Type RAS_MeshObject

**level**
The number of the lod level. (read only)

  Type integer

**distance**
Distance to begin using this level of detail. (read only)

  Type float (0.0 to infinite)

**hysteresis**
Minimum distance factor change required to transition to the previous level of detail in percent. (read only)

  Type float [0.0 to 100.0]

**useMesh**
Return True if the lod level uses a different mesh than the original object mesh. (read only)

  Type boolean
useMaterial
Return True if the lod level uses a different material than the original object mesh material. (read only)
   Type boolean
useHysteresis
Return true if the lod level uses hysteresis override. (read only)
   Type boolean

KX_LodManager(EXP_PyObjectPlus)
base class — EXP_PyObjectPlus
class KX_LodManager (EXP_PyObjectPlus)
This class contains a list of all levels of detail used by a game object.

levels
Return the list of all levels of detail of the lod manager.
   Type list of KX_LodLevel
distanceFactor
Method to multiply the distance to the camera.
   Type float

KX_Mesh(EXP_Value)
base class — EXP_Value
class KX_Mesh (EXP_Value)
A mesh object.
You can only change the vertex properties of a mesh object, not the mesh topology.
To use mesh objects effectively, you should know a bit about how the game engine handles them.

1. Mesh Objects are converted from Blender at scene load.
2. The Converter groups polygons by Material. This means they can be sent to the renderer efficiently. A
   material holds:
   1. The texture.
   2. The Blender material.
   3. The Tile properties
   4. The face properties - (From the “Texture Face” panel)
   5. Transparency & z sorting
   6. Light layer
   7. Polygon shape (triangle/quad)
   8. Game Object
3. Vertices will be split by face if necessary. Vertices can only be shared between faces if:
   1. They are at the same position
   2. UV coordinates are the same

8.1. Game Types (bge.types)
3. Their normals are the same (both polygons are “Set Smooth”)
4. They are the same color, for example: a cube has 24 vertices: 6 faces with 4 vertices per face.

The correct method of iterating over every `KX_VertexProxy` in a game object

```python
from bge import logic
cont = logic.getCurrentController()
object = cont.owner

for mesh in object.meshes:
    for m_index in range(len(mesh.materials)):
        for v_index in range(mesh.getVertexArrayLength(m_index)):
            vertex = mesh.getVertex(m_index, v_index)
            # Do something with vertex here...
            # ... eg: color the vertex red.
            vertex.color = [1.0, 0.0, 0.0, 1.0]
```

**materials**

Type list of `KX_BlenderMaterial` type

**numPolygons**

Type integer

**numMaterials**

Type integer

**polygons**

Returns the list of polygons of this mesh.

Type `KX_PolyProxy` list (read only)

**getMaterialName** *(matid)*

Gets the name of the specified material.

Parameters `matid` (integer) – the specified material.

Returns the attached material name.

Return type string

**getTextureName** *(matid)*

Gets the name of the specified material’s texture.

Parameters `matid` (integer) – the specified material

Returns the attached material’s texture name.

Return type string

**getVertexArrayLength** *(matid)*

Gets the length of the vertex array associated with the specified material.

There is one vertex array for each material.

Parameters `matid` (integer) – the specified material

Returns the number of vertices in the vertex array.

Return type integer

**getVertex** *(matid, index)*

Gets the specified vertex from the mesh object.
Parameters

- **matid** (**integer**) – the specified material
- **index** (**integer**) – the index into the vertex array.

Returns a vertex object.

Return type **KX_VertexProxy**

**getPolygon** (**index**)

Gets the specified polygon from the mesh.

Parameters **index** (**integer**) – polygon number

Returns a polygon object.

Return type **KX_PolyProxy**

**transform** (**matid**, **matrix**)  
Transforms the vertices of a mesh.

Parameters

- **matid** (**integer**) – material index, -1 transforms all.
- **matrix** (**4x4 matrix [[float]]**) – transformation matrix.

**transformUV** (**matid**, **matrix**, **uv_index=-1**, **uv_index_from=-1**)  
Transforms the vertices UV’s of a mesh.

Parameters

- **matid** (**integer**) – material index, -1 transforms all.
- **matrix** (**4x4 matrix [[float]]**) – transformation matrix.
- **uv_index** (**integer**) – optional uv index, -1 for all, otherwise 0 or 1.
- **uv_index_from** (**integer**) – optional uv index to copy from, -1 to transform the current uv.

**replaceMaterial** (**matid**, **material**)  
Replace the material in slot **matid** by the material **material**.

Parameters

- **matid** (**integer**) – The material index.
- **material** (**KX_BlenderMaterial**) – The material replacement.

**Warning:** Changing the material of a mesh used by many objects can be slow. This function should not be called every frames.

**copy** ()  
Return a duplicated mesh.

Returns a duplicated mesh of the current used.

Return type **KX_Mesh**
KX_MouseActuator(SCA_IActuator)

base class — SCA_IActuator

class KX_MouseActuator (SCA_IActuator)
    The mouse actuator gives control over the visibility of the mouse cursor and rotates the parent object according to mouse movement.

    reset()
        Undoes the rotation caused by the mouse actuator.

    visible
        The visibility of the mouse cursor.
            Type boolean

    use_axis_x
        Mouse movement along the x axis effects object rotation.
            Type boolean

    use_axis_y
        Mouse movement along the y axis effects object rotation.
            Type boolean

    threshold
        Amount of movement from the mouse required before rotation is triggered.
            Type list (vector of 2 floats)
            The values in the list should be between 0.0 and 0.5.

    reset_x
        Mouse is locked to the center of the screen on the x axis.
            Type boolean

    reset_y
        Mouse is locked to the center of the screen on the y axis.
            Type boolean

    object_axis
        The object's 3D axis to rotate with the mouse movement. ([x, y])
            Type list (vector of 2 integers from 0 to 2)
            • KX_ACT_MOUSE_OBJECT_AXIS_X
            • KX_ACT_MOUSE_OBJECT_AXIS_Y
            • KX_ACT_MOUSE_OBJECT_AXIS_Z

    local_x
        Rotation caused by mouse movement along the x axis is local.
            Type boolean

    local_y
        Rotation caused by mouse movement along the y axis is local.
            Type boolean
sensitivity
The amount of rotation caused by mouse movement along the x and y axis.

  Type  list (vector of 2 floats)
  Negative values invert the rotation.

limit_x
The minimum and maximum angle of rotation caused by mouse movement along the x axis in degrees.
limit_x[0] is minimum, limit_x[1] is maximum.

  Type  list (vector of 2 floats)

limit_y
The minimum and maximum angle of rotation caused by mouse movement along the y axis in degrees.
limit_y[0] is minimum, limit_y[1] is maximum.

  Type  list (vector of 2 floats)

angle
The current rotational offset caused by the mouse actuator in degrees.

  Type  list (vector of 2 floats)

KX_MouseFocusSensor(SCA_MouseSensor)

base class — SCA_MouseSensor

class KX_MouseFocusSensor (SCA_MouseSensor)
  The mouse focus sensor detects when the mouse is over the current game object.
  The mouse focus sensor works by transforming the mouse coordinates from 2d device space to 3d space then raycasting away from the camera.

  raySource
  The worldspace source of the ray (the view position).

    Type  list (vector of 3 floats)

  rayTarget
  The worldspace target of the ray.

    Type  list (vector of 3 floats)

  rayDirection
    The \text{rayTarget} - \text{raySource} normalized.

    Type  list (normalized vector of 3 floats)

  hitObject
    the last object the mouse was over.

    Type  KX_GameObject or None

  hitPosition
    The worldspace position of the ray intersecton.

    Type  list (vector of 3 floats)

  hitNormal
    the worldspace normal from the face at point of intersection.

    Type  list (normalized vector of 3 floats)
hitUV
the UV coordinates at the point of intersection.

Type list (vector of 2 floats)
If the object has no UV mapping, it returns [0, 0].
The UV coordinates are not normalized, they can be < 0 or > 1 depending on the UV mapping.

usePulseFocus
When enabled, moving the mouse over a different object generates a pulse. (only used when the ‘Mouse Over Any’ sensor option is set).

Type boolean

useXRay
If enabled it allows the sensor to see through game objects that don’t have the selected property or material.

Type boolean

mask
The collision mask (16 layers mapped to a 16-bit integer) combined with each object’s collision group, to hit only a subset of the objects in the scene. Only those objects for which collisionGroup & mask is true can be hit.

Type bitfield

propName
The property or material the sensor is looking for.

Type string

useMaterial
Determines if the sensor is looking for a property or material. KX_True = Find material; KX_False = Find property.

Type boolean

KX_MovementSensor(SCA_ISensor)

base class — SCA_ISensor
class KX_MovementSensor (SCA_ISensor)
Movement sensor detects movement of owner object at a given threshold.

axis
The axis which the movement will be detected. Can be one of these constants.

Type int

threshold
Minimum of movement needed to trigger the sensor.

Type float
**KX_NavMeshObject(KX_GameObject)**

**class KX_NavMeshObject (KX_GameObject)**
Python interface for using and controlling navigation meshes.

### findPath (start, goal)
Finds the path from start to goal points.

**Parameters**
- `start` – the start point
- `start` – 3D Vector
- `goal` – the goal point
- `start` – 3D Vector

**Returns** a path as a list of points

**Return type** list of points

### raycast (start, goal)
Raycast from start to goal points.

**Parameters**
- `start` – the start point
- `start` – 3D Vector
- `goal` – the goal point
- `start` – 3D Vector

**Returns** the hit factor

**Return type** float

### draw (mode)
Draws a debug mesh for the navigation mesh.

**Parameters**
- `mode` – the drawing mode (one of these constants)
- `mode` – integer

**Returns** None

### rebuild ()
Rebuild the navigation mesh.

**Returns** None

---

**KX_NearSensor(KX_CollisionSensor)**

**class KX_NearSensor (KX_CollisionSensor)**
A near sensor is a specialised form of touch sensor.

### distance
The near sensor activates when an object is within this distance.
Type  float

resetDistance
   The near sensor deactivates when the object exceeds this distance.
   Type  float

KX_NetworkMessageActuator(SCA_IActuator)

base class — SCA_IActuator

class KX_NetworkMessageActuator(SCA_IActuator)
   Message Actuator

   propName
      Messages will only be sent to objects with the given property name.
      Type  string

   subject
      The subject field of the message.
      Type  string

   body
      The body of the message.
      Type  string

   usePropBody
      Send a property instead of a regular body message.
      Type  boolean

KX_NetworkMessageSensor(SCA_ISensor)

base class — SCA_ISensor

class KX_NetworkMessageSensor(SCA_ISensor)
   The Message Sensor logic brick.
   Currently only loopback (local) networks are supported.

   subject
      The subject the sensor is looking for.
      Type  string

   frameMessageCount
      The number of messages received since the last frame. (read-only).
      Type  integer

   subjects
      The list of message subjects received. (read-only).
      Type  list of strings

   bodies
      The list of message bodies received. (read-only).
      Type  list of strings
KX_ObjectActuator(SCA_IActuator)

base class — SCA_IActuator

class KX_ObjectActuator(SCA_IActuator)
The object actuator ("Motion Actuator") applies force, torque, displacement, angular displacement, velocity, or angular velocity to an object. Servo control allows to regulate force to achieve a certain speed target.

force
The force applied by the actuator.
  Type Vector((x, y, z))

useLocalForce
A flag specifying if the force is local.
  Type boolean

torque
The torque applied by the actuator.
  Type Vector((x, y, z))

useLocalTorque
A flag specifying if the torque is local.
  Type boolean

dLoc
The displacement vector applied by the actuator.
  Type Vector((x, y, z))

useLocalDLoc
A flag specifying if the dLoc is local.
  Type boolean

dRot
The angular displacement vector applied by the actuator
  Type Vector((x, y, z))

useLocalDRot
A flag specifying if the dRot is local.
  Type boolean

linV
The linear velocity applied by the actuator.
  Type Vector((x, y, z))

useLocalLinV
A flag specifying if the linear velocity is local.
  Type boolean

Note: Since the displacement is applied every frame, you must adjust the displacement based on the frame rate, or you game experience will depend on the player’s computer speed.
Note: This is the target speed for servo controllers.

angV
The angular velocity applied by the actuator.

Type Vector((x, y, z))

useLocalAngV
A flag specifying if the angular velocity is local.

Type boolean

damping
The damping parameter of the servo controller.

Type short

forceLimitX
The min/max force limit along the X axis and activates or deactivates the limits in the servo controller.

Type list [min(float), max(float), bool]

forceLimitY
The min/max force limit along the Y axis and activates or deactivates the limits in the servo controller.

Type list [min(float), max(float), bool]

forceLimitZ
The min/max force limit along the Z axis and activates or deactivates the limits in the servo controller.

Type list [min(float), max(float), bool]

pid
The PID coefficients of the servo controller.

Type list of floats [proportional, integral, derivate]

reference
The object that is used as reference to compute the velocity for the servo controller.

Type KX_GameObject or None

KX_ParentActuator(SCA_IActuator)

base class — SCA_IActuator

class KX_ParentActuator (SCA_IActuator)
The parent actuator can set or remove an objects parent object.

object
the object this actuator sets the parent too.

Type KX_GameObject or None

mode
The mode of this actuator.

Type integer from 0 to 1.

compound
Whether the object shape should be added to the parent compound shape when parenting.

Effective only if the parent is already a compound shape.
Type boolean

**ghost**
Whether the object should be made ghost when parenting. Effective only if the shape is not added to the
parent compound shape.

Type boolean

---

**KX_PlanarMap(KX_TextureRenderer)**

base class — *KX_TextureRenderer*

class KX_PlanarMap(KX_TextureRenderer)

Python API for realtime planar map textures.

**normal**
Plane normal used to compute the reflection or refraction orientation of the render camera.

Type `mathutils.Vector`

---

**KX_PolyProxy(SCA_IObject)**

base class — *SCA_IObject*

class KX_PolyProxy(SCA_IObject)

A polygon holds the index of the vertex forming the polygon.

Note: The polygon attributes are read-only, you need to retrieve the vertex proxy if you want to change the
vertex settings.

**material_name**
The name of polygon material, empty if no material.

Type string

**material**
The material of the polygon.

Type `KX_BlenderMaterial`

**texture_name**
The texture name of the polygon.

Type string

**material_id**
The material index of the polygon, use this to retrieve vertex proxy from mesh proxy.

Type integer

**v1**
vertex index of the first vertex of the polygon, use this to retrieve vertex proxy from mesh proxy.

Type integer

**v2**
vertex index of the second vertex of the polygon, use this to retrieve vertex proxy from mesh proxy.

Type integer

**v3**
vertex index of the third vertex of the polygon, use this to retrieve vertex proxy from mesh proxy.
Type integer

v4
  Deprecated since version polygons: are triangles.
  Type integer

visible
  visible state of the polygon: 1=visible, 0=invisible.
  Type integer

collide
  collide state of the polygon: 1=receives collision, 0=collision free.
  Type integer

vertices
  Returns the list of vertices of this polygon.
  Type KX_VertexProxy list (read only)

getMaterialName()
  Returns the polygon material name with MA prefix
  Returns material name
  Return type string

getMaterial()
  Returns The polygon material
  Return type KX_BlenderMaterial

getTextureName()
  Returns The texture name
  Return type string

getMaterialIndex()
  Returns the material bucket index of the polygon. This index and the ones returned by getVertexIndex() are needed to retrieve the vertex proxy from MeshProxy.
  Returns the material index in the mesh
  Return type integer

getNumVertex()
  Returns the number of vertex of the polygon.
  Returns number of vertex.
  Return type integer

isVisible()
  Returns whether the polygon is visible or not
  Returns 0=invisible, 1=visible
  Return type boolean

isCollider()
  Returns whether the polygon is receives collision or not
  Returns 0=collision free, 1=receives collision
Return type integer

gGetVertexIndex(vertex)
Returns the mesh vertex index of a polygon vertex. This index and the one returned by getMaterialIndex() are needed to retrieve the vertex proxy from MeshProxy.

Parameters
• vertex – index of the vertex in the polygon: 0->2

getMesh()
Returns a mesh proxy

8.1. Game Types (bge.types)
```python
def start(self, args):
    self.move_speed = args['Move Speed']
    self.turn_speed = args['Turn Speed']

def update(self):
    keyboard = bge.logic.keyboard.events
    move = 0
    rotate = 0

    if keyboard[bge.events.WKEY]:
        move += self.move_speed
    if keyboard[bge.events.SKEY]:
        move -= self.move_speed
    if keyboard[bge.events.AKEY]:
        rotate += self.turn_speed
    if keyboard[bge.events.DKEY]:
        rotate -= self.turn_speed

    self.object.applyMovement((0, move, 0), True)
    self.object.applyRotation((0, 0, rotate), True)
```

Since the components are loaded for the first time outside the bge, then `bge` is a fake module that contains only
the class `KX_PythonComponent` to avoid importing all the bge modules. This behavior is safer but creates
some issues at loading when the user want to use functions or attributes from the bge modules other than the
`KX_PythonComponent` class. The way is to not call these functions at loading outside the bge. To detect it,
the bge module contains the attribute `__component__` when it’s imported outside the bge.

The following component example add a “Cube” object at initialization and move it along x for each update.
It shows that the user can use functions from scene and load the component outside the bge by setting global
attributes in a condition at the beginning of the script.

```python
import bge

if not hasattr(bge, "__component__"):
    global scene
    scene = bge.logic.getCurrentScene()

class Component(bge.types.KX_PythonComponent):
    args = {}

    def start(self, args):
        scene.addObject("Cube")

    def update(self):
        scene.objects["Cube"].worldPosition.x += 0.1
```

The property types supported are float, integer, boolean, string, set (for enumeration) and Vector 2D, 3D and
4D. The following example show all of these property types.

```python
from bge import *
from mathutils import *
from collections import OrderedDict

class Component(types.KX_PythonComponent):
    (continues on next page)```
object
The object owner of the component.

Type KX_GameObject

args
Dictionary of the component properties, the keys are string and the value can be: float, integer, Vector(2D/3D/4D), set, string.

Type dict

start (args)
Initialize the component.

Parameters args (dict) – The dictionary of the properties’ name and value.

Warning: This function must be inherited in the python component class.

update ()
Process the logic of the component.

Warning: This function must be inherited in the python component class.

KX_RadarSensor(KX_NearSensor)

base class — KX_NearSensor

class KX_RadarSensor (KX_NearSensor)
Radar sensor is a near sensor with a conical sensor object.

coneOrigin
The origin of the cone with which to test. The origin is in the middle of the cone. (read-only).

Type list of floats [x, y, z]

coneTarget
The center of the bottom face of the cone with which to test. (read-only).
Type list of floats $[x, y, z]$

distance
The height of the cone with which to test.
Type float

angle
The angle of the cone (in degrees) with which to test.
Type float

axis
The axis on which the radar cone is cast. Can be one of these constants.
Type integer from 0 to 5

KX_RaySensor(SCA_ISensor)

base class — SCA_ISensor
class KX_RaySensor(SCA_ISensor)
A ray sensor detects the first object in a given direction.

propName
The property the ray is looking for.
Type string

range
The distance of the ray.
Type float

useMaterial
Whether or not to look for a material (false = property).
Type boolean

useXRay
Whether or not to use XRay.
Type boolean

mask
The collision mask (16 layers mapped to a 16-bit integer) combined with each object’s collision group, to hit only a subset of the objects in the scene. Only those objects for which collisionGroup & mask is true can be hit.
Type bitfield

hitObject
The game object that was hit by the ray. (read-only).
Type KX_GameObject

hitPosition
The position (in worldcoordinates) where the object was hit by the ray. (read-only).
Type list $[x, y, z]$

hitNormal
The normal (in worldcoordinates) of the object at the location where the object was hit by the ray. (read-only).
Type  list [x, y, z]

**hitMaterial**
The material of the object in the face hit by the ray. (read-only).

Type  string

**rayDirection**
The direction from the ray (in world coordinates). (read-only).

Type  list [x, y, z]

**axis**
The axis the ray is pointing on. Can be one of these constants.

Type  integer from 0 to 5

---

**KX_SCA_AddObjectActuator(SCA_IActuator)**

base class — *SCA_IActuator*

**class KX_SCA_AddObjectActuator (SCA_IActuator)**
Edit Object Actuator (in Add Object Mode)

---

**Warning:** An Add Object actuator will be ignored if at game start, the linked object doesn’t exist (or is empty) or the linked object is in an active layer.

**Error:** GameObject *'Name'* has a AddObjectActuator *'ActuatorName'* without object in 'nonactive' layer

---

**object**
the object this actuator adds.

Type  *KX_GameObject* or None

**objectLastCreated**
the last added object from this actuator (read-only).

Type  *KX_GameObject* or None

**time**
the lifetime of added objects, in frames. Set to 0 to disable automatic deletion.

Type  float

**linearVelocity**
the initial linear velocity of added objects.

Type  list [vx, vy, vz]

**angularVelocity**
the initial angular velocity of added objects.

Type  list [vx, vy, vz]

**instantAddObject**
adds the object without needing to calling SCA_PythonController.activate()

---

**Note:** Use objectLastCreated to get the newly created object.
KX_SCA_DynamicActuator(SCA_IActuator)

base class — SCA_IActuator
class KX_SCA_DynamicActuator(SCA_IActuator)
    Dynamic Actuator.
    
    mode
        Type integer
        the type of operation of the actuator, 0-4
        • KX_DYN_RESTORE_DYNAMICS(0)
        • KX_DYN_DISABLE_DYNAMICS(1)
        • KX_DYN_ENABLE_RIGID_BODY(2)
        • KX_DYN_DISABLE_RIGID_BODY(3)
        • KX_DYN_SET_MASS(4)

    mass
        the mass value for the KX_DYN_SET_MASS operation.
        Type float

KX_SCA_EndObjectActuator(SCA_IActuator)

base class — SCA_IActuator
class KX_SCA_EndObjectActuator(SCA_IActuator)
    Edit Object Actuator (in End Object mode)
    This actuator has no python methods.

KX_SCA_ReplaceMeshActuator(SCA_IActuator)

base class — SCA_IActuator
class KX_SCA_ReplaceMeshActuator(SCA_IActuator)
    Edit Object actuator, in Replace Mesh mode.

Warning: Replace mesh actuators will be ignored if at game start, the named mesh doesn’t exist.
This will generate a warning in the console

Error: GameObject 'Name' ReplaceMeshActuator 'ActuatorName' without object

# Level-of-detail
# Switch a game object’s mesh based on its depth in the camera view.
# +------------------+ +------------------+ +------------------+
# | Always | +----+ Python | +----+ Edit Object (Replace Mesh) LOD.Mesh |
# +------------------+ +------------------+ +------------------+
from bge import logic

# List detail meshes here
# Mesh (name, near, far)
# Meshes overlap so that they don't 'pop' when on the edge of the distance.
meshes = (".Hi", 0.0, -20.0),
   (.Med", -15.0, -50.0),
   (.Lo", -40.0, -100.0)
}

cont = logic.getCurrentController()
object = cont.owner
actuator = cont.actuators["LOD." + obj.name]
camera = logic.getCurrentScene().active_camera

def Depth(pos, plane):

# Depth is negative and decreasing further from the camera
depth = Depth(object.position, camera.world_to_camera[2])

newmesh = None
curmesh = None
# Find the lowest detail mesh for depth
for mesh in meshes:
    if depth < mesh[1] and depth > mesh[2]:
        newmesh = mesh
    if "ME" + object.name + mesh[0] == actuator.getMesh():
        curmesh = mesh

if newmesh != None and "ME" + object.name + newmesh[0] != actuator.mesh:
    # The mesh is a different mesh - switch it.
    # Check the current mesh is not a better fit.
    if curmesh == None or curmesh[1] < depth or curmesh[2] > depth:
        actuator.mesh = object.name + newmesh[0]
        cont.activate(actuator)

mesh
    MeshProxy or the name of the mesh that will replace the current one.
    Set to None to disable actuator.

    Type MeshProxy or None if no mesh is set

useDisplayMesh
    when true the displayed mesh is replaced.

    Type boolean

usePhysicsMesh
    when true the physics mesh is replaced.

    Type boolean

instantReplaceMesh()
    Immediately replace mesh without delay.

KX_Scene(EXP_PyObjectPlus)

base class — EXP_PyObjectPlus

8.1. Game Types (bge.types) 209
class KX_Scene (EXP_PyObjectPlus)
An active scene that gives access to objects, cameras, lights and scene attributes.
The activity culling stuff is supposed to disable logic bricks when their owner gets too far from the active camera.
It was taken from some code lurking at the back of KX_Scene - who knows what it does!

```python
from bge import logic

# get the scene
scene = logic.getCurrentScene()

# print all the objects in the scene
for object in scene.objects:
    print(object.name)

# get an object named 'Cube'
object = scene.objects["Cube"]

# get the first object in the scene.
object = scene.objects[0]

# Get the depth of an object in the camera view.
from bge import logic

object = logic.getCurrentController().owner
cam = logic.getCurrentScene().active_camera

# Depth is negative and decreasing further from the camera
```

@bug: All attributes are read only at the moment.

**name**
The scene’s name, (read-only).

  **Type** string

**objects**
A list of objects in the scene, (read-only).

  **Type** EXP_ListValue of KX_GameObject

**objectsInactive**
A list of objects on background layers (used for the addObject actuator), (read-only).

  **Type** EXP_ListValue of KX_GameObject

**lights**
A list of lights in the scene, (read-only).

  **Type** EXP_ListValue of KX_LightObject

**cameras**
A list of cameras in the scene, (read-only).

  **Type** EXP_ListValue of KX_Camera

**texts**
A list of texts in the scene, (read-only).
active_camera
The current active camera.
Type: KX_Camera

Note: This can be set directly from python to avoid using the KX_SceneActuator.

overrideCullingCamera
The override camera used for scene culling, if set to None the culling is proceeded with the camera used to render.
Type: KX_Camera or None

world
The current active world, (read-only).
Type: KX_WorldInfo

filterManager
The scene’s 2D filter manager, (read-only).
Type: KX_2DFilterManager

suspended
True if the scene is suspended, (read-only).
Type: boolean

activityCulling
True if the scene allow object activity culling.
Type: boolean

dbvt_culling
True when Dynamic Bounding box Volume Tree is set (read-only).
Type: boolean

pre_draw
A list of callables to be run before the render step. The callbacks can take as argument the rendered camera.
Type: list

post_draw
A list of callables to be run after the render step.
Type: list

pre_draw_setup
A list of callables to be run before the drawing setup (i.e., before the model view and projection matrices are computed). The callbacks can take as argument the rendered camera, the camera could be temporary in case of stereo rendering.
Type: list

gravity
The scene gravity using the world x, y and z axis.
Type: Vector((gx, gy, gz))

addObject object, reference, time=0.0
Adds an object to the scene like the Add Object Actuator would.
Parameters

• **object** (*KX_GameObject* or string) – The (name of the) object to add.

• **reference** (*KX_GameObject* or string) – The (name of the) object which position, orientation, and scale to copy (optional), if the object to add is a light and there is not reference the light’s layer will be the same that the active layer in the blender scene.

• **time** (*float*) – The lifetime of the added object, in frames (assumes one frame is 1/50 second). A time of 0.0 means the object will last forever (optional).

**Returns**  The newly added object.

**Return type**  *KX_GameObject*

```python
end()
```
Removes the scene from the game.

```python
restart()
```
Restarts the scene.

```python
replace(scene)
```
Replaces this scene with another one.

**Parameters**  **scene** (*string*) – The name of the scene to replace this scene with.

**Returns**  True if the scene exists and was scheduled for addition, False otherwise.

**Return type**  *boolean*

```python
suspend()
```
Suspends this scene.

```python
resume()
```
Resume this scene.

```python
get(key, default=None)
```
Return the value matching key, or the default value if its not found. :return: The key value or a default.

```python
drawObstacleSimulation()
```
Draw debug visualization of obstacle simulation.

**KX_SceneActuator(SCA_IActuator)**

**base class**  *SCA_IActuator*

**class KX_SceneActuator (SCA_IActuator)**

Scene Actuator logic brick.

**Warning:** Scene actuators that use a scene name will be ignored if at game start, the named scene doesn’t exist or is empty

This will generate a warning in the console:

```
Error: GameObject 'Name' has a SceneActuator 'ActuatorName' (SetScene) without
˓
scene
```

**scene**

the name of the scene to change to/overlay/underlay/remove/suspend/resume.

**Type**  *string*
camera
the camera to change to.

Type `KX_Camera` on read, string or `KX_Camera` on write

Note: When setting the attribute, you can use either a `KX_Camera` or the name of the camera.

useRestart
Set flag to True to restart the scene.

Type boolean

dee
The mode of the actuator.

Type integer from 0 to 5.

**KX_SoundActuator(SCA_IActuator)**

class KX_SoundActuator(SCA_IActuator)
Sound Actuator.

The `startSound`, `pauseSound` and `stopSound` do not require the actuator to be activated - they act instantly provided that the actuator has been activated once at least.

**volume**
The volume (gain) of the sound.

Type float

**time**
The current position in the audio stream (in seconds).

Type float

**pitch**
The pitch of the sound.

Type float

**mode**
The operation mode of the actuator. Can be one of these constants

Type integer

**sound**
The sound the actuator should play.

Type Audaspace factory

**is3D**
Whether or not the actuator should be using 3D sound. (read-only)

Type boolean

**volume_maximum**
The maximum gain of the sound, no matter how near it is.

Type float
The minimum gain of the sound, no matter how far it is away.
    Type  float
distance_reference
    The distance where the sound has a gain of 1.0.
    Type  float
distance_maximum
    The maximum distance at which you can hear the sound.
    Type  float
attenuation
    The influence factor on volume depending on distance.
    Type  float
cone_angle_inner
    The angle of the inner cone.
    Type  float
cone_angle_outer
    The angle of the outer cone.
    Type  float
cone_volume_outer
    The gain outside the outer cone (the gain in the outer cone will be interpolated between this value and the
    normal gain in the inner cone).
    Type  float

    startSound()
    Starts the sound.
    Returns  None

    pauseSound()
    Pauses the sound.
    Returns  None

    stopSound()
    Stops the sound.
    Returns  None

KX_StateActuator(SCA_IActuator)

base class — SCA_IActuator
class KX_StateActuator (SCA_IActuator)
    State actuator changes the state mask of parent object.

    operation
    Type of bit operation to be applied on object state mask.
    You can use one of these constants
    Type  integer
**mask**
Value that defines the bits that will be modified by the operation.

The bits that are 1 in the mask will be updated in the object state.

The bits that are 0 are will be left unmodified expect for the Copy operation which copies the mask to the object state.

  Type  integer

**KX_SteeringActuator(SCA_IActuator)**

base class — *SCA_IActuator*

class **KX_SteeringActuator** (SCA_IActuator)
Steering Actuator for navigation.

  **behavior**
  
  The steering behavior to use.

  Type  one of these constants

  **velocity**
  
  Velocity magnitude

  Type  float

  **acceleration**
  
  Max acceleration

  Type  float

  **turnspeed**
  
  Max turn speed

  Type  float

  **distance**
  
  Relax distance

  Type  float

  **target**
  
  Target object

  Type  KX_GameObject

  **navmesh**
  
  Navigation mesh

  Type  KX_GameObject

  **selfterminated**
  
  Terminate when target is reached

  Type  boolean

  **enableVisualization**
  
  Enable debug visualization

  Type  boolean

  **pathUpdatePeriod**
  
  Path update period
Type int

path
Path point list.
Type list of mathutils.Vector

KX_TextureRenderer(EXP_Value)
base class — EXP_Value
class KX_TextureRenderer (EXP_Value)
Python API for object doing a render stored in a texture.

autoUpdate
Choose to update automatically each frame the texture renderer or not.
Type boolean

viewpointObject
The object where the texture renderer will render the scene.
Type KX_GameObject

enabled
Enable the texture renderer to render the scene.
Type boolean

ignoreLayers
The layers to ignore when rendering.
Type bitfield

clipStart
The projection view matrix near plane, used for culling.
Type float

clipEnd
The projection view matrix far plane, used for culling.
Type float

lodDistanceFactor
The factor to multiply distance to camera to adjust levels of detail. A float < 1.0f will make the distance to
camera used to compute levels of detail decrease.
Type float

update()
Request to update this texture renderer during the rendering stage. This function is effective only when
autoUpdate is disabled.

KX_TrackToActuator(SCA_IActuator)
base class — SCA_IActuator
class KX_TrackToActuator (SCA_IActuator)
Edit Object actuator in Track To mode.
Warning: Track To Actuators will be ignored if at game start, the object to track to is invalid.

This will generate a warning in the console:

```
GameObject 'Name' no object in EditObjectActuator 'ActuatorName'
```

**object**

the object this actuator tracks.

Type `KX_GameObject` or `None`

**time**

the time in frames with which to delay the tracking motion.

Type `integer`

**use3D**

the tracking motion to use 3D.

Type `boolean`

**upAxis**

The axis that points upward.

Type `integer` from 0 to 2

- `KX_TRACK_UPAXIS_POS_X`
- `KX_TRACK_UPAXIS_POS_Y`
- `KX_TRACK_UPAXIS_POS_Z`

**trackAxis**

The axis that points to the target object.

Type `integer` from 0 to 5

- `KX_TRACK_TRAXIS_POS_X`
- `KX_TRACK_TRAXIS_POS_Y`
- `KX_TRACK_TRAXIS_POS_Z`
- `KX_TRACK_TRAXIS_NEG_X`
- `KX_TRACK_TRAXIS_NEG_Y`
- `KX_TRACK_TRAXIS_NEG_Z`

---

**KX_VehicleWrapper**(EXP_PyObjectPlus)

base class — `EXP_PyObjectPlus`

class **KX_VehicleWrapper**(EXP_PyObjectPlus)

KX_VehicleWrapper

TODO - description

**addWheel**(`wheel, attachPos, downDir, axleDir, suspensionRestLength, wheelRadius, hasSteering`)

Add a wheel to the vehicle

Parameters
• **wheel** *(KX_GameObject or a KX_GameObject name)* – The object to use as a wheel.

• **attachPos** *(vector of 3 floats)* – The position to attach the wheel, relative to the chassis object center.

• **downDir** *(vector of 3 floats)* – The direction vector pointing down to where the vehicle should collide with the floor.

• **axleDir** *(vector of 3 floats)* – The axis the wheel rotates around, relative to the chassis.

• **suspensionRestLength** *(float)* – The length of the suspension when no forces are being applied.

• **wheelRadius** *(float)* – The radius of the wheel (half the diameter).

• **hasSteering** *(boolean)* – True if the wheel should turn with steering, typically used in front wheels.

**applyBraking** *(force, wheelIndex)*
Apply a braking force to the specified wheel

**Parameters**

• **force** *(float)* – the brake force

• **wheelIndex** *(integer)* – index of the wheel where the force needs to be applied

**applyEngineForce** *(force, wheelIndex)*
Apply an engine force to the specified wheel

**Parameters**

• **force** *(float)* – the engine force

• **wheelIndex** *(integer)* – index of the wheel where the force needs to be applied

**getConstraintId** *
Get the constraint ID

**Returns**  the constraint id

**Return type**  integer

**getConstraintType** *
Returns the constraint type.

**Returns**  constraint type

**Return type**  integer

**getNumWheels** *
Returns the number of wheels.

**Returns**  the number of wheels for this vehicle

**Return type**  integer

**getWheelOrientationQuaternion** *(wheelIndex)*
Returns the wheel orientation as a quaternion.

**Parameters**  **wheelIndex** *(integer)* – the wheel index

**Returns**  TODO Description

**Return type**  TODO - type should be quat as per method name but from the code it looks like a matrix
getWheelPosition\(\text{wheelIndex}\)  
Returns the position of the specified wheel  

Parameters \textbf{wheelIndex} (integer) – the wheel index  
Returns position vector  
Return type list\([x, y, z]\)

getWheelRotation\(\text{wheelIndex}\)  
Returns the rotation of the specified wheel  

Parameters \textbf{wheelIndex} (integer) – the wheel index  
Returns the wheel rotation  
Return type float

setRollInfluence\(\text{rollInfluence, wheelIndex}\)  
Set the specified wheel’s roll influence. The higher the roll influence the more the vehicle will tend to roll over in corners.  

Parameters  
• \textbf{rollInfluence} (float) – the wheel roll influence  
• \textbf{wheelIndex} (integer) – the wheel index

setSteeringValue\(\text{steering, wheelIndex}\)  
Set the specified wheel’s steering  

Parameters  
• \textbf{steering} (float) – the wheel steering  
• \textbf{wheelIndex} (integer) – the wheel index

setSuspensionCompression\(\text{compression, wheelIndex}\)  
Set the specified wheel’s compression  

Parameters  
• \textbf{compression} (float) – the wheel compression  
• \textbf{wheelIndex} (integer) – the wheel index

setSuspensionDamping\(\text{damping, wheelIndex}\)  
Set the specified wheel’s damping  

Parameters  
• \textbf{damping} (float) – the wheel damping  
• \textbf{wheelIndex} (integer) – the wheel index

setSuspensionStiffness\(\text{stiffness, wheelIndex}\)  
Set the specified wheel’s stiffness  

Parameters  
• \textbf{stiffness} (float) – the wheel stiffness  
• \textbf{wheelIndex} (integer) – the wheel index

setTyreFriction\(\text{friction, wheelIndex}\)  
Set the specified wheel’s tyre friction  

Parameters
• friction \((float)\) – the tyre friction
• wheelIndex \((integer)\) – the wheel index

\textbf{rayMask}
Set ray cast mask.
\begin{itemize}
  \item Type bitfield
\end{itemize}

\textbf{KX_VertexProxy(SCA_IObject)}

base class — \textit{SCA\_IObject}

class \textbf{KX\_VertexProxy}(SCA\_IObject)

A vertex holds position, UV, color and normal information.

Note: The physics simulation is NOT currently updated - physics will not respond to changes in the vertex position.

\textbf{XYZ}
The position of the vertex.
\begin{itemize}
  \item Type \textit{Vector}\((x, y, z)\)
\end{itemize}

\textbf{UV}
The texture coordinates of the vertex.
\begin{itemize}
  \item Type \textit{Vector}\((u, v)\)
\end{itemize}

\textbf{uvs}
The texture coordinates list of the vertex.
\begin{itemize}
  \item Type list of \textit{Vector}\((u, v)\)
\end{itemize}

\textbf{normal}
The normal of the vertex.
\begin{itemize}
  \item Type \textit{Vector}\((nx, ny, nz)\)
\end{itemize}

\textbf{color}
The color of the vertex.
\begin{itemize}
  \item Type \textit{Vector}\((r, g, b, a)\)
  \item Black = \([0.0, 0.0, 0.0, 1.0]\), White = \([1.0, 1.0, 1.0, 1.0]\)
\end{itemize}

\textbf{colors}
The color list of the vertex.
\begin{itemize}
  \item Type list of \textit{Vector}\((r, g, b, a)\)
\end{itemize}

\textbf{x}
The x coordinate of the vertex.
\begin{itemize}
  \item Type float
\end{itemize}

\textbf{y}
The y coordinate of the vertex.
\begin{itemize}
  \item Type float
\end{itemize}

\textbf{z}
The z coordinate of the vertex.
\begin{itemize}
  \item Type float
\end{itemize}
The u texture coordinate of the vertex.

Type float

The v texture coordinate of the vertex.

Type float

The second u texture coordinate of the vertex.

Type float

The second v texture coordinate of the vertex.

Type float

The red component of the vertex color. 0.0 <= r <= 1.0.

Type float

The green component of the vertex color. 0.0 <= g <= 1.0.

Type float

The blue component of the vertex color. 0.0 <= b <= 1.0.

Type float

The alpha component of the vertex color. 0.0 <= a <= 1.0.

Type float

getXYZ ()

Gets the position of this vertex.

Returns this vertex's position in local coordinates.

Return type Vector((x, y, z))

setXYZ (pos)

Sets the position of this vertex.

Type Vector((x, y, z))

Parameters pos – the new position for this vertex in local coordinates.

getUV ()

Gets the UV (texture) coordinates of this vertex.

Returns this vertex's UV (texture) coordinates.

Return type Vector((u, v))

setUV (uv)

Sets the UV (texture) coordinates of this vertex.

Type Vector((u, v))
getUV2()

Gets the 2nd UV (texture) coordinates of this vertex.

**Returns**
this vertexes UV (texture) coordinates.

**Return type**
*Vector((u, v))*

setUV2(*uv*, *unit*)

Sets the 2nd UV (texture) coordinates of this vertex.

**Type**
*Vector((u, v))*

**Parameters**

- *unit* – optional argument, FLAT==1, SECOND_UV==2, defaults to SECOND_UV
- *unit* – integer

getRGBA()

Gets the color of this vertex.

The color is represented as four bytes packed into an integer value. The color is packed as RGBA.

Since Python offers no way to get each byte without shifting, you must use the struct module to access color in an machine independent way.

Because of this, it is suggested you use the r, g, b and a attributes or the color attribute instead.

```python
import struct;
col = struct.unpack('4B', struct.pack('I', v.getRGBA()))
# col = (r, g, b, a)
# black = ( 0, 0, 0, 255)
# white = (255, 255, 255, 255)
```

**Returns**
packed color. 4 byte integer with one byte per color channel in RGBA format.

**Return type**
integer

setRGBA(*col*)

Sets the color of this vertex.

See getRGBA() for the format of col, and its relevant problems. Use the r, g, b and a attributes or the color attribute instead.

setRGBA() also accepts a four component list as argument col. The list represents the color as \([r, g, b, a]\) with black = \([0.0, 0.0, 0.0, 1.0]\) and white = \([1.0, 1.0, 1.0, 1.0]\)

```python
v.setRGBA(0xff0000ff) # Red
v.setRGBA(0xff00ff00) # Green on little endian, transparent purple on big endian
v.setRGBA([1.0, 0.0, 0.0, 1.0]) # Red
v.setRGBA([0.0, 1.0, 0.0, 1.0]) # Green on all platforms.
```

**Parameters**
*col* (*integer or list \([r, g, b, a]\)*) – the new color of this vertex in packed RGBA format.

getNormal()

Gets the normal vector of this vertex.

**Returns**
normalized normal vector.

**Return type**
*Vector((nx, ny, nz))*
**setNormal** *(normal)*
Sets the normal vector of this vertex.

*Type*  sequence of floats [r, g, b]

*Parameters* **normal** – the new normal of this vertex.

---

**KX_VisibilityActuator(SCA_IActuator)**

*base class* — *SCA_IActuator*

*class KX_VisibilityActuator(SCA_IActuator)*
Visibility Actuator.

  **visibility**
whether the actuator makes its parent object visible or invisible.

  *Type*  boolean

  **useOcclusion**
whether the actuator makes its parent object an occluder or not.

  *Type*  boolean

  **useRecursion**
whether the visibility/occlusion should be propagated to all children of the object.

  *Type*  boolean

---

**KX_WorldInfo(EXP_PyObjectPlus)**

*base class* — *EXP_PyObjectPlus*

*class KX_WorldInfo(EXP_PyObjectPlus)*
A world object.

```python
# Set the mist color to red.
import bge

sce = bge.logic.getCurrentScene()

sce.world.mistColor = [1.0, 0.0, 0.0]
```

---

**KX_MIST_QUADRATIC**
Type of quadratic attenuation used to fade mist.

**KX_MIST_LINEAR**
Type of linear attenuation used to fade mist.

**KX_MIST_INV_QUADRATIC**
Type of inverse quadratic attenuation used to fade mist.

**mistEnable**
Return the state of the mist.

  *Type*  bool

**mistStart**
The mist start point.

  *Type*  float

---

**8.1. Game Types (bge.types)**
**mistDistance**
The mist distance from the start point to reach 100% mist.

Type float

**mistIntensity**
The mist intensity.

Type float

**mistType**
The type of mist - must be KX_MIST_QUADRATIC, KX_MIST_LINEAR or KX_MIST_INV_QUADRATIC

**mistColor**
The color of the mist. Black = [0.0, 0.0, 0.0], White = [1.0, 1.0, 1.0]. Mist and background color should always set to the same color.

Type `mathutils.Color`

**horizonColor**
The horizon color. Black = [0.0, 0.0, 0.0, 1.0], White = [1.0, 1.0, 1.0, 1.0]. Mist and horizon color should always be set to the same color.

Type `mathutils.Vector`

**zenithColor**
The zenith color. Black = [0.0, 0.0, 0.0, 1.0], White = [1.0, 1.0, 1.0, 1.0].

Type `mathutils.Vector`

**ambientColor**
The color of the ambient light. Black = [0.0, 0.0, 0.0], White = [1.0, 1.0, 1.0].

Type `mathutils.Color`

**exposure**
Amount of exponential color correction for light.

Type float between 0.0 and 1.0 inclusive

**range**
The color range that will be mapped to 0 - 1.

Type float between 0.2 and 5.0 inclusive

**envLightEnergy**
The environment light energy.

Type float from 0.0 to infinite

**envLightEnabled**
Returns True if Environment Lighting is enabled. Else returns False

Type bool (read only)

**envLightColor**
White: returns 0 SkyColor: returns 1 SkyTexture: returns 2

Type int (read only)

---

**SCA_2DFilterActuator(SCA_IActuator)**

base class — `SCA_IActuator`
class SCA_2DFilterActuator (SCA_IActuator)
Create, enable and disable 2D filters.

The following properties don’t have an immediate effect. You must active the actuator to get the result. The actuator is not persistent: it automatically stops itself after setting up the filter but the filter remains active. To stop a filter you must activate the actuator with ‘type’ set to RAS_2DFILTER_DISABLED or RAS_2DFILTER_NOFILTER.

shaderText
shader source code for custom shader.

Type string

disableMotionBlur
action on motion blur: 0=enable, 1=disable.

Type integer

mode
Type of 2D filter, use one of these constants.

Type integer

passNumber
order number of filter in the stack of 2D filters. Filters are executed in increasing order of passNb.

Only be one filter can be defined per passNb.

Type integer (0-100)

value
argument for motion blur filter.

Type float (0.0-100.0)

SCA_ANDController (SCA_IController)
base class — SCA_IController

class SCA_ANDController (SCA_IController)
An AND controller activates only when all linked sensors are activated.

There are no special python methods for this controller.

SCA_ActuatorSensor (SCA_ISensor)
base class — SCA_ISensor

class SCA_ActuatorSensor (SCA_ISensor)
Actuator sensor detect change in actuator state of the parent object. It generates a positive pulse if the corresponding actuator is activated and a negative pulse if the actuator is deactivated.

actuator
the name of the actuator that the sensor is monitoring.

Type string
SCA_AlwaysSensor(SCA_ISensor)

base class — SCA_ISensor

class SCA_AlwaysSensor (SCA_ISensor)
This sensor is always activated.

SCA_DelaySensor(SCA_ISensor)

base class — SCA_ISensor

class SCA_DelaySensor (SCA_ISensor)
The Delay sensor generates positive and negative triggers at precise time, expressed in number of frames. The delay parameter defines the length of the initial OFF period. A positive trigger is generated at the end of this period.

The duration parameter defines the length of the ON period following the OFF period. There is a negative trigger at the end of the ON period. If duration is 0, the sensor stays ON and there is no negative trigger.

The sensor runs the OFF-ON cycle once unless the repeat option is set: the OFF-ON cycle repeats indefinitely (or the OFF cycle if duration is 0).

Use SCA_ISensor.reset at any time to restart sensor.

delay
length of the initial OFF period as number of frame, 0 for immediate trigger.

Type integer.

duration
length of the ON period in number of frame after the initial OFF period.

If duration is greater than 0, a negative trigger is sent at the end of the ON pulse.

Type integer

repeat
1 if the OFF-ON cycle should be repeated indefinitely, 0 if it should run once.

Type integer

SCA_IActuator(SCAILogicBrick)

base class — SCA_ILogicBrick

class SCA_IActuator (SCA_ILogicBrick)
Base class for all actuator logic bricks.

SCA_IController(SCAILogicBrick)

base class — SCA_ILogicBrick

class SCA_IController (SCA_ILogicBrick)
Base class for all controller logic bricks.

state
The controllers state bitmask. This can be used with the GameObject’s state to test if the controller is active.

Type int bitmask
**sensors**
A list of sensors linked to this controller.

*Type* sequence supporting index/string lookups and iteration.

---

**Note:** The sensors are not necessarily owned by the same object.

---

**Note:** When objects are instanced in dupligroups links may be lost from objects outside the dupligroup.

**actuators**
A list of actuators linked to this controller.

*Type* sequence supporting index/string lookups and iteration.

---

**Note:** The sensors are not necessarily owned by the same object.

---

**Note:** When objects are instanced in dupligroups links may be lost from objects outside the dupligroup.

**useHighPriority**
When set the controller executes always before all other controllers that dont have this set.

*Type* boolen

---

**Note:** Order of execution between high priority controllers is not guaranteed.

---

### SCA_ILogicBrick( EXP_Value )

base class — *EXP_Value*

**class SCA_ILogicBrick ( EXP_Value )**
Base class of all *Sensors, Controllers and Actuators*.

**executePriority**
This determines the order controllers are evaluated, and actuators are activated (lower priority is executed first).

*Type* int

**owner**
The game object this logic brick is attached to (read-only).

*Type* *KX_GameObject* or None in exceptional cases.

**name**
The name of this logic brick (read-only).

*Type* str

---

### SCA_IObject( EXP_Value )

base class — *EXP_Value*

---

8.1. Game Types (bge.types)
class SCA_IObject (EXP_Value)
    This class has no python functions

SCA_ISensor(SCA_ILogicBrick)

base class — SCA_ILogicBrick
class SCA_ISensor (SCA_ILogicBrick)
    Base class of all Sensors.
    usePosPulseMode
        Flag to turn positive pulse mode on and off.
        Type  bool
    useNegPulseMode
        Flag to turn negative pulse mode on and off.
        Type  bool
    frequency
        The frequency for pulse mode sensors.

        Warning: Deprecated, use SCA_ISensor.skippedTicks instead.

        Type  int
    skippedTicks
        Number of logic ticks skipped between 2 active pulses.
        Type  int
    level
        Level Option whether to detect level or edge transition when entering a state. It makes a difference only in case of logic state transition (state actuator). A level detector will immediately generate a pulse, negative or positive depending on the sensor condition, as soon as the state is activated. A edge detector will wait for a state change before generating a pulse. note: mutually exclusive with tap, enabling will disable tap.
        Type  bool
    tap
        When enabled only sensors that are just activated will send a positive event, after this they will be detected as negative by the controllers. This will make a key thats held act as if its only tapped for an instant. note: mutually exclusive with level, enabling will disable level.
        Type  boolean
    invert
        Flag to set if this sensor activates on positive or negative events.
        Type  boolean
    triggered
        True if this sensor brick is in a positive state. (read-only).
        Type  boolean
    positive
        True if this sensor brick is in a positive state. (read-only).
**Type** boolean

**pos_ticks**
The number of ticks since the last positive pulse (read-only).
**Type** int

**neg_ticks**
The number of ticks since the last negative pulse (read-only).
**Type** int

**status**
The status of the sensor (read-only): can be one of *these constants*.
**Type** int

**Note:** This convenient attribute combines the values of triggered and positive attributes.

**reset()**
Reset sensor internal state, effect depends on the type of sensor and settings.
The sensor is put in its initial state as if it was just activated.

**SCA_InputEvent(EXP_PyObjectPlus)**

*base class — EXP_PyObjectPlus*

**class SCA_InputEvent (EXP_PyObjectPlus)**
Events for a keyboard or mouse input.

**status**
A list of existing status of the input from the last frame. Can contain `bge.logic.KX_INPUT_NONE` and `bge.logic.KX_INPUT_ACTIVE`. The list always contains one value. The first value of the list is the last value of the list in the last frame. (read-only)
**Type** list of integer.

**queue**
A list of existing events of the input from the last frame. Can contain `bge.logic.KX_INPUT_JUST_ACTIVATED` and `bge.logic.KX_INPUT_JUST_RELEASED`. The list can be empty. (read-only)
**Type** list of integer.

**values**
A list of existing value of the input from the last frame. For keyboard it contains 1 or 0 and for mouse the coordinate of the mouse or the movement of the wheel mouse. The list contains always one value, the size of the list is the same than `queue` + 1 only for keyboard inputs. The first value of the list is the last value of the list in the last frame. (read-only)

Example to get the non-normalized mouse coordinates:

```python
import bge

x = bge.logic.mouse.inputs[bge.events.MOUSEX].values[-1]
y = bge.logic.mouse.inputs[bge.events.MOUSEY].values[-1]

print("Mouse non-normalized position: x: {0}, y: {1}".format(x, y))
```

8.1. Game Types (bge.types)
Type list of integer.

inactive
True if the input was inactive from the last frame.
Type boolean

active
True if the input was active from the last frame.
Type boolean

activated
True if the input was activated from the last frame.
Type boolean

released
True if the input was released from the last frame.
Type boolean

type
The type of the input. One of these constants
Type integer

SCA_JoystickSensor(SCA_ISensor)

base class — SCA_ISensor

class SCA_JoystickSensor(SCA_ISensor)
This sensor detects player joystick events.

axisValues
The state of the joysticks axis as a list of values numAxis long. (read-only).
Type list of ints.

Each specifying the value of an axis between -32767 and 32767 depending on how far the axis is pushed, 0 for nothing. The first 2 values are used by most joysticks and gamepads for directional control. 3rd and 4th values are only on some joysticks and can be used for arbitrary controls.

• left:[-32767, 0, …]
• right:[32767, 0, …]
• up:[0, -32767, …]
• down:[0, 32767, …]

axisSingle
like axisValues but returns a single axis value that is set by the sensor. (read-only).
Type integer

Note: Only use this for “Single Axis” type sensors otherwise it will raise an error.

hatValues (Deprecated. Use :data:button instead)
The state of the joysticks hats as a list of values numHats long. (read-only).
Type list of ints
Each specifying the direction of the hat from 1 to 12, 0 when inactive.
Hat directions are as follows...
- 0: None
- 1: Up
- 2: Right
- 4: Down
- 8: Left
- 3: Up - Right
- 6: Down - Right
- 12: Down - Left
- 9: Up - Left

hatSingle (Deprecated. Use :data:button instead)
Like hatValues but returns a single hat direction value that is set by the sensor. (read-only).
Type integer

numAxis
The number of axes for the joystick at this index. (read-only).
Type integer

numButtons
The number of buttons for the joystick at this index. (read-only).
Type integer

numHats (Deprecated. Use :data:numButtons instead)
The number of hats for the joystick at this index. (read-only).
Type integer

connected
True if a joystick is connected at this joystick's index. (read-only).
Type boolean

index
The joystick index to use (from 0 to 7). The first joystick is always 0.
Type integer

threshold
Axis threshold. Joystick axis motion below this threshold won't trigger an event. Use values between (0 and 32767), lower values are more sensitive.
Type integer

button
The button index the sensor reacts to (first button = 0). When the “All Events” toggle is set, this option has no effect.
Type integer

axis
The axis the sensor reacts to, as a list of two values [axisIndex, axisDirection]
• axisIndex: the axis index to use when detecting axis movement, 1=primary directional control, 2=secondary directional control.
• axisDirection: 0=right, 1=up, 2=left, 3=down.

    Type [integer, integer]

**hat** *(Deprecated. Use :data:button instead)*

The hat the sensor reacts to, as a list of two values: [hatIndex, hatDirection]

• hatIndex: the hat index to use when detecting hat movement, 1=primary hat, 2=secondary hat (4 max).
• hatDirection: 1-12.

    Type [integer, integer]

**getButtonActiveList()**

    Returns A list containing the indicies of the currently pressed buttons.
    Return type list

**getButtonStatus(buttonIndex)**

    Parameters buttonIndex (integer) – the button index, 0=first button
    Returns The current pressed state of the specified button.
    Return type boolean

**SCA_KeyboardSensor(SCA_ISensor)**

base class — *SCA_ISensor*

**class SCA_KeyboardSensor(SCA_ISensor)**

A keyboard sensor detects player key presses.

See module `bge.events` for keycode values.

**key**

The key code this sensor is looking for.

    Type keycode from `bge.events` module

**hold1**

The key code for the first modifier this sensor is looking for.

    Type keycode from `bge.events` module

**hold2**

The key code for the second modifier this sensor is looking for.

    Type keycode from `bge.events` module

**toggleProperty**

The name of the property that indicates whether or not to log keystrokes as a string.

    Type string

**targetProperty**

The name of the property that receives keystrokes in case a string is logged.

    Type string
**useAllKeys**
Flag to determine whether or not to accept all keys.

  **Type** boolean

**inputs**
A list of pressed input keys that have either been pressed, or just released, or are active this frame. (read-only).

  **Type** dictionary {**keycode**: SCA_InputEvent, ...}

**events**
a list of pressed keys that have either been pressed, or just released, or are active this frame. (read-only).

Deprecated since version use: **inputs**

  **Type** list [**[keycode, status]**, ...]

**getKeyStatus**(keycode)
Get the status of a key.

  **Parameters** keycode (**integer**) – The code that represents the key you want to get the state of, use one of **these constants**

  **Returns** The state of the given key, can be one of **these constants**

  **Return type** int

**SCA_MouseSensor**(SCA_ISensor)

base class — **SCA_ISensor**

**class SCA_MouseSensor**(SCA_ISensor)
Mouse Sensor logic brick.

**position**
current [x, y] coordinates of the mouse, in frame coordinates (pixels).

  **Type** [integer, integer]

**mode**
sensor mode.

  **Type** integer

  • KX_MOUSESENSORMODE_LEFTBUTTON(1)
  • KX_MOUSESENSORMODE_MIDDLEBUTTON(2)
  • KX_MOUSESENSORMODE_RIGHTBUTTON(3)
  • KX_MOUSESENSORMODE_WHEELUP(4)
  • KX_MOUSESENSORMODE_WHEELEDOWN(5)
  • KX_MOUSESENSORMODE_MOVEMENT(6)

**getButtonStatus**(button)
Get the mouse button status.

  **Parameters** button (**int**) – The code that represents the key you want to get the state of, use one of **these constants**

  **Returns** The state of the given key, can be one of **these constants**

  **Return type** int

8.1. Game Types (bge.types)
SCA_NANDController(SCA_IController)

base class — SCA_IController

class SCA_NANDController (SCA_IController)
   An NAND controller activates when all linked sensors are not active.
   There are no special python methods for this controller.

SCA_NORController(SCA_IController)

base class — SCA_IController

class SCA_NORController (SCA_IController)
   An NOR controller activates only when all linked sensors are de-activated.
   There are no special python methods for this controller.

SCA_ORController(SCA_IController)

base class — SCA_IController

class SCA_ORController (SCA_IController)
   An OR controller activates when any connected sensor activates.
   There are no special python methods for this controller.

SCA_PropertyActuator(SCA_IActuator)

base class — SCA_IActuator

class SCA_PropertyActuator (SCA_IActuator)
   Property Actuator
      propName
         the property on which to operate.
            Type  string
      value
         the value with which the actuator operates.
            Type  string
      mode
         TODO - add constants to game logic dict!.
            Type  integer

SCA_PropertySensor(SCA_ISensor)

base class — SCA_ISensor

class SCA_PropertySensor (SCA_ISensor)
   Activates when the game object property matches.
      mode
         Type of check on the property. Can be one of these constants
**Type** integer.

**propName**
the property the sensor operates.

**Type** string

**value**
the value with which the sensor compares to the value of the property.

**Type** string

**min**
the minimum value of the range used to evaluate the property when in interval mode.

**Type** string

**max**
the maximum value of the range used to evaluate the property when in interval mode.

**Type** string

**SCA_PythonController(SCA_IController)**

base class — **SCA_IController**

**class SCA_PythonController(SCA_IController)**
A Python controller uses a Python script to activate it’s actuators, based on it’s sensors.

**owner**
The object the controller is attached to.

**Type** KX_GameObject

**script**
The value of this variable depends on the execution method.

- When ‘Script’ execution mode is set this value contains the entire python script as a single string (not the script name as you might expect) which can be modified to run different scripts.

- When ‘Module’ execution mode is set this value will contain a single line string - module name and function “module.func” or “package.module.func” where the module names are python textblocks or external scripts.

**Type** string

**Note**: Once this is set the script name given for warnings will remain unchanged.

**mode**
the execution mode for this controller (read-only).

- Script: 0, Execute the **script** as a python code.

- Module: 1, Execute the **script** as a module and function.

**Type** integer

**activate**(actuator)
Activates an actuator attached to this controller.
Parameters `actuator` *(actuator or the actuator name as a string)* – The actuator to operate on.

`deactivate` *(actuator)*
Deactivates an actuator attached to this controller.

Parameters `actuator` *(actuator or the actuator name as a string)* – The actuator to operate on.

### SCA_PythonJoystick(\(\text{EXP}_\text{PyObjectPlus}\))

Base class — \(\text{EXP}_\text{PyObjectPlus}\)

**class SCA_PythonJoystick(\(\text{EXP}_\text{PyObjectPlus}\))**

A Python interface to a joystick.

- **name**
  The name assigned to the joystick by the operating system. (read-only)
  
  **Type** string

- **activeButtons**
  A list of active button values. (read-only)
  
  **Type** list

- **axisValues**
  The state of the joysticks axis as a list of values \(\text{numAxis}\) long. (read-only).
  
  **Type** list of ints.

  Each specifying the value of an axis between -1.0 and 1.0 depending on how far the axis is pushed, 0 for nothing. The first 2 values are used by most joysticks and gamepads for directional control. 3rd and 4th values are only on some joysticks and can be used for arbitrary controls.

  - left: [-1.0, 0.0, ...]
  - right: [1.0, 0.0, ...]
  - up: [0.0, -1.0, ...]
  - down: [0.0, 1.0, ...]

- **hatValues** *(Deprecated. Use :data:`activeButtons` instead)*
  The state of the joysticks hats as a list of values \(\text{numHats}\) long. (read-only).
  
  **Type** list of ints

  Each specifying the direction of the hat from 1 to 12, 0 when inactive.

  Hat directions are as follows...

  - 0: None
  - 1: Up
  - 2: Right
  - 4: Down
  - 8: Left
  - 3: Up - Right
  - 6: Down - Right
numAxis
  The number of axes for the joystick at this index. (read-only).
  Type integer

numButtons
  The number of buttons for the joystick at this index. (read-only).
  Type integer

numHats (Deprecated. Use :data:'numButtons' instead)
  The number of hats for the joystick at this index. (read-only).
  Type integer

SCA_PythonKeyboard(EXP_PyObjectPlus)

class SCA_PythonKeyboard(EXP_PyObjectPlus)
  The current keyboard.

inputs
  A dictionary containing the input of each keyboard key. (read-only).
  Type dictionary {keycode: SCA_InputEvent, ...}

events
  A dictionary containing the status of each keyboard event or key. (read-only).
  Deprecated since version use: inputs
  Type dictionary {keycode: status, ...}

activeInputs
  A dictionary containing the input of only the active keyboard keys. (read-only).
  Type dictionary {keycode: SCA_InputEvent, ...}

active_events
  A dictionary containing the status of only the active keyboard events or keys. (read-only).
  Deprecated since version use: activeInputs
  Type dictionary {keycode: status, ...}

text
  The typed unicode text from the last frame.
  Type string

getClipboard()
  Gets the clipboard text.
  Return type string

setClipboard(text)
  Sets the clipboard text.
  Parameters text (string) – New clipboard text
SCA_PythonMouse(EXP_PyObjectPlus)

base class — EXP_PyObjectPlus

class SCA_PythonMouse (EXP_PyObjectPlus)
The current mouse.

inputs
A dictionary containing the input of each mouse event. (read-only).

Type dictionary {keycode: SCA_InputEvent, ...}

events
a dictionary containing the status of each mouse event. (read-only).

Depreciated since version use: inputs

Type dictionary {keycode: status, ...}

activeInputs
A dictionary containing the input of only the active mouse events. (read-only).

Type dictionary {keycode: SCA_InputEvent, ...}

active_events
a dictionary containing the status of only the active mouse events. (read-only).

Depreciated since version use: activeInputs

Type dictionary {keycode: status, ...}

position
The normalized x and y position of the mouse cursor.

Type tuple (x, y)

visible
The visibility of the mouse cursor.

Type boolean

SCA_RandomActuator(SCA_IActuator)

base class — SCA_IActuator

class SCA_RandomActuator (SCA_IActuator)
Random Actuator

seed
Seed of the random number generator.

Type integer.

Equal seeds produce equal series. If the seed is 0, the generator will produce the same value on every call.

para1
the first parameter of the active distribution.

Type float, read-only.

Refer to the documentation of the generator types for the meaning of this value.

para2
the second parameter of the active distribution.
Type  float, read-only

Refer to the documentation of the generator types for the meaning of this value.

distribution
Distribution type. (read-only). Can be one of these constants

Type  integer

propName
the name of the property to set with the random value.

Type  string
If the generator and property types do not match, the assignment is ignored.

setBoolConst (value)
Sets this generator to produce a constant boolean value.

Parameters value (boolean) – The value to return.

setBoolUniform()
Sets this generator to produce a uniform boolean distribution.

The generator will generate True or False with 50% chance.

setBoolBernouilli (value)
Sets this generator to produce a Bernouilli distribution.

Parameters value (float) – Specifies the proportion of False values to produce.

• 0.0: Always generate True
• 1.0: Always generate False

setIntConst (value)
Sets this generator to always produce the given value.

Parameters value (integer) – the value this generator produces.

setIntUniform (lower_bound, upper_bound)
Sets this generator to produce a random value between the given lower and upper bounds (inclusive).

setIntPoisson (value)
Generate a Poisson-distributed number.

This performs a series of Bernouilli tests with parameter value. It returns the number of tries needed to achieve succes.

setFloatConst (value)
Always generate the given value.

setFloatUniform (lower_bound, upper_bound)
Generates a random float between lower_bound and upper_bound with a uniform distribution.

setFloatNormal (mean, standard_deviation)
Generates a random float from the given normal distribution.

Parameters
• **mean** (*float*) – The mean (average) value of the generated numbers
• **standard_deviation** (*float*) – The standard deviation of the generated numbers.

**setFloatNegativeExponential** (*half_life*)
Generate negative-exponentially distributed numbers.

The half-life ‘time’ is characterized by half_life.

---

**SCA_RandomSensor** (*SCA_ISensor*)

base class — *SCA_ISensor*

class **SCA_RandomSensor** (*SCA_ISensor*)

This sensor activates randomly.

  - **lastDraw**
    The seed of the random number generator.
    
    Type  integer
  
  - **seed**
    The seed of the random number generator.
    
    Type  integer

**SCA_VibrationActuator** (*SCA_IActuator*)

base class — *SCA_IActuator*

class **SCA_VibrationActuator** (*SCA_IActuator*)

Vibration Actuator.

  - **joyindex**
    Joystick index.
    
    Type  integer (0 to 7)

  - **strengthLeft**
    Strength of the Low frequency joystick’s motor (placed at left position usually).
    
    Type  float (0.0 to 1.0)

  - **strengthRight**
    Strength of the High frequency joystick’s motor (placed at right position usually).
    
    Type  float (0.0 to 1.0)

  - **duration**
    Duration of the vibration in milliseconds.
    
    Type  integer (0 to infinite)

  - **isVibrating**
    Check status of joystick vibration
    
    Type  bool (true vibrating and false stopped)

  - **hasVibration**
    Check if the joystick supports vibration
    
    Type  bool (true supported and false not supported)
\begin{verbatim}
startVibration()
    Starts the vibration.
    Returns  None

stopVibration()
    Stops the vibration.
    Returns  None
\end{verbatim}

SCA_XNORController(SCA_IController)

base class — SCA_IController
class SCA_XNORController(SCA_IController)
    An XNOR controller activates when all linked sensors are the same (activated or inative).
    There are no special python methods for this controller.

SCA_XORController(SCA_IController)

base class — SCA_IController
class SCA_XORController(SCA_IController)
    An XOR controller activates when there is the input is mixed, but not when all are on or off.
    There are no special python methods for this controller.

8.2 Game Logic (bge.logic)

8.2.1 Introduction

Module to access logic functions, imported automatically into the python controllers namespace.

\begin{verbatim}
# To get the controller thats running this python script:
cont = bge.logic.getCurrentController()  # bge.logic is automatically imported

# To get the game object this controller is on:
obj = cont.owner

KX_GameObject, KX_Camera or KX_LightObject methods are available depending on the type of object. See object types for more reference.

# To get a sensor linked to this controller.
# "sensorname" is the name of the sensor as defined in the Blender interface.
# +---------------------+ +--------+
# | Sensor "sensorname" +--+ Python +
# +---------------------+ +--------+
sens = cont.sensors["sensorname"]

# To get a sequence of all sensors:
sensors = co.sensors
\end{verbatim}

See the available sensors reference for available attributes.

You can also access actuators linked to the controller
To get an actuator attached to the controller:

```python
actuator = co.actuators['actuatorname']
```

Activate an actuator:

```python
controller.activate(actuator)
```

See the available actuators reference for available attributes.

Most logic brick’s methods are accessors for the properties available in the logic buttons. Consult the logic bricks documentation for more information on how each logic brick works.

There are also methods to access the current KX_Scene:

```python
scene = bge.logic.getCurrentScene()
```

```python
cam = scene.active_camera
```

Matrices as used by the game engine are row major

```python
matrix[row][col] = float
```

KX_Camera has some examples using matrices.

### 8.2.2 Variables

bge.logic.globalDict

A dictionary that is saved between loading blend files so you can use it to store inventory and other variables you want to store between scenes and blend files. It can also be written to a file and loaded later on with the game load/save actuators.

**Note:** only python built in types such as int/string/bool/float/tuples/lists can be saved. GameObjects, Actuators etc will not work as expected.

bge.logic.keyboard

The current keyboard wrapped in an SCA_PythonKeyboard object.

bge.logic.mouse

The current mouse wrapped in an SCA_PythonMouse object.

bge.logic.joysticks

A list of attached SCA_PythonJoystick. The list size is the maximum number of supported joysticks. If no joystick is available for a given slot, the slot is set to None.

### 8.2.3 General functions

bge.logic.getCurrentController()

Gets the Python controller associated with this Python script.

**Return type** SCA_PythonController

bge.logic.getCurrentScene()

Gets the current Scene.
bge.logic.getSceneList()

Gets a list of the current scenes loaded in the game engine.

Return type list of KX_Scene

Note: Scenes in your blend file that have not been converted won't be in this list. This list will only contain scenes such as overlays scenes.

bge.logic.getInactiveSceneNames()

Gets a list of the scene's names not loaded in the game engine.

Return type list of string

bge.logic.loadGlobalDict()

Loads bge.logic.globalDict from a file.

bge.logic.saveGlobalDict()

Saves bge.logic.globalDict to a file.

bge.logic.startGame(blend)

Loads the blend file.

Parameters

- **blend (string)** – The name of the blend file

bge.logic.endGame()

Ends the current game.

bge.logic.restartGame()

Restarts the current game by reloading the .blend file (the last saved version, not what is currently running).

bge.logic.LibLoad(blend, type, data, load_actions=False, verbose=False, load_scripts=True, async=False, scene=None)

Converts all of the datablocks of the given type from the given blend.

Parameters

- **blend (string)** – The path to the blend file (or the name to use for the library if data is supplied)
- **type (string)** – The datablock type (currently only “Action”, “Mesh” and “Scene” are supported)
- **data (bytes)** – Binary data from a blend file (optional)
- **load_actions (bool)** – Search for and load all actions in a given Scene and not just the “active” actions (Scene type only)
- **verbose (bool)** – Whether or not to print debugging information (e.g., “SceneName: Scene”)
- **load_scripts (bool)** – Whether or not to load text datablocks as well (can be disabled for some extra security)
- **async (bool)** – Whether or not to do the loading asynchronously (in another thread). Only the “Scene” type is currently supported for this feature.
- **scene (KX_Scene or string)** – Scene to merge loaded data to, if None use the current scene.

Return type KX_LibLoadStatus
Note: Asynchronously loaded libraries will not be available immediately after LibLoad() returns. Use the returned KX_LibLoadStatus to figure out when the libraries are ready.

`bge.logic.LibNew(name, type, data)`
Uses existing datablock data and loads in as a new library.

**Parameters**

- **name** (*string*) – A unique library name used for removal later
- **type** (*string*) – The datablock type (currently only “Mesh” is supported)
- **data** (*list of strings*) – A list of names of the datablocks to load

`bge.logic.LibFree(name)`
Frees a library, removing all objects and meshes from the currently active scenes.

**Parameters**

- **name** (*string*) – The name of the library to free (the name used in LibNew)

`bge.logic.LibList()`
Returns a list of currently loaded libraries.

**Return type**  list [str]

`bge.logic.addScene(name, overlay=1)`
Loads a scene into the game engine.

**Note:** This function is not effective immediately, the scene is queued and added on the next logic cycle where it will be available from `getSceneList`

**Parameters**

- **name** (*string*) – The name of the scene
- **overlay** (*integer*) – Overlay or underlay (optional)

`bge.logic.sendMessage(subject, body="", to="", message_from="")`
Sends a message to sensors in any active scene.

**Parameters**

- **subject** (*string*) – The subject of the message
- **body** (*string*) – The body of the message (optional)
- **to** (*string*) – The name of the object to send the message to (optional)
- **message_from** (*string*) – The name of the object that the message is coming from (optional)

`bge.logic.setGravity(gravity)`
Sets the world gravity.

**Parameters**

- **gravity** (*Vector((fx, fy, fz))*) – gravity vector

`bge.logic.getSpectrum()` *(Deprecated)*
Returns a 512 point list from the sound card. This only works if the fmod sound driver is being used.

**Return type**  list [float], len(getSpectrum()) == 512
bge.logic.getMaxLogicFrame()

Gets the maximum number of logic frames per render frame.

**Returns** The maximum number of logic frames per render frame

**Return type** integer

bge.logic.setMaxLogicFrame(maxlogic)

Sets the maximum number of logic frames that are executed per render frame. This does not affect the physic system that still runs at full frame rate.

**Parameters**

- **maxlogic** (integer) – The new maximum number of logic frames per render frame. Valid values: 1..5

bge.logic.getMaxPhysicsFrame()

Gets the maximum number of physics frames per render frame.

**Returns** The maximum number of physics frames per render frame

**Return type** integer

bge.logic.setMaxPhysicsFrame(maxphysics)

Sets the maximum number of physics timestep that are executed per render frame. Higher value allows physics to keep up with realtime even if graphics slows down the game. Physics timestep is fixed and equal to 1/ticrate (see setLogicTicRate) maxphysics/ticrate is the maximum delay of the renderer that physics can compensate.

**Parameters**

- **maxphysics** (integer) – The new maximum number of physics timestep per render frame. Valid values: 1..5.

bge.logic.getLogicTicRate()

Gets the logic update frequency.

**Returns** The logic frequency in Hz

**Return type** float

bge.logic.setLogicTicRate(ticrate)

Sets the logic update frequency.

The logic update frequency is the number of times logic bricks are executed every second. The default is 60 Hz.

**Parameters**

- **ticrate** (float) – The new logic update frequency (in Hz).

bge.logic.getPhysicsTicRate()

Gets the physics update frequency

**Returns** The physics update frequency in Hz

**Return type** float

bge.logic.setPhysicsTicRate(ticrate)

Sets the physics update frequency

The physics update frequency is the number of times the physics system is executed every second. The default is 60 Hz.

**Parameters**

- **ticrate** (float) – The new update frequency (in Hz).

bge.logic.getExitKey()

Gets the key used to exit the game engine

**Returns** The key (defaults to *bge.events.ESCKEY*)

**Return type** int
bge.logic.setExitKey(key)
Sets the key used to exit the game engine

Parameters key (int) – A key constant from bge.events

bge.logic.NextFrame()
Render next frame (if Python has control)

bge.logic.setRender(render)
Sets the global flag that controls the render of the scene. If True, the render is done after the logic frame. If False, the render is skipped and another logic frame starts immediately.

Note: GPU VSync no longer limits the number of frame per second when render is off, but the Use Frame Rate option still regulates the fps. To run as many frames as possible, untick this option (Render Properties, System panel).

Parameters render (bool) – the render flag

bge.logic.getRender()
Get the current value of the global render flag

Returns The flag value

Return type bool

8.2.4 Time related functions

bge.logic.getClockTime()
Get the current BGE render time, in seconds. The BGE render time is the simulation time corresponding to the next scene that will be rendered.

Return type double

bge.logic.getFrameTime()
Get the current BGE frame time, in seconds. The BGE frame time is the simulation time corresponding to the current call of the logic system. Generally speaking, it is what the user is interested in.

Return type double

bge.logic.getRealTime()
Get the number of real (system-clock) seconds elapsed since the beginning of the simulation.

Return type double

bge.logic.getTimeScale()
Get the time multiplier between real-time and simulation time. The default value is 1.0. A value greater than 1.0 means that the simulation is going faster than real-time, a value lower than 1.0 means that the simulation is going slower than real-time.

Return type double

bge.logic.setTimeScale(time_scale)
Set the time multiplier between real-time and simulation time. A value greater than 1.0 means that the simulation is going faster than real-time, a value lower than 1.0 means that the simulation is going slower than real-time. Note that a too large value may lead to some physics instabilities.

Parameters time_scale – The new time multiplier.
bge.logic.getUseExternalClock()
Get if the BGE use the inner BGE clock, or rely or on an external clock. The default is to use the inner BGE clock.

Return type  bool

bge.logic.setUseExternalClock(use_external_clock)
Set if the BGE use the inner BGE clock, or rely or on an external clock. If the user selects the use of an external clock, he should call regularly the setClockTime method.

Parameters use_external_clock – the new setting

bge.logic.setClockTime(new_time)
Set the next value of the simulation clock. It is preferable to use this method from a custom main function in python, as calling it in the logic block can easily lead to a blocked system (if the time does not advance enough to run at least the next logic step).

Parameters new_time – the next value of the BGE clock (in second).

8.2.5 Utility functions

bge.logic.expandPath(path)
Converts a blender internal path into a proper file system path. Use / as directory separator in path You can use ‘//’ at the start of the string to define a relative path; Blender replaces that string by the directory of the current .blend or runtime file to make a full path name. The function also converts the directory separator to the local file system format.

Parameters path(string) – The path string to be converted/expanded.
Returns The converted string
Return type  string

bge.logic.getAverageFrameRate()
Gets the estimated/average framerate for all the active scenes, not only the current scene.

Returns  The estimated average framerate in frames per second
Return type  float

bge.logic.getBlendFileList(path = "//")
Returns a list of blend files in the same directory as the open blend file, or from using the option argument.

Parameters path(string) – Optional directory argument, will be expanded (like expandPath) into the full path.

Returns  A list of filenames, with no directory prefix
Return type  list

bge.logic.getRandomFloat()
Returns a random floating point value in the range [0 - 1]

bge.logic.PrintGLInfo()
Prints GL Extension Info into the console

bge.logic.PrintMemInfo()
Prints engine statistics into the console

bge.logic.getProfileInfo()
Returns a Python dictionary that contains the same information as the on screen profiler. The keys are the profiler
categories and the values are tuples with the first element being time taken (in ms) and the second element being the percentage of total time.

8.2.6 Constants

bge.logic.KX_TRUE
   True value used by some modules.
   Value 1
bge.logic.KX_FALSE
   False value used by some modules.
   Value 2

Sensors

Sensor Status

See SCA_ISensor.status

bge.logic.KX_SENSOR_INACTIVE
   Value 0
bge.logic.KX_SENSOR_JUST_ACTIVATED
   Value 1
bge.logic.KX_SENSOR_ACTIVE
   Value 2
bge.logic.KX_SENSOR_JUST_DEACTIVATED
   Value 3

Armature Sensor

See KX_ArmatureSensor.type

bge.logic.KX_ARMSENSOR_STATE_CHANGED
   Detect that the constraint is changing state (active/inactive).
   Value 0
bge.logic.KX_ARMSENSOR_LIN_ERROR_BELOW
   Detect that the constraint linear error is above a threshold.
   Value 1
bge.logic.KX_ARMSENSOR_LIN_ERROR_ABOVE
   Detect that the constraint linear error is below a threshold.
   Value 2
bge.logic.KX_ARMSENSOR_ROT_ERROR_BELOW
   Detect that the constraint rotation error is above a threshold.
   Value 3
bge.logic.KX_ARMSENSOR_ROT_ERROR_ABOVE
   Detect that the constraint rotation error is below a threshold.
       Value 4

**Movement Sensor**

See `KX_MovementSensor.axis`

bge.logic.KX_MOVEMENT_ALL_AXIS
   Value 6

bge.logic.KX_MOVEMENT_AXIS_NEG_X
   Value 3

bge.logic.KX_MOVEMENT_AXIS_NEG_Y
   Value 4

bge.logic.KX_MOVEMENT_AXIS_NEG_Z
   Value 5

bge.logic.KX_MOVEMENT_AXIS_POS_X
   Value 1

bge.logic.KX_MOVEMENT_AXIS_POS_Y
   Value 0

bge.logic.KX_MOVEMENT_AXIS_POS_Z
   Value 2

**Property Sensor**

See `SCA_PropertySensor.mode`

bge.logic.KX_PROPSensor_EQUAL
   Activate when the property is equal to the sensor value.
       Value 1

bge.logic.KX_PROPSensor_NOTEQUAL
   Activate when the property is not equal to the sensor value.
       Value 2

bge.logic.KX_PROPSensor_INTERVAL
   Activate when the property is between the specified limits.
       Value 3

bge.logic.KX_PROPSensor_CHANGED
   Activate when the property changes.
       Value 4

bge.logic.KX_PROPSensor_EXPRESSION
   Activate when the expression matches.
Value 5
\texttt{bge.logic.KX_PROPSSENSOR_LESSTHAN}
Activate when the property is less than the sensor value.

Value 6
\texttt{bge.logic.KX_PROPSSENSOR_GREATERTHAN}
Activate when the property is greater than the sensor value.

\textbf{Radar Sensor}

See \texttt{KX_RadarSensor.axis}
\texttt{bge.logic.KX_RADAR_AXIS_POS_X}
Value 0
\texttt{bge.logic.KX_RADAR_AXIS_POS_Y}
Value 1
\texttt{bge.logic.KX_RADAR_AXIS_POS_Z}
Value 2
\texttt{bge.logic.KX_RADAR_AXIS_NEG_X}
Value 3
\texttt{bge.logic.KX_RADAR_AXIS_NEG_Y}
Value 4
\texttt{bge.logic.KX_RADAR_AXIS_NEG_Z}
Value 5

\textbf{Ray Sensor}

See \texttt{KX_RaySensor.axis}
\texttt{bge.logic.KX_RAY_AXIS_POS_X}
Value 1
\texttt{bge.logic.KX_RAY_AXIS_POS_Y}
Value 0
\texttt{bge.logic.KX_RAY_AXIS_POS_Z}
Value 2
\texttt{bge.logic.KX_RAY_AXIS_NEG_X}
Value 3
\texttt{bge.logic.KX_RAY_AXIS_NEG_Y}
Value 4
\texttt{bge.logic.KX_RAY_AXIS_NEG_Z}
Value 5

**Actuators**

**Action Actuator**

See `BL_ActionActuator`

```bge.logic.KX_ACTIONACT_PLAY```
Value 0

```bge.logic.KX_ACTIONACT_PINGPONG```
Value 1

```bge.logic.KX_ACTIONACT_FLIPPER```
Value 2

```bge.logic.KX_ACTIONACT_LOOPSTOP```
Value 3

```bge.logic.KX_ACTIONACT_LOOPEND```
Value 4

```bge.logic.KX_ACTIONACT_PROPERTY```
Value 6

**Armature Actuator**

See `BL_ArmatureActuator.type`

```bge.logic.KX_ACT_ARMATURE_RUN```

Just make sure the armature will be updated on the next graphic frame. This is the only persistent mode of the actuator: it executes automatically once per frame until stopped by a controller.

Value 0

```bge.logic.KX_ACT_ARMATURE_ENABLE```

Enable the constraint.

Value 1

```bge.logic.KX_ACT_ARMATURE_DISABLE```

Disable the constraint (runtime constraint values are not updated).

Value 2

```bge.logic.KX_ACT_ARMATURE_SETTARGET```

Change target and subtarget of constraint.

Value 3

```bge.logic.KX_ACT_ARMATURE_SETWEIGHT```

Change weight of constraint (IK only).

Value 4
**bge.logic.KX_ACT_ARMATURE_SETINFLUENCE**
Change influence of constraint.

Value 5

**Constraint Actuator**

See `KX_ConstraintActuator.option`

- Applicable to Distance constraint:

**bge.logic.KX_CONSTRAINTACT_NORMAL**
Activate alignment to surface.

Value 64

**bge.logic.KX_CONSTRAINTACT_DISTANCE**
Activate distance control.

Value 512

**bge.logic.KX_CONSTRAINTACT_LOCAL**
Direction of the ray is along the local axis.

Value 1024

- Applicable to Force field constraint:

**bge.logic.KX_CONSTRAINTACT_DOROTFH**
Force field act on rotation as well.

Value 2048

- Applicable to both:

**bge.logic.KX_CONSTRAINTACT_MATERIAL**
Detect material rather than property.

Value 128

**bge.logic.KX_CONSTRAINTACT_PERMANENT**
No deactivation if ray does not hit target.

Value 256

See `KX_ConstraintActuator.limit`

**bge.logic.KX_CONSTRAINTACT_LOCX**
Limit X coord.

Value 1

**bge.logic.KX_CONSTRAINTACT_LOCY**
Limit Y coord.

Value 2

**bge.logic.KX_CONSTRAINTACT_LOCZ**
Limit Z coord.

Value 3

**bge.logic.KX_CONSTRAINTACT_ROTX**
Limit X rotation.
Value 4
bge.logic.KX_CONSTRAINTACT_ROTY
 Limit Y rotation.

Value 5
bge.logic.KX_CONSTRAINTACT_ROTZ
 Limit Z rotation.

Value 6
bge.logic.KX_CONSTRAINTACT_DIRNX
 Set distance along negative X axis.

Value 10
bge.logic.KX_CONSTRAINTACT_DIRNY
 Set distance along negative Y axis.

Value 11
bge.logic.KX_CONSTRAINTACT_DIRNZ
 Set distance along negative Z axis.

Value 12
bge.logic.KX_CONSTRAINTACT_DIRPX
 Set distance along positive X axis.

Value 7
bge.logic.KX_CONSTRAINTACT_DIRPY
 Set distance along positive Y axis.

Value 8
bge.logic.KX_CONSTRAINTACT_DIRPZ
 Set distance along positive Z axis.

Value 9
bge.logic.KX_CONSTRAINTACT_ORIX
 Set orientation of X axis.

Value 13
bge.logic.KX_CONSTRAINTACT_ORIY
 Set orientation of Y axis.

Value 14
bge.logic.KX_CONSTRAINTACT_ORIZ
 Set orientation of Z axis.

Value 15
bge.logic.KX_CONSTRAINTACT_FHNX
 Set force field along negative X axis.

Value 19
bge.logic.KX_CONSTRAINTACT_FHNY
 Set force field along negative Y axis.
bge.logic.KX_CONSTRAINTACT_FHNZ
    Set force field along negative Z axis.
    Value 21

bge.logic.KX_CONSTRAINTACT_FHPX
    Set force field along positive X axis.
    Value 16

bge.logic.KX_CONSTRAINTACT_FHPY
    Set force field along positive Y axis.
    Value 17

bge.logic.KX_CONSTRAINTACT_FHPZ
    Set force field along positive Z axis.
    Value 18

Dynamic Actuator

See KX_SCA_DynamicActuator

bge.logic.KX_DYN_RESTORE_DYNAMICS
    Value 0

bge.logic.KX_DYN_DISABLE_DYNAMICS
    Value 1

bge.logic.KX_DYN_ENABLE_RIGID_BODY
    Value 2

bge.logic.KX_DYN_DISABLE_RIGID_BODY
    Value 3

bge.logic.KX_DYN_SET_MASS
    Value 4

Game Actuator

See KX_GameActuator

bge.logic.KX_GAME_LOAD
    Value 1

bge.logic.KX_GAME_START
    Value 2

bge.logic.KX_GAME_RESTART
    Value 3

bge.logic.KX_GAME_QUIT
    Value 4

bge.logic.KX_GAME_SAVECFG
Value 5
bge.logic.KX_GAME_LOADCFG
Value 6

Mouse Actuator

See KX_MouseActuator
bge.logic.KX_ACT_MOUSE_OBJECT_AXIS_X
Value 0
bge.logic.KX_ACT_MOUSE_OBJECT_AXIS_Y
Value 1
bge.logic.KX_ACT_MOUSE_OBJECT_AXIS_Z
Value 2

Parent Actuator

See KX_ParentActuator
bge.logic.KX_PARENT_REMOVE
Value 2
bge.logic.KX_PARENT_SET
Value 1

Random Distributions

See SCA_RandomActuator
bge.logic.KX_RANDOMACT_BOOL_CONST
Value 1
bge.logic.KX_RANDOMACT_BOOL_UNIFORM
Value 2
bge.logic.KX_RANDOMACT_BOOL_BERNOUILLI
Value 3
bge.logic.KX_RANDOMACT_INT_CONST
Value 4
bge.logic.KX_RANDOMACT_INT_UNIFORM
Value 5
bge.logic.KX_RANDOMACT_INT_POISSON
Value 6
bge.logic.KX_RANDOMACT_FLOAT_CONST

8.2. Game Logic (bge.logic) 255
Value 7
bge.logic.KX_RANDOMACT_FLOAT_UNIFORM
Value 8
bge.logic.KX_RANDOMACT_FLOAT_NORMAL
Value 9
bge.logic.KX_RANDOMACT_FLOAT_NEGATIVE_EXPONENTIAL
Value 10

**Scene Actuator**

See *KX_SceneActuator*

bge.logic.KX_SCENE_RESTART
Value 1
bge.logic.KX_SCENE_SET_SCENE
Value 2
bge.logic.KX_SCENE_SET_CAMERA
Value 3
bge.logic.KX_SCENE_ADD_FRONT_SCENE
Value 4
bge.logic.KX_SCENE_ADD_BACK_SCENE
Value 5
bge.logic.KX_SCENE_REMOVE_SCENE
Value 6
bge.logic.KX_SCENE_SUSPEND
Value 7
bge.logic.KX_SCENE_RESUME
Value 8

**Sound Actuator**

See *KX_SoundActuator*

bge.logic.KX_SOUNDACT_PLAYSTOP
Value 1
bge.logic.KX_SOUNDACT_PLAYEND
Value 2
bge.logic.KX_SOUNDACT_LOOPSTOP
Value 3
bge.logic.KX_SOUNDACT_LOOPEN
Value 4
bge.logic.KX_SOUNDACT_LOOPBIDIRECTIONAL
Value 5
bge.logic.KX_SOUNDACT_LOOPBIDIRECTIONAL_STOP
Value 6

**Steering Actuator**

See [KX_SteeringActuator.behavior](#)
bge.logic.KX_STEERING_SEEK
Value 1
bge.logic.KX_STEERING_FLEE
Value 2
bge.logic.KX_STEERING_PATHFOLLOWING
Value 3

**TrackTo Actuator**

See [KX_TrackToActuator](#)
bge.logic.KX_TRACK_UPAXIS_POS_X
Value 0
bge.logic.KX_TRACK_UPAXIS_POS_Y
Value 1
bge.logic.KX_TRACK_UPAXIS_POS_Z
Value 2
bge.logic.KX_TRACK_TRAXIS_POS_X
Value 0
bge.logic.KX_TRACK_TRAXIS_POS_Y
Value 1
bge.logic.KX_TRACK_TRAXIS_POS_Z
Value 2
bge.logic.KX_TRACK_TRAXIS_NEG_X
Value 3
bge.logic.KX_TRACK_TRAXIS_NEG_Y
Value 4
bge.logic.KX_TRACK_TRAXIS_NEG_Z
Value 5
Various

2D Filter

See KX_2DFilterActuator.mode

bge.logic.RAS_2DFILTER_BLUR
  Value 2

bge.logic.RAS_2DFILTER_CUSTOMFILTER
  Customer filter, the code code is set via shaderText property.
  Value 12

bge.logic.RAS_2DFFILTER_DILATION
  Value 4

bge.logic.RAS_2DFFILTER_DISABLED
  Disable the filter that is currently active.
  Value -1

bge.logic.RAS_2DFFILTER_ENABLED
  Enable the filter that was previously disabled.
  Value -2

bge.logic.RAS_2DFILTER_EROSION
  Value 5

bge.logic.RAS_2DFFILTER_GRAYSCALE
  Value 9

bge.logic.RAS_2DFFILTER_INVERT
  Value 11

bge.logic.RAS_2DFFILTER_LAPLACIAN
  Value 6

bge.logic.RAS_2DFFILTER_MOTIONBLUR
  Create and enable preset filters.
  Value 1

bge.logic.RAS_2DFFILTER_NOFILTER
  Disable and destroy the filter that is currently active.
  Value 0

bge.logic.RAS_2DFFILTER_PREWITT
  Value 8

bge.logic.RAS_2DFFILTER_SEPIA
  Value 10

bge.logic.RAS_2DFFILTER_SHARPEN
  Value 3

bge.logic.RAS_2DFFILTER SOBEL
Armature Channel

See `BL_ArmatureChannel.rotation_mode`

- **bge.logic.ROT_MODE_QUAT**
  - Use quaternion in rotation attribute to update bone rotation.
  - Value 0
- **bge.logic.ROT_MODE_XYZ**
  - Use euler_rotation and apply angles on bone’s Z, Y, X axis successively.
  - Value 1
- **bge.logic.ROT_MODE_XZY**
  - Use euler_rotation and apply angles on bone’s Y, Z, X axis successively.
  - Value 2
- **bge.logic.ROT_MODE_YXZ**
  - Use euler_rotation and apply angles on bone’s Z, X, Y axis successively.
  - Value 3
- **bge.logic.ROT_MODE_YZX**
  - Use euler_rotation and apply angles on bone’s X, Z, Y axis successively.
  - Value 4
- **bge.logic.ROT_MODE_ZXY**
  - Use euler_rotation and apply angles on bone’s Y, X, Z axis successively.
  - Value 5
- **bge.logic.ROT_MODE_ZYX**
  - Use euler_rotation and apply angles on bone’s X, Y, Z axis successively.
  - Value 6

Armature Constraint

See `BL_ArmatureConstraint.type`

- **bge.logic.CONTRAINT_TYPE_TRACKTO**
  - Value 2
- **bge.logic.CONTRAIN_TYPE_KINEMATIC**
  - Value 3
- **bge.logic.CONTRAIN_TYPE_ROTLIKE**
  - Value 8
- **bge.logic.CONTRAIN_TYPE_LOCLIKE**
  - Value 9
- **bge.logic.CONTRAIN_TYPE_MINMAX**
  - Value 16
bge.logic.CONSTRAINT_TYPE_SIZELIKE
  Value 10
bge.logic.CONSTRAINT_TYPE_LOCKTRACK
  Value 13
bge.logic.CONSTRAINT_TYPE_STRETCHTO
  Value 15
bge.logic.CONSTRAINT_TYPE_CLAMPTO
  Value 18
bge.logic.CONSTRAINT_TYPE_TRANSFORM
  Value 19
bge.logic.CONSTRAINT_TYPE_DISTLIMIT
  Value 14

See BL_ArmatureConstraint.ik_type

bge.logic.CONSTRAINT_IK_COPYPOSE
  Constraint is trying to match the position and eventually the rotation of the target.
  Value 0

bge.logic.CONSTRAINT_IK_DISTANCE
  Constraint is maintaining a certain distance to target subject to ik_mode.
  Value 1

See BL_ArmatureConstraint.ik_flag

bge.logic.CONSTRAINT_IK_FLAG_TIP
  Set when the constraint operates on the head of the bone and not the tail.
  Value 1

bge.logic.CONSTRAINT_IK_FLAG_ROT
  Set when the constraint tries to match the orientation of the target.
  Value 2

bge.logic.CONSTRAINT_IK_FLAG_STRETCH
  Set when the armature is allowed to stretch (only the bones with stretch factor > 0.0).
  Value 16

bge.logic.CONSTRAINT_IK_FLAG_POS
  Set when the constraint tries to match the position of the target.
  Value 32

See BL_ArmatureConstraint.ik_mode

bge.logic.CONSTRAINT_IK_MODE_INSIDE
  The constraint tries to keep the bone within ik_dist of target.
  Value 0

bge.logic.CONSTRAINT_IK_MODE_OUTSIDE
  The constraint tries to keep the bone outside ik_dist of the target.
  Value 1
bge.logic.CONSTRAINT_IK_MODE_ONSURFACE

The constraint tries to keep the bone exactly at ik_dist of the target.

Value 2

Blender Material

bge.logic.BL_DST_ALPHA

Value 8
bge.logic.BL_DST_COLOR

Value 4
bge.logic.BL_ONE

Value 1
bge.logic.BL_ONE_MINUS_DST_ALPHA

Value 9
bge.logic.BL_ONE_MINUS_DST_COLOR

Value 5
bge.logic.BL_ONE_MINUS_SRC_ALPHA

Value 7
bge.logic.BL_ONE_MINUS_SRC_COLOR

Value 3
bge.logic.BL_SRC_ALPHA

Value 6
bge.logic.BL_SRC_ALPHA_SATURATE

Value 10
bge.logic.BL_SRC_COLOR

Value 2
bge.logic.BL_ZERO

Value 0

Input Status

See SCA_PythonKeyboard, SCA_PythonMouse, SCA_MouseSensor, SCA_KeyboardSensor
bge.logic.KX_INPUT_NONE

Value 0
bge.logic.KX_INPUT_JUST_ACTIVATED

Value 1
bge.logic.KX_INPUT_ACTIVE

Value 2
bge.logic.KX_INPUT_JUST_RELEASED
   Value 3

**KX_GameObject**

See `KX_GameObject.playAction`

bge.logic.KX_ACTION_MODE_PLAY
   Play the action once.
   
   Value 0

bge.logic.KX_ACTION_MODE_LOOP
   Loop the action (repeat it).
   
   Value 1

bge.logic.KX_ACTION_MODE_PING_PONG
   Play the action one direct then back the other way when it has completed.
   
   Value 2

bge.logic.KX_ACTION_BLEND_BLEND
   Blend layers using linear interpolation.
   
   Value 0

bge.logic.KX_ACTION_BLEND_ADD
   Adds the layers together.
   
   Value 1

**Mouse Buttons**

See `SCA_MouseSensor`

bge.logic.KX_MOUSE_BUT_LEFT
   Value 116

bge.logic.KX_MOUSE_BUT_MIDDLE
   Value 117

bge.logic.KX_MOUSE_BUT_RIGHT
   Value 118

**Navigation Mesh Draw Modes**

bge.logic.RM_WALLS
   Draw only the walls.
   
   Value 0

bge.logic.RM_POLYS
   Draw only polygons.
   
   Value 1
bge.logic.RM_TRIS
   Draw triangle mesh.
       Value 2

Shader

bge.logic.VIEWMATRIX
   Value 0
bge.logic.VIEWMATRIX_INVERSE
   Value 10
bge.logic.VIEWMATRIX_INVERSETRANSPOSE
   Value 11
bge.logic.VIEWMATRIX_TRANSPOSE
   Value 9
bge.logic.MODELMATRIX
   Value 4
bge.logic.MODELMATRIX_INVERSE
   Value 6
bge.logic.MODELMATRIX_INVERSETRANSPOSE
   Value 7
bge.logic.MODELMATRIX_TRANSPOSE
   Value 5
bge.logic.MODELVIEWMATRIX
   Value 0
bge.logic.MODELVIEWMATRIX_INVERSE
   Value 2
bge.logic.MODELVIEWMATRIX_INVERSETRANSPOSE
   Value 3
bge.logic.MODELVIEWMATRIX_TRANSPOSE
   Value 1
bge.logic.CAM_POS
   Current camera position
       Value 12
bge.logic.CONSTANT_TIMER
   Value 13
bge.logic.EYE
   User a timer for the uniform value.
       Value 14

8.2. Game Logic (bge.logic) 263
bge.logic.SHD_TANGENT
  Value 1

States

See KX_STATE

bge.logic.KX_STATE1
  Value 1
bge.logic.KX_STATE2
  Value 2
bge.logic.KX_STATE3
  Value 4
bge.logic.KX_STATE4
  Value 8
bge.logic.KX_STATE5
  Value 16
bge.logic.KX_STATE6
  Value 32
bge.logic.KX_STATE7
  Value 64
bge.logic.KX_STATE8
  Value 128
bge.logic.KX_STATE9
  Value 256
bge.logic.KX_STATE10
  Value 512
bge.logic.KX_STATE11
  Value 1024
bge.logic.KX_STATE12
  Value 2048
bge.logic.KX_STATE13
  Value 4096
bge.logic.KX_STATE14
  Value 8192
bge.logic.KX_STATE15
  Value 16384
bge.logic.KX_STATE16
Value 32768
bge.logic.KX_STATE17
Value 65536
bge.logic.KX_STATE18
Value 131072
bge.logic.KX_STATE19
Value 262144
bge.logic.KX_STATE20
Value 524288
bge.logic.KX_STATE21
Value 1048576
bge.logic.KX_STATE22
Value 2097152
bge.logic.KX_STATE23
Value 4194304
bge.logic.KX_STATE24
Value 8388608
bge.logic.KX_STATE25
Value 16777216
bge.logic.KX_STATE26
Value 33554432
bge.logic.KX_STATE27
Value 67108864
bge.logic.KX_STATE28
Value 134217728
bge.logic.KX_STATE29
Value 268435456
bge.logic.KX_STATE30
Value 536870912

See KX_StateActuator.operation

bge.logic.KX_STATE_OP_CLR
    Subtract bits to state mask.
    Value 2

bge.logic.KX_STATE_OP_CPY
    Copy state mask.
    Value 0

8.2. Game Logic (bge.logic)
Invert bits to state mask.

Value 3

Add bits to state mask.

Value 1

### 8.3 Rasterizer (bge.render)

#### 8.3.1 Intro

Example of using a *SCA_MouseSensor*, and two *KX_ObjectActuator* to implement MouseLook:

Note: This can also be achieved with the *KX_MouseActuator*.

```python
# To use a mouse movement sensor "Mouse" and a
# motion actuator to mouse look:
import bge

# scale sets the speed of motion
scale = 1.0, 0.5

co = bge.logic.getCurrentController()
obj = co.owner
mouse = co.sensors["Mouse"]
lmotion = co.actuators["LMove"]
wmotion = co.actuators["WMove"]

def mousePos():
    x = (bge.render.getWindowWidth() / 2 - mouse.position[0]) * scale[0]
    y = (bge.render.getWindowHeight() / 2 - mouse.position[1]) * scale[1]
    return (x, y)

pos = mousePos()

# Set the amount of motion: X is applied in world coordinates...
wmotion.useLocalTorque = False
wmotion.torque = ((0.0, 0.0, pos[0]))

# ...Y is applied in local coordinates
lmotion.useLocalTorque = True
lmotion.torque = ((-pos[1], 0.0, 0.0))

# Activate both actuators
co.activate(lmotion)
co.activate(wmotion)

# Centre the mouse
bge.render.setMousePosition(int(bge.render.getWindowWidth() / 2), int(bge.render.getWindowHeight() / 2))
```
8.3.2 Functions

bge.render.getWindowWidth()
Gets the width of the window (in pixels)

Return type  integer

bge.render.getWindowHeight()
Gets the height of the window (in pixels)

Return type  integer

bge.render.setWindowSize(width, height)
Set the width and height of the window (in pixels). This also works for fullscreen applications.

Note: Only works in the standalone player, not the Blender-embedded player.

Parameters

• width(integer) – width in pixels
• height(integer) – height in pixels

bge.render.setFullScreen(enable)
Set whether or not the window should be fullscreen.

Note: Only works in the standalone player, not the Blender-embedded player.

Parameters enable(bool) – True to set full screen, False to set windowed.

bge.render.getFullScreen()
Returns whether or not the window is fullscreen.

Note: Only works in the standalone player, not the Blender-embedded player; there it always returns False.

Return type  bool

bge.render.getDisplayDimensions()
Get the display dimensions, in pixels, of the display (e.g., the monitor). Can return the size of the entire view, so the combination of all monitors; for example, (3840, 1080) for two side-by-side 1080p monitors.

Return type  tuple (width, height)

bge.render.makeScreenshot(filename)
Writes an image file with the displayed image at the frame end.

The image is written to 'filename'. The path may be absolute (eg. /home/foo/image) or relative when started with // (eg. //image). Note that absolute paths are not portable between platforms. If the filename contains a #, it will be replaced by an incremental index so that screenshots can be taken multiple times without overwriting the previous ones (eg. image-#).

Settings for the image are taken from the render settings (file format and respective settings, gamma and colospace conversion, etc). The image resolution matches the framebuffer, meaning, the window size and aspect

8.3. Rasterizer (bge.render)
ratio. When running from the standalone player, instead of the embedded player, only PNG files are supported. Additional color conversions are also not supported.

**Parameters**

`filename` *(string)* – path and name of the file to write

`bge.render.enableVisibility` *(visible)*

Deprecated; doesn’t do anything.

`bge.render.showMouse` *(visible)*

Enables or disables the operating system mouse cursor.

**Parameters**

`visible` *(boolean)* –

`bge.render.setMousePosition` *(x, y)*

Sets the mouse cursor position.

**Parameters**

- `x` *(integer)* – X-coordinate in screen pixel coordinates.
- `y` *(integer)* – Y-coordinate in screen pixel coordinates.

`bge.render.setBackgroundColor` *(rgba)*

Deprecated and no longer functional. Use `KX_WorldInfo.horizonColor` or `KX_WorldInfo.zenithColor` instead.

`bge.render.setEyeSeparation` *(eyesep)*

Sets the eye separation for stereo mode. Usually Focal Length/30 provides a comfortable value.

**Parameters**

`eyesep` *(float)* – The distance between the left and right eye.

`bge.render.getEyeSeparation` *

Gets the current eye separation for stereo mode.

**Return type**

`float`

`bge.render.setFocalLength` *(focallength)*

Sets the focal length for stereo mode. It uses the current camera focal length as initial value.

**Parameters**

`focallength` *(float)* – The focal length.

`bge.render.getFocalLength` *

Gets the current focal length for stereo mode.

**Return type**

`float`

`bge.render.getStereoEye` *

Gets the current stereoscopy eye being rendered. This function is mainly used in a `KX_Scene.pre_draw` callback function to customize the camera projection matrices for each stereoscopic eye.

**Return type**

`LEFT_EYE, RIGHT_EYE`

`bge.render.setMaterialMode` *(mode)*

Deprecated and no longer functional.

Set the material mode to use for OpenGL rendering.

**Parameters**

`mode` *(`KX_TEXFACE_MATERIAL, KX_BLENDER_MULTITEX_MATERIAL, KX_BLENDER_GLSL_MATERIAL`)* – material mode

**Note:** Changes will only affect newly created scenes.
bge.render.getMaterialMode(mode)
Deprecated and no longer functional.
Get the material mode to use for OpenGL rendering.

**Return type** KX_TEXFACE_MATERIAL, KX_BLENDER_MULTITEX_MATERIAL, KX_BLENDER_GLSL_MATERIAL

bge.render.setGLSLMaterialSetting(setting, enable)
Enables or disables a GLSL material setting.

**Parameters**
- **setting** (string (lights, shaders, shadows, ramps, nodes, extra_textures))
- **enable** (boolean)

bge.render.getGLSLMaterialSetting(setting)
Get the state of a GLSL material setting.

**Parameters**
- **setting** (string (lights, shaders, shadows, ramps, nodes, extra_textures))

**Return type** boolean

bge.render.setAnisotropicFiltering(level)
Set the anisotropic filtering level for textures.

**Parameters**
- **level** (integer (must be one of 1, 2, 4, 8, 16)) – The new anisotropic filtering level to use

**Note:** Changing this value can cause all textures to be recreated, which can be slow.

bge.render.getAnisotropicFiltering()
Get the anisotropic filtering level used for textures.

**Return type** integer (one of 1, 2, 4, 8, 16)

bge.render.setMipmapping(value)
Change how to use mipmapping.

**Note:** Changing this value can cause all textures to be recreated, which can be slow.

bge.render.getMipmapping()
Get the current mipmapping setting.

**Return type** RAS_MIPMAP_NONE, RAS_MIPMAP_NEAREST, RAS_MIPMAP_LINEAR

bge.render.drawLine(fromVec, toVec, color)
Draw a line in the 3D scene.

**Parameters**
- **fromVec** (list [x, y, z]) – the origin of the line
- **toVec** (list [x, y, z]) – the end of the line
- **color** (list [r, g, b, a]) – the color of the line
bge.render.enableMotionBlur (factor)
Enable the motion blur effect.

Parameters factor (float [0.0 - 1.0]) – the amount of motion blur to display.

bge.render.disableMotionBlur()
Disable the motion blur effect.

bge.render.showFramerate (enable)
Show or hide the framerate.

Parameters enable (boolean) –

bge.render.showProfile (enable)
Show or hide the profile.

Parameters enable (boolean) –

bge.render.showProperties (enable)
Show or hide the debug properties.

Parameters enable (boolean) –

bge.render.autoDebugList (enable)
Enable or disable auto adding debug properties to the debug list.

Parameters enable (boolean) –

bge.render.clearDebugList()
Clears the debug property list.

bge.render.setVsync (value)
Set the vsync value

Parameters value (integer) – One of VSYNC_OFF, VSYNC_ON, VSYNC_ADAPTIVE

bge.render.getVsync()
Get the current vsync value

Return type One of VSYNC_OFF, VSYNC_ON, VSYNC_ADAPTIVE

### 8.3.3 Constants

#### General

bge.render.KX_BLENDER_MULTITEX_MATERIAL
Deprecated.

Materials approximating blender materials with multitexturing.

Value 1

bge.render.KX_BLENDER_GLSL_MATERIAL
Deprecated.

Materials approximating blender materials with GLSL.

Value 2

bge.render.VSYNC_OFF
Disables vsync.

Value 1
bge.render.VSYNC_ON
  Enables vsync.
  Value 0

bge.render.VSYNC_ADAPTIVE
  Enables adaptive vsync, if supported. Adaptive vsync enables vsync if the framerate is above the monitors refresh rate. Otherwise, vsync is disabled if the framerate is too low.
  Value 2

bge.render.LEFT_EYE
  Left eye being used during stereoscopic rendering.
  Value 0

bge.render.RIGHT_EYE
  Right eye being used during stereoscopic rendering.
  Value 1

HDR

bge.render.HDR_NONE
  Use 8 bit per channel image format.
  Value 0

bge.render.HDR_HALF_FLOAT
  Use 16 bit float per channel image format.
  Value 1

bge.render.HDR_FULL_FLOAT
  Use 32 bit float per channel image format.
  Value 2

Mipmap

bge.render.RAS_MIPMAP_NONE
  Disables mipmaps.
  Value 0

bge.render.RAS_MIPMAP_NEAREST
  Nearest mipmaps.
  Value 1

bge.render.RAS_MIPMAP_LINEAR
  Linear mipmaps.
  Value 2
8.4 Video Texture (bge.texture)

8.4.1 Introduction

The `bge.texture` module allows you to manipulate textures during the game. Several sources for texture are possible: video files, image files, video capture, memory buffer, camera render or a mix of that. The video and image files can be loaded from the Internet using a URL instead of a file name. In addition, you can apply filters on the images before sending them to the GPU, allowing video effect: blue screen, color band, gray, normal map. `bge.texture` uses FFmpeg to load images and videos. All the formats and codecs that FFmpeg supports are supported by `bge.texture`, including but not limited to:

- AVI
- Ogg
- Xvid
- Theora
- dv1394 camera
- video4linux capture card (this includes many webcams)
- videoForWindows capture card (this includes many webcams)
- JPG

How it works

The principle is simple: first you identify a texture on an existing object using the `bge.texture.materialID` function, then you create a new texture with dynamic content and swap the two textures in the GPU.

The game engine is not aware of the substitution and continues to display the object as always, except that you are now in control of the texture.

When the texture object is deleted, the new texture is deleted and the old texture restored.

Game Preparation

Before you can use the `bge.texture` module, you must have objects with textures applied appropriately.

Imagine you want to have a television showing live broadcast programs in the game. You will create a television object and UV-apply a different texture at the place of the screen, for example `tv.png`. What this texture looks like is not important; probably you want to make it dark gray to simulate power-off state. When the television must be turned on, you create a dynamic texture from a video capture card and use it instead of `tv.png`: the TV screen will come to life.

You have two ways to define textures that `bge.texture` can grab:

- Simple UV texture.
- Blender material with image texture channel.

Because `bge.texture` works at texture level, it is compatible with all the Blender Game Engine’s fancy texturing features: GLSL, multi-texture, custom shaders, etc.
8.4.2 Examples

Basic Video Playback

Example of how to replace a texture in game with a video. It needs to run everyframe.

```python
import bge
from bge import texture
from bge import logic

cont = logic.getCurrentController()
obj = cont.owner

# the creation of the texture must be done once: save the
# texture object in an attribute of bge.logic module makes it persistent
if not hasattr(logic, 'video'):

    # identify a static texture by name
    matID = texture.materialID(obj, 'IMvideo.png')

    # create a dynamic texture that will replace the static texture
    logic.video = texture.Texture(obj, matID)

    # define a source of image for the texture, here a movie
    movie = logic.expandPath('//trailer_400p.ogg')
    logic.video.source = texture.VideoFFmpeg(movie)
    logic.video.source.scale = True

    # quick off the movie, but it wont play in the background
    logic.video.source.play()

# you need to call this function every frame to ensure update of the texture.
logic.video.refresh(True)
```

Texture Replacement

Example of how to replace a texture in game with an external image. `createTexture()` and `removeTexture()` are to be called from a module Python Controller.

```python
from bge import logic
from bge import texture

def createTexture(cont):
    """Create a new Dynamic Texture""
    obj = cont.owner

    # get the reference pointer (ID) of the internal texture
    ID = texture.materialID(obj, 'IMoriginal.png')

    # create a texture object
    object_texture = texture.Texture(obj, ID)

    # create a new source with an external image
    url = logic.expandPath('//newtexture.jpg')
    new_source = texture.ImageFFmpeg(url)
```

(continues on next page)
## Video Capture with DeckLink

Video frames captured with DeckLink cards have pixel formats that are generally not directly usable by OpenGL, they must be processed by a shader. The three shaders presented here should cover all common video capture cases.

This file reflects the current video transfer method implemented in the Decklink module: whenever possible the video images are transferred as float texture because this is more compatible with GPUs. Of course, only the pixel formats that have a correspondent GL format can be transferred as float. Look for fg_shaders in this file for an exhaustive list.

Other pixel formats will be transferred as 32 bits integer red-channel texture but this won’t work with certain GPU (Intel GMA); the corresponding shaders are not shown here. However, it should not be necessary to use any of them as the list below covers all practical cases of video capture with all types of Decklink product.

In other words, only use one of the pixel format below and you will be fine. Note that depending on the video stream, only certain pixel formats will be allowed (others will throw an exception). For example, to capture a PAL video stream, you must use one of the YUV formats.

To find which pixel format is suitable for a particular video stream, use the ‘Media Express’ utility that comes with the Decklink software: if you see the video in the ‘Log and Capture’ Window, you have selected the right pixel format and you can use the same in Blender.

**Note:** These shaders only decode the RGB channel and set the alpha channel to a fixed value. It’s up to you to add postprocessing to the color.

**Note:** These shaders are compatible with 2D and 3D video stream

```python
import bge
from bge import logic
from bge import texture as vt

# The default vertex shader, because we need one
# VertexShader = """version 130
# void main()
# {
```
gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;

gl_TexCoord[0] = gl_MultiTexCoord0;
}

"
"

# For use with RGB video stream: the pixel is directly usable
#
FragmentShader_R10l = ""
#version 130
uniform sampler2D tex;
// stereo = 1.0 if 2D image, =0.5 if 3D (left eye below, right eye above)
uniform float stereo;
// eye = 0.0 for the left eye, 0.5 for the right eye
uniform float eye;

void main(void)
{
    vec4 color;
    float tx, ty;
    tx = gl_TexCoord[0].x;
    ty = eye+gl_TexCoord[0].y*stereo;
    color = texture(tex, vec2(tx,ty));
    color.a = 0.7;
    gl_FragColor = color;
}
"
"

# For use with YUV video stream
#
FragmentShader_2vuy = ""
#version 130
uniform sampler2D tex;
// stereo = 1.0 if 2D image, =0.5 if 3D (left eye below, right eye above)
uniform float stereo;
// eye = 0.0 for the left eye, 0.5 for the right eye
uniform float eye;

void main(void)
{
    vec4 color;
    float tx, ty, width, Y, Cb, Cr;
    int px;
    tx = gl_TexCoord[0].x;
    ty = eye+gl_TexCoord[0].y*stereo;
    width = float(textureSize(tex, 0).x);
    color = texture(tex, vec2(tx, ty));
    px = int(floor(fract(tx*width)*2.0));
    switch (px) {
    case 0:
        Y = color.g;
        break;
    case 1:
        Y = color.a;
        break;
    }
    Y = (Y - 0.0625) * 1.168949772;
}
Cb = (color.b - 0.0625) * 1.142857143 - 0.5;
Cr = (color.r - 0.0625) * 1.142857143 - 0.5;
color.r = Y + 1.5748 * Cr;
color.g = Y - 0.1873 * Cb - 0.4681 * Cr;
color.b = Y + 1.8556 * Cb;
color.a = 0.7;
gl_FragColor = color;
}
"

# For use with high resolution YUV
#
FragmentShader_v210 = ""
#version 130
uniform sampler2D tex;
// stereo = 1.0 if 2D image, =0.5 if 3D (left eye below, right eye above)
uniform float stereo;
// eye = 0.0 for the left eye, 0.5 for the right eye
uniform float eye;

void main(void)
{
  vec4 color, color1, color2, color3;
  int px;
  float tx, ty, width, sx, dx, bx, Y, Cb, Cr;
  tx = gl_TexCoord[0].x;
  ty = eye+gl_TexCoord[0].y*stereo;
  width = float(textureSize(tex, 0).x);
  // to sample macro pixels (6 pixels in 4 words)
  sx = tx*width*0.25+0.01;
  // index of display pixel in the macro pixel 0..5
  px = int(floor(fract(sx)*6.0));
  // increment as we sample the macro pixel
  dx = 1.0/width;
  // base x coord of macro pixel
  bx = (floor(sx)+0.01)*dx*4.0;
  color = texture(tex, vec2(bx, ty));
  color1 = texture(tex, vec2(bx+dx, ty));
  color2 = texture(tex, vec2(bx+dx*2.0, ty));
  color3 = texture(tex, vec2(bx+dx*3.0, ty));
  switch (px) {
    case 0:
      Cb = color.b;
      Cr = color.r;
      break;
    case 1:
      Cb = color1.g;
      Cr = color2.b;
      break;
    case 2:
      Cb = color2.r;
      Cr = color3.g;
      break;
    default:
      Cb = color2.r;
      Cr = color3.g;
      break;
  }
  switch (px) { (continues on next page)
case 0:
    Y = color.g;
    break;
case 1:
    Y = color1.b;
    break;
case 2:
    Y = color1.r;
    break;
case 3:
    Y = color2.g;
    break;
case 4:
    Y = color3.b;
    break;
default:
    Y = color3.r;
    break;
}
Y = (Y - 0.0625) * 1.168949772;
Cb = (Cb - 0.0625) * 1.142857143 - 0.5;
Cr = (Cr - 0.0625) * 1.142857143 - 0.5;
color.r = Y + 1.5748 * Cr;
color.g = Y - 0.1873 * Cb - 0.4681 * Cr;
color.b = Y + 1.8556 * Cb;
color.a = 0.7;
gl_FragColor = color;
}

# The exhausitve list of pixel formats that are transferred as float texture
# Only use those for greater efficiency and compatibility.
#
gf_shaders = {
    '2vuy' :FragmentShader_2vuy,
    '8BitYUV' :FragmentShader_2vuy,
    'v210' :FragmentShader_v210,
    '10BitYUV' :FragmentShader_v210,
    '8BitBGRA' :FragmentShader_R10l,
    'BGRA' :FragmentShader_R10l,
    '8BitRGBXLE' :FragmentShader_R10l,
    '10BitRGBXLE' :FragmentShader_R10l,
    'R10l' :FragmentShader_R10l
}

# Helper function to attach a pixel shader to the material that receives the video frame.
#
def config_video(obj, format, pixel, is3D=False, mat=0, card=0):
    if pixel not in gf_shaders:
        raise('Unsupported shader')
    shader = obj.meshes[0].materials[mat].getShader()
    if shader is not None and not shader.isValid():
        shader.setSource(VertexShader, gf_shaders[pixel], True)
shader.setSampler('tex', 0)
shader.setUniformEyef("eye")
shader.setUniform1f("stereo", 0.5 if is3D else 1.0)
tex = vt.Texture(obj, mat)
tex.source = vt.VideoDeckLink(format + "/" + pixel + ("/3D" if is3D else ""), card)
print("frame rate: ", tex.source.framerate)
tex.source.play()
obj["video"] = tex

# Attach this function to an object that has a material with texture
# and call it once to initialize the object
#
def init(cont):
    # config_video(cont.owner, 'HD720p5994', '8BitBGRA')
    # config_video(cont.owner, 'HD720p5994', '8BitYUV')
    # config_video(cont.owner, 'pal ', '10BitYUV')
    config_video(cont.owner, 'pal ', '8BitYUV')

# To be called on every frame
#
def play(cont):
    obj = cont.owner
    video = obj.get("video")
    if video is not None:
        video.refresh(True)

8.4.3 Video classes

class bge.texture.VideoFFmpeg(file, capture=-1, rate=25.0, width=0, height=0)

FFmpeg video source, used for video files, video captures, or video streams.

Parameters
- **file**(str) – Path to the video to load; if capture >= 0 on Windows, this parameter will not be used.
- **capture**(int) – Capture device number; if >= 0, the corresponding webcam will be used. (optional)
- **rate**(float) – Capture rate. (optional, used only if capture >= 0)
- **width**(int) – Capture width. (optional, used only if capture >= 0)
- **height**(int) – Capture height. (optional, used only if capture >= 0)

**status**
Video status. (readonly)

    Type  int
    Value  see FFmpeg Video and Image Status.

**range**
The start and stop time of the video playback, expressed in seconds from beginning. By default the entire video.
Type sequence of two floats

repeat
   Number of times to replay the video. -1 for infinite repeat.
   Type int

framerate
   Relative frame rate, <1.0 for slow, >1.0 for fast.
   Type float

valid
   Tells if an image is available. (readonly)
   Type bool

image
   Image data. (readonly)
   Type bgl.Buffer or None

size
   Image size. (readonly)
   Type tuple of two ints

scale
   Set to True to activate fast nearest neighbor scaling algorithm. Texture width and height must be a power of 2. If the video picture size is not a power of 2, rescaling is required. By default bge.texture uses the precise but slow gluScaleImage() function. Best is to rescale the video offline so that no scaling is necessary at runtime!
   Type bool

flip
   If True the imaged will be flipped vertically. FFmpeg always delivers the image upside down, so this attribute is set to True by default.
   Type bool

filter
   An additional filter that is applied on the video before sending it to the GPU.
   Type one of:
   - FilterBGR24
   - FilterBlueScreen
   - FilterColor
   - FilterGray
   - FilterLevel
   - FilterNormal
   - FilterRGB24
   - FilterRGBA32

preseek
   Number of frames of preseek.
   Type int
deinterlace
Deinterlace image.

Type bool

play()
Play (restart) video.

Returns Whether the video was ready or stopped.

Return type bool

pause()
Pause video.

Returns Whether the video was playing.

Return type bool

stop()
Stop video (play will replay it from start).

Returns Whether the video was playing.

Return type bool

refresh(buffer=None, format="RGBA", timestamp=-1.0)
Refresh video - get its status and optionally copy the frame to an external buffer.

Parameters

• buffer (any buffer type) – An optional object that implements the buffer protocol. If specified, the image is copied to the buffer, which must be big enough or an exception is thrown.

• format (str) – An optional image format specifier for the image that will be copied to the buffer. Only valid values are “RGBA” or “BGRA”

• timestamp (float) – An optional timestamp (in seconds from the start of the movie) of the frame to be copied to the buffer.

Returns see FFmpeg Video and Image Status.

Return type int

8.4.4 Image classes

class bge.texture.ImageFFmpeg(file)
FFmpeg image source, used for image files and web based images.

Parameters file (str) – Path to the image to load.

status
Image status. (readonly)

Type int

Value see FFmpeg Video and Image Status.

valid
Tells if an image is available. (readonly)

Type bool
image
Image data. (readonly)
Type  Buffer or None

size
Image size. (readonly)
Type  tuple of two ints

scale
Fast scale of image (near neighbour).
Type  bool

flip
Flip image vertically.
Type  bool

filter
Pixel filter.
Type  one of...
• FilterBGR24
• FilterBlueScreen
• FilterColor
• FilterGray
• FilterLevel
• FilterNormal
• FilterRGB24
• FilterRGBA32

refresh (buffer=None, format=“RGBA”)
Refresh image, get its status and optionally copy the frame to an external buffer.

Parameters
• buffer (any buffer type) – An optional object that implements the buffer protocol. If specified, the image is copied to the buffer, which must be big enough or an exception is thrown.
• format (str) – An optional image format specifier for the image that will be copied to the buffer. Only valid values are “RGBA” or “BGRA”

Returns  see FFmpeg Video and Image Status.

Return type  int

reload (newname=None)
Reload image, i.e. reopen it.

Parameters  newname (str) – Path to a new image. (optional)

class  bge.texture.ImageBuff (width, height, color=0, scale=False)
Image from application memory. For computer generated images, drawing applications.

Parameters
• width (int) – Width of the image.
• **height** *(int)* – Height of the image.

• **color** *(int in [0, 255])* – Value to initialize RGB channels with. The initialized buffer will have all pixels set to (color, color, color, 255). (optional)

• **scale** *(bool)* – Image uses scaling. (optional)

**filter**

Pixel filter.

Type one of...

• FilterBGR24
• FilterBlueScreen
• FilterColor
• FilterGray
• FilterLevel
• FilterNormal
• FilterRGB24
• FilterRGBA32

**flip**

Flip image vertically.

Type bool

**image**

Image data. (readonly)

Type Buffer or None

**load** *(imageBuffer, width, height)*

Load image from buffer.

Parameters

• **imageBuffer** *(Buffer or Python object implementing the buffer protocol (e.x. bytes))* – Buffer to load the image from.

• **width** *(int)* – Width of the image to load.

• **height** *(int)* – Height of the image to load.

**plot** *(imageBuffer, width, height, positionX, positionY, mode=IMB_BLEND_COPY)*

Update image buffer.

Parameters

• **imageBuffer** *(Buffer, ImageBuff or Python object implementing the buffer protocol (e.x. bytes))* – Buffer to load the new data from.

• **width** *(int)* – Width of the data to load.

• **height** *(int)* – Height of the data to load.

• **positionX** *(int)* – Left boundary of the region to be drawn on.

• **positionY** *(int)* – Upper boundary of the region to be drawn on.

• **mode** *(int)* – Drawing mode, see Image Blending Modes.
scale
    Fast scale of image (near neighbour).
    Type  bool

size
    Image size. (readonly)
    Type  tuple of two ints

valid
    Tells if an image is available. (readonly)
    Type  bool

class bge.texture.ImageMirror (scene, observer, mirror, material=0, width, height, samples, hdr)
    Image source from mirror.

Parameters
    • scene (KX_Scene) – Scene in which the image has to be taken.
    • observer (KX_GameObject) – Reference object for the mirror (the object from which
    the mirror has to be looked at, for example a camera).
    • mirror (KX_GameObject) – Object holding the mirror.
    • material (int) – ID of the mirror’s material to be used for mirroring. (optional)
    • width (integer) – Off-screen render buffer width (optional).
    • height (integer) – Off-screen render buffer height (optional).
    • samples (integer) – Off-screen render buffer samples (optional).
    • hdr (One of these constants) – Off-screen image format (optional).

alpha
    Use alpha in texture.
    Type  bool

horizon
    Horizon color.
    Type  float list [r, g, b, a] in [0.0, 1.0]

zenith
    Zenith color.
    Type  float list [r, g, b, a] in [0.0, 1.0]

background
    Type  float list [r, g, b, a] in [0.0, 1.0]
    Deprecated use bge.texture.ImageMirror.horizon or bge.texture.ImageMirror.
    zenith instead.

updateShadow
    Choose to force shadow buffer update if there is a gap between image rendered and shadows.
    Type  bool

colorBindCode
    Off-screen color texture bind code.
    Type  integer
capsize
Size of render area.
    Type  sequence of two ints

clip
Clipping distance.
    Type  float in [0.01, 5000.0]

filter
Pixel filter.
    Type  one of...
        • FilterBGR24
        • FilterBlueScreen
        • FilterColor
        • FilterGray
        • FilterLevel
        • FilterNormal
        • FilterRGB24
        • FilterRGBA32

flip
Flip image vertically.
    Type  bool

image
Image data. (readonly)
    Type  Buffer or None

refresh(buff=None, format="RGBA")
Refresh image - render and copy the image to an external buffer (optional) then invalidate its current content.

    Parameters
        • buffer (any buffer type) – An optional object that implements the buffer protocol. If specified, the image is rendered and copied to the buffer, which must be big enough or an exception is thrown.
        • format (str) – An optional image format specifier for the image that will be copied to the buffer. Only valid values are “RGBA” or “BGRA”

scale
Fast scale of image (near neighbour).
    Type  bool

size
Image size (readonly).
    Type  tuple of two ints

valid
Tells if an image is available. (readonly)
Type `bool`

`whole`
Use whole viewport to render.

`whole` `bool`

`class bge.texture.ImageMix`  
Image mixer used to mix multiple image sources together.

`filter`  
Pixel filter.

`filter` `Type` one of...

- `FilterBGR24`
- `FilterBlueScreen`
- `FilterColor`
- `FilterGray`
- `FilterLevel`
- `FilterNormal`
- `FilterRGB24`
- `FilterRGBA32`

`flip`  
Flip image vertically.

`flip` `Type` `bool`

`getSource(id)`  
Get image source.

`Parameters`  
`id(str)` – Identifier of the source to get.

`Returns`  
Image source.

`Return type` one of...

- `VideoFFmpeg`
- `ImageFFmpeg`
- `ImageBuff`
- `ImageMirror`
- `ImageMix`
- `ImageRender`
- `ImageViewport`

`getWeight(id)`  
Get image source weight.

`Parameters`  
`id(str)` – Identifier of the source.

`Returns`  
Weight of the source.

`Return type`  
`int`
image
Image data. (readonly)
Type Buffer or None

refresh(buffer=None, format="RGBA")
Refresh image - calculate and copy the image to an external buffer (optional) then invalidate its current content.

Parameters

• buffer (any buffer type) – An optional object that implements the buffer protocol. If specified, the image is calculated and copied to the buffer, which must be big enough or an exception is thrown.

• format (str) – An optional image format specifier for the image that will be copied to the buffer. Only valid values are “RGBA” or “BGRA”

ga
scale
Fast scale of image (near neighbour).
Type bool

size
Image size. (readonly)
Type tuple of two ints

setSource(id, image)
Set image source - all sources must have the same size.

Parameters

• id (str) – Identifier of the source to set.

• image – Image source of type...
  – VideoFFmpeg
  – ImageFFmpeg
  – ImageBuff
  – ImageMirror
  – ImageMix
  – ImageRender
  – ImageViewport

setWeight(id, weight)
Set image source weight - the sum of the weights should be 256 to get full color intensity in the output.

Parameters

• id (str) – Identifier of the source.

• weight (int) – Weight of the source.

valid
Tells if an image is available. (readonly)
Type bool
class bge.texture.ImageRender (scene, camera, width, height, samples, hdr)

Image source from a render of a non active camera. The render is done on a custom framebuffer object if fbo is specified, otherwise on the default framebuffer.

Parameters

- **scene (KX_Scene)** – Scene in which the image has to be taken.
- **camera (KX_Camera)** – Camera from which the image has to be taken.
- **width (integer)** – Off-screen render buffer width (optional).
- **height (integer)** – Off-screen render buffer height (optional).
- **samples (integer)** – Off-screen render buffer samples (optional).
- **hdr (One of these constants)** – Off-screen image format (optional).

**alpha**

Use alpha in texture.

**Type** bool

**horizon**

Horizon color.

**Type** float list [r, g, b, a] in [0.0, 1.0]

**zenith**

Zenith color.

**Type** float list [r, g, b, a] in [0.0, 1.0]

**background**

Background color.

**Type** float list [r, g, b, a] in [0.0, 1.0]

Deprecated use bge.texture.ImageRender.horizon() or bge.texture.ImageRender.zenith() instead.

**updateShadow**

Choose to force shadow buffer update if there is a gap between image rendered and shadows.

**Type** bool

**colorBindCode**

Off-screen color texture bind code.

**Type** integer

**capsize**

Size of render area.

**Type** sequence of two ints

**filter**

Pixel filter.

**Type** one of...

- FilterBGR24
- FilterBlueScreen
- FilterColor
- FilterGray
• FilterLevel
• FilterNormal
• FilterRGB24
• FilterRGBA32

flip
   Flip image vertically.
   Type bool

image
   Image data. (readonly)
   Type Buffer or None

scale
   Fast scale of image (near neighbour).
   Type bool

size
   Image size. (readonly)
   Type tuple of two ints

valid
   Tells if an image is available. (readonly)
   Type bool

whole
   Use whole viewport to render.
   Type bool

depth
   Use depth component of render as array of float - not suitable for texture source, should only be used with bge.texture.imageToArray(mode='F').
   Type bool

zbuff
   Use depth component of render as grayscale color - suitable for texture source.
   Type bool

render()
   Render the scene but do not extract the pixels yet. The function returns as soon as the render commands have been send to the GPU. The render will proceed asynchronously in the GPU while the host can perform other tasks. To complete the render, you can either call refresh() directly of refresh the texture of which this object is the source. This method is useful to implement asynchronous render for optimal performance: call render() on frame n and refresh() on frame n+1 to give as much as time as possible to the GPU to render the frame while the game engine can perform other tasks.

   Returns True if the render was initiated, False if the render cannot be performed (e.g. the camera is active)

   Return type bool

refresh()
refresh \((\text{buffer}, \text{format}="\text{RGBA}\))\

Refresh video - render and optionally copy the image to an external buffer then invalidate its current content. The render may have been started earlier with the \texttt{render()} method, in which case this function simply waits for the render operations to complete. When called without argument, the pixels are not extracted but the render is guaranteed to be completed when the function returns. This only makes sense with offscreen render on texture target (see \texttt{bge.render.offScreenCreate()}).

Parameters

- **buffer** \((\text{any buffer type of sufficient size})\) – An object that implements the buffer protocol. If specified, the image is copied to the buffer, which must be big enough or an exception is thrown. The transfer to the buffer is optimal if no processing of the image is needed. This is the case if \texttt{flip=False, alpha=True, scale=False, whole=True, depth=False, zbuff=False} and no filter is set.

- **format** \((\text{str})\) – An optional image format specifier for the image that will be copied to the buffer. Only valid values are “RGBA” or “BGRA”

Returns True if the render is complete, False if the render cannot be performed (e.g. the camera is active)

Return type bool

\begin{verbatim}
class bge.texture.ImageViewport
    Image source from viewport rendered by the active camera. To render from a non active camera see ImageRender.

    alpha
        Use alpha in texture.
        Type bool

    capsize
        Size of viewport area being captured.
        Type sequence of two ints

    filter
        Pixel filter.
        Type one of...
        \begin{itemize}
            \item FilterBGR24
            \item FilterBlueScreen
            \item FilterColor
            \item FilterGray
            \item FilterLevel
            \item FilterNormal
            \item FilterRGB24
            \item FilterRGBA32
        \end{itemize}

    flip
        Flip image vertically.
        Type bool
\end{verbatim}
image
Image data. (readonly)

    Type Buffer or None

position
Upper left corner of the captured area.

    Type sequence of two ints

refresh (buffer=None, format="RGBA")
Refresh video - copy the viewport to an external buffer (optional) then invalidate its current content.

    Parameters

    • buffer (any buffer type) – An optional object that implements the buffer protocol. If specified, the image is copied to the buffer, which must be big enough or an exception is thrown. The transfer to the buffer is optimal if no processing of the image is needed. This is the case if flip=False, alpha=True, scale=False, whole=True, depth=False, zbuff=False and no filter is set.

    • format (str) – An optional image format specifier for the image that will be copied to the buffer. Only valid values are “RGBA” or “BGRA”

scale
Fast scale of image (near neighbour).

    Type bool

depth
Use depth component of viewport as array of float - not suitable for texture source, should only be used with bge.texture.imageToArray(mode='F').

    Type bool

depth
Use depth component of viewport as grayscale color - suitable for texture source.

    Type bool

class bge.texture.VideoDeckLink (format, capture=0)
Image source from an external video stream captured with a DeckLink video card from Black Magic Design. Before this source can be used, a DeckLink hardware device must be installed, it can be a PCIe card or a USB device, and the ‘Desktop Video’ software package (version 10.4 or above must be installed) on the host as described in the DeckLink documentation. If in addition you have a recent nVideo Quadro card, you can benefit from the ‘GPUDirect’ technology to push the captured video frame very efficiently to the GPU. For this you need to install the ‘DeckLink SDK’ version 10.4 or above and copy the ‘dvp.dll’ runtime library to Blender’s installation directory or to any other place where Blender can load a DLL from.
Parameters

- **format** (*str*) – string describing the video format to be captured.
- **capture** (*int*) – Card number from which the input video must be captured.

The format argument must be written as `<displayMode>/<pixelFormat>[/3D][:<cacheSize>]` where `<displayMode>` describes the frame size and rate and `<pixelFormat>` the encoding of the pixels. The optional `/3D` suffix is to be used if the video stream is stereo with a left and right eye feed. The optional `:<cacheSize>` suffix determines the number of the video frames kept in cache, by default 8. Some DeckLink cards won’t work below a certain cache size. The default value 8 should be sufficient for all cards. You may try to reduce the cache size to reduce the memory footprint. For example the The 4K Extreme is known to work with 3 frames only, the Extreme 2 needs 4 frames and the Intensity Shuttle needs 6 frames, etc. Reducing the cache size may be useful when Decklink is used in conjunction with GPUDirect: all frames must be locked in memory in that case and that puts a lot of pressure on memory. If you reduce the cache size too much, you’ll get no error but no video feed either.

The valid `<displayMode>` values are copied from the `BMDDisplayMode` enum in the DeckLink API without the ‘bmdMode’ prefix. In case a mode that is not in this list is added in a later version of the SDK, it is also possible to specify the 4 letters of the internal code for that mode. You will find the internal code in the `DeckLinkAPIModes.h` file that is part of the SDK. Here is for reference the full list of supported display modes with their equivalent internal code:

**Internal Codes**

- NTSC ‘ntsc’
- NTSC2398 ‘nt23’
- PAL ‘pal’
- NTSCp ‘ntsp’
- PALp ‘palp’

**HD 1080 Modes**

- HD1080p2398 ‘23ps’
- HD1080p24 ‘24ps’
- HD1080p25 ‘Hp25’
- HD1080p2997 ‘Hp29’
- HD1080p30 ‘Hp30’
- HD1080i50 ‘Hi50’
- HD1080i5994 ‘Hi59’
- HD1080i6000 ‘Hi60’
- HD1080p50 ‘Hp50’
- HD1080p5994 ‘Hp59’
- HD1080p6000 ‘Hp60’

**HD 720 Modes**

- HD720p50 ‘hp50’
- HD720p5994 ‘hp59’
- HD720p60 ‘hp60’
# 2k Modes
- 2k2398 ‘2k23’
- 2k24 ‘2k24’
- 2k25 ‘2k25’

## 4k Modes
- 4K2160p2398 ‘4k23’
- 4K2160p24 ‘4k24’
- 4K2160p25 ‘4k25’
- 4K2160p2997 ‘4k29’
- 4K2160p30 ‘4k30’
- 4K2160p50 ‘4k50’
- 4K2160p5994 ‘4k59’
- 4K2160p60 ‘4k60’

Most of names are self explanatory. If necessary refer to the DeckLink API documentation for more information.

Similarly, `<pixelFormat>` is copied from the BMDPixelFormat enum.

Here is for reference the full list of supported pixel format and their equivalent internal code:

### Pixel Formats
- 8BitYUV ‘2vuy’
- 10BitYUV ‘v210’
- 8BitARGB * no equivalent code *
- 8BitBGRA ‘BGRA’
- 10BitRGB ‘r210’
- 12BitRGB ‘R12B’
- 12BitRGBLE ‘R12L’
- 10BitRGBXLE ‘R10l’
- 10BitRGBX ‘R10b’

Refer to the DeckLink SDK documentation for a full description of these pixel format. It is important to understand them as the decoding of the pixels is NOT done in VideoTexture for performance reason. Instead a specific shader must be used to decode the pixel in the GPU. Only the ‘8BitARGB’, ‘8BitBGRA’ and ‘10BitRGBXLE’ pixel formats are mapped directly to OpenGL RGB float textures. The ‘8BitYUV’ and ‘10BitYUV’ pixel formats are mapped to openGL RGB float texture but require a shader to decode. The other pixel formats are sent as a `GL_RED_INTEGER` texture (i.e. a texture with only the red channel coded as an unsigned 32 bit integer) and are not recommended for use.

Example: HD1080p24/10BitYUV/3D:4 is equivalent to 24ps/v210/3D:4 and represents a full HD stereo feed at 24 frame per second and 4 frames cache size.

Although video format auto detection is possible with certain DeckLink devices, the corresponding API is NOT implemented in the BGE. Therefore it is important to specify the format string that matches exactly the video feed. If the format is wrong, no frame will be captured. It should be noted that the pixel format that you need to specify is not necessarily the actual format in the video feed. For example, the 4K Extreme card delivers 8bit...
RGBs pixels in the ‘10BitRGBXLE’ format. Use the ‘Media Express’ application included in ‘Desktop Video’ to discover which pixel format works for a particular video stream.

status
Status of the capture: 1=ready to use, 2=capturing, 3=stopped

Type int

framerate
Capture frame rate as computed from the video format.

Type float

valid
Tells if the image attribute can be used to retrieve the image. Always False in this implementation (the image is not available at python level)

Type bool

image
The image data. Always None in this implementation.

Type Buffer or None

size
The size of the frame in pixel. Stereo frames have double the height of the video frame, i.e. 3D is delivered to the GPU as a single image in top-bottom order, left eye on top.

Type (int,int)

scale
Not used in this object.

Type bool

flip
Not used in this object.

Type bool

filter
Not used in this object.

play()
Kick-off the capture after creation of the object.

Returns True if the capture could be started, False otherwise.

Return type bool

pause()
Temporary stops the capture. Use play() to restart it.

Returns True if the capture could be paused, False otherwise.

Return type bool

stop()
Stops the capture.

Returns True if the capture could be stopped, False otherwise.

Return type bool
8.4.5 Texture classes

class bge.texture.Texture(gameObj, materialID=0, textureID=0, textureObj=None)
    Class that creates the Texture object that loads the dynamic texture on the GPU.

    Parameters

    • gameObj(KX_GameObject) – Game object to be created a video texture on.
    • materialID(int) – Material ID default, 0 is the first material. (optional)
    • textureID(int) – Texture index in case of multi-texture channel, 0 = first channel by default. In case of UV texture, this parameter should always be 0. (optional)
    • textureObj(Texture) – Reference to another Texture object with shared bindId which he user might want to reuse the texture. If this argument is used, you should not create any source on this texture and there is no need to refresh it either: the other Texture object will provide the texture for both materials/textures.(optional)

    bindId
    OpenGL Bind Name. (readonly)

    Type int

    close()
    Close dynamic texture and restore original.

    mipmap
    Mipmap texture.

    Type bool

    refresh(refresh_source, timestamp=-1.0)
    Refresh texture from source.

    Parameters

    • refresh_source(bool) – Whether to also refresh the image source of the texture.
    • timestamp(float) – If the texture controls a VideoFFmpeg object: timestamp (in seconds from the start of the movie) of the frame to be loaded; this can be used for video-sound synchroniztion by passing KX_SoundActuator.time to it. (optional)

    source
    Source of texture.

    Type one of...

    • VideoFFmpeg
    • VideoDeckLink
    • ImageFFmpeg
    • ImageBuff
    • ImageMirror
    • ImageMix
    • ImageRender
    • ImageViewport
class bge.texture.DeckLink(cardIdx=0, format='')

Certain DeckLink devices can be used to playback video: the host sends video frames regularly for immediate or scheduled playback. The video feed is outputted on HDMI or SDI interfaces. This class supports the immediate playback mode: it has a source attribute that is assigned one of the source object in the bge.texture module. Refreshing the DeckLink object causes the image source to be computed and sent to the DeckLink device for immediate transmission on the output interfaces. Keying is supported: it allows to composite the frame with an input video feed that transits through the DeckLink card.

Parameters

- **cardIdx (int)** – Number of the card to be used for output (0=first card). It should be noted that DeckLink devices are usually half duplex: they can either be used for capture or playback but not both at the same time.

- **format (str)** – String representing the display mode of the output feed.

The default value of the format argument is reserved for auto detection but it is currently not supported (it will generate a runtime error) and thus the video format must be explicitly specified. If keying is the goal (see keying attributes), the format must match exactly the input video feed, otherwise it can be any format supported by the device (there will be a runtime error if not). The format of the string is `<displayMode>[/3D]`.

Refer to VideoDeckLink to get the list of acceptable `<displayMode>`. The optional `/3D` suffix is used to create a stereo 3D feed. In that case the ‘right’ attribute must also be set to specify the image source for the right eye.

Note: The pixel format is not specified here because it is always BGRA. The alpha channel is used in keying to mix the source with the input video feed, otherwise it is not used. If a conversion is needed to match the native video format, it is done inside the DeckLink driver or device.

**source**

This attribute must be set to one of the image sources. If the image size does not fit exactly the frame size, the extend attribute determines what to do.

For best performance, the source image should match exactly the size of the output frame. A further optimization is achieved if the image source object is ImageViewport or ImageRender set for whole viewport, flip disabled and no filter: the GL frame buffer is copied directly to the image buffer and directly from there to the DeckLink card (hence no buffer to buffer copy inside VideoTexture).

Type one of... - VideoFFmpeg - VideoDeckLink - ImageFFmpeg - ImageBuff - ImageMirror - ImageMix - ImageRender - ImageViewport

**right**

If the video format is stereo 3D, this attribute should be set to an image source object that will produce the right eye images. If the goal is to render the BGE scene in 3D, it can be achieved with 2 cameras, one for each eye, used by 2 ImageRender with an offscreen render buffer that is just the size of the video frame.

Type one of... - VideoFFmpeg - VideoDeckLink - ImageFFmpeg - ImageBuff - ImageMirror - ImageMix - ImageRender - ImageViewport

**keying**

Specify if keying is enabled. False (default): the output frame is sent unmodified on the output interface (in that case no input video is required). True: the output frame is mixed with the input video, using the alpha channel to blend the two images and the combination is sent on the output interface.

Type bool

**level**

If keying is enabled, sets the keying level from 0 to 255. This value is a global alpha value that multiplies the alpha channel of the image source. Use 255 (the default) to keep the alpha channel unmodified, 0 to make the output frame totally transparent.
Type int

extend
Determines how the image source should be mapped if the size does not fit the video frame size. * False (the default): map the image pixel by pixel. If the image size is smaller than the frame size, extra space around the image is filled with 0-alpha black. If it is larger, the image is cropped to fit the frame size. * True: the image is scaled by the nearest neighbor algorithm to fit the frame size. The scaling is fast but poor quality. For best results, always adjust the image source to match the size of the output video.

Type bool
close()
Close the DeckLink device and release all resources. After calling this method, the object cannot be reactivated, it must be destroyed and a new DeckLink object created from fresh to restart the output.

refresh(refresh_source, ts)
This method must be called frequently to update the output frame in the DeckLink device.

Parameters

• refresh_source (bool) – True if the source objects image buffer should be invalidated after being used to compute the output frame. This triggers the recomputing of the source image on next refresh, which is normally the desired effect. False if the image source buffer should stay valid and reused on next refresh. Note that the DeckLink device stores the output frame and replays until a new frame is sent from the host. Thus, it is not necessary to refresh the DeckLink object if it is known that the image source has not changed.

• ts (float) – The timestamp value passed to the image source object to compute the image. If unspecified, the BGE clock is used.

8.4.6 Filter classes

class bge.texture.FilterBGR24
Source filter BGR24.

class bge.texture.FilterBlueScreen
Filter for Blue Screen. The RGB channels of the color are left unchanged, while the output alpha is obtained as follows:

• if the square of the euclidian distance between the RGB color and the filter’s reference color is smaller than the filter’s lower limit, the output alpha is set to 0;

• if that square is bigger than the filter’s upper limit, the output alpha is set to 255;

• otherwise the output alpha is linearly extrapoled between 0 and 255 in the interval of the limits.

color
Reference color.

Type sequence of three ints

Default (0, 0, 255)

limits
Reference color limits.

Type sequence of two ints

Default (64, 64)
Previous pixel filter.

**Type** one of...
- FilterBGR24
- FilterBlueScreen
- FilterColor
- FilterGray
- FilterLevel
- FilterNormal
- FilterRGB24
- FilterRGBA32

### class bge.texture.FilterColor

Filter for color calculations. The output color is obtained by multiplying the reduced 4x4 matrix with the input color and adding the remaining column to the result.

**matrix**

Matrix [4][5] for color calculation.

**Type** sequence of four sequences of five ints

**Default** ((256, 0, 0, 0, 0), (0, 256, 0, 0, 0), (0, 0, 256, 0, 0), (0, 0, 0, 256, 0))

 Previous pixel filter.

**Type** one of...
- FilterBGR24
- FilterBlueScreen
- FilterColor
- FilterGray
- FilterLevel
- FilterNormal
- FilterRGB24
- FilterRGBA32

### class bge.texture.FilterGray

Filter for grayscale effect. Proportions of R, G and B contributions in the output grayscale are 28:151:77.

 Previous pixel filter.

**Type** one of...
- FilterBGR24
- FilterBlueScreen
- FilterColor
- FilterGray

8.4. Video Texture (bge.texture)
- **FilterLevel**
- **FilterNormal**
- **FilterRGB24**
- **FilterRGBA32**

class bge.texture.FilterLevel

Filter for levels calculations. Each output color component is obtained as follows:

- if it is smaller than its corresponding min value, it is set to 0;
- if it is bigger than its corresponding max value, it is set to 255;
- Otherwise it is linearly extrapolated between 0 and 255 in the (min, max) interval.

**levels**

Levels matrix [4] (min, max).

**Type** sequence of four sequences of two ints

**Default** ((0, 255), (0, 255), (0, 255), (0, 255))

**previous**

Previous pixel filter.

**Type** one of...

- **FilterBGR24**
- **FilterBlueScreen**
- **FilterColor**
- **FilterGray**
- **FilterLevel**
- **FilterNormal**
- **FilterRGB24**
- **FilterRGBA32**

class bge.texture.FilterNormal

Normal map filter.

**colorIdx**

Index of color used to calculate normal (0 - red, 1 - green, 2 - blue, 3 - alpha).

**Type** int in [0, 3]

**Default** 0

**depth**

Depth of relief.

**Type** float

**Default** 4.0

**previous**

Previous pixel filter.

**Type** one of...

- **FilterBGR24**
- FilterBlueScreen
- FilterColor
- FilterGray
- FilterLevel
- FilterNormal
- FilterRGB24
- FilterRGBA32

class bge.texture.FilterRGB24
Returns a new input filter object to be used with ImageBuff object when the image passed to the ImageBuff.load() function has the 3-bytes pixel format BGR.

class bge.texture.FilterRGBA32
Source filter RGBA32.

8.4.7 Functions

bge.texture.getError()
Last error that occurred in a bge.texture function.

Returns The description of the last error occurred in a bge.texture function.

Return type str

bge.texture.imageToArray(image, mode)
Returns a Buffer corresponding to the current image stored in a texture source object.

Parameters

- **image** – Image source object of type:
  - VideoFFmpeg
  - ImageFFmpeg
  - ImageBuff
  - ImageMirror
  - ImageMix
  - ImageRender
  - ImageViewport
- **mode** (str) – Optional argument representing the pixel format.
  - You can use the characters R, G, B for the 3 color channels, A for the alpha channel, 0 to force a fixed 0 color channel and 1 to force a fixed 255 color channel.

Examples:

* "BGR" will return 3 bytes per pixel with the Blue, Green and Red channels in that order.
* "RGB1" will return 4 bytes per pixel with the Red, Green, Blue channels in that order and the alpha channel forced to 255.
– A special mode “F” allows to return the image as an array of float. This mode should only be used to retrieve the depth buffer of the class: ImageViewport and ImageRender objects. The default mode is “RGBA”.

**Returns** An object representing the image as one dimensional array of bytes of size (pixel_size*width*height), line by line starting from the bottom of the image. The pixel size and format is determined by the mode parameter. For mode ‘F’, the array is a one dimensional array of float of size (width*height).

**Return type** Buffer

```python
bge.texture.materialID(object, name)
```

Returns a numeric value that can be used in Texture to create a dynamic texture.

The value corresponds to an internal material number that uses the texture identified by name. name is a string representing a texture name with IM prefix if you want to identify the texture directly. This method works for basic tex face and for material, provided the material has a texture channel using that particular texture in first position of the texture stack. name can also have MA prefix if you want to identify the texture by material. In that case the material must have a texture channel in first position.

If the object has no material that matches name, it generates a runtime error. Use try/except to catch the exception.

**Ex:** bge.texture.materialID(obj, 'IMvideo.png')

**Parameters**

- **object (KX_GameObject)** – The game object that uses the texture you want to make dynamic.
- **name (str)** – Name of the texture/material you want to make dynamic.

**Returns** The internal material number.

**Return type** int

```python
bge.texture.setLogFile(filename)
```

Sets the name of a text file in which runtime error messages will be written, in addition to the printing of the messages on the Python console. Only the runtime errors specific to the VideoTexture module are written in that file, ordinary runtime time errors are not written.

**Parameters** filename (str) – Name of the error log file.

**Returns** -1 if the parameter name is invalid (not of type string), else 0.

**Return type** int

### 8.4.8 Constants

**FFmpeg Video and Image Status**

```python
bge.texture.SOURCE_ERROR
```

**Value** -1

```python
bge.texture.SOURCE_EMPTY
```

**Value** 0

```python
bge.texture.SOURCE_READY
```

**Value** 1
bge.texture.SOURCE_PLAYING
    Value 2
bge.texture.SOURCE_STOPPED
    Value 3

Image Blending Modes

See Wikipedia’s Blend Modes for reference.

bge.texture.IMB_BLEND_MIX
    Value 0
bge.texture.IMB_BLEND_ADD
    Value 1
bge.texture.IMB_BLEND_SUB
    Value 2
bge.texture.IMB_BLEND_MUL
    Value 3
bge.texture.IMB_BLEND_LIGHTEN
    Value 4
bge.texture.IMB_BLEND_DARKEN
    Value 5
bge.texture.IMB_BLEND_ERASE_ALPHA
    Value 6
bge.texture.IMB_BLEND_ADD_ALPHA
    Value 7
bge.texture.IMB_BLEND_OVERLAY
    Value 8
bge.texture.IMB_BLEND_HARDLIGHT
    Value 9
bge.texture.IMB_BLEND_COLORBURN
    Value 10
bge.texture.IMB_BLEND_LINEARBURN
    Value 11
bge.texture.IMB_BLEND_COLORDODGE
    Value 12
bge.texture.IMB_BLEND_SCREEN
    Value 13
bge.texture.IMB_BLEND_SOFTLIGHT
Value 14
bge.texture.IMB_BLEND_PINLIGHT

Value 15
bge.texture.IMB_BLEND_VIVIDLIGHT

Value 16
bge.texture.IMB_BLEND_LINEARLIGHT

Value 17
bge.texture.IMB_BLEND_DIFFERENCE

Value 18
bge.texture.IMB_BLEND_EXCLUSION

Value 19
bge.texture.IMB_BLEND_HUE

Value 20
bge.texture.IMB_BLEND_SATURATION

Value 21
bge.texture.IMB_BLEND_LUMINOSITY

Value 22
bge.texture.IMB_BLEND_COLOR

Value 23
bge.texture.IMB_BLEND_COPY

Value 1000
bge.texture.IMB_BLEND_COPY_RGB

Value 1001
bge.texture.IMB_BLEND_COPY_ALPHA

Value 1002

8.5 Game Keys (bge.events)

8.5.1 Intro

This module holds key constants for the SCA_KeyboardSensor.

```python
# Set a connected keyboard sensor to accept F1
import bge

c = bge.logic.getCurrentController()
# 'Keyboard' is a keyboard sensor
sensor = c.sensors["Keyboard"]
sensor.key = bge.events.F1KEY
```
# Do the all keys thing

```python
import bge

c0 = bge.logic.getCurrentController()
# 'Keyboard' is a keyboard sensor
sensor = co.sensors["Keyboard"]

for key, input in sensor.inputs:
    # key[0] == bge.events.keycode = event.type, key[1] = input
    if bge.logic.KX_INPUT_JUST_ACTIVATED in input.queue:
        if key == bge.events.WKEY:
            # Activate Forward!
        if key == bge.events.SKEY:
            # Activate Backward!
        if key == bge.events.AKEY:
            # Activate Left!
        if key == bge.events.DKEY:
            # Activate Right!
```

# The all keys thing without a keyboard sensor (but you will
# need an always sensor with pulse mode on)

```python
import bge

# Just shortening names here
keyboard = bge.logic.keyboard
JUST_ACTIVATED = bge.logic.KX_INPUT_JUST_ACTIVATED

if JUST_ACTIVATED in keyboard.inputs[bge.events.WKEY].queue:
    print("Activate Forward!")
if JUST_ACTIVATED in keyboard.inputs[bge.events.SKEY].queue:
    print("Activate Backward!")
if JUST_ACTIVATED in keyboard.inputs[bge.events.AKEY].queue:
    print("Activate Left!")
if JUST_ACTIVATED in keyboard.inputs[bge.events.DKEY].queue:
    print("Activate Right!")
```

## 8.5.2 Functions

**bge.events.EventToString** (*event*)

Return the string name of a key event. Will raise a ValueError error if its invalid.

**Parameters**

- **event** (*int*) – key event constant from `bge.events` or the keyboard sensor.

**Return type**

`string`

**bge.events.EventToCharacter** (*event*, *shift*)

Return the string name of a key event. Returns an empty string if the event cant be represented as a character.

**Parameters**

- **event** (*int*) – key event constant from `bge.events` or the keyboard sensor.
- **shift** (*bool*) – set to true if shift is held.

**Return type**

`string`
8.5.3 Constants

Mouse Keys

bge.events.LEFTMOUSE
  Value 116
bge.events.MIDDLETMOUSE
  Value 117
bge.events.RIGHTMOUSE
  Value 118
bge.events.WHEELUPMOUSE
  Value 120
bge.events.WHEELEDOWNMOUSE
  Value 121
bge.events.MOUSEX
  Value 122
bge.events.MOUSEY
  Value 123

Alphabet Keys

bge.events.AKEY
  Value 23
bge.events.BKEY
  Value 24
bge.events.CKEY
  Value 25
bge.events.DKEY
  Value 26
bge.events.EKEY
  Value 27
bge.events.FKEY
  Value 28
bge.events.GKEY
  Value 29
bge.events.HKEY
  Value 30
bge.events.IKEY
Value 31
bge.events.JKEY
Value 32
bge.events.KKEY
Value 33
bge.events.LKEY
Value 34
bge.events.MKEY
Value 35
bge.events.NKEY
Value 36
bge.events.OKEY
Value 37
bge.events.PKEY
Value 38
bge.events.QKEY
Value 39
bge.events.RKEY
Value 40
bge.events.SKEY
Value 41
bge.events.TKEY
Value 42
bge.events.UKEY
Value 43
bge.events.VKEY
Value 44
bge.events.WKEY
Value 45
bge.events.XKEY
Value 46
bge.events.YKEY
Value 47
bge.events.ZKEY
Value 48
Number Keys

bge.events.ZEROKEY
Value 13
bge.events.ONEKEY
Value 14
bge.events.TWOKEY
Value 15
bge.events.THREEKEY
Value 16
bge.events.FOURKEY
Value 17
bge.events.FIVEKEY
Value 18
bge.events.SIXKEY
Value 19
bge.events.SEVENKEY
Value 20
bge.events.EIGHTKEY
Value 21
bge.events.NINEKEY
Value 22

Modifiers Keys

bge.events.CAPSLOCKKEY
Value 49
bge.events.LEFTCTRLKEY
Value 50
bge.events.LEFTALTKEY
Value 51
bge.events.RIGHTALTKEY
Value 52
bge.events.RIGHTCTRLKEY
Value 53
bge.events.RIGHTSHIFTKEY
Value 54
bge.events.LEFTSHIFTKEY
Value 55

**Arrow Keys**

bge.events.LEFTARROWKEY
Value 69

bge.events.DOWNARROWKEY
Value 70

bge.events.RIGHTARROWKEY
Value 71

bge.events.UPARROWKEY
Value 72

**Numberpad Keys**

bge.events.PAD0
Value 84

bge.events.PAD1
Value 77

bge.events.PAD2
Value 73

bge.events.PAD3
Value 78

bge.events.PAD4
Value 74

bge.events.PAD5
Value 79

bge.events.PAD6
Value 75

bge.events.PAD7
Value 80

bge.events.PAD8
Value 76

bge.events.PAD9
Value 71

bge.events.PADPERIOD
Value 82

bge.events.PADSLASHKEY
Value 83
bge.events.PADASTERKEY
Value 9
bge.events.PADMINUS
Value 85
bge.events.PADENTER
Value 86
bge.events.PADPLUSKEY
Value 87

Function Keys

bge.events.F1KEY
Value 88
bge.events.F2KEY
Value 89
bge.events.F3KEY
Value 90
bge.events.F4KEY
Value 91
bge.events.F5KEY
Value 92
bge.events.F6KEY
Value 93
bge.events.F7KEY
Value 94
bge.events.F8KEY
Value 95
bge.events.F9KEY
Value 96
bge.events.F10KEY
Value 97
bge.events.F11KEY
Value 98
bge.events.F12KEY
Value 99
bge.events.F13KEY
Value 100
bge.events.F14KEY

Value 101
bge.events.F15KEY

Value 102
bge.events.F16KEY

Value 103
bge.events.F17KEY

Value 104
bge.events.F18KEY

Value 105
bge.events.F19KEY

Value 106

Other Keys

bge.events.ACCENTGRAVEKEY

Value 63
bge.events.BACKSLASHKEY

Value 65
bge.events.BACKSPACEKEY

Value 59
bge.events.COMMAKEY

Value 10
bge.events.DELKEY

Value 60
bge.events.ENDKEY

Value 113
bge.events.EQUALKEY

Value 66
bge.events.ESCKEY

Value 56
bge.events.HOMEKEY

Value 110
bge.events.INSERTKEY

Value 109
bge.events.LEFTBRACKETKEY
Value 67
bge.events.LINEFEEDKEY
Value 58
bge.events.MINUSKEY
Value 11
bge.events.PAGEDOWNKEY
Value 112
bge.events.PAGEUPKEY
Value 111
bge.events.PAUSEKEY
Value 108
bge.events.PERIODKEY
Value 12
bge.events.QUOTEKEY
Value 62
bge.events.RIGHTBRACKETKEY
Value 68
bge.events.RETKEY

Warning: Deprecated, use bge.events.ENTERKEY() instead.

Value 7
bge.events.ENTERKEY
Value 7
bge.events.SEMICOLONKEY
Value 61
bge.events.SLASHKEY
Value 64
bge.events.SPACETKEY
Value 8
bge.events.TABKEY
Value 57
8.6 Physics Constraints (bge.constraints)

Bullet Physics provides collision detection and rigid body dynamics for the Blender Game Engine.

Features:

- Vehicle simulation.
- Rigid body constraints: hinge and point to point (ball socket).
- Access to internal physics settings, like deactivation time, and debugging features

Note: Note about parameter settings

Since this API is not well documented, it can be unclear what kind of values to use for setting parameters. In general, damping settings should be in the range of 0 to 1 and stiffness settings should not be much higher than about 10.

8.6.1 Examples

See also:

For more examples of Bullet physics and how to use them see the pybullet forum.

Example of how to create a hinge Physics Constraint between two objects.

```python
from bge import logic
from bge import constraints

# get object list
objects = logic.getCurrentScene().objects

# get object named Object1 and Object2
object_1 = objects["Object1"]
object_2 = objects["Object2"]

# want to use Edge constraint type
constraint_type = 2

# get Object1 and Object2 physics IDs
physics_id_1 = object_1.getPhysicsId()
physics_id_2 = object_2.getPhysicsId()

# use bottom right edge of Object1 for hinge position
edge_position_x = 1.0
edge_position_y = 0.0
edge_position_z = -1.0

# rotate the pivot z axis about 90 degrees
edge_angle_x = 0.0
edge_angle_y = 0.0
edge_angle_z = 90.0
```

(continues on next page)
# create an edge constraint

custom.createConstraint(physics_id_1, physics_id_2, 
constraint_type, 
edge_position_x, edge_position_y, edge_position_z, 
edge_angle_x, edge_angle_y, edge_angle_z)

## 8.6.2 Functions

### bge.constraints.createConstraint

Creates a constraint.

**Parameters**

- `physicsid_1 (int)` – The physics id of the first object in constraint.
- `physicsid_2 (int)` – The physics id of the second object in constraint.
- `constraint_type (int)` – The type of the constraint, see Create Constraint Constants.
- `pivot_x (float)` – Pivot X position. (optional)
- `pivot_y (float)` – Pivot Y position. (optional)
- `pivot_z (float)` – Pivot Z position. (optional)
- `axis_x (float)` – X axis angle in degrees. (optional)
- `axis_y (float)` – Y axis angle in degrees. (optional)
- `axis_z (float)` – Z axis angle in degrees. (optional)
- `flag (int)` – 128 to disable collision between linked bodies. (optional)

**Returns**

A constraint wrapper.

**Return type**

*KX_ConstraintWrapper*

### bge.constraints.createVehicle

Creates a vehicle constraint.

**Parameters**

- `physicsid (int)` – The physics id of the chassis object in constraint.

**Returns**

A vehicle constraint wrapper.

**Return type**

*KX_VehicleWrapper*

### bge.constraints.exportBulletFile

Exports a file representing the dynamics world (usually using .bullet extension).

**See** Bullet binary serialization.

**Parameters**

- `filename (str)` – File path.

### bge.constraints.getAppliedImpulse

**Parameters**

- `constraintId (int)` – The id of the constraint.

**Returns**

The most recent applied impulse.

**Return type**

*float*
**bge.constraints.getVehicleConstraint**(*constraintId*)

**Parameters**

`constraintId (int)` – The id of the vehicle constraint.

**Returns**

A vehicle constraint object.

**Return type**

`KX_VehicleWrapper`

**bge.constraints.getCharacter**(*gameobj*)

**Parameters**

`gameobj (KX_GameObject)` – The game object with the character physics.

**Returns**

Character wrapper.

**Return type**

`KX_CharacterWrapper`

**bge.constraints.removeConstraint**(*constraintId*)

Removes a constraint.

**Parameters**

`constraintId (int)` – The id of the constraint to be removed.

**bge.constraints.setCcdMode**(*ccdMode*)

**Note:**

Very experimental, not recommended

Sets the CCD (Continous Collision Detection) mode in the Physics Environment.

**Parameters**

`ccdMode (int)` – The new CCD mode.

**bge.constraints.setContactBreakingTreshold**(*breakingTreshold*)

**Note:**

Reasonable default is 0.02 (if units are meters)

Sets thresholds to do with contact point management.

**Parameters**

`breakingTreshold (float)` – The new contact breaking threshold.

**bge.constraints.setDeactivationAngularTreshold**(*angularTreshold*)

Sets the angular velocity threshold.

**Parameters**

`angularTreshold (float)` – New deactivation angular threshold.

**bge.constraints.setDeactivationLinearTreshold**(*linearTreshold*)

Sets the linear velocity threshold.

**Parameters**

`linearTreshold (float)` – New deactivation linear threshold.

**bge.constraints.setDeactivationTime**(*time*)

Sets the time after which a resting rigidbody gets deactivated.

**Parameters**

`time (float)` – The deactivation time.

**bge.constraints.setDebugMode**(*mode*)

Sets the debug mode.

**Parameters**

`mode (int)` – The new debug mode, see Debug Mode Constants.

**bge.constraints.setGravity**(*x, y, z*)

Sets the gravity force.

**Parameters**
• \texttt{x (float)} – Gravity X force.
• \texttt{y (float)} – Gravity Y force.
• \texttt{z (float)} – Gravity Z force.

\texttt{bge.constraints.setLinearAirDamping(damping)}

\textbf{Note:} Not implemented

Sets the linear air damping for rigidbodies.

\texttt{bge.constraints.setNumIterations(numiter)}

Sets the number of iterations for an iterative constraint solver.

\textbf{Parameters} \texttt{numiter (int)} – New number of iterations.

\texttt{bge.constraints.setNumTimeSubSteps(numsubstep)}

Sets the number of substeps for each physics proceed. Tradeoff quality for performance.

\textbf{Parameters} \texttt{nums substep (int)} – New number of substeps.

\texttt{bge.constraints.setSolverDamping(damping)}

\textbf{Note:} Very experimental, not recommended

Sets the damper constant of a penalty based solver.

\textbf{Parameters} \texttt{damping (float)} – New damping for the solver.

\texttt{bge.constraints.setSolverTau(tau)}

\textbf{Note:} Very experimental, not recommended

Sets the spring constant of a penalty based solver.

\textbf{Parameters} \texttt{tau (float)} – New tau for the solver.

\texttt{bge.constraints.setSolverType(solverType)}

\textbf{Note:} Very experimental, not recommended

Sets the solver type.

\textbf{Parameters} \texttt{solverType (int)} – The new type of the solver.

\texttt{bge.constraints.setSorConstant(sor)}

\textbf{Note:} Very experimental, not recommended

Sets the successive overrelaxation constant.
Parameters sor \((float)\) – New sor value.

```python
bge.constraints.setUseEpa(epa)
```

**Note:** Not implemented

### 8.6.3 Constants

**bge.constraints.error**

Symbolic constant string that indicates error.

**Type** str

#### Debug Mode Constants

Debug mode to be used with `setDebugMode()`.

- **bge.constraints.DBG_NODEBUG**
  
  No debug.
  
  **Value** 0

- **bge.constraints.DBG_DRAWWIREFRAME**
  
  Draw wireframe in debug.
  
  **Value** 1

- **bge.constraints.DBG_DRAWAABB**
  
  Draw Axis Aligned Bounding Box in debug.
  
  **Value** 2

- **bge.constraints.DBG_DRAWFEATURETEXT**
  
  Draw features text in debug.
  
  **Value** 4

- **bge.constraints.DBG_DRAWCONTACTPOINTS**
  
  Draw contact points in debug.
  
  **Value** 8

- **bge.constraints.DBG_NOHELPTEXT**
  
  Debug without help text.
  
  **Value** 32

- **bge.constraints.DBG_DRAWTEXT**
  
  Draw text in debug.
  
  **Value** 64

- **bge.constraints.DBG_PROFILETIMINGS**
  
  Draw profile timings in debug.
  
  **Value** 128

- **bge.constraints.DBG_ENABLESATCOMPARISION**
  
  Enable sat comparision in debug.
bge.constraints(DBG_DISABLEBULLETLCP
Dispose Bullet LCP.

Value 256

bge.constraints(DBG_ENABLECCD
Enable Continuous Collision Detection in debug.

Value 512

bge.constraints(DBG_DRAWCONSTRAINTS
Draw constraints in debug.

Value 1024

bge.constraints(DBG_DRAWCONSTRAINTLIMITS
Draw constraint limits in debug.

Value 4096

bge.constraints(DBG_FASTWIREFRAME
Draw a fast wireframe in debug.

Value 8192

Create Constraint Constants

Constraint type to be used with createConstraint().

bge.constraints.POINTTOPOINT_CONSTRAINT

Value 1

bge.constraints.LINEHINGE_CONSTRAINT

Value 2

bge.constraints.ANGULAR_CONSTRAINT

Value 3

bge.constraints.CONE_TWIST_CONSTRAINT

Value 4

bge.constraints.VEHICLE_CONSTRAINT

Value 11

bge.constraints.GENERIC_6DOF_CONSTRAINT

Value 12

8.7 Application Data (bge.app)

Module to access application values that remain unchanged during runtime.

bge.app.version

The Blender/BGE version as a tuple of 3 ints, eg. (2, 79, 1).
### bge.app.version_string
The Blender/BGE version formatted as a string, eg. “2.79 (sub 1)”.

**Type** str

### bge.app.version_char
The Blender/BGE version character (for minor releases).

**Type** str

### bge.app.upbge_version
The UPBGE version as a tuple of 3 ints, eg. (0, 2, 3).

**Note:** Version tuples can be compared simply with (in)equality symbols; for example, (0, 2, 3) < (0, 2, 3) returns True (lexical order).

**Type** tuple of three ints

### bge.app.upbge_version_string
The UPBGE version formatted as a string, eg. “0.2 (sub 3)”.

**Type** str

### bge.app.has_texture_ffmpeg
True if the BGE has been built with FFmpeg support, enabling use of `ImageFFmpeg` and `VideoFFmpeg`.

**Type** bool

### bge.app.has_joystick
True if the BGE has been built with joystick support.

**Type** bool

### bge.app.has_physics
True if the BGE has been built with physics support.

**Type** bool
Audio System (aud)

This module provides access to the audaspace audio library.

### 9.1 Basic Sound Playback

This script shows how to use the classes: `Device`, `Factory` and `Handle`.

```python
""
Basic Sound Playback
++++++++++++++++++++
This script shows how to use the classes: `Device`, `Factory` and `Handle`.
""
import aud
device = aud.device()
# load sound file (it can be a video file with audio)
factory = aud.Factory('music.ogg')

# play the audio, this return a handle to control play/pause
handle = device.play(factory)
# if the audio is not too big and will be used often you can buffer it
factory_buffered = aud.Factory.buffer(factory)
handle_buffered = device.play(factory_buffered)

# stop the sounds (otherwise they play until their ends)
handle.stop()
handle_buffered.stop()
```
# 9.2 Functions

```python
aud.device()
```

Returns the application’s `Device`.

```python
default
    return The application’s `Device`.
```

`rtype Device`

## 9.3 Device

```python
class aud.Device
```

Device objects represent an audio output backend like OpenAL or SDL, but might also represent a file output or RAM buffer output.

```python
lock()
```

Locks the device so that it’s guaranteed, that no samples are read from the streams until `unlock()` is called. This is useful if you want to do start/stop/pause/resume some sounds at the same time.

**Note:** The device has to be unlocked as often as locked to be able to continue playback.

**Warning:** Make sure the time between locking and unlocking is as short as possible to avoid clicks.

```python
play(factory, keep=False)
```

Plays a factory.

**Parameters**

- `factory (Factory)` – The factory to play.
- `keep (bool)` – See `Handle.keep`.

**Returns** The playback handle with which playback can be controlled with.

**Return type** `Handle`

```python
stopAll()
```

Stops all playing and paused sounds.

```python
unlock()
```

Unlocks the device after a lock call, see `lock()` for details.

```python
channels
```

The channel count of the device.

```python
distance_model
```

The distance model of the device.

**See also:**

OpenAL documentation &lt;https://www.openal.org/documentation&gt;
**doppler_factor**
The doppler factor of the device. This factor is a scaling factor for the velocity vectors in doppler calculation. So a value bigger than 1 will exaggerate the effect as it raises the velocity.

**format**
The native sample format of the device.

**listener_location**
The listeners's location in 3D space, a 3D tuple of floats.

**listener_orientation**
The listener's orientation in 3D space as quaternion, a 4 float tuple.

**listener_velocity**
The listener's velocity in 3D space, a 3D tuple of floats.

**rate**
The sampling rate of the device in Hz.

**speed_of_sound**
The speed of sound of the device. The speed of sound in air is typically 343.3 m/s.

**volume**
The overall volume of the device.

---

### 9.4 Factory

**class aud.Factory**
Factory objects are immutable and represent a sound that can be played simultaneously multiple times. They are called factories because they create reader objects internally that are used for playback.

**file (filename)**
Creates a factory object of a sound file.

- **Parameters** `filename (string)` – Path of the file.
- **Returns** The created Factory object.
- **Return type** Factory

**Warning:** If the file doesn’t exist or can’t be read you will not get an exception immediately, but when you try to start playback of that factory.

**sine (frequency, rate=48000)**
Creates a sine factory which plays a sine wave.

- **Parameters**
  - `frequency (float)` – The frequency of the sine wave in Hz.
  - `rate (int)` – The sampling rate in Hz. It’s recommended to set this value to the playback device’s sampling rate to avoid resampling.

- **Returns** The created Factory object.
- **Return type** Factory

**buffer()**
Buffers a factory into RAM. This saves CPU usage needed for decoding and file access if the underlying factory reads from a file on the harddisk, but it consumes a lot of memory.

**Returns** The created `Factory` object.

**Return type** `Factory`

**Note:** Only known-length factories can be buffered.

---

**Warning:** Raw PCM data needs a lot of space, only buffer short factories.

---

**delay** *(time)*

Delays by playing adding silence in front of the other factory’s data.

**Parameters**
- `time` *(float)* – How many seconds of silence should be added before the factory.

**Returns** The created `Factory` object.

**Return type** `Factory`

**fadein** *(start, length)*

Fades a factory in by raising the volume linearly in the given time interval.

**Parameters**
- `start` *(float)* – Time in seconds when the fading should start.
- `length` *(float)* – Time in seconds how long the fading should last.

**Returns** The created `Factory` object.

**Return type** `Factory`

**Note:** Before the fade starts it plays silence.

---

**fadeout** *(start, length)*

Fades a factory in by lowering the volume linearly in the given time interval.

**Parameters**
- `start` *(float)* – Time in seconds when the fading should start.
- `length` *(float)* – Time in seconds how long the fading should last.

**Returns** The created `Factory` object.

**Return type** `Factory`

**Note:** After the fade this factory plays silence, so that the length of the factory is not altered.

---

**filter** *(b, a = (1))*

Filters a factory with the supplied IIR filter coefficients. Without the second parameter you’ll get a FIR filter. If the first value of the a sequence is 0 it will be set to 1 automatically. If the first value of the a sequence is neither 0 nor 1, all filter coefficients will be scaled by this value so that it is 1 in the end, you don’t have to scale yourself.
Parameters

• \( b \) (sequence of float) – The nominator filter coefficients.
• \( a \) (sequence of float) – The denominator filter coefficients.

Returns The created Factory object.

Return type Factory

highpass \((frequency, Q=0.5)\)

Creates a second order highpass filter based on the transfer function \( H(s) = \frac{s^2}{s^2 + s/Q + 1} \)

Parameters

• frequency (float) – The cut off frequency of the highpass.
• Q (float) – Q factor of the lowpass.

Returns The created Factory object.

Return type Factory

join \((factory)\)

Plays two factories in sequence.

Parameters factory (Factory) – The factory to play second.

Returns The created Factory object.

Return type Factory

Note: The two factories have to have the same specifications (channels and samplerate).

limit \((start, end)\)

Limits a factory within a specific start and end time.

Parameters

• start (float) – Start time in seconds.
• end (float) – End time in seconds.

Returns The created Factory object.

Return type Factory

loop \((count)\)

Loops a factory.

Parameters count (integer) – How often the factory should be looped. Negative values mean endlessly.

Returns The created Factory object.

Return type Factory

Note: This is a filter function, you might consider using Handle.loop_count instead.

lowpass \((frequency, Q=0.5)\)

Creates a second order lowpass filter based on the transfer function \( H(s) = \frac{1}{s^2 + s/Q + 1} \)
Parameters

- **frequency** (*float*) – The cut off frequency of the lowpass.
- **Q** (*float*) – Q factor of the lowpass.

**Returns** The created Factory object.

**Return type** Factory

**mix** (*factory*)
Mixes two factories.

**Parameters** **factory** (*Factory*) – The factory to mix over the other.

**Returns** The created Factory object.

**Return type** Factory

**Note:** The two factories have to have the same specifications (channels and samplerate).

**pingpong**()
Plays a factory forward and then backward. This is like joining a factory with its reverse.

**Returns** The created Factory object.

**Return type** Factory

**pitch** (*factor*)
Changes the pitch of a factory with a specific factor.

**Parameters** **factor** (*float*) – The factor to change the pitch with.

**Returns** The created Factory object.

**Return type** Factory

**Note:** This is done by changing the sample rate of the underlying factory, which has to be an integer, so the factor value rounded and the factor may not be 100% accurate.

**Note:** This is a filter function, you might consider using Handle.pitch instead.

**reverse**()
Plays a factory reversed.

**Returns** The created Factory object.

**Return type** Factory

**Note:** The factory has to have a finite length and has to be seekable. It’s recommended to use this only with factories with fast and accurate seeking, which is not true for encoded audio files, such ones should be buffered using buffer() before being played reversed.
Warning: If seeking is not accurate in the underlying factory you’ll likely hear skips/jumps/cracks.

**square** *(threshold = 0)*

Makes a square wave out of an audio wave by setting all samples with a amplitude >= threshold to 1, all <= -threshold to -1 and all between to 0.

- **Parameters** `threshold` *(float)* – Threshold value over which an amplitude counts non-zero.
- **Returns** The created `Factory` object.
- **Return type** `Factory`

**volume** *(volume)*

Changes the volume of a factory.

- **Parameters** `volume` *(float)* – The new volume.
- **Returns** The created `Factory` object.
- **Return type** `Factory`

**Note:** Should be in the range [0, 1] to avoid clipping.

**Note:** This is a filter function, you might consider using `Handle.volume` instead.

### 9.5 Handle

**class aud.Handle**

Handle objects are playback handles that can be used to control playback of a sound. If a sound is played back multiple times then there are as many handles.

- **pause()**
  
  Pauses playback.
  
  - **Returns** Whether the action succeeded.
  - **Return type** `bool`

- **resume()**
  
  Resumes playback.
  
  - **Returns** Whether the action succeeded.
  - **Return type** `bool`

- **stop()**
  
  Stops playback.
  
  - **Returns** Whether the action succeeded.
  - **Return type** `bool`
Note: This makes the handle invalid.

**attenuation**
This factor is used for distance based attenuation of the source.

See also:
*Device.distance_model*

**cone_angle_inner**
The opening angle of the inner cone of the source. If the cone values of a source are set there are two (audible) cones with the apex at the *location* of the source and with infinite height, heading in the direction of the source's *orientation*. In the inner cone the volume is normal. Outside the outer cone the volume will be *cone_volume_outer* and in the area between the volume will be interpolated linearly.

**cone_angle_outer**
The opening angle of the outer cone of the source.

See also:
*cone_angle_inner*

**cone_volume_outer**
The volume outside the outer cone of the source.

See also:
*cone_angle_inner*

**distance_maximum**
The maximum distance of the source. If the listener is further away the source volume will be 0.

See also:
*Device.distance_model*

**distance_reference**
The reference distance of the source. At this distance the volume will be exactly *volume*.

See also:
*Device.distance_model*

**keep**
Whether the sound should be kept paused in the device when its end is reached. This can be used to seek the sound to some position and start playback again.

**Warning:** If this is set to true and you forget stopping this equals a memory leak as the handle exists until the device is destroyed.

**location**
The source's location in 3D space, a 3D tuple of floats.

**loop_count**
The (remaining) loop count of the sound. A negative value indicates infinity.

**orientation**
The source's orientation in 3D space as quaternion, a 4 float tuple.
pitch
   The pitch of the sound.

position
   The playback position of the sound in seconds.

relative
   Whether the source’s location, velocity and orientation is relative or absolute to the listener.

status
   Whether the sound is playing, paused or stopped (=invalid).

velocity
   The source’s velocity in 3D space, a 3D tuple of floats.

device
   The volume of the sound.

volume
   The volume of the sound.

volume_maximum
   The maximum volume of the source.

   See also:
   Device.distance_model

volume_minimum
   The minimum volume of the source.

   See also:
   Device.distance_model

class aud.error

9.6 Constants

aud.AUD_DEVICE_JACK
   constant value 3

aud.AUD_DEVICE_NULL
   constant value 0

aud.AUD_DEVICE_OPENAL
   constant value 1

aud.AUD_DEVICE_SDL
   constant value 2

aud.AUD_DISTANCE_MODEL_EXPONENT
   constant value 5

aud.AUD_DISTANCE_MODEL_EXPONENT_CLAMPED
   constant value 6

aud.AUD_DISTANCE_MODEL_INVALID
   constant value 0

aud.AUD_DISTANCE_MODEL_INVERSE
   constant value 1

aud.AUD_DISTANCE_MODEL_INVERSE_CLAMPED
   constant value 2
aud.\texttt{AUD\_DISTANCE\_MODEL\_LINEAR}
    constant value 3

aud.\texttt{AUD\_DISTANCE\_MODEL\_LINEAR\_CLAMPED}
    constant value 4

aud.\texttt{AUD\_FORMAT\_FLOAT32}
    constant value 36

aud.\texttt{AUD\_FORMAT\_FLOAT64}
    constant value 40

aud.\texttt{AUD\_FORMAT\_INVALID}
    constant value 0

aud.\texttt{AUD\_FORMAT\_S16}
    constant value 18

aud.\texttt{AUD\_FORMAT\_S24}
    constant value 19

aud.\texttt{AUD\_FORMAT\_S32}
    constant value 20

aud.\texttt{AUD\_FORMAT\_U8}
    constant value 1

aud.\texttt{AUD\_STATUS\_INVALID}
    constant value 0

aud.\texttt{AUD\_STATUS\_PAUSED}
    constant value 2

aud.\texttt{AUD\_STATUS\_PLAYING}
    constant value 1

aud.\texttt{AUD\_STATUS\_STOPPED}
    constant value 3
CHAPTER 10

Math Types & Utilities (mathutils)

This module provides access to math operations.

Note: Classes, methods and attributes that accept vectors also accept other numeric sequences, such as tuples, lists.

Submodules:

10.1 BVHTree Utilities (mathutils.bvhtree)

BVH tree structures for proximity searches and ray casts on geometry.

```
class mathutils.bvhtree.BVHTree

classmethod FromBMesh(bmesh, epsilon=0.0)
    BVH tree based on BMesh data.
    Parameters
    - bmesh (BMesh) – BMesh data.
    - epsilon (float) – Increase the threshold for detecting overlap and raycast hits.

classmethod FromObject(object, scene, deform=True, render=False, cage=False, epsilon=0.0)
    BVH tree based on Object data.
    Parameters
    - object (Object) – Object data.
    - scene (Scene) – Scene data to use for evaluating the mesh.
    - deform (bool) – Use mesh with deformations.
    - render (bool) – Use render settings.
```
• **cage** (*bool*) – Use render settings.

• **epsilon** (*float*) – Increase the threshold for detecting overlap and raycast hits.

**classmethod FromPolygons** (*vertices, polygons, all_triangles=False, epsilon=0.0*)

BVH tree constructed geometry passed in as arguments.

**Parameters**

• **vertices** (*float triplet sequence*) – float triplets each representing *(x, y, z)*

• **polygons** (*Sequence of sequences containing ints*) – Sequence of polygons, each containing indices to the vertices argument.

• **all_triangles** (*bool*) – Use when all **polygons** are triangles for more efficient conversion.

• **epsilon** (*float*) – Increase the threshold for detecting overlap and raycast hits.

**find_nearest** (*origin, distance=1.84467e+19*)

Find the nearest element (typically face index) to a point.

**Parameters**

• **co** (*Vector*) – Find nearest element to this point.

• **distance** (*float*) – Maximum distance threshold.

**Returns** Returns a tuple *(Vector location, Vector normal, int index, float distance)*, Values will all be None if no hit is found.

**Return type** tuple

**find_nearest_range** (*origin, distance=1.84467e+19*)

Find the nearest elements (typically face index) to a point in the distance range.

**Parameters**

• **co** (*Vector*) – Find nearest elements to this point.

• **distance** (*float*) – Maximum distance threshold.

**Returns** Returns a list of tuples *(Vector location, Vector normal, int index, float distance)*,

**Return type** list

**overlap** (*other_tree*)

Find overlapping indices between 2 trees.

**Parameters** other_tree (*BVHTree*) – Other tree to preform overlap test on.

**Returns** Returns a list of unique index pairs, the first index referencing this tree, the second referencing the other_tree.

**Return type** list

**ray_cast** (*origin, direction, distance=sys.float_info.max*)

Cast a ray onto the mesh.

**Parameters**

• **co** (*Vector*) – Start location of the ray in object space.

• **direction** (*Vector*) – Direction of the ray in object space.

• **distance** (*float*) – Maximum distance threshold.
Returns Returns a tuple (Vector location, Vector normal, int index, float distance), Values will all be None if no hit is found.

Return type tuple

10.2 Geometry Utilities (mathutils.geometry)

The Blender geometry module

mathutils.geometry.area_tri(v1, v2, v3)
Returns the area size of the 2D or 3D triangle defined.

Parameters
• v1 (mathutils.Vector) – Point1
• v2 (mathutils.Vector) – Point2
• v3 (mathutils.Vector) – Point3

Return type float

mathutils.geometry.barycentric_transform(point, tri_a1, tri_a2, tri_a3, tri_a1, tri_a2, tri_a3, tri_a1, tri_a2, tri_a3)
Return a transformed point, the transformation is defined by 2 triangles.

Parameters
• point (mathutils.Vector) – The point to transform.
• tri_a1 (mathutils.Vector) – source triangle vertex.
• tri_a2 (mathutils.Vector) – source triangle vertex.
• tri_a3 (mathutils.Vector) – source triangle vertex.
• tri_a1 – target triangle vertex.
• tri_a2 – target triangle vertex.
• tri_a3 – target triangle vertex.

Returns The transformed point

Return type mathutils.Vector’s

mathutils.geometry.box_fit_2d(points)
Returns an angle that best fits the points to an axis aligned rectangle

Parameters points (list) – list of 2d points.

Returns angle

Return type float

mathutils.geometry.box_pack_2d(boxes)
Returns the normal of the 3D tri or quad.

Parameters boxes (list) – list of boxes, each box is a list where the first 4 items are [x, y, width, height, …] other items are ignored.

Returns the width and height of the packed bounding box

Return type tuple, pair of floats
mathutils.geometry.convex_hull_2d(points)
Returns a list of indices into the list given

Parameters:
- points (list) – list of 2d points.

Returns:
- a list of indices

Return type:
- list of ints

mathutils.geometry.distance_point_to_plane(pt, plane_co, plane_no)
Returns the signed distance between a point and a plane (negative when below the normal).

Parameters:
- pt (mathutils.Vector) – Point
- plane_co (mathutils.Vector) – A point on the plane
- plane_no (mathutils.Vector) – The direction the plane is facing

Return type:
- float

mathutils.geometry.interpolate_bezier(knot1, handle1, handle2, knot2, resolution)
Interpolate a bezier spline segment.

Parameters:
- knot1 (mathutils.Vector) – First bezier spline point.
- handle1 (mathutils.Vector) – First bezier spline handle.
- handle2 (mathutils.Vector) – Second bezier spline handle.
- knot2 (mathutils.Vector) – Second bezier spline point.
- resolution (int) – Number of points to return.

Returns:
- The interpolated points

Return type:
- list of mathutils.Vector’s

mathutils.geometry.intersect_line_line(v1, v2, v3, v4)
Returns a tuple with the points on each line respectively closest to the other.

Parameters:
- v1 (mathutils.Vector) – First point of the first line
- v2 (mathutils.Vector) – Second point of the first line
- v3 (mathutils.Vector) – First point of the second line
- v4 (mathutils.Vector) – Second point of the second line

Return type:
- tuple of mathutils.Vector’s

mathutils.geometry.intersect_line_line_2d(lineA_p1, lineA_p2, lineB_p1, lineB_p2)
Takes 2 segments (defined by 4 vectors) and returns a vector for their point of intersection or None.

Warning: Despite its name, this function works on segments, and not on lines.

Parameters:
- lineA_p1 (mathutils.Vector) – First point of the first line
- lineA_p2 (mathutils.Vector) – Second point of the first line
• **lineB_p1** (*mathutils.Vector*) – First point of the second line
• **lineB_p2** (*mathutils.Vector*) – Second point of the second line

**Returns**  The point of intersection or None when not found

**Return type**  *mathutils.Vector* or None

```python
mathutils.geometry.intersect_line_plane(line_a, line_b, plane_co, plane_no, no_flip=False)
```

Calculate the intersection between a line (as 2 vectors) and a plane. Returns a vector for the intersection or None.

**Parameters**

• **line_a** (*mathutils.Vector*) – First point of the first line
• **line_b** (*mathutils.Vector*) – Second point of the first line
• **plane_co** (*mathutils.Vector*) – A point on the plane
• **plane_no** (*mathutils.Vector*) – The direction the plane is facing

**Returns**  The point of intersection or None when not found

**Return type**  *mathutils.Vector* or None

```python
mathutils.geometry.intersect_line_sphere(line_a, line_b, sphere_co, sphere_radius, clip=True)
```

Takes a line (as 2 points) and a sphere (as a point and a radius) and returns the intersection

**Parameters**

• **line_a** (*mathutils.Vector*) – First point of the line
• **line_b** (*mathutils.Vector*) – Second point of the line
• **sphere_co** (*mathutils.Vector*) – The center of the sphere
• **sphere_radius** (*sphere_radius*) – Radius of the sphere

**Returns**  The intersection points as a pair of vectors or None when there is no intersection

**Return type**  A tuple pair containing *mathutils.Vector* or None

```python
mathutils.geometry.intersect_line_sphere_2d(line_a, line_b, sphere_co, sphere_radius, clip=True)
```

Takes a line (as 2 points) and a sphere (as a point and a radius) and returns the intersection

**Parameters**

• **line_a** (*mathutils.Vector*) – First point of the line
• **line_b** (*mathutils.Vector*) – Second point of the line
• **sphere_co** (*mathutils.Vector*) – The center of the sphere
• **sphere_radius** (*sphere_radius*) – Radius of the sphere

**Returns**  The intersection points as a pair of vectors or None when there is no intersection

**Return type**  A tuple pair containing *mathutils.Vector* or None

```python
mathutils.geometry.intersect_plane_plane(plane_a_co, plane_a_no, plane_b_co, plane_b_no)
```

Return the intersection between two planes

**Parameters**

• **plane_a_co** (*mathutils.Vector*) – Point on the first plane
• plane_a_no (mathutils.Vector) – Normal of the first plane
• plane_b_co (mathutils.Vector) – Point on the second plane
• plane_b_no (mathutils.Vector) – Normal of the second plane

Returns The line of the intersection represented as a point and a vector

Return type tuple pair of mathutils.Vector or None if the intersection can’t be calculated

mathutils.geometry.intersect_point_line (pt, line_p1, line_p2)
Takes a point and a line and returns a tuple with the closest point on the line and its distance from the first point of the line as a percentage of the length of the line.

Parameters
• pt (mathutils.Vector) – Point
• line_p1 (mathutils.Vector) – First point of the line
• line_p1 – Second point of the line

Return type (mathutils.Vector, float)

mathutils.geometry.intersect_point_quad_2d (pt, quad_p1, quad_p2, quad_p3, quad_p4)
Takes 5 vectors (using only the x and y coordinates): one is the point and the next 4 define the quad, only the x and y are used from the vectors. Returns 1 if the point is within the quad, otherwise 0. Works only with convex quads without singular edges.

Parameters
• pt (mathutils.Vector) – Point
• quad_p1 (mathutils.Vector) – First point of the quad
• quad_p2 (mathutils.Vector) – Second point of the quad
• quad_p3 (mathutils.Vector) – Third point of the quad
• quad_p4 (mathutils.Vector) – Fourth point of the quad

Return type int

mathutils.geometry.intersect_point_tri (pt, tri_p1, tri_p2, tri_p3)
Takes 4 vectors: one is the point and the next 3 define the triangle.

Parameters
• pt (mathutils.Vector) – Point
• tri_p1 (mathutils.Vector) – First point of the triangle
• tri_p2 (mathutils.Vector) – Second point of the triangle
• tri_p3 (mathutils.Vector) – Third point of the triangle

Returns Point on the triangles plane or None if its outside the triangle

Return type mathutils.Vector or None

mathutils.geometry.intersect_point_tri_2d (pt, tri_p1, tri_p2, tri_p3)
Takes 4 vectors (using only the x and y coordinates): one is the point and the next 3 define the triangle. Returns 1 if the point is within the triangle, otherwise 0.

Parameters
• pt (mathutils.Vector) – Point
• tri_p1 (mathutils.Vector) – First point of the triangle
- `tri_p2(mathutils.Vector)` – Second point of the triangle
- `tri_p3(mathutils.Vector)` – Third point of the triangle

**Return type** int

`mathutils.geometry.intersect_ray_tri(v1, v2, v3, ray, orig, clip=True)`

Returns the intersection between a ray and a triangle, if possible, returns None otherwise.

**Parameters**
- `v1(mathutils.Vector)` – Point1
- `v2(mathutils.Vector)` – Point2
- `v3(mathutils.Vector)` – Point3
- `ray(mathutils.Vector)` – Direction of the projection
- `orig(mathutils.Vector)` – Origin
- `clip(boolean)` – When False, don’t restrict the intersection to the area of the triangle, use the infinite plane defined by the triangle.

**Returns** The point of intersection or None if no intersection is found

**Return type** `mathutils.Vector` or None

`mathutils.geometry.intersect_sphere_sphere_2d(p_a, radius_a, p_b, radius_b)`

Returns 2 points on between intersecting circles.

**Parameters**
- `p_a(mathutils.Vector)` – Center of the first circle
- `radius_a(float)` – Radius of the first circle
- `p_b(mathutils.Vector)` – Center of the second circle
- `radius_b(float)` – Radius of the second circle

**Return type** tuple of `mathutils.Vector`’s or None when there is no intersection

`mathutils.geometry.normal(vectors)`

Returns the normal of a 3D polygon.

**Parameters** `vectors` (sequence of 3 or more 3d vector) – Vectors to calculate normals with

**Return type** `mathutils.Vector`

`mathutils.geometry.points_in_planes(planes)`

Returns a list of points inside all planes given and a list of index values for the planes used.

**Parameters** `planes` (list of `mathutils.Vector`) – List of planes (4D vectors).

**Returns** two lists, once containing the vertices inside the planes, another containing the plane indices used

**Return type** pair of lists

`mathutils.geometry.tessellate_polygon(veclist_list)`

Takes a list of polylines (each point a vector) and returns the point indices for a polyline filled with triangles.

**Parameters** `veclist_list` – list of polylines

**Return type** list
mathutils.geometry.volume_tetrahedron(v1, v2, v3, v4)

Return the volume formed by a tetrahedron (points can be in any order).

Parameters

• v1 (mathutils.Vector) – Point 1
• v2 (mathutils.Vector) – Point 2
• v3 (mathutils.Vector) – Point 3
• v4 (mathutils.Vector) – Point 4

Return type float

10.3 Interpolation Utilities (mathutils.interpolate)

The Blender interpolate module

mathutils.interpolate.poly_3d_calc(veclist, pt)

Calculate barycentric weights for a point on a polygon.

Parameters

• veclist – list of vectors
• pt – point :rtype: list of per-vector weights

10.4 KDTree Utilities (mathutils.kdtree)

Generic 3-dimentional kd-tree to perform spatial searches.

```python
import mathutils

# create a kd-tree from a mesh
from bpy import context
obj = context.object

# 3d cursor relative to the object data
co_find = context.scene.cursor_location * obj.matrix_world.inverted()

mesh = obj.data
size = len(mesh.vertices)
k = mathutils.kdtree.KDTree(size)
for i, v in enumerate(mesh.vertices):
    k.insert(v.co, i)
k.balance()

# Find the closest point to the center
co_find = (0.0, 0.0, 0.0)
co, index, dist = k.find(co_find)
print("Close to center:", co, index, dist)
```

(continues on next page)
# Find the closest 10 points to the 3d cursor
print("Close 10 points")
for (co, index, dist) in kd.find_n(co_find, 10):
    print("  ", co, index, dist)

# Find points within a radius of the 3d cursor
print("Close points within 0.5 distance")
co_find = context.scene.cursor_location
for (co, index, dist) in kd.find_range(co_find, 0.5):
    print("  ", co, index, dist)

class mathutils.kdtree.KDTree
    KdTree(size) -> new kd-tree initialized to hold size items.

    Note: KDTree.balance must have been called before using any of the find methods.

    balance()
        Balance the tree.

    Note: This builds the entire tree, avoid calling after each insertion.

    find(co, filter=None)
        Find nearest point to co.

        Parameters
            • co(float triplet) – 3d coordinates.
            • filter(callable) – function which takes an index and returns True for indices to include in the search.

        Returns Returns (Vector, index, distance).

        Return type tuple

    find_n(co, n)
        Find nearest n points to co.

        Parameters
            • co(float triplet) – 3d coordinates.
            • n(int) – Number of points to find.

        Returns Returns a list of tuples (Vector, index, distance).

        Return type list

    find_range(co, radius)
        Find all points within radius of co.

        Parameters
            • co(float triplet) – 3d coordinates.
            • radius(float) – Distance to search for points.

        Returns Returns a list of tuples (Vector, index, distance).
Return type list

\texttt{insert(co, index)}

Insert a point into the KDTree.

Parameters

- \texttt{co(float triplet)} – Point 3d position.
- \texttt{index(int)} – The index of the point.

10.5 Noise Utilities (mathutils.noise)

The Blender noise module

\texttt{mathutils.noise.cell(position)}

Returns cell noise value at the specified position.

Parameters \texttt{position(mathutils.Vector)} – The position to evaluate the selected noise function at.

Returns The cell noise value.

Return type float

\texttt{mathutils.noise.cell\_vector(position)}

Returns cell noise vector at the specified position.

Parameters \texttt{position(mathutils.Vector)} – The position to evaluate the selected noise function at.

Returns The cell noise vector.

Return type \texttt{mathutils.Vector}

\texttt{mathutils.noise.fractal(position, H, lacunarity, octaves, noise\_basis=noise.types.STDPERLIN)}

Returns the fractal Brownian motion (fBm) noise value from the noise basis at the specified position.

Parameters

- \texttt{position(mathutils.Vector)} – The position to evaluate the selected noise function at.
- \texttt{H(float)} – The fractal increment factor.
- \texttt{lacunarity(float)} – The gap between successive frequencies.
- \texttt{octaves(int)} – The number of different noise frequencies used.
- \texttt{noise\_basis(Value in noise.types or int)} – The type of noise to be evaluated.

Returns The fractal Brownian motion noise value.

Return type float

\texttt{mathutils.noise.hetero\_terrain(position, H, lacunarity, octaves, offset, noise\_basis=noise.types.STDPERLIN)}

Returns the heterogeneous terrain value from the noise basis at the specified position.

Parameters

- \texttt{position(mathutils.Vector)} – The position to evaluate the selected noise function at.
• \( H (\text{float}) \) – The fractal dimension of the roughest areas.
• \( \text{lacunarity} (\text{float}) \) – The gap between successive frequencies.
• \( \text{octaves} (\text{int}) \) – The number of different noise frequencies used.
• \( \text{offset} (\text{float}) \) – The height of the terrain above ‘sea level’.
• \( \text{noise\_basis}(\text{Value in noise.types or int}) \) – The type of noise to be evaluated.

Returns The heterogeneous terrain value.

Return type float

mathutils.noise\_hybrid\_multi\_fractal \((\text{position}, \ H, \ \text{lacunarity}, \ \text{octaves}, \ \text{offset}, \ \text{gain}, \ \text{noise\_basis}=\text{noise.types.STDPERLIN})\)

Returns hybrid multifractal value from the noise basis at the specified position.

Parameters

• \( \text{position}(\text{mathutils.Vector}) \) – The position to evaluate the selected noise function at.
• \( H (\text{float}) \) – The fractal dimension of the roughest areas.
• \( \text{lacunarity}(\text{float}) \) – The gap between successive frequencies.
• \( \text{octaves}(\text{int}) \) – The number of different noise frequencies used.
• \( \text{offset}(\text{float}) \) – The height of the terrain above ‘sea level’.
• \( \text{gain}(\text{float}) \) – Scaling applied to the values.
• \( \text{noise\_basis}(\text{Value in noise.types or int}) \) – The type of noise to be evaluated.

Returns The hybrid multifractal value.

Return type float

mathutils.noise\_multi\_fractal \((\text{position}, \ H, \ \text{lacunarity}, \ \text{octaves}, \ \text{noise\_basis}=\text{noise.types.STDPERLIN})\)

Returns multifractal noise value from the noise basis at the specified position.

Parameters

• \( \text{position}(\text{mathutils.Vector}) \) – The position to evaluate the selected noise function at.
• \( H (\text{float}) \) – The fractal increment factor.
• \( \text{lacunarity}(\text{float}) \) – The gap between successive frequencies.
• \( \text{octaves}(\text{int}) \) – The number of different noise frequencies used.
• \( \text{noise\_basis}(\text{Value in noise.types or int}) \) – The type of noise to be evaluated.

Returns The multifractal noise value.

Return type float

mathutils.noise\_noise \((\text{position}, \ \text{noise\_basis}=\text{noise.types.STDPERLIN})\)

Returns noise value from the noise basis at the position specified.

Parameters
• **position**(*mathutils.Vector*) – The position to evaluate the selected noise function at.

• **noise_basis**(*Value in noise.types or int*) – The type of noise to be evaluated.

**Returns** The noise value.

**Return type** float

`mathutils.noise.noise_vector(position, noise_basis=noise.types.STDPERLIN)`

Returns the noise vector from the noise basis at the specified position.

**Parameters**

• **position**(*mathutils.Vector*) – The position to evaluate the selected noise function at.

• **noise_basis**(*Value in noise.types or int*) – The type of noise to be evaluated.

**Returns** The noise vector.

**Return type** `mathutils.Vector`

`mathutils.noise.random()`

Returns a random number in the range [0, 1].

**Returns** The random number.

**Return type** float

`mathutils.noise.random_unit_vector(size=3)`

Returns a unit vector with random entries.

**Parameters**

• **size**(*Int*) – The size of the vector to be produced.

**Returns** The random unit vector.

**Return type** `mathutils.Vector`

`mathutils.noise.ridged_multi_fractal(position, H, lacunarity, octaves, offset, gain, noise_basis=noise.types.STDPERLIN)`

Returns ridged multifractal value from the noise basis at the specified position.

**Parameters**

• **position**(*mathutils.Vector*) – The position to evaluate the selected noise function at.

• **H**(*float*) – The fractal dimension of the roughest areas.

• **lacunarity**(*float*) – The gap between successive frequencies.

• **octaves**(*int*) – The number of different noise frequencies used.

• **offset**(*float*) – The height of the terrain above ‘sea level’.

• **gain**(*float*) – Scaling applied to the values.

• **noise_basis**(*Value in noise.types or int*) – The type of noise to be evaluated.

**Returns** The ridged multifractal value.

**Return type** float
mathutils.noise.seed_set(seed)
Sets the random seed used for random_unit_vector, random_vector and random.

Parameters

- **seed** *(Int)* – Seed used for the random generator. When seed is zero, the current time will be used instead.

mathutils.noise.turbulence(position, octaves, hard, noise_basis=noise.types.STDPERLIN, amplitude_scale=0.5, frequency_scale=2.0)
Returns the turbulence value from the noise basis at the specified position.

Parameters

- **position** *(mathutils.Vector)* – The position to evaluate the selected noise function at.
- **octaves** *(int)* – The number of different noise frequencies used.
- **hard** *(boolean)* – Specifies whether returned turbulence is hard (sharp transitions) or soft (smooth transitions).
- **noise_basis** *(Value in mathutils.noise.types or int)* – The type of noise to be evaluated.
- **amplitude_scale** *(float)* – The amplitude scaling factor.
- **frequency_scale** *(Value in noise.types or int)* – The frequency scaling factor

Returns
The turbulence value.

Return type
**float**

mathutils.noise.turbulence_vector(position, octaves, hard, noise_basis=noise.types.STDPERLIN, amplitude_scale=0.5, frequency_scale=2.0)
Returns the turbulence vector from the noise basis at the specified position.

Parameters

- **position** *(mathutils.Vector)* – The position to evaluate the selected noise function at.
- **octaves** *(int)* – The number of different noise frequencies used.
- **hard** *(boolean)* – Specifies whether returned turbulence is hard (sharp transitions) or soft (smooth transitions).
- **noise_basis** *(Value in mathutils.noise.types or int)* – The type of noise to be evaluated.
- **amplitude_scale** *(float)* – The amplitude scaling factor.
- **frequency_scale** *(Value in noise.types or int)* – The frequency scaling factor

Returns
The turbulence vector.

Return type **mathutils.Vector**

mathutils.noise.variable_lacunarity(position, distortion, noise_type1=noise.types.STDPERLIN, noise_type2=noise.types.STDPERLIN)
Returns variable lacunarity noise value, a distorted variety of noise, from noise type 1 distorted by noise type 2 at the specified position.

Parameters
• **position** (*mathutils.Vector*) – The position to evaluate the selected noise function at.

• **distortion** (*float*) – The amount of distortion.

• **noise_type1** (*Value in noise.types or int*) – The type of noise to be distorted.

• **noise_type2** (*Value in noise.types or int*) – The type of noise used to distort noise_type1.

**Returns** The variable lacunarity noise value.

**Return type** float

`mathutils.noise.voronoi(position, distance_metric=noise.distance_metrics.DISTANCE, exponent=2.5)`

Returns a list of distances to the four closest features and their locations.

**Parameters**

• **position** (*mathutils.Vector*) – The position to evaluate the selected noise function at.

• **distance_metric** (*Value in noise.distance_metrics or int*) – Method of measuring distance.

• **exponent** (*float*) – The exponent for Minkowski distance metric.

**Returns** A list of distances to the four closest features and their locations.

**Return type** list of four floats, list of four `mathutils.Vector` types

The `mathutils` module provides the following classes:

• **Color**,

• **Euler**,

• **Matrix**,

• **Quaternion**,  

• **Vector**,

```python
import mathutils
from math import radians

vec = mathutils.Vector((1.0, 2.0, 3.0))

mat_rot = mathutils.Matrix.Rotation(radians(90.0), 4, 'X')
mat_trans = mathutils.Matrix.Translation(vec)

mat = mat_trans * mat_rot
mat.invert()

mat3 = mat.to_3x3()
quat1 = mat.to_quaternion()
quat2 = mat3.to_quaternion()

quat_diff = quat1.rotation_difference(quat2)

print(quat_diff.angle)
```
10.6 Color

class mathutils.Color(rgb)

This object gives access to Colors in Blender.

Parameters rgb (3d vector) – (r, g, b) color values

```python
import mathutils

col = mathutils.Color((0.0, 0.0, 1.0))

col.s *= 0.5

print("Color R:", col.r)
print("Color G:", col.g)
print("Color B:", col.b)
print("Color HSV: %.2f, %.2f, %.2f", col[:])

col[:] = 0.0, 0.5, 1.0

col += mathutils.Color((0.25, 0.0, 0.0))

print("Color: %d, %d, %d" % (col * 255.0)[:])

print("Hexidecimal: %.2x%.2x%.2x" % (col * 255.0)[:])
```

copy()

Returns a copy of this color.

Returns A copy of the color.

Return type Color

Note: use this to get a copy of a wrapped color with no reference to the original data.

freeze()

Make this object immutable.

After this the object can be hashed, used in dictionaries & sets.

Returns An instance of this object.

b

Blue color channel.

Type float
g  
Green color channel.  
Type float

h  
HSV Hue component in [0, 1].  
Type float

hsv  
HSV Values in [0, 1].  
Type float triplet

is_frozen  
True when this object has been frozen (read-only).  
Type boolean

is_wrapped  
True when this object wraps external data (read-only).  
Type boolean

owner  
The item this is wrapping or None (read-only).

r  
Red color channel.  
Type float

s  
HSV Saturation component in [0, 1].  
Type float

v  
HSV Value component in [0, 1].  
Type float

10.7 Euler

class mathutils.Euler(angles, order='XYZ')  
This object gives access to Eulers in Blender.

Parameters

• angles(3d vector) – Three angles, in radians.
• order(str) – Optional order of the angles, a permutation of XYZ.

| import mathutils  
| import math  

# create a new euler with default axis rotation order
eul = mathutils.Euler((0.0, math.radians(45.0), 0.0), 'XYZ')

# rotate the euler
eul.rotate_axis('Z', math.radians(10.0))

(continues on next page)
# you can access its components by attribute or index
print("Euler X", eul.x)
print("Euler Y", eul[1])
print("Euler Z", eul[-1])

# components of an existing euler can be set
eul[:] = 1.0, 2.0, 3.0

# components of an existing euler can use slice notation to get a tuple
print("Values: %f, %f, %f" % eul[:])

# the order can be set at any time too
eul.order = "ZYX"

# eulers can be used to rotate vectors
vec = mathutils.Vector((0.0, 0.0, 1.0))
vec.rotate(eul)

# often its useful to convert the euler into a matrix so it can be used as
# transformations with more flexibility
mat_rot = eul.to_matrix()
mat_loc = mathutils.Matrix.Translation((2.0, 3.0, 4.0))
mat = mat_loc * mat_rot.to_4x4()

```python
copy()
  Returns a copy of this euler.
  
  Returns A copy of the euler.
  
  Return type Euler

  Note: use this to get a copy of a wrapped euler with no reference to the original data.

freeze()
  Make this object immutable.
  After this the object can be hashed, used in dictionaries & sets.
  
  Returns An instance of this object.

make_compatible(other)
  Make this euler compatible with another, so interpolating between them works as intended.

  Note: the rotation order is not taken into account for this function.

rotate(other)
  Rotates the euler by another mathutils value.
  
  Parameters other (Euler, Quaternion or Matrix) – rotation component of mathutils value

rotate_axis(axis, angle)
  Rotates the euler a certain amount and returning a unique euler rotation (no 720 degree pitches).
  
  Parameters
• **axis** *(string)* – single character in ['X', 'Y', 'Z'].

• **angle** *(float)* – angle in radians.

**to_matrix()**
Return a matrix representation of the euler.

Returns A 3x3 rotation matrix representation of the euler.

Return type *Matrix*

**to_quaternion()**
Return a quaternion representation of the euler.

Returns Quaternion representation of the euler.

Return type *Quaternion*

**zero()**
Set all values to zero.

**is_frozen**
True when this object has been frozen (read-only).

Type boolean

**is_wrapped**
True when this object wraps external data (read-only).

Type boolean

**order**
Euler rotation order.

Type string in ['XYZ', 'XZY', 'YXZ', 'YZX', 'ZXY', 'ZYX']

**owner**
The item this is wrapping or None (read-only).

**x**
Euler axis angle in radians.

Type float

**y**
Euler axis angle in radians.

Type float

**z**
Euler axis angle in radians.

Type float

### 10.8 Matrix

**class** *mathutils.Matrix([rows])*
This object gives access to Matrices in Blender, supporting square and rectangular matrices from 2x2 up to 4x4.

**Parameters**
rows *(2d number sequence)* – Sequence of rows. When ommitted, a 4x4 identity matrix is constructed.
import mathutils
import math

# create a location matrix
mat_loc = mathutils.Matrix.Translation((2.0, 3.0, 4.0))

# create an identity matrix
mat_sca = mathutils.Matrix.Scale(0.5, 4, (0.0, 0.0, 1.0))

# create a rotation matrix
mat_rot = mathutils.Matrix.Rotation(math.radians(45.0), 4, 'X')

# combine transformations
mat_out = mat_loc * mat_rot * mat_sca
print(mat_out)

# extract components back out of the matrix
loc, rot, sca = mat_out.decompose()
print(loc, rot, sca)

# it can also be useful to access components of a matrix directly
mat = mathutils.Matrix()
mat[0][0], mat[1][0], mat[2][0] = 0.0, 1.0, 2.0
mat[0][0:3] = 0.0, 1.0, 2.0

# each item in a matrix is a vector so vector utility functions can be used
mat[0].xyz = 0.0, 1.0, 2.0

classmethod Identity(size)
    Create an identity matrix.
    Parameters size(int) – The size of the identity matrix to construct [2, 4].
    Returns A new identity matrix.
    Return type Matrix

classmethod OrthoProjection(axis, size)
    Create a matrix to represent an orthographic projection.
    Parameters
    • axis (string or Vector) – Can be any of the following: ['X', 'Y', 'XY', 'XZ', 'YZ'],
      where a single axis is for a 2D matrix. Or a vector for an arbitrary axis
    • size(int) – The size of the projection matrix to construct [2, 4].
    Returns A new projection matrix.
    Return type Matrix

classmethod Rotation(angle, size, axis)
    Create a matrix representing a rotation.
    Parameters
    • angle(float) – The angle of rotation desired, in radians.
    • size(int) – The size of the rotation matrix to construct [2, 4].
    • axis (string or Vector) – a string in ['X', 'Y', 'Z'] or a 3D Vector Object (optional when size is 2).
Returns  A new rotation matrix.

Return type  Matrix

classmethod Scale (factor, size, axis)
Create a matrix representing a scaling.

Parameters

• factor (float) – The factor of scaling to apply.

• size (int) – The size of the scale matrix to construct [2, 4].

• axis (Vector) – Direction to influence scale. (optional).

Returns  A new scale matrix.

Return type  Matrix

classmethod Shear (plane, size, factor)
Create a matrix to represent an shear transformation.

Parameters

• plane (string) – Can be any of the following: ['X', 'Y', 'XY', 'XZ', 'YZ'], where a single axis is for a 2D matrix only.

• size (int) – The size of the shear matrix to construct [2, 4].

• factor (float or float pair) – The factor of shear to apply. For a 3 or 4 size matrix pass a pair of floats corresponding with the plane axis.

Returns  A new shear matrix.

Return type  Matrix

classmethod Translation (vector)
Create a matrix representing a translation.

Parameters  vector (Vector) – The translation vector.

Returns  An identity matrix with a translation.

Return type  Matrix

adjugate ()
Set the matrix to its adjugate.

Note:  When the matrix cannot be adjugated a ValueError exception is raised.

See also:


adjugated ()
Return an adjugated copy of the matrix.

Returns  the adjugated matrix.

Return type  Matrix

Note:  When the matrix cant be adjugated a ValueError exception is raised.
copy ()
Returns a copy of this matrix.

    Returns  an instance of itself

    Return type  Matrix

decompose ()
Return the translation, rotation and scale components of this matrix.

    Returns  trans, rot, scale triple.

    Return type  (Vector, Quaternion, Vector)
determinant ()
Return the determinant of a matrix.

    Returns  Return the determinant of a matrix.

    Return type  float

    See also:

freeze ()
Make this object immutable.

After this the object can be hashed, used in dictionaries & sets.

    Returns  An instance of this object.

identity ()
Set the matrix to the identity matrix.

    Note:  An object with a location and rotation of zero, and a scale of one will have an identity matrix.

    See also:

invert (fallback=None)
Set the matrix to its inverse.

    Parameters  fallback (Matrix) – Set the matrix to this value when the inverse cannot be calculated (instead of raising a ValueError exception).

    See also:

invert_safe ()
Set the matrix to its inverse, will never error. If degenerated (e.g. zero scale on an axis), add some epsilon to its diagonal, to get an invertible one. If tweaked matrix is still degenerated, set to the identity matrix instead.

    See also:

inverted (fallback=None)
Return an inverted copy of the matrix.

    Parameters  fallback (any) – return this when the inverse can’t be calculated (instead of raising a ValueError).
Returns the inverted matrix or fallback when given.

Return type Matrix

inverted_safe()
Return an inverted copy of the matrix, will never error. If degenerated (e.g. zero scale on an axis), add some epsilon to its diagonal, to get an invertible one. If tweaked matrix is still degenerated, return the identity matrix instead.

Returns the inverted matrix.

Return type Matrix

lerp(other, factor)
Returns the interpolation of two matrices.

Parameters

• other (Matrix) – value to interpolate with.
• factor (float) – The interpolation value in [0.0, 1.0].

Returns The interpolated matrix.

Return type Matrix

normalize()
Normalize each of the matrix columns.

normalized()
Return a column normalized matrix

Returns a column normalized matrix

Return type Matrix

resize_4x4()
Resize the matrix to 4x4.

rotate(other)
Rotates the matrix by another mathutils value.

Parameters other (Euler, Quaternion or Matrix) – rotation component of mathutils value

Note: If any of the columns are not unit length this may not have desired results.

to_3x3()
Return a 3x3 copy of this matrix.

Returns a new matrix.

Return type Matrix

to_4x4()
Return a 4x4 copy of this matrix.

Returns a new matrix.

Return type Matrix

to_euler(order, euler_compat)
Return an Euler representation of the rotation matrix (3x3 or 4x4 matrix only).

Parameters
• **order**(string) – Optional rotation order argument in ['XYZ', 'XZY', 'YXZ', 'YZX', 'ZXY', 'ZYX'].

• **euler_compat**(Euler) – Optional euler argument the new euler will be made compatible with (no axis flipping between them). Useful for converting a series of matrices to animation curves.

  Returns  Euler representation of the matrix.

  Return type  Euler

to_quaternion()  
Return a quaternion representation of the rotation matrix.

  Returns  Quaternion representation of the rotation matrix.

  Return type  Quaternion

to_scale()  
Return the scale part of a 3x3 or 4x4 matrix.

  Returns  Return the scale of a matrix.

  Return type  Vector

Note:  This method does not return a negative scale on any axis because it is not possible to obtain this data from the matrix alone.

to_translation()  
Return the translation part of a 4 row matrix.

  Returns  Return the translation of a matrix.

  Return type  Vector

transpose()  
Set the matrix to its transpose.

  See also:  

transposed()  
Return a new, transposed matrix.

  Returns  a transposed matrix

  Return type  Matrix

zero()  
Set all the matrix values to zero.

  Return type  Matrix

col  
Access the matix by columns, 3x3 and 4x4 only, (read-only).

  Type  Matrix Access

is_frozen  
True when this object has been frozen (read-only).

  Type  boolean
**is_negative**
True if this matrix results in a negative scale, 3x3 and 4x4 only, (read-only).

*Type* bool

**is_orthogonal**
True if this matrix is orthogonal, 3x3 and 4x4 only, (read-only).

*Type* bool

**is_orthogonal_axis_vectors**
True if this matrix has got orthogonal axis vectors, 3x3 and 4x4 only, (read-only).

*Type* bool

**is_wrapped**
True when this object wraps external data (read-only).

*Type* boolean

**median_scale**
The average scale applied to each axis (read-only).

*Type* float

**owner**
The item this is wrapping or None (read-only).

**row**
Access the matix by rows (default), (read-only).

*Type* Matrix Access

**translation**
The translation component of the matrix.

*Type* Vector

### 10.9 Quaternion

**class** `mathutils.Quaternion([seq[, angle]])`
This object gives access to Quaternions in Blender.

*Parameters*

- `seq` (*Vector*) – size 3 or 4
- `angle` (*float*) – rotation angle, in radians

The constructor takes arguments in various forms:

- `()` *no args*  Create an identity quaternion
- `(wxyz)` Create a quaternion from a \((w, x, y, z)\) vector.
- `(exponential_map)` Create a quaternion from a 3d exponential map vector.

**See also:**
- `to_exponential_map()`
- `(axis, angle)` Create a quaternion representing a rotation of `angle` radians over `axis`.

**See also:**
import mathutils
import math

# a new rotation 90 degrees about the Y axis
quat_a = mathutils.Quaternion((0.7071068, 0.0, 0.7071068, 0.0))

# passing values to Quaternion's directly can be confusing so axis, angle
# is supported for initializing too
quat_b = mathutils.Quaternion((0.0, 1.0, 0.0), math.radians(90.0))

print("Check quaternions match", quat_a == quat_b)

# like matrices, quaternions can be multiplied to accumulate rotational values
quat_a = mathutils.Quaternion((0.0, 1.0, 0.0), math.radians(90.0))
quat_b = mathutils.Quaternion((0.0, 0.0, 1.0), math.radians(45.0))
quat_out = quat_a * quat_b

# print the quat, euler degrees for mere mortals and (axis, angle)
print("Final Rotation:")
print(quat_out)
print("%.2f, %.2f, %.2f", tuple(math.degrees(a) for a in quat_out.to_euler()))
print("(%(%.2f, %.2f, %.2f), %.2f)", (quat_out.axis[:], math.degrees(quat_out.angle), ))

# multiple rotations can be interpolated using the exponential map
quat_c = mathutils.Quaternion((1.0, 0.0, 0.0), math.radians(15.0))
exp_avg = (quat_a.to_exponential_map() +
          quat_b.to_exponential_map() +
          quat_c.to_exponential_map()) / 3.0
quat_avg = mathutils.Quaternion(exp_avg)
print("Average rotation:")
print(quat_avg)

conjugate()
    Set the quaternion to its conjugate (negate x, y, z).

conjugated()
    Return a new conjugated quaternion.
    Returns a new quaternion.
    Return type Quaternion

copy()
    Returns a copy of this quaternion.
    Returns A copy of the quaternion.
    Return type Quaternion

Note: use this to get a copy of a wrapped quaternion with no reference to the original data.

cross(other)
    Return the cross product of this quaternion and another.
    Parameters other (Quaternion) – The other quaternion to perform the cross product with.
    Returns The cross product.
Return type: Quaternion

dot (other)
Return the dot product of this quaternion and another.

Parameters

other (Quaternion) – The other quaternion to perform the dot product with.

Returns The dot product.

Return type Quaternion

freeze()
Make this object immutable.

After this the object can be hashed, used in dictionaries & sets.

Returns An instance of this object.

identity()
Set the quaternion to an identity quaternion.

Return type Quaternion

invert()
Set the quaternion to its inverse.

inverted()
Return a new, inverted quaternion.

Returns the inverted value.

Return type Quaternion

negate()
Set the quaternion to its negative.

Return type Quaternion

normalize()
Normalize the quaternion.

normalized()
Return a new normalized quaternion.

Returns a normalized copy.

Return type Quaternion

rotate (other)
Rotates the quaternion by another mathutils value.

Parameters

other (Euler, Quaternion or Matrix) – rotation component of mathutils value

rotation_difference (other)
Returns a quaternion representing the rotational difference.

Parameters

other (Quaternion) – second quaternion.

Returns the rotational difference between the two quat rotations.

Return type Quaternion

slerp (other, factor)
Returns the interpolation of two quaternions.

Parameters
- **other** (*Quaternion*) – value to interpolate with.
- **factor** (*float*) – The interpolation value in [0.0, 1.0].

Returns The interpolated rotation.

Return type *Quaternion*

to_axis_angle()
Return the axis, angle representation of the quaternion.

Returns axis, angle.

Return type *(Vector, float)* pair

to_euler(order, euler_compat)
Return Euler representation of the quaternion.

Parameters
- **order** (*string*) – Optional rotation order argument in ['XYZ', 'XZY', 'YXZ', 'YZX', 'ZXY', 'ZYX'].
- **euler_compat** (*Euler*) – Optional euler argument the new euler will be made compatible with (no axis flipping between them). Useful for converting a series of matrices to animation curves.

Returns Euler representation of the quaternion.

Return type *Euler*

to_exponential_map()
Return the exponential map representation of the quaternion.

This representation consist of the rotation axis multiplied by the rotation angle. Such a representation is useful for interpolation between multiple orientations.

Returns exponential map.

Return type *Vector* of size 3

To convert back to a quaternion, pass it to the *Quaternion* constructor.

to_matrix()
Return a matrix representation of the quaternion.

Returns A 3x3 rotation matrix representation of the quaternion.

Return type *Matrix*

gle
Angle of the quaternion.

Type *float*

axis
Quaternion axis as a vector.

Type *Vector*

is_frozen
True when this object has been frozen (read-only).

Type *boolean*

is_wrapped
True when this object wraps external data (read-only).
Type boolean

**magnitude**
Size of the quaternion (read-only).
Type float

**owner**
The item this is wrapping or None (read-only).

**w**
Quaternion axis value.
Type float

**x**
Quaternion axis value.
Type float

**y**
Quaternion axis value.
Type float

**z**
Quaternion axis value.
Type float

## 10.10 Vector

**class** `mathutils.Vector(seq)`
This object gives access to Vectors in Blender.

**Parameters seq** *(sequence of numbers)* – Components of the vector, must be a sequence of at least two

```python
import mathutils

# zero length vector
vec = mathutils.Vector((0.0, 0.0, 1.0))

# unit length vector
vec_a = vec.normalized()

vec_b = mathutils.Vector((0.0, 1.0, 2.0))
vec2d = mathutils.Vector((1.0, 2.0))
vec3d = mathutils.Vector((1.0, 0.0, 0.0))
vec4d = vec_a.to_4d()

# other mathutils types
quat = mathutils.Quaternion()
matrix = mathutils.Matrix()

# Comparison operators can be done on Vector classes:

# (In)equality operators == and != test component values, e.g. 1,2,3 != 3,2,1
vec_a == vec_b
```

(continues on next page)
vec_a != vec_b

# Ordering operators >, >=, > and <= test vector length.
vec_a > vec_b  
vec_a >= vec_b  
vec_a < vec_b  
vec_a <= vec_b

# Math can be performed on Vector classes
vec_a + vec_b  
vec_a - vec_b  
vec_a * vec_b  
vec_a * 10.0  
matrix * vec_a  
quat * vec_a  
vec_a + vec_b  
-vec_a

# You can access a vector object like a sequence
x = vec_a[0]  
len(vec)  
vec_a[:] = vec_b  
vec_a[:] = 1.0, 2.0, 3.0  
vec2d[:] = vec3d[:2]

# Vectors support 'swizzle' operations
# See https://en.wikipedia.org/wiki/Swizzling_(computer_graphics)
vec.xyz = vec.zyx  
vec.xy = vec4d.zw  
vec.xyz = vec4d.wzz  
vec4d.wxyz = vec.yyx

```python

classmethod Fill(size, fill=0.0)
    Create a vector of length size with all values set to fill.

    Parameters
    "size" (int) – The length of the vector to be created.
    "fill" (float) – The value used to fill the vector.

classmethod Linspace(start, stop, size)
    Create a vector of the specified size which is filled with linearly spaced values between start and stop values.

    Parameters
    "start" (int) – The start of the range used to fill the vector.
    "stop" (int) – The end of the range used to fill the vector.
    "size" (int) – The size of the vector to be created.

classmethod Range(start=0, stop, step=1)
    Create a filled with a range of values.

    Parameters
```

• **start** (*int*) – The start of the range used to fill the vector.
• **stop** (*int*) – The end of the range used to fill the vector.
• **step** (*int*) – The step between successive values in the vector.

**classmethod** Repeat (*vector, size*)
Create a vector by repeating the values in vector until the required size is reached.

**Parameters**
- **tuple** (*mathutils.Vector*) – The vector to draw values from.
- **size** (*int*) – The size of the vector to be created.

**angle** (*other, fallback=None*)
Return the angle between two vectors.

**Parameters**
- **other** (*Vector*) – another vector to compare the angle with
- **fallback** (*any*) – return this when the angle can’t be calculated (zero length vector), (instead of raising a `ValueError`).

**Returns** angle in radians or fallback when given

**Return type** float

**angle_signed** (*other, fallback*)
Return the signed angle between two 2D vectors (clockwise is positive).

**Parameters**
- **other** (*Vector*) – another vector to compare the angle with
- **fallback** (*any*) – return this when the angle can’t be calculated (zero length vector), (instead of raising a `ValueError`).

**Returns** angle in radians or fallback when given

**Return type** float

**copy** ()
Returns a copy of this vector.

**Returns** A copy of the vector.

**Return type** Vector

---

**Note:** use this to get a copy of a wrapped vector with no reference to the original data.

**cross** (*other*)
Return the cross product of this vector and another.

**Parameters** **other** (*Vector*) – The other vector to perform the cross product with.

**Returns** The cross product.

**Return type** *Vector* or float when 2D vectors are used

---

**Note:** both vectors must be 2D or 3D
dot \text{(}\textit{other}\text{)}
Return the dot product of this vector and another.

Parameters \text{other} \text{(Vector)} – The other vector to perform the dot product with.

Returns The dot product.

Return type Vector

freeze()
Make this object immutable.

After this the object can be hashed, used in dictionaries & sets.

Returns An instance of this object.

lerp \text{(}\textit{other, factor}\text{)}
Returns the interpolation of two vectors.

Parameters

• \textit{other} \text{(Vector)} – value to interpolate with.

• \textit{factor} \text{(float)} – The interpolation value in \([0.0, 1.0]\).

Returns The interpolated vector.

Return type Vector

negate()
Set all values to their negative.

normalize()
Normalize the vector, making the length of the vector always 1.0.

\textbf{Warning: } Normalizing a vector where all values are zero has no effect.

\underline{Note: } Normalize works for vectors of all sizes, however 4D Vectors w axis is left untouched.

normalized()
Return a new, normalized vector.

Returns a normalized copy of the vector

Return type Vector

orthogonal()
Return a perpendicular vector.

Returns a new vector 90 degrees from this vector.

Return type Vector

\underline{Note: } the axis is undefined, only use when any orthogonal vector is acceptable.

project \text{(}\textit{other}\text{)}
Return the projection of this vector onto the \textit{other}.

Parameters \textit{other} \text{(Vector)} – second vector.

Returns the parallel projection vector
Return type `Vector`

**reflect** *(mirror)*
Return the reflection vector from the `mirror` argument.

**Parameters**
- **mirror** *(Vector)* – This vector could be a normal from the reflecting surface.

**Returns**
The reflected vector matching the size of this vector.

Return type `Vector`

**resize**(size=3)
Resize the vector to have size number of elements.

**resize_2d**()
Resize the vector to 2D (x, y).

**resize_3d**()
Resize the vector to 3D (x, y, z).

**resize_4d**()
Resize the vector to 4D (x, y, z, w).

**resized**(size=3)
Return a resized copy of the vector with size number of elements.

**Returns**
a new vector

Return type `Vector`

**rotate**(other)
Rotate the vector by a rotation value.

**Parameters**
- **other** *(Euler, Quaternion or Matrix)* – rotation component of mathutils value

**rotation_difference**(other)
Returns a quaternion representing the rotational difference between this vector and another.

**Parameters**
- **other** *(Vector)* – second vector.

**Returns**
the rotational difference between the two vectors.

Return type `Quaternion`

---

**Note:** 2D vectors raise an `AttributeError`.

**slerp**(other, factor, fallback=None)
Returns the interpolation of two non-zero vectors (spherical coordinates).

**Parameters**
- **other** *(Vector)* – value to interpolate with.
- **factor** *(float)* – The interpolation value typically in [0.0, 1.0].
- **fallback** *(any)* – return this when the vector can’t be calculated (zero length vector or direct opposites), (instead of raising a `ValueError`).

**Returns**
The interpolated vector.

Return type `Vector`

**to_2d**()
Return a 2d copy of the vector.
Returns a new vector

**Return type** *Vector*

**to_3d()**

Return a 3d copy of the vector.

Returns a new vector

**Return type** *Vector*

**to_4d()**

Return a 4d copy of the vector.

Returns a new vector

**Return type** *Vector*

**to_track_quat** (*track*, *up*)

Return a quaternion rotation from the vector and the track and up axis.

**Parameters**

- *track* (*string*) – Track axis in ['X', 'Y', 'Z', '-X', '-Y', '-Z'].

- *up* (*string*) – Up axis in ['X', 'Y', 'Z'].

Returns rotation from the vector and the track and up axis.

**Return type** *Quaternion*

**to_tuple** (*precision*=-1)

Return this vector as a tuple with.

**Parameters**

- *precision* (*int*) – The number to round the value to in [-1, 21].

Returns the values of the vector rounded by *precision*

**Return type** tuple

**zero()**

Set all values to zero.

**is_frozen**

True when this object has been frozen (read-only).

**Type** boolean

**is_wrapped**

True when this object wraps external data (read-only).

**Type** boolean

**length**

Vector Length.

**Type** float

**length_squared**

Vector length squared (v.dot(v)).

**Type** float

**magnitude**

Vector Length.

**Type** float
owner
The item this is wrapping or None (read-only).

w
Vector W axis (4D Vectors only).
  Type float

ww
Undocumented

www
Undocumented

www
Undocumented

wwwx
Undocumented

wwwy
Undocumented

wwwz
Undocumented

wwx
Undocumented

wwxw
Undocumented

wwxx
Undocumented

wwxy
Undocumented

wwxz
Undocumented

wwy
Undocumented

wwyw
Undocumented

wwyx
Undocumented

wwyy
Undocumented

wwyz
Undocumented

wwz
Undocumented

www
Undocumented

wwzw
Undocumented

wwzx
Undocumented
Undocumented
undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented
Undocumented

10.10. Vector
wxzy
Undocumented
wxzz
Undocumented
wy
Undocumented
wyw
Undocumented
wyww
Undocumented
wywx
Undocumented
wywy
Undocumented
wywz
Undocumented
wyx
Undocumented
wyxw
Undocumented
wyxx
Undocumented
wyxy
Undocumented
wyxz
Undocumented
wyx
Undocumented
wyw
Undocumented
wywx
Undocumented
wywy
Undocumented
wywz
Undocumented
wyx
Undocumented
wyxw
Undocumented
wyxx
Undocumented
wyxy
Undocumented
wyxz
Undocumented
wyy
Undocumented
wyyw
Undocumented
wyyx
Undocumented
wyyy
Undocumented
wyzz
Undocumented
wyzz
Undocumented
wyzz
Undocumented
wyzz
Undocumented
wyzz
Undocumented
wyzz
Undocumented
wyzz
Undocumented
**wzzy**
Undocumented

**wzzz**
Undocumented

**x**
Vector X axis.

  **Type** float

**xw**
Undocumented

**xww**
Undocumented

**xwww**
Undocumented

**xwwx**
Undocumented

**xwwy**
Undocumented

**xwwz**
Undocumented

**xwx**
Undocumented

**xwxw**
Undocumented

**xwxx**
Undocumented

**xwxy**
Undocumented

**xwxz**
Undocumented

**xwy**
Undocumented

**xwyw**
Undocumented

**xwyx**
Undocumented

**xwyy**
Undocumented

**xwyz**
Undocumented

**xwz**
Undocumented

**xwzw**
Undocumented

---

**366 Chapter 10. Math Types & Utilities (mathutils)**
xwzx
  Undocumented
xwzy
  Undocumented
xwzz
  Undocumented
xx
  Undocumented
xxw
  Undocumented
xxww
  Undocumented
xxwx
  Undocumented
xxwy
  Undocumented
xxwz
  Undocumented
xxx
  Undocumented
xxxw
  Undocumented
xxxx
  Undocumented
xxxxx
  Undocumented
xxxxy
  Undocumented
xxxxz
  Undocumented
xy
  Undocumented
xyw
  Undocumented
xyx
  Undocumented
xyy
  Undocumented
xyz
  Undocumented
xz
  Undocumented
xzw
  Undocumented
xxzx
Undocumented

xxzy
Undocumented

xxzz
Undocumented

xy
Undocumented

xyw
Undocumented

xyww
Undocumented

xywx
Undocumented

xywy
Undocumented

xywz
Undocumented

xyx
Undocumented

xyxw
Undocumented

xyxx
Undocumented

xyxy
Undocumented

xyxz
Undocumented

xyy
Undocumented

xyyw
Undocumented

xyyx
Undocumented

xyyy
Undocumented

xyyz
Undocumented

xyz
Undocumented

xyzw
Undocumented
xyzx
Undocumented

xyzy
Undocumented

xyzz
Undocumented

xz
Undocumented

xzw
Undocumented

xzww
Undocumented

xzwx
Undocumented

xzwy
Undocumented

xzwz
Undocumented

xzx
Undocumented

xzwx
Undocumented

xzz
Undocumented

xzy
Undocumented

xzyw
Undocumented

xzyx
Undocumented

xzyy
Undocumented

xzyz
Undocumented

xzz
Undocumented

xzzw
Undocumented
Undocumented

Vector Y axis.

Type float

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented
ywzw
Undocumented
ywzx
Undocumented
ywzy
Undocumented
ywzz
Undocumented
yx
Undocumented
yxw
Undocumented
yxww
Undocumented
yxwx
Undocumented
yxwy
Undocumented
yxwz
Undocumented
yxx
Undocumented
yxxw
Undocumented
yxxx
Undocumented
yxxz
Undocumented
yxy
Undocumented
yxyw
Undocumented
yxyx
Undocumented
yxyy
Undocumented
yxyz
Undocumented
yxz
Undocumented
Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented
Undocumented
Undocumented

Undocumented

Undocumented

Undocumented

Vector Z axis (3D Vectors only).

**Type** float

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented
zwz
Undocumented

zwzw
Undocumented

zwzx
Undocumented

zwzy
Undocumented

zwzz
Undocumented

zx
Undocumented

zxw
Undocumented

zxww
Undocumented

zxwx
Undocumented

zxwy
Undocumented

zxwz
Undocumented

zxzx
Undocumented

zxww
Undocumented

zxxx
Undocumented

zxx
Undocumented

zxxw
Undocumented

zxxz
Undocumented

zxy
Undocumented

zxyw
Undocumented

zxyx
Undocumented

zxyy
Undocumented

zxyz
Undocumented

10.10. Vector
Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented

Undocumented
zyz
Undocumented
zyzw
Undocumented
zyzx
Undocumented
zyzy
Undocumented
zyzz
Undocumented
zz
Undocumented
zzw
Undocumented
zzww
Undocumented
zzwx
Undocumented
zzwy
Undocumented
zzwz
Undocumented
zzx
Undocumented
zzxw
Undocumented
zzxx
Undocumented
zzxy
Undocumented
zzxz
Undocumented
zzy
Undocumented
zzyw
Undocumented
zzyx
Undocumented
zzyy
Undocumented
zzyz
Undocumented
zzz
Undocumented

zzzw
Undocumented

zzzx
Undocumented

zzzy
Undocumented

zzzz
Undocumented
CHAPTER 11

OpenGL Wrapper (bgl)

This module wraps OpenGL constants and functions, making them available from within Blender Python.

The complete list can be retrieved from the module itself, by listing its contents: dir(bgl). A simple search on the web can point to more than enough material to teach OpenGL programming, from books to many collections of tutorials.

Here is a comprehensive list of books (non free). The arcsynthesis tutorials is one of the best resources to learn modern OpenGL and g-truc offers a set of extensive examples, including advanced features.

Note: You can use the Image type to load and set textures. See Image.gl_load and Image.gl_load, for example.

**glAccum** *(op, value)*:
Operate on the accumulation buffer.

See also:
OpenGL Docs

Parameters

- **op** *(Enumerated constant)* – The accumulation buffer operation.
- **value** *(float)* – a value used in the accumulation buffer operation.

**glAlphaFunc** *(func, ref)*:
Specify the alpha test function.

See also:
OpenGL Docs

Parameters

- **func** *(Enumerated constant)* – Specifies the alpha comparison function.
- **ref** *(float)* – The reference value that incoming alpha values are compared to. Clamped between 0 and 1.
glAreTexturesResident(n, textures, residences):
  Determine if textures are loaded in texture memory

  See also:
  OpenGL Docs

  Parameters
  • n (int) – Specifies the number of textures to be queried.
  • textures (bgl.Buffer object I{type GL_INT}) – Specifies an array containing the names of the textures to be queried
  • residences (bgl.Buffer object I{type GL_INT}(boolean)) – An array in which the texture residence status in returned. The residence status of a texture named by an element of textures is returned in the corresponding element of residences.

glBegin(mode):
  Delimit the vertices of a primitive or a group of like primatives

  See also:
  OpenGL Docs

  Parameters mode (Enumerated constant) – Specifies the primitive that will be create from vertices between glBegin and glEnd.

glBindTexture(target, texture):
  Bind a named texture to a texturing target

  See also:
  OpenGL Docs

  Parameters
  • target (Enumerated constant) – Specifies the target to which the texture is bound.
  • texture (unsigned int) – Specifies the name of a texture.

glBitmap(width, height, xorig, yorig, xmove, ymove, bitmap):
  Draw a bitmap

  See also:
  OpenGL Docs

  Parameters
  • height (width,) – Specify the pixel width and height of the bitmap image.
  • yorig (xorig,) – Specify the location of the origin in the bitmap image. The origin is measured from the lower left corner of the bitmap, with right and up being the positive axes.
  • ymove (xmove,) – Specify the x and y offsets to be added to the current raster position after the bitmap is drawn.
  • bitmap (bgl.Buffer object I{type GL_BYTE}) – Specifies the address of the bitmap image.

glBlendFunc(sfactor, dfactor):
  Specify pixel arithmetic

  See also:
OpenGL Docs

Parameters

- **sfactor** (*Enumerated constant*) – Specifies how the red, green, blue, and alpha source blending factors are computed.
- **dfactor** (*Enumerated constant*) – Specifies how the red, green, blue, and alpha destination blending factors are computed.

**glCallList(list):**
Execute a display list

See also:
OpenGL Docs

Parameters list (*unsigned int*) – Specifies the integer name of the display list to be executed.

**glCallLists(n, type, lists):**
Execute a list of display lists

See also:
OpenGL Docs

Parameters

- **n** (*int*) – Specifies the number of display lists to be executed.
- **type** (*Enumerated constant*) – Specifies the type of values in lists.
- **lists** (*bgl.Buffer* object) – Specifies the address of an array of name offsets in the display list. The pointer type is void because the offsets can be bytes, shorts, ints, or floats, depending on the value of type.

**glClear(mask):**
Clear buffers to preset values

See also:
OpenGL Docs

Parameters mask (*Enumerated constant (s)*) – Bitwise OR of masks that indicate the buffers to be cleared.

**glClearAccum(red, green, blue, alpha):**
Specify clear values for the accumulation buffer

See also:
OpenGL Docs

Parameters green, blue, alpha (*red*) – Specify the red, green, blue, and alpha values used when the accumulation buffer is cleared. The initial values are all 0.

**glClearColor(red, green, blue, alpha):**
Specify clear values for the color buffers

See also:
OpenGL Docs
Parameters `green`, `blue`, `alpha` (*red,*) — Specify the red, green, blue, and alpha values used when the color buffers are cleared. The initial values are all 0.

`glClearDepth(depth):`
Specify the clear value for the depth buffer

See also:
OpenGL Docs

Parameters `depth` (*int*) — Specifies the depth value used when the depth buffer is cleared. The initial value is 1.

`glClearIndex(c):`
Specify the clear value for the color index buffers

See also:
OpenGL Docs

Parameters `c` (*float*) — Specifies the index used when the color index buffers are cleared. The initial value is 0.

`glClearStencil(s):`
Specify the clear value for the stencil buffer

See also:
OpenGL Docs

Parameters `s` (*int*) — Specifies the index used when the stencil buffer is cleared. The initial value is 0.

`glClipPlane (plane, equation):`
Specify a plane against which all geometry is clipped

See also:
OpenGL Docs

Parameters
- `plane` (*Enumerated constant*) — Specifies which clipping plane is being positioned.
- `equation` (*bgl.Buffer* object I{type GL_FLOAT}(*double*)) — Specifies the address of an array of four double-precision floating-point values. These values are interpreted as a plane equation.

`glColor (red, green, blue, alpha):`

B{`glColor3b`, `glColor3d`, `glColor3f`, `glColor3i`, `glColor3s`, `glColor3ub`, `glColor3us`, `glColor3vb`, `glColor3vs`, `glColor4b`, `glColor4d`, `glColor4f`, `glColor4i`, `glColor4s`, `glColor4ub`, `glColor4ui`, `glColor4us`, `glColor4vb`, `glColor4vs`, `glColor3fv`, `glColor3iv`, `glColor3sv`, `glColor3ubv`, `glColor3uiv`, `glColor3usv`, `glColor4fv`, `glColor4iv`, `glColor4sv`, `glColor4ubv`, `glColor4uiv`, `glColor4usv`}

Set a new color.

See also:
OpenGL Docs

Parameters
- `green`, `blue` (*red,*) — Specify new red, green, and blue values for the current color.
• **alpha** – Specifies a new alpha value for the current color. Included only in the four-argument `glColor4` commands. (With ‘4’ colors only)

```cpp
void glColorMask(red, green, blue, alpha);
```

Enable and disable writing of frame buffer color components

**See also:**
OpenGL Docs

**Parameters**

- **green, blue, alpha** *(red,)* – Specify whether red, green, blue, and alpha can or cannot be written into the frame buffer. The initial values are all GL_TRUE, indicating that the color components can be written.

```cpp
void glColorMaterial(face, mode);
```

Cause a material color to track the current color

**See also:**
OpenGL Docs

**Parameters**

- **face** *(Enumerated constant)* – Specifies whether front, back, or both front and back material parameters should track the current color.
- **mode** *(Enumerated constant)* – Specifies which of several material parameters track the current color.

```cpp
void glCopyPixels(x, y, width, height, type);
```

Copy pixels in the frame buffer

**See also:**
OpenGL Docs

**Parameters**

- **y** *(x,)* – Specify the window coordinates of the lower left corner of the rectangular region of pixels to be copied.
- **width, height** – Specify the dimensions of the rectangular region of pixels to be copied. Both must be non-negative.
- **type** *(Enumerated constant)* – Specifies whether color values, depth values, or stencil values are to be copied.

```cpp
def glCopyTexImage2D(target, level, internalformat, x, y, width, height, border):
```

Copy pixels into a 2D texture image

**See also:**
OpenGL Docs

**Parameters**

- **target** *(Enumerated constant)* – Specifies the target texture.
- **level** *(int)* – Specifies the level-of-detail number. Level 0 is the base image level. Level n is the nth mipmap reduction image.
- **internalformat** *(int)* – Specifies the number of color components in the texture.
• \( y(x, \cdot) \) – Specify the window coordinates of the first pixel that is copied from the frame buffer. This location is the lower left corner of a rectangular block of pixels.

• \textit{width} (\textit{int}) – Specifies the width of the texture image. Must be \( 2n+2(\text{border}) \) for some integer \( n \). All implementations support texture images that are at least 64 texels wide.

• \textit{height} (\textit{int}) – Specifies the height of the texture image. Must be \( 2m+2(\text{border}) \) for some integer \( m \). All implementations support texture images that are at least 64 texels high.

• \textit{border} (\textit{int}) – Specifies the width of the border. Must be either 0 or 1.

\texttt{glCullFace(mode)}:
Specify whether front- or back-facing facets can be culled

\texttt{See also:}
OpenGL Docs

\begin{itemize}
\item \texttt{Parameters mode (Enumerated constant)} – Specifies whether front- or back-facing facets are candidates for culling.
\end{itemize}

\texttt{glDeleteLists(list, range)}:
Delete a contiguous group of display lists

\texttt{See also:}
OpenGL Docs

\begin{itemize}
\item \texttt{Parameters}
\begin{itemize}
\item \texttt{list} (\texttt{unsigned int}) – Specifies the integer name of the first display list to delete
\item \texttt{range} (\textit{int}) – Specifies the number of display lists to delete
\end{itemize}
\end{itemize}

\texttt{glDeleteTextures(n, textures)}:
Delete named textures

\texttt{See also:}
OpenGL Docs

\begin{itemize}
\item \texttt{Parameters}
\begin{itemize}
\item \texttt{n} (\textit{int}) – Specifies the number of textures to be deleted
\item \texttt{textures} (\texttt{bgl.Buffer I(GL_INT)}) – Specifies an array of textures to be deleted
\end{itemize}
\end{itemize}

\texttt{glDepthFunc(func)}:
Specify the value used for depth buffer comparisons

\texttt{See also:}
OpenGL Docs

\begin{itemize}
\item \texttt{Parameters func (Enumerated constant)} – Specifies the depth comparison function.
\end{itemize}

\texttt{glDepthMask(flag)}:
Enable or disable writing into the depth buffer

\texttt{See also:}
OpenGL Docs
Parameters flag(int boolean) – Specifies whether the depth buffer is enabled for writing. If flag is GL_FALSE, depth buffer writing is disabled. Otherwise, it is enabled. Initially, depth buffer writing is enabled.

glDepthRange(zNear, zFar):
Specify mapping of depth values from normalized device coordinates to window coordinates

See also:
OpenGL Docs

Parameters

- zNear(int) – Specifies the mapping of the near clipping plane to window coordinates. The initial value is 0.
- zFar(int) – Specifies the mapping of the far clipping plane to window coordinates. The initial value is 1.

glDisable(cap):
Disable server-side GL capabilities

See also:
OpenGL Docs

Parameters cap(Enumerated constant) – Specifies a symbolic constant indicating a GL capability.

glDrawBuffer(mode):
Specify which color buffers are to be drawn into

See also:
OpenGL Docs

Parameters mode(Enumerated constant) – Specifies up to four color buffers to be drawn into.

glDrawPixels(width, height, format, type, pixels):
Write a block of pixels to the frame buffer

See also:
OpenGL Docs

Parameters

- height(width,) – Specify the dimensions of the pixel rectangle to be written into the frame buffer.
- format(Enumerated constant) – Specifies the format of the pixel data.
- type(Enumerated constant) – Specifies the data type for pixels.
- pixels(bgl.Buffer object) – Specifies a pointer to the pixel data.

glEdgeFlag (flag):
B{glEdgeFlag, glEdgeFlagv}
Flag edges as either boundary or non-boundary

See also:
OpenGL Docs
Parameters **flag** *(Depends of function prototype)* – Specifies the current edge flag value. The initial value is GL_TRUE.

**glEnable** *(cap)* :
Enable server-side GL capabilities

See also:
OpenGL Docs

**Parameters** **cap** *(Enumerated constant)* – Specifies a symbolic constant indicating a GL capability.

glEnd() :
Delimit the vertices of a primitive or group of like primitives

See also:
OpenGL Docs

glEndList() :
Create or replace a display list

See also:
OpenGL Docs

glEvalCoord *(u,v)* :
B{glEvalCoord1d, glEvalCoord1f, glEvalCoord2d, glEvalCoord2f, glEvalCoord1dv, glEvalCoord1fv, glEvalCoord2dv, glEvalCoord2fv}
Evaluate enabled one- and two-dimensional maps

See also:
OpenGL Docs

**Parameters**

- **u** *(Depends on function prototype.)* – Specifies a value that is the domain coordinate u to the basis function defined in a previous glMap1 or glMap2 command. If the function prototype ends in 'v' then u specifies a pointer to an array containing either one or two domain coordinates. The first coordinate is u. The second coordinate is v, which is present only in glEvalCoord2 versions.

- **v** *(Depends on function prototype. (only with '2' prototypes)* – Specifies a value that is the domain coordinate v to the basis function defined in a previous glMap2 command. This argument is not present in a glEvalCoord1 command.

glEvalMesh *(mode, i1, i2)* :
B{glEvalMesh1 or glEvalMesh2}
Compute a one- or two-dimensional grid of points or lines

See also:
OpenGL Docs

**Parameters**

- **mode** *(Enumerated constant)* – In glEvalMesh1, specifies whether to compute a one-dimensional mesh of points or lines.

- **i2** *(i1,)* – Specify the first and last integer values for the grid domain variable i.
glEvalPoint (i, j):
B{glEvalPoint1 and glEvalPoint2}
Generate and evaluate a single point in a mesh
See also:
OpenGL Docs

Parameters
• i (int) – Specifies the integer value for grid domain variable i.
• j (int (only with '2' prototypes)) – Specifies the integer value for grid domain variable j (glEvalPoint2 only).

glFeedbackBuffer (size, type, buffer):
Controls feedback mode
See also:
OpenGL Docs

Parameters
• size (int) – Specifies the maximum number of values that can be written into buffer.
• type (Enumerated constant) – Specifies a symbolic constant that describes the information that will be returned for each vertex.
• buffer (bgl.Buffer object I{GL_FLOAT}) – Returns the feedback data.

glFinish():
Block until all GL execution is complete
See also:
OpenGL Docs

glFlush():
Force Execution of GL commands in finite time
See also:
OpenGL Docs

glFog (pname, param):
B{glFogf, glFogi, glFogfv, glFogiv}
Specify fog parameters
See also:
OpenGL Docs

Parameters
• pname (Enumerated constant) – Specifies a single-valued fog parameter. If the function prototype ends in ‘v’ specifies a fog parameter.
• param (Depends on function prototype.) – Specifies the value or values to be assigned to pname. GL_FOG_COLOR requires an array of four values. All other parameters accept an array containing only a single value.
glFrontFace(mode):
    Define front- and back-facing polygons

    See also:
    OpenGL Docs

    Parameters
    mode (Enumerated constant) – Specifies the orientation of front-facing polygons.

glFrustum(left, right, bottom, top, zNear, zFar):
    Multiply the current matrix by a perspective matrix

    See also:
    OpenGL Docs

    Parameters
    • right (left,) – Specify the coordinates for the left and right vertical clipping planes.
    • bottom (top,) – Specify the coordinates for the bottom and top horizontal clipping planes.
    • zFar (zNear,) – Specify the distances to the near and far depth clipping planes. Both distances must be positive.

glGenLists(range):
    Generate a contiguous set of empty display lists

    See also:
    OpenGL Docs

    Parameters
    range (int) – Specifies the number of contiguous empty display lists to be generated.

glGenTextures(n, textures):
    Generate texture names

    See also:
    OpenGL Docs

    Parameters
    • n (int) – Specifies the number of textures name to be generated.
    • textures (bgl.Buffer object I{type GL_INT}) – Specifies an array in which the generated textures names are stored.

glGet (pname, param):
    B{glGetBooleanv, glGetfloatv, glGetFloatv, glGetIntegerv}
    Return the value or values of a selected parameter

    See also:
    OpenGL Docs

    Parameters
    • pname (Enumerated constant) – Specifies the parameter value to be returned.
    • param (Depends on function prototype.) – Returns the value or values of the specified parameter.
**glGetClipPlane** (plane, equation):
Return the coefficients of the specified clipping plane

See also:
OpenGL Docs

**Parameters**

- **plane** (*Enumerated constant*) – Specifies a clipping plane. The number of clipping planes depends on the implementation, but at least six clipping planes are supported. They are identified by symbolic names of the form GL_CLIP_PLANEi where 0 < i < GL_MAX_CLIP_PLANES.
- **equation** (*bgl.Buffer object* I{type GL_FLOAT}) – Returns four float (double)-precision values that are the coefficients of the plane equation of plane in eye coordinates. The initial value is (0, 0, 0, 0).

**glGetError()**:
Return error information

See also:
OpenGL Docs

**glGetLight** (light, pname, params):
B{glGetLightfv and glGetLightiv}
Return light source parameter values

See also:
OpenGL Docs

**Parameters**

- **light** (*Enumerated constant*) – Specifies a light source. The number of possible lights depends on the implementation, but at least eight lights are supported. They are identified by symbolic names of the form GL_LIGHTi where 0 < i < GL_MAX_LIGHTS.
- **pname** (*Enumerated constant*) – Specifies a light source parameter for light.
- **params** (*bgl.Buffer object. Depends on function prototype.*) – Returns the requested data.

**glGetMap** (target, query, v):
B{glGetMapdv, glGetMapfv, glGetMapiv}
Return evaluator parameters

See also:
OpenGL Docs

**Parameters**

- **target** (*Enumerated constant*) – Specifies the symbolic name of a map.
- **query** (*Enumerated constant*) – Specifies which parameter to return.
- **v** (*bgl.Buffer object. Depends on function prototype.*) – Returns the requested data.

**glGetMaterial** (face, pname, params):
B{glGetMaterialfv, glGetMaterialiv}
Return material parameters

See also:
OpenGL Docs

Parameters

- **face** (*Enumerated constant*) – Specifies which of the two materials is being queried, representing the front and back materials, respectively.
- **pname** (*Enumerated constant*) – Specifies the material parameter to return.
- **params** (*bgl.Buffer* object. Depends on function prototype.) – Returns the requested data.

`glGetPixelMap (map, values):`  
B{`glGetPixelMapfv`, `glGetPixelMapuiv`, `glGetPixelMapusv`}

Return the specified pixel map

See also:
OpenGL Docs

Parameters

- **map** (*Enumerated constant*) – Specifies the name of the pixel map to return.
- **values** (*bgl.Buffer* object. Depends on function prototype.) – Returns the pixel map contents.

`glGetPolygonStipple (mask):`  
Return the polygon stipple pattern

See also:
OpenGL Docs

Parameters **mask** (*bgl.Buffer* object I{type GL_BYTE}) – Returns the stipple pattern. The initial value is all 1’s.

`glGetString (name):`  
Return a string describing the current GL connection

See also:
OpenGL Docs

Parameters **name** (*Enumerated constant*) – Specifies a symbolic constant.

`glGetTexEnv (target, pname, params):`  
B{`glGetTexEnvfv`, `glGetTexEnviv`}

Return texture environment parameters

See also:
OpenGL Docs

Parameters

- **target** (*Enumerated constant*) – Specifies a texture environment. Must be GL_TEXTURE_ENV.
• **pname** (*Enumerated constant*) – Specifies the symbolic name of a texture environment parameter.

• **params** (*bgl.Buffer* object. Depends on function prototype.) – Returns the requested data.

`glGetTexGen (coord, pname, params):`

Return texture coordinate generation parameters

See also:
OpenGL Docs

**Parameters**

• **coord** (*Enumerated constant*) – Specifies a texture coordinate.

• **pname** (*Enumerated constant*) – Specifies the symbolic name of the value(s) to be returned.

• **params** (*bgl.Buffer* object. Depends on function prototype.) – Returns the requested data.

`glGetTexImage (target, level, format, type, pixels):`

Return a texture image

See also:
OpenGL Docs

**Parameters**

• **target** (*Enumerated constant*) – Specifies which texture is to be obtained.

• **level** (*int*) – Specifies the level-of-detail number of the desired image. Level 0 is the base image level. Level n is the nth mipmap reduction image.

• **format** (*Enumerated constant*) – Specifies a pixel format for the returned data.

• **type** (*Enumerated constant*) – Specifies a pixel type for the returned data.

• **pixels** (*bgl.Buffer* object.) – Returns the texture image. Should be a pointer to an array of the type specified by type

`glGetTexLevelParameter (target, level, pname, params):`

B{glGetTexParameterfv, glGetTexParameteriv}

return texture parameter values for a specific level of detail

See also:
OpenGL Docs

**Parameters**

• **target** (*Enumerated constant*) – Specifies the symbolic name of the target texture.

• **level** (*int*) – Specifies the level-of-detail number of the desired image. Level 0 is the base image level. Level n is the nth mipmap reduction image.

• **pname** (*Enumerated constant*) – Specifies the symbolic name of a texture parameter.

• **params** (*bgl.Buffer* object. Depends on function prototype.) – Returns the requested data.
**glGetTexParameter (target, pname, params):**

B{glGetTexParameterfv, glGetTexParameteriv}

Return texture parameter values

**See also:**

OpenGL Docs

**Parameters**

- **target** (*Enumerated constant*) – Specifies the symbolic name of the target texture.
- **pname** (*Enumerated constant*) – Specifies the symbolic name the target texture.
- **params** (*bgl.Buffer object. Depends on function prototype.*) – Returns the texture parameters.

**glHint (target, mode):**

Specify implementation-specific hints

**See also:**

OpenGL Docs

**Parameters**

- **target** (*Enumerated constant*) – Specifies a symbolic constant indicating the behavior to be controlled.
- **mode** (*Enumerated constant*) – Specifies a symbolic constant indicating the desired behavior.

**glIndex (c):**

B{glIndexd, glIndexf, glIndexi, glIndexs, glIndexdv, glIndexfv, glIndexiv, glIndexsv}

Set the current color index

**See also:**

OpenGL Docs

**Parameters**

- **c** (*bgl.Buffer object. Depends on function prototype.*) – Specifies a pointer to a one element array that contains the new value for the current color index.

**glIndexMask (mask):**

Control the writing of individual bits in the color index buffers

**See also:**

OpenGL Docs

**Parameters**

- **mask** (*int*) – Specifies a bit mask to enable and disable the writing of individual bits in the color index buffers. Initially, the mask is all 1’s.

**glInitNames ():**

Initialize the name stack

**See also:**

OpenGL Docs
glIsEnabled(cap):
  Test whether a capability is enabled

  See also:
  OpenGL Docs

  Parameters cap (Enumerated constant) – Specifies a constant representing a GL capability.

glIsList(list):
  Determine if a name corresponds to a display-list

  See also:
  OpenGL Docs

  Parameters list (unsigned int) – Specifies a potential display-list name.

glIsTexture(texture):
  Determine if a name corresponds to a texture

  See also:
  OpenGL Docs

  Parameters texture (unsigned int) – Specifies a value that may be the name of a texture.

glLight (light, pname, param):
  B{glLightf, glLighti, glLightfv, glLightiv}

  Set the light source parameters

  See also:
  OpenGL Docs

  Parameters

    • light (Enumerated constant) – Specifies a light. The number of lights depends on the implementation, but at least eight lights are supported. They are identified by symbolic names of the form GL_LIGHTi where 0 < i < GL_MAX_LIGHTS.

    • pname (Enumerated constant) – Specifies a single-valued light source parameter for light.

    • param (Depends on function prototype.) – Specifies the value that parameter pname of light source light will be set to. If function prototype ends in ‘v’ specifies a pointer to the value or values that parameter pname of light source light will be set to.

glLightModel (pname, param):
  B{glLightModelf, glLightModeli, glLightModelfv, glLightModeliv}

  Set the lighting model parameters

  See also:
  OpenGL Docs

  Parameters

    • pname (Enumerated constant) – Specifies a single-value light model parameter.
\textbf{UPBGE Manual Documentation, Release latest}

- \textit{param} (\textit{Depends on function prototype.}) \textendash \textit{Specifies the value that param will be set to. If function prototype ends in ‘v’ specifies a pointer to the value or values that param will be set to.}

\texttt{glLineStipple(factor, pattern):}
\begin{itemize}
  \item \texttt{factor (int)} \textendash \textit{Specifies a multiplier for each bit in the line stipple pattern. If factor is 3, for example, each bit in the pattern is used three times before the next bit in the pattern is used. factor is clamped to the range \([1, 256]\) and defaults to 1.}
  \item \texttt{pattern (unsigned short int)} \textendash \textit{Specifies a 16-bit integer whose bit pattern determines which fragments of a line will be drawn when the line is rasterized. Bit zero is used first; the default pattern is all 1’s.}
\end{itemize}

\texttt{glLineWidth(width):}
\begin{itemize}
  \item \texttt{width (float)} \textendash \textit{Specifies the width of rasterized lines. The initial value is 1.}
\end{itemize}

\texttt{glListBase(base):}
\begin{itemize}
  \item \texttt{base (unsigned int)} \textendash \textit{Specifies an integer offset that will be added to glCallLists offsets to generate display-list names. The initial value is 0.}
\end{itemize}

\texttt{glLoadIdentity():}
\begin{itemize}
  \item \textit{Replace the current matrix with the identity matrix}
\end{itemize}

\texttt{glLoadMatrix (m):}
\begin{itemize}
  \item \textit{B{glLoadMatrixd, glLoadMatixf}}
  \item \textit{Replace the current matrix with the specified matrix}
\end{itemize}

\texttt{glLoadName(name):}
\begin{itemize}
  \item \textit{Load a name onto the name stack.}
\end{itemize}
OpenGL Docs

Parameters name (unsigned int) – Specifies a name that will replace the top value on the name stack.

glLogicOp (opcode):
Specify a logical pixel operation for color index rendering
See also:
OpenGL Docs

Parameters opcode (Enumerated constant) – Specifies a symbolic constant that selects a logical operation.

glMap1 (target, u1, u2, stride, order, points):
B{glMap1d, glMap1f}
Define a one-dimensional evaluator
See also:
OpenGL Docs

Parameters

• target (Enumerated constant) – Specifies the kind of values that are generated by the evaluator.

• u1, u2 – Specify a linear mapping of u, as presented to glEvalCoord1, to the variable that is evaluated by the equations specified by this command.

• stride (int) – Specifies the number of floats or float (double)s between the beginning of one control point and the beginning of the next one in the data structure referenced in points. This allows control points to be embedded in arbitrary data structures. The only constraint is that the values for a particular control point must occupy contiguous memory locations.

• order (int) – Specifies the number of control points. Must be positive.

• points (bgl.Buffer object. Depends on function prototype.) – Specifies a pointer to the array of control points.

glMap2 (target, u1, u2, ustride, uorder, v1, v2, vstride, vorder, points):
B{glMap2d, glMap2f}
Define a two-dimensional evaluator
See also:
OpenGL Docs

Parameters

• target (Enumerated constant) – Specifies the kind of values that are generated by the evaluator.

• u1, u2 – Specify a linear mapping of u, as presented to glEvalCoord2, to the variable that is evaluated by the equations specified by this command. Initially u1 is 0 and u2 is 1.

• ustride (int) – Specifies the number of floats or float (double)s between the beginning of control point R and the beginning of control point R ij, where i and j are the u and v control point indices, respectively. This allows control points to be embedded in arbitrary
data structures. The only constraint is that the values for a particular control point must occupy contiguous memory locations. The initial value of ustride is 0.

- **uorder (int)** – Specifies the dimension of the control point array in the u axis. Must be positive. The initial value is 1.

- **v2 (v1, )** – Specify a linear mapping of v, as presented to glEvalCoord2, to ^, one of the two variables that are evaluated by the equations specified by this command. Initially, v1 is 0 and v2 is 1.

- **vstride (int)** – Specifies the number of floats or float (double)s between the beginning of control point R and the beginning of control point R ij, where i and j are the u and v control point(indices, respectively. This allows control points to be embedded in arbitrary data structures. The only constraint is that the values for a particular control point must occupy contiguous memory locations. The initial value of vstride is 0.

- **vorder (int)** – Specifies the dimension of the control point array in the v axis. Must be positive. The initial value is 1.

- **points (bgl.Buffer object. Depends on function prototype.)** – Specifies a pointer to the array of control points.

### glMapGrid (un, u1, u2, vn, v1, v2):

B{glMapGrid1d, glMapGrid1f, glMapGrid2d, glMapGrid2f}

Define a one- or two-dimensional mesh

**See also:**

OpenGL Docs

**Parameters**

- **un (int)** – Specifies the number of partitions in the grid range interval [u1, u2]. Must be positive.

- **u2 (u1, )** – Specify the mappings for integer grid domain values i=0 and i=un.

- **vn (int)** – Specifies the number of partitions in the grid range interval [v1, v2] (glMapGrid2 only).

- **v2 (v1, )** – Specify the mappings for integer grid domain values j=0 and j=vn (glMapGrid2 only).

### glMaterial (face, pname, params):

Specify material parameters for the lighting model.

**See also:**

OpenGL Docs

**Parameters**

- **face (Enumerated constant)** – Specifies which face or faces are being updated. Must be one of:

- **pname (Enumerated constant)** – Specifies the single-valued material parameter of the face or faces that is being updated. Must be GL_SHININESS.

- **params (int)** – Specifies the value that parameter GL_SHININESS will be set to. If function prototype ends in ‘v’ specifies a pointer to the value or values that pname will be set to.
**glMatrixMode(mode):**
Specify which matrix is the current matrix.

*See also:*
OpenGL Docs

**Parameters**
mode (*Enumerated constant*) – Specifies which matrix stack is the target for subsequent matrix operations.

**glMultMatrix (m):**
B{glMultMatrixd, glMultMatrixf}
Multiply the current matrix with the specified matrix

*See also:*
OpenGL Docs

**Parameters**

- m (*bgl.Buffer object. Depends on function prototype.*) – Points to 16 consecutive values that are used as the elements of a 4x4 column major matrix.

**glNewList(list, mode):**
Create or replace a display list

*See also:*
OpenGL Docs

**Parameters**

- list (*unsigned int*) – Specifies the display list name
- mode (*Enumerated constant*) – Specifies the compilation mode.

**glNormal3 (nx, ny, nz, v):**
B{Normal3b, Normal3bv, Normal3d, Normal3dv, Normal3f, Normal3fv, Normal3i, Normal3iv, Normal3s, Normal3sv}
Set the current normal vector

*See also:*
OpenGL Docs

**Parameters**

- nx, ny, nz (*nx,*) – Specify the x, y, and z coordinates of the new current normal. The initial value of the current normal is the unit vector, (0, 0, 1).
- v (*bgl.Buffer object. Depends on function prototype. (*v* prototypes]*) – Specifies a pointer to an array of three elements: the x, y, and z coordinates of the new current normal.

**glOrtho(left, right, bottom, top, zNear, zFar):**
Multiply the current matrix with an orthographic matrix

*See also:*
OpenGL Docs

**Parameters**

- right (*left,*) – Specify the coordinates for the left and right vertical clipping planes.
• **top** *(bottom,)* – Specify the coordinates for the bottom and top horizontal clipping planes.
• **zFar** *(zNear,)* – Specify the distances to the nearer and farther depth clipping planes. These values are negative if the plane is to be behind the viewer.

**glPassThrough** *(token):*

Place a marker in the feedback buffer

See also:
OpenGL Docs

Parameters

* **token** *(float)* – Specifies a marker value to be placed in the feedback buffer following a GL_PASS_THROUGH_TOKEN.

**glPixelMap** *(map, mapsiz, values):*

B{glPixelMapf, glPixelMapuiv, glPixelMapusv}  
Set up pixel transfer maps

See also:
OpenGL Docs

Parameters

* **map** *(Enumerated constant)* – Specifies a symbolic map name.
* **mapsize** *(int)* – Specifies the size of the map being defined.

**glPixelStore** *(pname, param):*

B{glPixelStoref, glPixelStorei}  
Set pixel storage modes

See also:
OpenGL Docs

Parameters

* **pname** *(Enumerated constant)* – Specifies the symbolic name of the parameter to be set. Six values affect the packing of pixel data into memory. Six more affect the unpacking of pixel data from memory.

* **param** *(Depends on function prototype.)* – Specifies the value that pname is set to.

**glPixelTransfer** *(pname, param):*

B{glPixelTransferf, glPixelTransferi}  
Set pixel transfer modes

See also:
OpenGL Docs

Parameters

* **pname** *(Enumerated constant)* – Specifies the symbolic name of the pixel transfer parameter to be set.
• **param** *(depends on function prototype)* – Specifies the value that `pname` is set to.

**glPixelZoom(xfactor, yfactor):**
Specify the pixel zoom factors

**See also:**
OpenGL Docs

**Parameters**

- `xfactor` *(float)* – Specifies the x zoom factor.
- `yfactor` *(float)* – Specifies the y zoom factor.

**glPointSize(size):**
Specify the diameter of rasterized points

**See also:**
OpenGL Docs

**Parameters**

- `size` *(float)* – Specifies the diameter of rasterized points. The initial value is 1.

**glPolygonMode(face, mode):**
Select a polygon rasterization mode

**See also:**
OpenGL Docs

**Parameters**

- `face` *(Enumerated constant)* – Specifies the polygons that mode applies to. Must be `GL_FRONT` for front-facing polygons, `GL_BACK` for back-facing polygons, or `GL_FRONT_AND_BACK` for front- and back-facing polygons.
- `mode` *(Enumerated constant)* – Specifies how polygons will be rasterized. The initial value is `GL_FILL` for both front- and back-facing polygons.

**glPolygonOffset(factor, units):**
Set the scale and units used to calculate depth values

**See also:**
OpenGL Docs

**Parameters**

- `factor` *(float)* – Specifies a scale factor that is used to create a variable depth offset for each polygon. The initial value is 0.
- `units` *(float)* – Is multiplied by an implementation-specific value to create a constant depth offset. The initial value is 0.

**glPolygonStipple(mask):**
Set the polygon stippling pattern

**See also:**
OpenGL Docs

**Parameters**

- `mask` *(bgl.Buffer object I{type GL_BYTE})* – Specifies a pointer to a 32x32 stipple pattern that will be unpacked from memory in the same way that `glDrawPixels` unpacks pixels.
glPopAttrib():
    Pop the server attribute stack
    See also:
    OpenGL Docs

glPopClientAttrib():
    Pop the client attribute stack
    See also:
    OpenGL Docs

glPopMatrix():
    Pop the current matrix stack
    See also:
    OpenGL Docs

glPopName():
    Pop the name stack
    See also:
    OpenGL Docs

glPrioritizeTextures(n, textures, priorities):
    Set texture residence priority
    See also:
    OpenGL Docs

Parameters

- \text{n} (\text{int}) – Specifies the number of textures to be prioritized.
- \text{textures} (\text{bgl.Buffer I\{type GL_INT\}}) – Specifies an array containing the names of the textures to be prioritized.
- \text{priorities} (\text{bgl.Buffer I\{type GL_FLOAT\}}) – Specifies an array containing the texture priorities. A priority given in an element of priorities applies to the texture named by the corresponding element of textures.

glPushAttrib(mask):
    Push the server attribute stack
    See also:
    OpenGL Docs

Parameters \text{mask} (\text{Enumerated constant(s)}) – Specifies a mask that indicates which attributes to save.

glPushClientAttrib(mask):
    Push the client attribute stack
    See also:
    OpenGL Docs

Parameters \text{mask} (\text{Enumerated constant(s)}) – Specifies a mask that indicates which attributes to save.
glPushMatrix():
    Push the current matrix stack

    See also:
    OpenGL Docs

glPushName(name):
    Push the name stack

    See also:
    OpenGL Docs

    Parameters name (unsigned int) – Specifies a name that will be pushed onto the name stack.

glRasterPos (x, y, z, w):
    B{glRasterPos2d, glRasterPos2f, glRasterPos2i, glRasterPos2s, glRasterPos3d, glRasterPos3f, glRasterPos3i, 
glRasterPos3s, glRasterPos4d, glRasterPos4f, glRasterPos4i, glRasterPos4s, glRasterPos2dv, glRasterPos2fv, 
glRasterPos2iv, glRasterPos2sv, glRasterPos3dv, glRasterPos3fv, glRasterPos3iv, glRasterPos3sv, glRasterPos4dv, 
glRasterPos4fv, glRasterPos4iv, glRasterPos4sv}

    Specify the raster position for pixel operations

    See also:
    OpenGL Docs

    Parameters y, z, w (x, ) – Specify the x, y, z, and w object coordinates (if present) for the raster position. If function prototype ends in ‘v’ specifies a pointer to an array of two, three, or four elements, specifying x, y, z, and w coordinates, respectively.

    Note: If you are drawing to the 3d view with a Scriptlink of a space handler the zoom level of the panels will scale the glRasterPos by the view matrix. so a X of 10 will not always offset 10 pixels as you would expect.

    To work around this get the scale value of the view matrix and use it to scale your pixel values.

import bgl
xval, yval= 100, 40

    # Get the scale of the view matrix
view_matrix = bgl.Buffer(bgl.GL_FLOAT, 16)
bgl.glGetFloatv(bgl.GL_MODELVIEW_MATRIX, view_matrix)
f = 1.0 / view_matrix[0]

    # Instead of the usual glRasterPos2i(xval, yval)
bgl.glRasterPos2f(xval * f, yval * f)

    glReadBuffer (mode):
    Select a color buffer source for pixels.

    See also:
    OpenGL Docs

    Parameters mode (Enumerated constant) – Specifies a color buffer.

glReadPixels (x, y, width, height, format, type, pixels):
    Read a block of pixels from the frame buffer

    See also:
OpenGL Docs

Parameters

- $y(x,)$ – Specify the window coordinates of the first pixel that is read from the frame buffer. This location is the lower left corner of a rectangular block of pixels.
- **height**($width,$) – Specify the dimensions of the pixel rectangle. width and height of one correspond to a single pixel.
- **format**(Enumerated constant) – Specifies the format of the pixel data.
- **type**(Enumerated constant) – Specifies the data type of the pixel data.
- **pixels**(bgl.Buffer object) – Returns the pixel data.

$\text{glRect}(x_1,y_1,x_2,y_2,v_1,v_2)$:
B{glRectd, glRectf, glRecti, glRects, glRectdv, glRectfv, glRectiv, glRectsv}

Draw a rectangle

See also:

OpenGL Docs

Parameters

- $y_1(x_1,)$ – Specify one vertex of a rectangle
- $y_2(x_2,)$ – Specify the opposite vertex of the rectangle
- $v_2(v_1,)$ – Specifies a pointer to one vertex of a rectangle and the pointer to the opposite vertex of the rectangle

$\text{glRenderMode}(\text{mode})$:
Set rasterization mode

See also:

OpenGL Docs

Parameters \textbf{mode}(Enumerated constant) – Specifies the rasterization mode.

$\text{glRotate}(\text{angle}, x, y, z)$:
B{glRotated, glRotatef}

Multiply the current matrix by a rotation matrix

See also:

OpenGL Docs

Parameters

- \textbf{angle}(Depends on function prototype.) – Specifies the angle of rotation in degrees.
- $y, z(x,)$ – Specify the $x, y,$ and $z$ coordinates of a vector respectively.

$\text{glScale}(x,y,z)$:
B{glScaled, glScalef}

Multiply the current matrix by a general scaling matrix

See also:
OpenGL Docs

Parameters **y, z** (*x*) – Specify scale factors along the x, y, and z axes, respectively.

**glScissor**(*x, y, width, height*):
Define the scissor box

See also:
OpenGL Docs

Parameters

• **y** (*x*) – Specify the lower left corner of the scissor box. Initially (0, 0).
• **height** (*width*) – Specify the width and height of the scissor box. When a GL context is first attached to a window, width and height are set to the dimensions of that window.

**glSelectBuffer**(*size, buffer*):
Establish a buffer for selection mode values

See also:
OpenGL Docs

Parameters

• **size** (*int*) – Specifies the size of buffer
• **buffer** (*bgl.Buffer* (type GL_INT)) – Returns the selection data

**glShadeModel**(*mode*):
Select flat or smooth shading

See also:
OpenGL Docs

Parameters **mode** (*Enumerated constant*) – Specifies a symbolic value representing a shading technique.

**glStencilFunc**(*func, ref, mask*):
Set function and reference value for stencil testing

See also:
OpenGL Docs

Parameters

• **func** (*Enumerated constant*) – Specifies the test function.
• **ref** (*int*) – Specifies the reference value for the stencil test. ref is clamped to the range \([0,2^n-1]\), where \(n\) is the number of bitplanes in the stencil buffer. The initial value is 0.
• **mask** (*unsigned int*) – Specifies a mask that is ANDed with both the reference value and the stored stencil value when the test is done. The initial value is all 1’s.

**glStencilMask**(*mask*):
Control the writing of individual bits in the stencil planes

See also:
OpenGL Docs
**Parameters**

- **mask** (*unsigned int*) – Specifies a bit mask to enable and disable writing of individual bits in the stencil planes. Initially, the mask is all 1’s.

**glStencilOp** *(fail, zfail, zpass):*

Set stencil test actions

**See also:**
OpenGL Docs

**Parameters**

- **fail** (*Enumerated constant*) – Specifies the action to take when the stencil test fails. The initial value is GL_KEEP.
- **zfail** (*Enumerated constant*) – Specifies the stencil action when the stencil test passes, but the depth test fails. zfail accepts the same symbolic constants as fail. The initial value is GL_KEEP.
- **zpass** (*Enumerated constant*) – Specifies the stencil action when both the stencil test and the depth test pass, or when the stencil test passes and either there is no depth buffer or depth testing is not enabled. zpass accepts the same symbolic constants as fail. The initial value is GL_KEEP.

**glTexCoord** *(s,t,r,q,v):

Set the current texture coordinates

**See also:**
OpenGL Docs

**Parameters**

- **t, r, q (s)** – Specify s, t, r, and q texture coordinates. Not all parameters are present in all forms of the command.
- **v** (*bgl.Buffer* object. Depends on function prototype. (for ‘v’ prototypes only)) – Specifies a pointer to an array of one, two, three, or four elements, which in turn specify the s, t, r, and q texture coordinates.

**glTexEnv** *(target, pname, param):*

Set texture environment parameters

**See also:**
OpenGL Docs

**Parameters**

- **target** (*Enumerated constant*) – Specifies a texture environment. Must be GL_TEXTURE_ENV.
- **pname** (*Enumerated constant*) – Specifies the symbolic name of a single-valued texture environment parameter. Must be GL_TEXTURE_ENV_MODE.
• **param** *(Depends on function prototype.)* – Specifies a single symbolic constant. If function prototype ends in ‘v’ specifies a pointer to a parameter array that contains either a single symbolic constant or an RGBA color.

**glTexGen** *(coord, pname, param):*

\[ \text{B}\{\text{glTexGend, glTexGenf, glTexGeni, glTexGenfv, glTexGeniv}\} \]

Control the generation of texture coordinates

**See also:**
OpenGL Docs

**Parameters**

• **coord** *(Enumerated constant)* – Specifies a texture coordinate.

• **pname** *(Enumerated constant)* – Specifies the symbolic name of the texture-coordinate generation function.

• **param** *(Depends on function prototype.)* – Specifies a single-valued texture generation parameter. If function prototype ends in ‘v’ specifies a pointer to an array of texture generation parameters. If **pname** is **GL_TEXTURE_GEN_MODE**, then the array must contain a single symbolic constant. Otherwise, params holds the coefficients for the texture-coordinate generation function specified by *pname*.

**glTexImage1D** *(target, level, internalformat, width, border, format, type, pixels):*

Specify a one-dimensional texture image

**See also:**
OpenGL Docs

**Parameters**

• **target** *(Enumerated constant)* – Specifies the target texture.

• **level** *(int)* – Specifies the level-of-detail number. Level 0 is the base image level. Level n is the nth mipmap reduction image.

• **internalformat** *(int)* – Specifies the number of color components in the texture.

• **width** *(int)* – Specifies the width of the texture image. Must be \(2^n+2\) (border) for some integer \(n\). All implementations support texture images that are at least 64 texels wide. The height of the 1D texture image is 1.

• **border** *(int)* – Specifies the width of the border. Must be either 0 or 1.

• **format** *(Enumerated constant)* – Specifies the format of the pixel data.

• **type** *(Enumerated constant)* – Specifies the data type of the pixel data.

• **pixels** *(**gl.Buffer** object.)* – Specifies a pointer to the image data in memory.

**glTexImage2D** *(target, level, internalformat, width, height, border, format, type, pixels):*

Specify a two-dimensional texture image

**See also:**
OpenGL Docs

**Parameters**

• **target** *(Enumerated constant)* – Specifies the target texture.
• **level** *(int)* – Specifies the level-of-detail number. Level 0 is the base image level. Level \( n \) is the \( n \)th mipmap reduction image.

• **internalformat** *(int)* – Specifies the number of color components in the texture.

• **width** *(int)* – Specifies the width of the texture image. Must be \( 2n+2 \) (border) for some integer \( n \). All implementations support texture images that are at least 64 texels wide.

• **height** *(int)* – Specifies the height of the texture image. Must be \( 2m+2 \) (border) for some integer \( m \). All implementations support texture images that are at least 64 texels high.

• **border** *(int)* – Specifies the width of the border. Must be either 0 or 1.

• **format** *(Enumerated constant)* – Specifies the format of the pixel data.

• **type** *(Enumerated constant)* – Specifies the data type of the pixel data.

• **pixels** *(bgl.Buffer object)* – Specifies a pointer to the image data in memory.

```c
glTexParameter (target, pname, param):
    B{glTexParameterf, glTexParameteri, glTexParameterfv, glTexParameteriv}
```

Set texture parameters

See also:
OpenGL Docs

**Parameters**

• **target** *(Enumerated constant)* – Specifies the target texture.

• **pname** *(Enumerated constant)* – Specifies the symbolic name of a single-valued texture parameter.

• **param** *(Depends on function prototype.)* – Specifies the value of **pname**. If function prototype ends in ‘v’ specifies a pointer to an array where the value or values of **pname** are stored.

```c
glTranslate (x, y, z):
    B{glTranslatef, glTranslated}
```

Multiply the current matrix by a translation matrix

See also:
OpenGL Docs

**Parameters** \( y, z \) \((x, )\) – Specify the \( x, y, \) and \( z \) coordinates of a translation vector.

```c
glVertex (x,y,z,w,v):
    B{glVertex2d, glVertex2f, glVertex2i, glVertex2s, glVertex3d, glVertex3f, glVertex3i, glVertex3s, glVertex4d, glVertex4f, glVertex4i, glVertex4s, glVertex2dv, glVertex2fv, glVertex2iv, glVertex2sv, glVertex3dv, glVertex3fv, glVertex3iv, glVertex3sv, glVertex4dv, glVertex4fv, glVertex4iv, glVertex4sv}
```

Specify a vertex

See also:
OpenGL Docs

**Parameters**

• **y, z, w** \((x, )\) – Specify \( x, y, z, \) and \( w \) coordinates of a vertex. Not all parameters are present in all forms of the command.
• \texttt{v} (\texttt{bgl.Buffer} object). Depends on function prototype (for ‘v’ prototypes only)—Specifies a pointer to an array of two, three, or four elements. The elements of a two-element array are \(x\) and \(y\); of a three-element array, \(x\), \(y\), and \(z\); and of a four-element array, \(x\), \(y\), \(z\), and \(w\).

\texttt{glViewport(x,y,width,height)}:
  
  Set the viewport

  \textbf{See also:}

  OpenGL Docs

  \textbf{Parameters}

  • \texttt{x} (double) – Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0,0).

  • \texttt{y} (double) – Specify the width and height of the viewport. When a GL context is first attached to a window, width and height are set to the dimensions of that window.

\texttt{gluPerspective(fovY, aspect, zNear, zFar)}:
  
  Set up a perspective projection matrix.

  \textbf{See also:}

  https://www.opengl.org/sdk/docs/man2/xhtml/gluPerspective.xml

  \textbf{Parameters}

  • \texttt{fovY} (double) – Specifies the field of view angle, in degrees, in the y direction.

  • \texttt{aspect} (double) – Specifies the aspect ratio that determines the field of view in the x direction. The aspect ratio is the ratio of x (width) to y (height).

  • \texttt{zNear} (double) – Specifies the distance from the viewer to the near clipping plane (always positive).

  • \texttt{zFar} (double) – Specifies the distance from the viewer to the far clipping plane (always positive).

\texttt{gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)}:
  
  Define a viewing transformation.

  \textbf{See also:}

  https://www.opengl.org/sdk/docs/man2/xhtml/gluLookAt.xml

  \textbf{Parameters}

  • \texttt{eyex}, \texttt{eyey}, \texttt{eyez} (\texttt{eyex},) – Specifies the position of the eye point.

  • \texttt{centery}, \texttt{centerz} (\texttt{centerx},) – Specifies the position of the reference point.

  • \texttt{upx}, \texttt{upy}, \texttt{upz} (\texttt{upx},) – Specifies the direction of the up vector.

\texttt{gluOrtho2D(left, right, bottom, top)}:
  
  Define a 2-D orthographic projection matrix.

  \textbf{See also:}

  https://www.opengl.org/sdk/docs/man2/xhtml/gluOrtho2D.xml

  \textbf{Parameters}

  • \texttt{right} (\texttt{left},) – Specify the coordinates for the left and right vertical clipping planes.
• **top** *(bottom,)* – Specify the coordinates for the bottom and top horizontal clipping planes.

`gluPickMatrix(x, y, width, height, viewport):`
Define a picking region.

See also:
https://www.opengl.org/sdk/docs/man2/xhtml/gluPickMatrix.xml

**Parameters**

• **y** *(x,)* – Specify the center of a picking region in window coordinates.

• **height** *(width,)* – Specify the width and height, respectively, of the picking region in window coordinates.

• **viewport** *(bgl.Buffer object. [int])* – Specifies the current viewport.

`gluProject(objx, objy, objz, modelMatrix, projMatrix, viewport, winx, winy, winz):`
Map object coordinates to window coordinates.

See also:
https://www.opengl.org/sdk/docs/man2/xhtml/gluProject.xml

**Parameters**

• **objy, objz** *(objx,)* – Specify the object coordinates.

• **modelMatrix** *(bgl.Buffer object. [double])* – Specifies the current modelview matrix (as from a glGetDoublev call).

• **projMatrix** *(bgl.Buffer object. [double])* – Specifies the current projection matrix (as from a glGetDoublev call).

• **viewport** *(bgl.Buffer object. [int])* – Specifies the current viewport (as from a glGetIntegerv call).

• **winy, winz** *(winx,)* – Return the computed window coordinates.

`gluUnProject(winx, winy, winz, modelMatrix, projMatrix, viewport, objx, objy, objz):`
Map object coordinates to window coordinates.

See also:
https://www.opengl.org/sdk/docs/man2/xhtml/gluUnProject.xml

**Parameters**

• **winy, winz** *(winx,)* – Specify the window coordinates to be mapped.

• **modelMatrix** *(bgl.Buffer object. [double])* – Specifies the current modelview matrix (as from a glGetDoublev call).

• **projMatrix** *(bgl.Buffer object. [double])* – Specifies the current projection matrix (as from a glGetDoublev call).

• **viewport** *(bgl.Buffer object. [int])* – Specifies the current viewport (as from a glGetIntegerv call).

• **objy, objz** *(objx,)* – Return the computed object coordinates.
glUseProgram(program):
Installs a program object as part of current rendering state

See also:
OpenGL Docs

Parameters program(int) – Specifies the handle of the program object whose executables are
to be used as part of current rendering state.

glValidateProgram(program):
Validates a program object

See also:
OpenGL Docs

Parameters program(int) – Specifies the handle of the program object to be validated.

glLinkProgram(program):
Links a program object.

See also:
OpenGL Docs

Parameters program(int) – Specifies the handle of the program object to be linked.

glActiveTexture(texture):
Select active texture unit.

See also:
OpenGL Docs

Parameters texture(int) – Constant in GL_TEXTURE0 - 8

glAttachShader(program, shader):
Attaches a shader object to a program object.

See also:
OpenGL Docs

Parameters
• program(int) – Specifies the program object to which a shader object will be attached.
• shader(int) – Specifies the shader object that is to be attached.

glCompileShader(shader):
Compiles a shader object.

See also:
OpenGL Docs

Parameters shader(int) – Specifies the shader object to be compiled.

glCreateProgram():
Creates a program object

See also:
OpenGL Docs
Return type  int

Returns  The new program or zero if an error occurs.

`glCreateShader(shaderType):`
Creates a shader object.

See also:
OpenGL Docs

Parameters  shaderType  (Specifies the type of shader to be created. Must be one of  GL_VERTEX_SHADER, GL_TESS_CONTROL_SHADER, GL_TESS_EVALUATION_SHADER, GL_GEOMETRY_SHADER, or GL_FRAGMENT_SHADER.) –

Return type  int

Returns  0 if an error occurs.

`glDeleteProgram(program):`
Deletes a program object.

See also:
OpenGL Docs

Parameters  program(int) – Specifies the program object to be deleted.

`glDeleteShader(shader):`
Deletes a shader object.

See also:
OpenGL Docs

Parameters  shader(int) – Specifies the shader object to be deleted.

`glDetachShader(program, shader):`
Detaches a shader object from a program object to which it is attached.

See also:
OpenGL Docs

Parameters
- program(int) – Specifies the program object from which to detach the shader object.
- shader(int) – Specifies the program object from which to detach the shader object.

`glGetAttachedShaders(program, maxCount, count, shaders):`
Returns the handles of the shader objects attached to a program object.

See also:
OpenGL Docs

Parameters
- program(int) – Specifies the program object to be queried.
- maxCount(int) – Specifies the size of the array for storing the returned object names.
• `count (bgl.Buffer int buffer.)` – Returns the number of names actually returned in objects.

• `shaders (bgl.Buffer int buffer.)` – Specifies an array that is used to return the names of attached shader objects.

`glGetProgramInfoLog(program, maxLength, length, infoLog):`
Returns the information log for a program object.

See also:
OpenGL Docs

Parameters

• `program (int)` – Specifies the program object whose information log is to be queried.

• `maxLength (int)` – Specifies the size of the character buffer for storing the returned information log.

• `length (bgl.Buffer int buffer.)` – Returns the length of the string returned in `infoLog` (excluding the null terminator).

• `infoLog (bgl.Buffer char buffer.)` – Specifies an array of characters that is used to return the information log.

`glGetShaderInfoLog(program, maxLength, length, infoLog):`
Returns the information log for a shader object.

See also:
OpenGL Docs

Parameters

• `shader (int)` – Specifies the shader object whose information log is to be queried.

• `maxLength (int)` – Specifies the size of the character buffer for storing the returned information log.

• `length (bgl.Buffer int buffer.)` – Returns the length of the string returned in `infoLog` (excluding the null terminator).

• `infoLog (bgl.Buffer char buffer.)` – Specifies an array of characters that is used to return the information log.

`glGetProgramiv(program, pname, params):`
Returns a parameter from a program object.

See also:
OpenGL Docs

Parameters

• `program (int)` – Specifies the program object to be queried.

• `pname (int)` – Specifies the object parameter.

• `params (bgl.Buffer int buffer.)` – Returns the requested object parameter.

`glIsShader(shader):`
Determines if a name corresponds to a shader object.

See also:
OpenGL Docs

Parameters shader (int) – Specifies a potential shader object.

glIsProgram(program):
    Determines if a name corresponds to a program object

See also:
OpenGL Docs

Parameters program (int) – Specifies a potential program object.

glGetShaderSource(shader, bufSize, length, source):
    Returns the source code string from a shader object

See also:
OpenGL Docs

Parameters

• shader (int) – Specifies the shader object to be queried.

• bufSize (int) – Specifies the size of the character buffer for storing the returned source code string.

• length (bgl.Buffer int buffer.) – Returns the length of the string returned in source (excluding the null terminator).

• source (bgl.Buffer char.) – Specifies an array of characters that is used to return the source code string.

glShaderSource(shader, shader_string):
    Replaces the source code in a shader object.

See also:
OpenGL Docs

Parameters

• shader (int) – Specifies the handle of the shader object whose source code is to be replaced.

• shader_string (string) – The shader string.

class bgl.Buffer
    The Buffer object is simply a block of memory that is delineated and initialized by the user. Many OpenGL functions return data to a C-style pointer, however, because this is not possible in python the Buffer object can be used to this end. Wherever pointer notation is used in the OpenGL functions the Buffer object can be used in it’s bgl wrapper. In some instances the Buffer object will need to be initialized with the template parameter, while in other instances the user will want to create just a blank buffer which will be zeroed by default.

```python
import bgl

myByteBuffer = bgl.Buffer(bgl.GL_BYTE, [32, 32])
bgl.glGetPolygonStipple(myByteBuffer)

print(myByteBuffer.dimensions)
print(myByteBuffer.to_list())
```

(continues on next page)
sliceBuffer = myByteBuffer[0:16]
print(sliceBuffer)

dimensions
The number of dimensions of the Buffer.

to_list()
The contents of the Buffer as a python list.

__init__(type, dimensions, template = None):
This will create a new Buffer object for use with other bgl OpenGL commands. Only the type of argument to store in the buffer and the dimensions of the buffer are necessary. Buffers are zeroed by default unless a template is supplied, in which case the buffer is initialized to the template.

Parameters

• type (int) – The format to store data in. The type should be one of GL_BYTE, GL_SHORT, GL_INT, or GL_FLOAT.

• dimensions (An int or sequence object specifying the dimensions of the buffer.) – If the dimensions are specified as an int a linear array will be created for the buffer. If a sequence is passed for the dimensions, the buffer becomes n-Dimensional, where n is equal to the number of parameters passed in the sequence. Example: [256,2] is a two-dimensional buffer while [256,256,4] creates a three-dimensional buffer. You can think of each additional dimension as a sub-item of the dimension to the left. i.e. [10,2] is a 10 element array each with 2 sub-items. [(0,0), (0,1), (1,0), (1,1), (2,0), ...] etc.

• template (A python sequence object (optional)) – A sequence of matching dimensions which will be used to initialize the Buffer. If a template is not passed in all fields will be initialized to 0.

Return type Buffer object

Returns The newly created buffer as a PyObject.
This module provides access to blender's text drawing functions.

## 12.1 Text Example

Blender Game Engine example of using the blf module. For this module to work we need to use the OpenGL wrapper `bgl` as well.

```python
""
Hello World Text Example
++++++++++++++++++++++++
Example of using the blf module. For this module to work we need to use the OpenGL wrapper :class:`bgl` as well.
""

# import stand alone modules
import bgl
import blf
import bpy

font_info = {
    "font_id": 0,
    "handler": None,
}

def init():
    """init function - runs once"""

    import os
    # Create a new font object, use external ttf file.
    font_path = bpy.path.abspath('//Zeyada.ttf')
    # Store the font indice - to use later.
    if os.path.exists(font_path):
        font_info["font_id"] = blf.load(font_path)
```

(continues on next page)
else:
    # Default font.
    font_info['font_id'] = 0

    # set the font drawing routine to run every frame
    font_info['handler'] = bpy.types.SpaceView3D.draw_handler_add(
        draw_callback_px, (None, None), 'WINDOW', 'POST_PIXEL')

def draw_callback_px(self, context):
    """Draw on the viewports""
    # BLF drawing routine
    font_id = font_info['font_id']
    blf.position(font_id, 2, 80, 0)
    blf.size(font_id, 50, 72)
    blf.draw(font_id, "Hello World")

if __name__ == '__main__':
    init()

### Chapter 12. Font Drawing (blf)

#### 12.2 Functions

**blf.aspect** *(fontid, aspect)*

Set the aspect for drawing text.

**Parameters**

- **fontid** *(int)* – The id of the typeface as returned by `blf.load()`, for default font use 0.
- **aspect** *(float)* – The aspect ratio for text drawing to use.

**blf.blur** *(fontid, radius)*

Set the blur radius for drawing text.

**Parameters**

- **fontid** *(int)* – The id of the typeface as returned by `blf.load()`, for default font use 0.
- **radius** *(int)* – The radius for blurring text (in pixels).

**blf.clipping** *(fontid, xmin, ymin, xmax, ymax)*

Set the clipping, enable/disable using CLIPPING.

**Parameters**

- **fontid** *(int)* – The id of the typeface as returned by `blf.load()`, for default font use 0.
- **xmin** *(float)* – Clip the drawing area by these bounds.
- **ymin** *(float)* – Clip the drawing area by these bounds.
- **xmax** *(float)* – Clip the drawing area by these bounds.
- **ymax** *(float)* – Clip the drawing area by these bounds.
**blf.dimensions** *(fontid, text)*

Return the width and height of the text.

**Parameters**

- **fontid** *(int)* – The id of the typeface as returned by `blf.load()`, for default font use 0.
- **text** *(string)* – the text to draw.

**Returns** the width and height of the text.

**Return type** tuple of 2 floats

**blf.disable** *(fontid, option)*

Disable option.

**Parameters**

- **fontid** *(int)* – The id of the typeface as returned by `blf.load()`, for default font use 0.
- **option** *(int)* – One of ROTATION, CLIPPING, SHADOW or KERNING_DEFAULT.

**blf.draw** *(fontid, text)*

Draw text in the current context.

**Parameters**

- **fontid** *(int)* – The id of the typeface as returned by `blf.load()`, for default font use 0.
- **text** *(string)* – the text to draw.

**blf.enable** *(fontid, option)*

Enable option.

**Parameters**

- **fontid** *(int)* – The id of the typeface as returned by `blf.load()`, for default font use 0.
- **option** *(int)* – One of ROTATION, CLIPPING, SHADOW or KERNING_DEFAULT.

**blf.load** *(filename)*

Load a new font.

**Parameters**

- **filename** *(string)* – the filename of the font.

**Returns** the new font's fontid or -1 if there was an error.

**Return type** integer

**blf.position** *(fontid, x, y, z)*

Set the position for drawing text.

**Parameters**

- **fontid** *(int)* – The id of the typeface as returned by `blf.load()`, for default font use 0.
- **x** *(float)* – X axis position to draw the text.
- **y** *(float)* – Y axis position to draw the text.
- **z** *(float)* – Z axis position to draw the text.
**blf.rotation** *(fontid, angle)*

Set the text rotation angle, enable/disable using ROTATION.

**Parameters**

- **fontid** *(int)* – The id of the typeface as returned by *blf.load()* for default font use 0.
- **angle** *(float)* – The angle for text drawing to use.

**blf.shadow** *(fontid, level, r, g, b, a)*

Shadow options, enable/disable using SHADOW.

**Parameters**

- **fontid** *(int)* – The id of the typeface as returned by *blf.load()* for default font use 0.
- **level** *(int)* – The blur level, can be 3, 5 or 0.
- **r** *(float)* – Shadow color (red channel 0.0 - 1.0).
- **g** *(float)* – Shadow color (green channel 0.0 - 1.0).
- **b** *(float)* – Shadow color (blue channel 0.0 - 1.0).
- **a** *(float)* – Shadow color (alpha channel 0.0 - 1.0).

**blf.shadow_offset** *(fontid, x, y)*

Set the offset for shadow text.

**Parameters**

- **fontid** *(int)* – The id of the typeface as returned by *blf.load()* for default font use 0.
- **x** *(float)* – Vertical shadow offset value in pixels.
- **y** *(float)* – Horizontal shadow offset value in pixels.

**blf.size** *(fontid, size, dpi)*

Set the size and dpi for drawing text.

**Parameters**

- **fontid** *(int)* – The id of the typeface as returned by *blf.load()* for default font use 0.
- **size** *(int)* – Point size of the font.
- **dpi** *(int)* – Dots per inch value to use for drawing.

**blf.unload** *(filename)*

Unload an existing font.

**Parameters**

- **filename** *(string)* – The filename of the font.

**blf.word_wrap** *(fontid, wrap_width)*

Set the wrap width, enable/disable using WORD_WRAP.

**Parameters**

- **fontid** *(int)* – The id of the typeface as returned by *blf.load()* for default font use 0.
- **wrap_width** *(int)* – The width (in pixels) to wrap words at.
12.3 Constants

*blf.* **CLIPPING**
constant value 2

*blf.* **KERNING_DEFAULT**
constant value 8

*blf.* **ROTATION**
constant value 1

*blf.* **SHADOW**
constant value 4

*blf.* **WORD_WRAP**
constant value 128
This module provides access to materials GLSL shaders.

Submodules:

13.1 GPU Off-Screen Buffer (gpu.offscreen)

This module provides access to offscreen rendering functions.

new(width, height, samples=0)

Return a GPUOffScreen.

- **param width** Horizontal dimension of the buffer.
- **type width** int
- **param height** Vertical dimension of the buffer.
- **type height** int
- **param samples** OpenGL samples to use for MSAA or zero to disable.
- **type samples** int
- **return** Newly created off-screen buffer.
- **rtype** gpu.GPUOffscreen

```python
class gpu.offscreen.GPUOffscreen
This object gives access to off screen buffers.
bind(save=True)

Bind the offscreen object.

- **param save** save OpenGL current states.
- **type save** bool
```
draw_view3d(scene, view3d, region, modelview_matrix, projection_matrix)

Draw the 3d viewport in the offscreen object.

  **param scene**   Scene to draw.
  **type scene**   bpy.types.Scene
  **param view3d**  3D View to get the drawing settings from.
  **type view3d**   bpy.types.SpaceView3D
  **param region**  Region of the 3D View.
  **type region**   bpy.types.Region
  **param modelview_matrix**  ModelView Matrix.
  **type modelview_matrix**   mathutils.Matrix
  **param projection_matrix**  Projection Matrix.
  **type projection_matrix**   mathutils.Matrix

free()

Free the offscreen object The framebuffer, texture and render objects will no longer be accessible.

unbind(restore=True)

Unbind the offscreen object.

  **param restore**   restore OpenGL previous states.
  **type restore**   bool

  **color_texture** Color texture.
    **Type**   int
  **height** Texture height.
    **Type**   int
  **width** Texture width.
    **Type**   int

### 13.2 Intro

Module to provide functions concerning the GPU implementation in Blender, in particular the GLSL shaders that blender generates automatically to render materials in the 3D view and in the game engine.

**Warning:** The API provided by this module is subject to change. The data exposed by the API are are closely related to Blender’s internal GLSL code and may change if the GLSL code is modified (e.g. new uniform type).
13.3 Constants

13.3.1 GLSL Data Type

Type of GLSL data. For shader uniforms, the data type determines which `glUniform` function variant to use to send the uniform value to the GPU. For vertex attributes, the data type determines which `glVertexAttrib` function variant to use to send the vertex attribute to the GPU.

See `export_shader`

```
gpu.GPU_DATA_1I  
one integer

gpu.GPU_DATA_1F  
one float

gpu.GPU_DATA_2F  
two floats

gpu.GPU_DATA_3F  
three floats

gpu.GPU_DATA_4F  
four floats

gpu.GPU_DATA_9F  
matrix 3x3 in column-major order

gpu.GPU_DATA_16F 
matrix 4x4 in column-major order

gpu.GPU_DATA_4UB 
four unsigned byte
```

13.3.2 GLSL Uniform Types

Constants that specify the type of uniform used in a GLSL shader. The uniform type determines the data type, origin and method of calculation used by Blender to compute the uniform value.

The calculation of some of the uniforms is based on matrices available in the scene:

- **mat4_cam_to_world** Model matrix of the camera. OpenGL 4x4 matrix that converts camera local coordinates to world coordinates. In blender this is obtained from the ‘matrix_world’ attribute of the camera object.

  Some uniform will need the `mat4_world_to_cam` matrix computed as the inverse of this matrix.

- **mat4_object_to_world** Model matrix of the object that is being rendered. OpenGL 4x4 matrix that converts object local coordinates to world coordinates. In blender this is obtained from the ‘matrix_world’ attribute of the object.

  Some uniform will need the `mat4_world_to_object` matrix, computed as the inverse of this matrix.

- **mat4_lamp_to_world** Model matrix of the lamp lighting the object. OpenGL 4x4 matrix that converts lamp local coordinates to world coordinates. In blender this is obtained from the ‘matrix_world’ attribute of the lamp object.

  Some uniform will need the `mat4_world_to_lamp` matrix computed as the inverse of this matrix.


Note: Any uniforms used for view projections or transformations (object, lamp matrices for eg), can only be set once per frame.

GLSL Object Uniforms

Note:

- Object transformations and color must be set before drawing the object.
- There is at most one uniform of these types per shader.

```cpp
gpu.GPU_DYNAMIC_OBJECT_VIEWMAT
   A matrix that converts world coordinates to camera coordinates (see mat4_world_to_cam).
   Type matrix4x4

gpu.GPU_DYNAMIC_OBJECT_MAT
   A matrix that converts object coordinates to world coordinates (see mat4_object_to_world).
   Type matrix4x4

gpu.GPU_DYNAMIC_OBJECT_VIEWIMAT
   The uniform is a 4x4 GL matrix that converts coordinates in camera space to world coordinates (see mat4_cam_to_world).
   Type matrix4x4

gpu.GPU_DYNAMIC_OBJECT_IMAT
   The uniform is a 4x4 GL matrix that converts world coordinates to object coordinates (see mat4_world_to_object).
   Type matrix4x4

gpu.GPU_DYNAMIC_OBJECT_COLOR
   An RGB color + alpha defined at object level. Each values between 0.0 and 1.0.
   See bpy.types.Object.color.
   Type float4

gpu.GPU_DYNAMIC_OBJECT_AUTOBUMPSCALE
   Multiplier for bump-map scaling.
   Type float
```

GLSL Lamp Uniforms

Note: There is one uniform of that type per lamp lighting the material.

```cpp
gpu.GPU_DYNAMIC_LAMP_DYNVEC
   Represents the direction of light in camera space.
   Computed as: mat4_world_to_cam * (-vec3_lamp_Z_axis)
```

Note:
• The lamp Z axis points to the opposite direction of light.
• The norm of the vector should be unit length.

**Type** float3

gpu.GPU_DYNAMIC_LAMP_DYNCO
Represents the position of the light in camera space.

**Computed as:** \( \text{mat4\_world\_to\_cam} \times \text{vec3\_lamp\_pos} \)

**Type** float3

gpu.GPU_DYNAMIC_LAMP_DYNIMAT
Matrix that converts vector in camera space to lamp space.

**Computed as:** \( \text{mat4\_world\_to\_lamp} \times \text{mat4\_cam\_to\_world} \)

**Type** matrix4x4


gpu.GPU_DYNAMIC_LAMP_DYNPERSMAT
Matrix that converts a vector in camera space to shadow buffer depth space.

**Computed as:** \( \text{mat4\_perspective\_to\_depth} \times \text{mat4\_lamp\_to\_perspective} \times \text{mat4\_world\_to\_lamp} \times \text{mat4\_cam\_to\_world} \)

\( \text{mat4\_perspective\_to\_depth} \) is a fixed matrix defined as follow:

\[
\begin{bmatrix}
0.5 & 0.0 & 0.0 & 0.5 \\
0.0 & 0.5 & 0.0 & 0.5 \\
0.0 & 0.0 & 0.5 & 0.5 \\
0.0 & 0.0 & 0.0 & 1.0 \\
\end{bmatrix}
\]

**Note:**
• There is one uniform of that type per lamp casting shadow in the scene.

**Type** matrix4x4

gpu.GPU_DYNAMIC_LAMP_DYNENERGY
See bpy.types.Lamp.energy.

**Type** float

gpu.GPU_DYNAMIC_LAMP_DYNCOL
See bpy.types.Lamp.color.

**Type** float3

gpu.GPU_DYNAMIC_LAMP_DISTANCE
See bpy.types.Lamp.distance.

**Type** float

gpu.GPU_DYNAMIC_LAMP_ATT1
Type float

gpu.GPU_DYNAMIC_LAMP_ATT2
    Type float

gpu.GPU_DYNAMIC_LAMP_SPOTSIZE
    See bpy.types.SpotLamp.spot_size.
    Type float

gpu.GPU_DYNAMIC_LAMP_SPOTBLEND
    See bpy.types.SpotLamp.spot_blend.
    Type float

gpu.GPU_DYNAMIC_LAMP_SPOTSCALE
    Represents the SpotLamp local scale.
    Type float2

GLSL Sampler Uniforms

gpu.GPU_DYNAMIC_SAMPLER_2DBUFFER
    Represents an internal texture used for certain effect (color band, etc).
    Type integer

gpu.GPU_DYNAMIC_SAMPLER_2DIMAGE
    Represents a texture loaded from an image file.
    Type integer

gpu.GPU_DYNAMIC_SAMPLER_2DSHADOW
    Represents a texture loaded from a shadow buffer file.
    Type integer

GLSL Mist Uniforms

GPU_DYNAMIC_MIST_ENABLE:
    See bpy.types.WorldMistSettings.use_mist.
    Type float (0 or 1)

gpu.GPU_DYNAMIC_MIST_START
    See bpy.types.WorldMistSettings.start.
    Type float
    See bpy.types.WorldMistSettings.depth.

gpu.GPU_DYNAMIC_MIST_DISTANCE
    Type float
    See bpy.types.WorldMistSettings.intensity.

gpu.GPU_DYNAMIC_MIST_INTENSITY
    Type float
gpu.GPU_DYNAMIC_MIST_TYPE
   See bpy.types.WorldMistSettings.falloff.
   Type float (used as an index into the type)

gpu.GPU_DYNAMIC_MIST_COLOR

GLSL World Uniforms

gpu.GPU_DYNAMIC_HORIZON_COLOR
   See bpy.types.World.horizon_color.
   Type float3

gpu.GPU_DYNAMIC_AMBIENT_COLOR
   See bpy.types.World.ambient_color.
   Type float3

GLSL Material Uniforms

gpu.GPU_DYNAMIC_MAT_DIFFRGB
   See bpy.types.Material.diffuse_color.
   Type float3

gpu.GPU_DYNAMIC_MAT_REF
   See bpy.types.Material.diffuse_intensity.
   Type float

gpu.GPU_DYNAMIC_MAT_SPECRGB
   See bpy.types.Material.specular_color.
   Type float3

gpu.GPU_DYNAMIC_MAT_SPEC
   See bpy.types.Material.specular_intensity.
   Type float

gpu.GPU_DYNAMIC_MAT_HARD
   See bpy.types.Material.specular_hardness.
   Type float

gpu.GPU_DYNAMIC_MAT_EMIT
   See bpy.types.Material.emit.
   Type float

gpu.GPU_DYNAMIC_MAT_AMB
   See bpy.types.Material.ambient.
   Type float

gpu.GPU_DYNAMIC_MAT_ALPHA
   See bpy.types.Material.alpha.
   Type float
13.3.3 GLSL Attribute Type

Type of the vertex attribute used in the GLSL shader. Determines the mesh custom data layer that contains the vertex attribute.

gpu.CD_MTFACE

Vertex attribute is a UV Map. Data type is vector of 2 float.

There can be more than one attribute of that type, they are differenciated by name. In blender, you can retrieve the attribute data with:

```python
mesh.uv_layers[attribute["name"]]
```

gpu.CD_MCOL

Vertex attribute is color layer. Data type is vector 4 unsigned byte (RGBA).

There can be more than one attribute of that type, they are differenciated by name. In blender you can retrieve the attribute data with:

```python
mesh.vertex_colors[attribute["name"]]
```

gpu.CD_ORCO

Vertex attribute is original coordinates. Data type is vector 3 float.

There can be only 1 attribute of that type per shader. In blender you can retrieve the attribute data with:

```python
mesh.vertices
```

gpu.CD_TANGENT

Vertex attribute is the tangent vector. Data type is vector 4 float.

There can be only 1 attribute of that type per shader. There is currently no way to retrieve this attribute data via the RNA API but a standalone C function to compute the tangent layer from the other layers can be obtained from blender.org.

13.4 Functions

gpu.export_shader (scene, material)

Extracts the GLSL shader producing the visual effect of material in scene for the purpose of reusing the shader in an external engine.

This function is meant to be used in material exporter so that the GLSL shader can be exported entirely.

The return value is a dictionary containing the shader source code and all associated data.

Parameters

- `scene` (`bpy.types.Scene`) – the scene in which the material in rendered.
- `material` (`bpy.types.Material`) – the material that you want to export the GLSL shader

Returns the shader source code and all associated data in a dictionary

Return type dictionary

The dictionary contains the following elements:

- `"fragment"`: `string` fragment shader source code.
- `"vertex"`: `string` vertex shader source code.
• "uniforms": sequence list of uniforms used in fragment shader, can be empty list. Each element of the sequence is a dictionary with the following elements:
  
  - "varname": string name of the uniform in the fragment shader. Always of the form 'unf<number>'.
  
  - "datatype": integer data type of the uniform variable. Can be one of the following:
    * gpu.GPU_DATA_1I: use glUniform1i
    * gpu.GPU_DATA_1F: use glUniform1fv
    * gpu.GPU_DATA_2F: use glUniform2fv
    * gpu.GPU_DATA_3F: use glUniform3fv
    * gpu.GPU_DATA_4F: use glUniform4fv
    * gpu.GPU_DATA_9F: use glUniformMatrix3fv
    * gpu.GPU_DATA_16F: use glUniformMatrix4fv
  
  - "type": integer type of uniform, determines the origin and method of calculation. See uniform-type. Depending on the type, more elements will be be present.
  
  - "lamp": bpy.types.Object Reference to the lamp object from which the uniforms value are extracted. Set for the following uniforms types:
    * gpu.GPU_DYNAMIC_LAMP_DYNVEC
    * gpu.GPU_DYNAMIC_LAMP_DYNCO
    * gpu.GPU_DYNAMIC_LAMP_DYNIMAT
    * gpu.GPU_DYNAMIC_LAMP_DYNPERSMAT
    * gpu.GPU_DYNAMIC_LAMP_DYNENERGY
    * gpu.GPU_DYNAMIC_LAMP_DYNCO
    * gpu.GPU_DYNAMIC_SAMPLER_2DSHADOW

Notes:

* The uniforms gpu.GPU_DYNAMIC_LAMP_DYNVEC, gpu.GPU_DYNAMIC_LAMP_DYNCO, gpu.GPU_DYNAMIC_LAMP_DYNIMAT and gpu.GPU_DYNAMIC_LAMP_DYNPERSMAT refer to the lamp object position and orientation, both of can be derived from the object world matrix:

  ```
  obmat = uniform["lamp"].matrix_world
  ```

  where obmat is the mat4_lamp_to_world matrix of the lamp as a 2 dimensional array, the lamp world location location is in obmat[3].

* The uniform types gpu.GPU_DYNAMIC_LAMP_DYNENERGY and gpu.GPU_DYNAMIC_LAMP_DYNCO refer to the lamp data bloc that you get from:

  ```
  la = uniform["lamp"].data
  ```

  from which you get lamp.energy and lamp.color

* Lamp duplication is not supported: if you have duplicated lamps in your scene (i.e. lamp that are instantiated by dupligroup, etc), this element will only give you a reference to the original lamp and you will not know which instance of the lamp it is referring too. You can still handle that case in the exporter by distributing the uniforms amongst the duplicated lamps.

- "image": bpy.types.Image Reference to the image databloc. Set for uniform type gpu.GPU_DYNAMIC_SAMPLER_2DIMAGE. You can get the image data from:
# full path to image file
uniform["image"].filepath
# image size as a 2-dimensional array of int
uniform["image"].size

- ["texnumber"]: integer Channel number to which the texture is bound when drawing the object. Set for uniform types `gpu.GPU_DYNAMIC_SAMPLER_2DBUFFER`, `gpu.GPU_DYNAMIC_SAMPLER_2DIMAGE` and `gpu.GPU_DYNAMIC_SAMPLER_2DSHADOW`.

This is provided for information only: when reusing the shader outside Blender, you are free to assign the textures to the channel of your choice and to pass that number channel to the GPU in the uniform.

- ["texpixels"]: byte array texture data for uniform type `gpu.GPU_DYNAMIC_SAMPLER_2DBUFFER`. Although the corresponding uniform is a 2D sampler, the texture is always a 1D texture of n x 1 pixel. The texture size n is provided in ["texsize"] element. These textures are only used for computer generated texture (colorband, etc). The texture data is provided so that you can make a real image out of it in the exporter.

- ["texsize"]: integer horizontal size of texture for uniform type `gpu.GPU_DYNAMIC_SAMPLER_2DBUFFER`. The texture data is in ["texpixels"].

• ["attributes"]: sequence list of attributes used in vertex shader, can be empty. Blender doesn’t use standard attributes except for vertex position and normal. All other vertex attributes must be passed using the generic `glVertexAttrib` functions. The attribute data can be found in the derived mesh custom data using RNA. Each element of the sequence is a dictionary containing the following elements:

  - ["varname"]: string name of the uniform in the vertex shader. Always of the form ‘att<number>’.
  
  - ["datatype"]: integer data type of vertex attribute, can be one of the following:
    
      * `gpu.GPU_DATA_2F`: use `glVertexAttrib2fv`
      * `gpu.GPU_DATA_3F`: use `glVertexAttrib3fv`
      * `gpu.GPU_DATA_4F`: use `glVertexAttrib4fv`
      * `gpu.GPU_DATA_4UB`: use `glVertexAttrib4ubv`

  - ["number"]: integer Generic attribute number. This is provided for information only. Blender doesn’t use `glBindAttribLocation` to place generic attributes at specific location, it lets the shader compiler place the attributes automatically and query the placement with `glGetAttribLocation`. The result of this placement is returned in this element.

When using this shader in a render engine, you should either use `glBindAttribLocation` to force the attribute at this location or use `glGetAttribLocation` to get the placement chosen by the compiler of your GPU.

  - ["type"]: integer type of the mesh custom data from which the vertex attribute is loaded. See `attribute-type`.

  - ["name"]: string or integer custom data layer name, used for attribute type `gpu.CD_MTFACE` and `gpu.CD_MCOL`.

Example:
import gpu

# get GLSL shader of material Mat.001 in scene Scene.001
scene = bpy.data.scenes["Scene.001"]
material = bpy.data.materials["Mat.001"]
shader = gpu.export_shader(scene, material)

# scan the uniform list and find the images used in the shader
for uniform in shader["uniforms"]:  
    if uniform["type"] == gpu.GPU_DYNAMIC_SAMPLER_2DIMAGE:
        print("uniform {0} is using image {1}".format(uniform["varname"], uniform["image"].filepath))

# scan the attribute list and find the UV Map used in the shader
for attribute in shader["attributes"]:  
    if attribute["type"] == gpu.CD_MTFACE:
        print("attribute {0} is using UV Map {1}".format(attribute["varname"], attribute["name"]))

13.5 Notes

1. Calculation of the mat4_lamp_to_perspective matrix for a spot lamp.
   The following pseudo code shows how the mat4_lamp_to_perspective matrix is computed in blender for uniforms of `gpu.GPU_DYNAMIC_LAMP_DYNPERSMAT` type:

   # Get the lamp datablock with:
lamp = bpy.data.objects[uniform["lamp"]].data

   # Compute the projection matrix:
   # You will need these lamp attributes:
   # lamp.clipsta : near clip plane in world unit
   # lamp.clipend : far clip plane in world unit
   # lamp.spotsize : angle in degree of the spot light

   # The size of the projection plane is computed with the usual formula:
   wsize = lamp.clista * tan(lamp.spotsize/2)

   # And the projection matrix:
   mat4_lamp_to_perspective = glFrustum(-wsize, wsize, -wsize, wsize, lamp.clista, lamp.clipend)

2. Creation of the shadow map for a spot lamp.
   The shadow map is the depth buffer of a render performed by placing the camera at the spot light position. The size of the shadow map is given by the attribute `lamp.bufsize`: shadow map size in pixel, same size in both dimensions.
## Python Module Index

<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>319</td>
</tr>
<tr>
<td>aud</td>
<td>319</td>
</tr>
<tr>
<td>b</td>
<td>316</td>
</tr>
<tr>
<td>bge.app</td>
<td>316</td>
</tr>
<tr>
<td>bge.constraints</td>
<td>311</td>
</tr>
<tr>
<td>bge.events</td>
<td>302</td>
</tr>
<tr>
<td>bge.logic</td>
<td>241</td>
</tr>
<tr>
<td>bge.render</td>
<td>266</td>
</tr>
<tr>
<td>bge.texture</td>
<td>278</td>
</tr>
<tr>
<td>bge.types</td>
<td>129</td>
</tr>
<tr>
<td>blf</td>
<td>415</td>
</tr>
<tr>
<td>g</td>
<td>421</td>
</tr>
<tr>
<td>gpu</td>
<td>421</td>
</tr>
<tr>
<td>gpu.offscreen</td>
<td>421</td>
</tr>
<tr>
<td>m</td>
<td>329</td>
</tr>
<tr>
<td>mathutils</td>
<td>329</td>
</tr>
<tr>
<td>mathutils.bvhtree</td>
<td>329</td>
</tr>
<tr>
<td>mathutils.geometry</td>
<td>331</td>
</tr>
<tr>
<td>mathutils.interpolate</td>
<td>336</td>
</tr>
<tr>
<td>mathutils.kdtree</td>
<td>336</td>
</tr>
<tr>
<td>mathutils.noise</td>
<td>338</td>
</tr>
</tbody>
</table>
Index

A
   a (KX_VertexProxy attribute), 221
   acceleration (KX_SteeringActuator attribute), 215
   ACCENTGRAVEKEY (in module bge.events), 309
   action (BL_ActionActuator attribute), 132
   activate() (SCA_PythonController method), 235
   activated (SCA_InputEvent attribute), 230
   active (BL_ArmatureConstraint attribute), 140
   active (SCA_InputEvent attribute), 230
   active_camera (KX_Scene attribute), 211
   active_events (SCA_PythonKeyboard attribute), 237
   active_events (SCA_PythonMouse attribute), 238
   activeButtons (SCA_PythonJoystick attribute), 236
   activeInputs (SCA_PythonKeyboard attribute), 237
   activeInputs (SCA_PythonMouse attribute), 238
   activityCulling (KX_Camera attribute), 157
   activityCulling (KX_Scene attribute), 211
   actuator (SCA_ActuatorSensor attribute), 225
   actuators (KX_GameObject attribute), 176
   actuators (SCA_JController attribute), 227
   addDebugProperty() (KX_GameObject method), 187
   addFilter() (KX_2DFilterManager method), 150
   addObject() (KX_Scene method), 211
   addOffScreen() (KX_2DFilter method), 149
   addScene() (in module bge.logic), 244
   addWheel() (KX_VehicleWrapper method), 217
   adjugate() (mathutils.Matrix method), 349
   adjugated() (mathutils.Matrix method), 348
   AKEY (in module bge.events), 304
   alignAxisToVect() (KX_GameObject method), 178
   alpha (bge.texture.ImageMirror attribute), 283
   alpha (bge.texture.ImageRender attribute), 287
   alpha (bge.texture.ImageViewport attribute), 289
   alpha (BL_Texture attribute), 145
   alpha (KX_BlenderMaterial attribute), 154
   ambientColor (KX_WorldInfo attribute), 224
   angle (KX_MouseActuator attribute), 195
   angle (KX_RadarSensor attribute), 206
   angle (mathutils.Quaternion attribute), 355
   ANGULAR_CONSTRAINT (in module bge.constraints), 316
   angularDamping (KX_GameObject attribute), 217
   angularVelocity (KX_SCA_AddObjectActuator attribute), 207
   angularVelocityMax (KX_GameObject attribute), 172
   angularVelocityMin (KX_GameObject attribute), 172
   angV (KX_ObjectActuator attribute), 200
   append() (EXP_ListValue method), 147
   appliedImpulse (KX_CollisionContactPoint attribute), 163
   applyBraking() (KX_VehicleWrapper method), 218
   applyEngineForce() (KX_VehicleWrapper method), 218
   applyForce() (KX_GameObject method), 179
   applyImpulse() (KX_GameObject method), 181
   applyMovement() (KX_GameObject method), 178
   applyRotation() (KX_GameObject method), 179
   applyTorque() (KX_GameObject method), 179
   area_tri() (in module mathutils.geometry), 331
   args (KX_PythonComponent attribute), 205
   arm_head (BL_ArmatureBone attribute), 134
   arm_mat (BL_ArmatureBone attribute), 134
   arm_tail (BL_ArmatureBone attribute), 134
   aspect() (in module bgl), 416
   attenuation (aud.Handle attribute), 326
   attenuation (KX_SoundActuator attribute), 214
   attrDict (KX_GameObject attribute), 214
   aud (module), 319
   AUD_DEVICE_JACK (in module aud), 327
   AUD_DEVICE_NULL (in module aud), 327
   AUD_DEVICE_OPENAL (in module aud), 327
   AUD_DEVICE_SDL (in module aud), 327

435
AUD_DISTANCE_MODEL_EXPONENT (in module aud), 327
AUD_DISTANCE_MODEL_EXPONENT_CLAMPED (in module aud), 327
AUD_DISTANCE_MODEL_INVALID (in module aud), 327
AUD_DISTANCE_MODEL_INVERSE (in module aud), 327
AUD_DISTANCE_MODEL_INVERSE_CLAMPED (in module aud), 327
AUD_DISTANCE_MODEL_LINEAR (in module aud), 327
AUD_DISTANCE_MODEL_LINEAR_CLAMPED (in module aud), 328
AUD_FORMAT_FLOAT32 (in module aud), 328
AUD_FORMAT_FLOAT64 (in module aud), 328
AUD_FORMAT_INVALID (in module aud), 328
AUD_FORMAT_S16 (in module aud), 328
AUD_FORMAT_S24 (in module aud), 328
AUD_FORMAT_U8 (in module aud), 328
AUD_STATUS_INVALID (in module aud), 328
AUD_STATUS_PAUSED (in module aud), 328
AUD_STATUS_PLAYING (in module aud), 328
AUD_STATUS_STOPPED (in module aud), 328
autoDebugList () (in module bge.render), 270
autoUpdate (KX_BoundingBox attribute), 156
autoUpdate (KX_TextureRenderer attribute), 216
axis (KX_CameraActuator attribute), 161
axis (KX_MovementSensor attribute), 196
axis (KX_RadarSensor attribute), 206
axis (KX_RaySensor attribute), 207
axis (mathutils.Quaternion attribute), 355
axis (SCA_JoystickSensor attribute), 231
axisSingle (SCA_JoystickSensor attribute), 230
axisValues (SCA_JoystickSensor attribute), 230
axisValues (SCA_PythonJoystick attribute), 236

B
b (KX_VertexProxy attribute), 221
b (mathutils.Color attribute), 343
background (bge.texture.ImageMirror attribute), 283
background (bge.texture.ImageRender attribute), 287
BACKSLASHKEY (in module bge.events), 309
BACKSPACEKEY (in module bge.events), 309
balance () (mathutils.kdtree.KDTree method), 337
barycentric_transform () (in module mathutils.geometry), 331
batchGroup (KX.GameObject attribute), 176
bbone_segments (BL_ArmatureBone attribute), 134
behavior (KX_SteeringActuator attribute), 215
bge.app (module), 316
bge.constraints (module), 311
bge.events (module), 302
bge.logic (module), 241
bge.render (module), 266
bge.texture (module), 278
bge.types (module), 129
bg1 (module), 379
bind Callbacks (BL_Shader attribute), 141
bindCode (BL_Texture attribute), 146
bindId (bge.texture.Texture attribute), 294
BKEY (in module bge.events), 304
BL_ActionActuator (built-in class), 132
BL_ArmatureActuator (built-in class), 133
BL_ArmatureBone (built-in class), 133
BL_ArmatureChannel (built-in class), 135
BL_ArmatureConstraint (built-in class), 139
BL_ArmatureObject (built-in class), 141
BL_DST_ALPHA (in module bge.logic), 261
BL_DST_COLOR (in module bge.logic), 261
BL_ONE (in module bge.logic), 261
BL_ONE_MINUS_DST_ALPHA (in module bge.logic), 261
BL_ONE_MINUS_DST_COLOR (in module bge.logic), 261
BL_ONE_MINUS_SRC_ALPHA (in module bge.logic), 261
BL_ZERO (in module bge.logic), 261
BL_DST_ALPHA (in module bge.logic), 261
BL_ONE (in module bge.logic), 261
BL_ONE_MINUS_SRC_ALPHA (in module bge.logic), 261
BL_ONE_MINUS_SRC_COLOR (in module bge.logic), 261
BL_TEXTURE (in module bge.logic), 145
BL_ZERO (in module bge.logic), 261
blendIn (BL_ActionActuator attribute), 132
blending (KX_BlenderMaterial attribute), 153
blf (module), 415
blur () (in module bgl), 416
bodies (KX_NetworkMessageSensor attribute), 198
body (KX_NetworkMessageActuator attribute), 198
bone (BL_ArmatureChannel attribute), 135
bone_mat (BL_ArmatureBone attribute), 134
box_fit_2d () (in module mathutils.geometry), 331
box_pack_2d () (in module mathutils.geometry), 331
boxInsideFrustum () (KX_Camera method), 158
breakingThreshold (KX.ConstraintWrapper attribute), 168
Buffer (class in bgl), 412
button (SCA_JoystickSensor attribute), 231
BVHTree (class in mathutils.bvhtree), 329

C
CAM_POS (in module bge.logic), 263
camera (KX_SceneActuator attribute), 212
camera_to_world (KX_Camera attribute), 157
Index 439

distance_model (aud.Device attribute), 320
distance_point_to_plane() (in module mathutils.geometry), 332
distance_reference (aud.Handle attribute), 326
distance_reference (KX_SoundActuator attribute), 214
distanceFactor (KX_LodManager attribute), 191
distribution (SCA_RandomActuator attribute), 239
dLoc (KX_ObjectActuator attribute), 199
doppler_factor (aud.Device attribute), 320
dot() (mathutils.Quaternion method), 354
dot() (mathutils.Vector method), 358
dOWNARROWKEY (in module bge.events), 307
draw() (BL_ArmatureObject method), 141
draw() (in module bgl), 417
draw() (KX_NavMeshObject method), 197
drawLine() (in module bge.render), 269
drawObstacleSimulation() (KX_Scene method), 212
dRot (KX_ObjectActuator attribute), 199
duration (SCA_DelaySensor attribute), 226
duration (SCA_VibrationActuator attribute), 240

e
EIGHTKEY (in module bge.events), 306
EKEY (in module bge.events), 304
emit (BL_Texture attribute), 145
emit (KX_BlenderMaterial attribute), 154
enable() (in module bfl), 417
enabled (BL_Shader attribute), 141
enabled (KX_ConstraintWrapper attribute), 168
enabled (KX_TextureRenderer attribute), 216
enableMotionBlur() (in module bge.render), 269
enableRigidBody() (KX_GameObject method), 182
enableVisibility() (in module bge.render), 268
enableVisualization (KX_SteeringActuator attribute), 215
end() (KX_Scene method), 212
endGame() (in module bge.logic), 243
ENDKEY (in module bge.events), 309
endObject() (KX_GameObject method), 177
energy (KX_LightObject attribute), 188
enforce (BL_ArmatureConstraint attribute), 139
ENTERKEY (in module bge.events), 310
evalLightColor (KX_WorldInfo attribute), 224
evalLightEnabled (KX_WorldInfo attribute), 224
evalLightEnergy (KX_WorldInfo attribute), 224
EQUALKEY (in module bge.events), 309
error (class in aud), 327
error (in module bge.constraints), 315
ESCKEY (in module bge.events), 309
Euler (class in mathutils), 344
Euler.copy() (in module mathutils), 345
Euler.freeze() (in module mathutils), 345
events (SCA_KeyboardSensor attribute), 233
events (SCA_PythonKeyboard attribute), 237
events (SCA_PythonMouse attribute), 238
EventToCharacter() (in module bge.events), 303
EventToString() (in module bge.events), 303
executePriority (SCAILogicBrick attribute), 227
EXP_ListValue (built-in class), 146
EXP_PropValue (built-in class), 148
EXP_PyObjectPlus (built-in class), 148
EXP_Value (built-in class), 148
expandPath() (in module bge.logic), 247
export_shader() (in module gpu), 428
exportBulletFile() (in module bge.constraints), 312
exposure (KX_WorldInfo attribute), 224
extend (bge.texture.DeckLink attribute), 296
EYE (in module bge.logic), 263

F

F10KEY (in module bge.events), 308
F11KEY (in module bge.events), 308
F12KEY (in module bge.events), 308
F13KEY (in module bge.events), 308
F14KEY (in module bge.events), 309
F15KEY (in module bge.events), 309
F16KEY (in module bge.events), 309
F17KEY (in module bge.events), 309
F18KEY (in module bge.events), 309
F19KEY (in module bge.events), 309
F1KEY (in module bge.events), 308
F2KEY (in module bge.events), 308
F3KEY (in module bge.events), 308
F4KEY (in module bge.events), 308
F5KEY (in module bge.events), 308
F6KEY (in module bge.events), 308
F7KEY (in module bge.events), 308
F8KEY (in module bge.events), 308
F9KEY (in module bge.events), 308
Factory (class in aud), 321
Factory.buffer() (in module aud), 321
Factory.delay() (in module aud), 322
Factory.fadein() (in module aud), 322
Factory.fadeout() (in module aud), 322
Factory.file() (in module aud), 321
Factory.filter() (in module aud), 322
Factory.highpass() (in module aud), 323
Factory.join() (in module aud), 323
Factory.limit() (in module aud), 323
Factory.loop() (in module aud), 323
Factory.lowpass() (in module aud), 323
Factory.mix() (in module aud), 324
Factory.pingpong() (in module aud), 324
Factory.pitch() (in module aud), 324
Factory.reverse() (in module aud), 324
Factory.sin() (in module aud), 321
Factory.square() (in module aud), 325
Factory.volume() (in module aud), 325
fallSpeed (KX_CharacterWrapper attribute), 162
far (KX_Camera attribute), 157
fileName (KX_GameActuator attribute), 169
Fill() (mathutils.Vector class method), 357
filter (bge.texture.ImageBuff attribute), 282
filter (bge.texture.ImageFFmpeg attribute), 281
filter (bge.texture.ImageMirror attribute), 284
filter (bge.texture.ImageMix attribute), 285
filter (bge.texture.ImageViewport attribute), 289
filter (bge.texture.VideoDeckLink attribute), 293
filter (bge.texture.VideoFFmpeg attribute), 279
filter() (EXP_ListValue method), 147
FilterBGR24 (class in bge.texture), 296
FilterBlueScreen (class in bge.texture), 296
FilterColor (class in bge.texture), 297
FilterGray (class in bge.texture), 297
FilterLevel (class in bge.texture), 298
filterManager (KX_NavMeshObject method), 337
FilterNormal (class in bge.texture), 299
FilterRGB24 (class in bge.texture), 299
filterRGBA32 (class in bge.texture), 299
find() (mathutils.kdtree.KDTree method), 337
find_nearest() (mathutils.kdtree.KDTree method), 337
find_nearest() (mathutils.bvhtree.BVHTree method), 330
find_nearest_range() (mathutils.bvhtree.BVHTree method), 330
find_range() (mathutils.kdtree.KDTree method), 337
findPath() (KX_NavMeshObject method), 197
finished (KX_LibLoadStatus attribute), 188
FIVEKEY (in module bge.events), 306
FKEY (in module bge.events), 304
flip (bge.texture.ImageBuff attribute), 282
flip (bge.texture.ImageFFmpeg attribute), 281
flip (bge.texture.ImageMirror attribute), 284
flip (bge.texture.ImageMix attribute), 285
flip (bge.texture.ImageViewport attribute), 289
flip (bge.texture.VideoDeckLink attribute), 293
flip (bge.texture.VideoFFmpeg attribute), 279
force (KX_ObjectActuator attribute), 199
forceLimitX (KX_ObjectActuator attribute), 200
forceLimitY (KX_ObjectActuator attribute), 200
forceLimitZ (KX_ObjectActuator attribute), 200
format (aud.Device attribute), 321
FOURKEY (in module bge.events), 306
fov (KX_Camera attribute), 157
fractal() (in module mathutils.noise), 338
frame (BL_ActionActuator attribute), 132
frameEnd (BL_ActionActuator attribute), 132
frameMessageCount (KX_NetworkMessageSensor attribute), 198
framePropName (BL_ActionActuator attribute), 132
 framerate (bge.texture.VideoDeckLink attribute), 293
framerate (bge.texture.VideoFFmpeg attribute), 279
frameStart (BL_ActionActuator attribute), 132
frequency (SCA_ISensor attribute), 228
from_id() (EXP_ListValue method), 147
FromBMesh() (mathutils.bvhtree.BVHTree class method), 329
FromObject() (mathutils.bvhtree.BVHTree class method), 329
FromPolygons() (mathutils.bvhtree.BVHTree class method), 330
frustum_culling (KX_Camera attribute), 157
G
g (KX_VertexProxy attribute), 221

g (mathutils.Color attribute), 343

GILINEAR (in module bge.render), 316
get() (EXP_ListValue method), 147
get() (KX_GameObject method), 186
get() (KX_Scene method), 212
getActionFrame() (KX_GameObject method), 186
getActionName() (KX_GameObject method), 187
getAngularVelocity() (KX_GameObject method), 180
getAnisotropicFiltering() (in module bge.render), 269
getAppliedImpulse() (in module bge.constraints), 312
getAverageFrameRate() (in module bge.logic), 247
getAxisVect() (KX_GameObject method), 178
getBlendFileList() (in module bge.logic), 247
getButtonActiveList() (SCA_JoystickSensor method), 232
getButtonStatus() (SCA_JoystickSensor method), 232
gButtonStatus() (SCA_MouseSensor method), 233
getCameraToWorld() (KX_Camera method), 159
getCharacter() (in module bge.constraints), 313
getClipboard() (SCA_PythonKeyboard method), 237
gGetClockTime() (in module bge.logic), 246
getConstraintId() (KX_ConstraintWrapper method), 218
getConstraintId() (KX_VehicleWrapper method), 218

UPBGE Manual Documentation, Release latest

Index
getConstraintType() (KX_VehicleWrapper method), 218
getCurrentController() (in module bge.logic), 242
gGetCurrentScene() (in module bge.logic), 242
gGetDisplayDimensions() (in module bge.render), 267
gGetDistanceTo() (KX_GameObject method), 182
gGetExitKey() (in module bge.logic), 268
gGetEyeSeparation() (in module bge.logic), 245
gGetFilter() (KX_2DFilterManager method), 150
gGetFocalLength() (in module bge.render), 268
gGetFragmentProg() (BL_Shader method), 142
gGetFrameTime() (in module bge.logic), 246
gGetFullScreen() (in module bge.logic), 267
gGetGLSLMaterialSetting() (in module bge.render), 269
gGetInactiveSceneNames() (in module bge.logic), 243
gGetKeyStatus() (SCA_KeyboardSensor method), 233
gGetLastError() (in module bge.texture), 299
gGetLinearVelocity() (KX_GameObject method), 179
gGetLogicTicRate() (in module bge.logic), 245
gGetMaterial() (KX_PolyProxy method), 202
gGetMaterialIndex() (KX_PolyProxy method), 202
gGetMaterialMode() (in module bge.render), 268
gGetMaterialName() (KX_Mesh method), 192
gGetMaterialName() (KX_PolyProxy method), 202
gGetMaxLogicFrame() (in module bge.logic), 244
gGetMaxPhysicsFrame() (in module bge.logic), 245
gGetMesh() (KX_PolyProxy method), 203
gGetMipmapping() (in module bge.render), 269
gGetNormal() (KX_VertexProxy method), 222
gGetNumVertex() (KX_PolyProxy method), 202
gGetNumWheels() (KX_VehicleWrapper method), 218
gGetParam() (KX_ConstraintWrapper method), 167
gGetPhysicsId() (KX_GameObject method), 182
gGetPhysicsTicRate() (in module bge.logic), 245
gGetPolygon() (KX_Mesh method), 193
gGetProfileInfo() (in module bge.logic), 247
gGetPropertyNames() (KX_GameObject method), 182
gGetRandomFloat() (in module bge.logic), 247
gGetReactionForce() (KX_GameObject method), 180
gGetRealTime() (in module bge.logic), 246
gGetRender() (in module bge.logic), 246
gGetRGBA() (KX_VertexProxy method), 222
gGetSceneList() (in module bge.logic), 243
gGetScreenPosition() (KX_Camera method), 160
gGetScreenRay() (KX_Camera method), 160
gGetScreenVect() (KX_Camera method), 160
gGetShade() (KX_BlenderMaterial method), 153
gGetSource() (bge.texture.ImageMix method), 285
gGetSpectrum() (in module bge.logic), 244
gGetStereoEye() (in module bge.render), 268
gGetTextureBindcode() (KX_BlenderMaterial method), 154
gGetTextureName() (KX_Mesh method), 192
gGetTextureName() (KX_PolyProxy method), 202
gGetTimeScale() (in module bge.logic), 246
gGetUseExternalClock() (in module bge.logic), 246
gGetUV() (KX_VertexProxy method), 221
gGetUV2() (KX_VertexProxy method), 221
gGet VectTo() (KX_GameObject method), 182
gGetVehicleConstraint() (in module bge.constraints), 312
gGetVelocity() (KX_GameObject method), 180
gGetVertex() (KX_Mesh method), 192
gGetVertexArrayLength() (KX_Mesh method), 192
gGetVertexIndex() (KX_PolyProxy method), 203
gGet VertexProg() (BL_Shader method), 142
gGetVsync() (in module bge.render), 270
gGetWeight() (bge.texture.ImageMix method), 285
gGetWheelOrientationQuaternion() (KX_VehicleWrapper method), 218
gGetWheelPosition() (KX_VehicleWrapper method), 218
gGetWheelRotation() (KX_VehicleWrapper method), 219
gGetWindowHeight() (in module bge.render), 267
gGetWindowWidth() (in module bge.render), 267
gGetWorldToCamera() (KX_Camera method), 159
gGetXYZ() (KX_VertexProxy method), 221
gGet ghost (KX_ParentActuator attribute), 201
gGet KKEY (in module bge.events), 422
get globalDict (in module bge.logic), 242
gpu (module), 421
gpu.offscreen (module), 421
GPU_DATA_16F (in module gpu), 423
GPU_DATA_1F (in module gpu), 423
GPU_DATA_1I (in module gpu), 423
GPU_DATA_2F (in module gpu), 423
GPU_DATA_3F (in module gpu), 423
GPU_DATA_4F (in module gpu), 423
GPU_DATA_4UB (in module gpu), 423
GPU_DATA_9F (in module gpu), 423
GPU_DYNAMIC_AMBIENT_COLOR (in module gpu), 427
GPU_DYNAMIC_HORIZON_COLOR (in module gpu), 427
GPU_DYNAMIC_LAMP_ATT1 (in module gpu), 425
GPU_DYNAMIC_LAMP_ATT2 (in module gpu), 426
UPBGE Manual Documentation, Release latest

Index 441
GPU_DYNAMIC_LAMP_DISTANCE (in module gpu), 425
GPU_DYNAMIC_LAMP_DYNCO (in module gpu), 425
GPU_DYNAMIC_LAMP_DYNCOL (in module gpu), 425
GPU_DYNAMIC_LAMP_DYNENERGY (in module gpu), 425
GPU_DYNAMIC_LAMP_DYNMAT (in module gpu), 425
GPU_DYNAMIC_LAMP_DYNPERSMAT (in module gpu), 425
GPU_DYNAMIC_LAMP_DYNVEC (in module gpu), 424
GPU_DYNAMIC_LAMP_SPOTBLEND (in module gpu), 426
GPU_DYNAMIC_LAMP_SPOTSCALE (in module gpu), 426
GPU_DYNAMIC_LAMP_SPOTSIZE (in module gpu), 426
GPU_DYNAMIC_MAT_ALPHA (in module gpu), 427
GPU_DYNAMIC_MAT_AMB (in module gpu), 427
GPU_DYNAMIC_MAT_DIFFRGB (in module gpu), 427
GPU_DYNAMIC_MAT_HARD (in module gpu), 427
GPU_DYNAMIC_MAT_REF (in module gpu), 427
GPU_DYNAMIC_MAT_]_ (in module gpu), 427
GPU_DYNAMIC_MAT_SPEC (in module gpu), 427
GPU_DYNAMIC_MAT_SPECRGB (in module gpu), 427
GPU_DYNAMIC_MAT_SPEC (in module gpu), 427
GPU_DYNAMIC_MIST_COLOR (in module gpu), 427
GPU_DYNAMIC_MIST_DISTANCE (in module gpu), 426
GPU_DYNAMIC_MIST_INTENSITY (in module gpu), 426
GPU_DYNAMIC_MIST_START (in module gpu), 426
GPU_DYNAMIC_OBJECT_VIEWMAT (in module gpu), 424
GPU_DYNAMIC_SAMPLER_2DBUFFER (in module gpu), 426
GPU_DYNAMIC_SAMPLER_2DIMAGE (in module gpu), 426
GPU_DYNAMIC_SAMPLER_2DSHADOW (in module gpu), 426
GPUOffscreen (class in gpu.offscreen), 421
gravity (KX_CharacterWrapper attribute), 161
gravity (KX_GameObject attribute), 176
gravity (KX_Scene attribute), 211
groupMembers (KX_GameObject attribute), 172
groupObject (KX_GameObject attribute), 172

H
h (mathutils.Color attribute), 344
Handle (class in aud), 325
Handle.pause() (in module aud), 325
Handle.resume() (in module aud), 325
Handle.stop() (in module aud), 325
hardness (BL_Texture attribute), 145
hardness (KX_BlenderMaterial attribute), 154
has_i_k (BL_ArmatureChannel attribute), 135
has_joystick (in module bge.app), 317
has_physical (in module bge.app), 317
hasVibration (SCA_VibrationActuator attribute), 240
hat (SCA_JoystickSensor attribute), 232
hatSingle (SCA_JoystickSensor attribute), 231
hatValues (SCA_JoystickSensor attribute), 230
hatValues (SCA_PythonJoystick attribute), 236
HDR_FULL_FLOAT (in module bge.render), 271
HDR_HALF_FLOAT (in module bge.render), 271
HDR_NONE (in module bge.render), 271
head (BL_ArmatureBone attribute), 134
headtail (BL_ArmatureConstraint attribute), 139
height (gpu.offscreen.GPUOffscreen attribute), 422
height (KX_CameraAttribute attribute), 161
hetero_terrain() (in module mathutils.noise), 338
hinge (BL_ArmatureBone attribute), 134
hitMaterial (KX_CollisionSensor attribute), 164
hitMaterial (KX_RaySensor attribute), 207
hitNormal (KX_MouseFocusSensor attribute), 195
hitNormal (KX_RaySensor attribute), 206
hitObject (KX_CollisionSensor attribute), 164
hitObject (KX_MouseFocusSensor attribute), 195
hitObject (KX_RaySensor attribute), 206
hitObjectList (KX_CollisionSensor attribute), 164
hitPosition (KX_MouseFocusSensor attribute), 195
hitPosition (KX_RaySensor attribute), 206
hitUV (KX_MouseFocusSensor attribute), 195
HKEY (in module bge.events), 304
hold1 (SCA_KeyboardSensor attribute), 232
hold2 (SCA_KeyboardSensor attribute), 232
HOMEKEY (in module bge.events), 309
horizon (bge.texture.ImageMirror attribute), 283
horizon (bge.texture.ImageRender attribute), 287
horizonColor (KX_WorldInfo attribute), 224
hsv (mathutils.Color attribute), 344
hybrid_multi_fractal() (in module mathutils.noise), 339
hysteresis (KX_LodLevel attribute), 190

| Identity() (mathutils.Matrix class method), 347
| identity() (mathutils.Matrix method), 349
intersect_line_plane()  (in module mathutils.geometry), 333
intersect_line_sphere()  (in module mathutils.geometry), 333
intersect_line_sphere_2d()  (in module mathutils.geometry), 333
intersect_planar_line()  (in module mathutils.geometry), 333
intersect_point_line()  (in module mathutils.geometry), 334
intersect_point_quad_2d()  (in module mathutils.geometry), 334
intersect_point_tri()  (in module mathutils.geometry), 334
intersect_point_tri_2d()  (in module mathutils.geometry), 334
intersect_ray_tri()  (in module mathutils.geometry), 335
intersect_sphere_sphere_2d()  (in module mathutils.geometry), 335
invalid(EXP_PyObjectPlus attribute), 148
invert()  (SCA_IonSensor attribute), 228
invert_safe()  (mathutils.Matrix method), 349
 inverted_safe()  (mathutils.Matrix method), 350
is3D (BL_Texture attribute), 146
is_frozen (mathutils.Color attribute), 344
is_frozen (mathutils.Euler attribute), 346
is_frozen (mathutils.Matrix attribute), 351
is_frozen (mathutils.Quaternion attribute), 355
is_frozen (mathutils.Vector attribute), 361
is_negative (mathutils.Matrix attribute), 351
is_orthogonal (mathutils.Matrix attribute), 352
is_orthogonal_axis_vectors (mathutils.Matrix attribute), 352
is_wrapped (mathutils.Color attribute), 344
is_wrapped (mathutils.Euler attribute), 346
is_wrapped (mathutils.Matrix attribute), 352
is_wrapped (mathutils.Quaternion attribute), 355
is_wrapped (mathutils.Vector attribute), 361
isCollider()  (KX_PolyProxy method), 202
isPlayingAction()  (KX_GameObject method), 187
isSuspendDynamics()  (KX_GameObject attribute), 171
isValid()  (BL_Shader method), 142
isVibrating (SCA_VibrationActuator attribute), 240
isVisible()  (KX_PolyProxy method), 202

J
JKEY (in module bge.events), 305
KX_LightObject (built-in class), 188
KX_LightObject.HEMI (built-in variable), 188
KX_LightObject.NORMAL (built-in variable), 188
KX_LightObject.Spot (built-in variable), 188
KX_LightObject.SUN (built-in variable), 188
KX_LodLevel (built-in class), 190
KX_LodManager (built-in class), 191
KX_Mesh (built-in class), 191
KX_MOUSE_BUT_LEFT (in module bge.logic), 262
KX_MOUSE_BUT_MIDDLE (in module bge.logic), 262
KX_MOUSE_BUT_RIGHT (in module bge.logic), 262
KX_MouseActuator (built-in class), 194
KX_MouseFocusSensor (built-in class), 195
KX_MOVEMENT_AXIS_ALL_AXIS (in module bge.logic), 249
KX_MOVEMENT_AXIS_NEG_X (in module bge.logic), 249
KX_MOVEMENT_AXIS_NEG_Y (in module bge.logic), 249
KX_MOVEMENT_AXIS_NEG_Z (in module bge.logic), 249
KX_MOVEMENT_AXIS_POS_X (in module bge.logic), 249
KX_MOVEMENT_AXIS_POS_Y (in module bge.logic), 249
KX_MOVEMENT_AXIS_POS_Z (in module bge.logic), 249
KX_MOVEMENT_AXIS_POS_X (in module bge.logic), 249
KX_MOVEMENT_AXIS_POS_Y (in module bge.logic), 249
KX_MOVEMENT_AXIS_POS_Z (in module bge.logic), 249
KX_RADAR_AXIS_POS_X (in module bge.logic), 250
KX_RADAR_AXIS_POS_Y (in module bge.logic), 250
KX_RADAR_AXIS_POS_Z (in module bge.logic), 250
KX_RadarSensor (built-in class), 205
KX_RANDOMACT_BOOL_BERNOULLI (in module bge.logic), 255
KX_RANDOMACT_BOOL_CONST (in module bge.logic), 255
KX_RANDOMACT_BOOL_UNIFORM (in module bge.logic), 255
KX_RANDOMACT_FLOAT_CONST (in module bge.logic), 255
KX_RANDOMACT_FLOAT_UNIFORM (in module bge.logic), 255
KX_RANDOMACT_FLOAT_NORMAL (in module bge.logic), 255
KX_RANDOMACT_FLOAT_UNIFORM (in module bge.logic), 255
KX_RANDOMACT_FLOAT_NEGATIVE_EXPONENTIAL (in module bge.logic), 255
KX_RANDOMACT_FLOAT_NORMAL (in module bge.logic), 255
KX_RANDOMACT_FLOAT_UNIFORM (in module bge.logic), 255
KX_RANDOMACT_FLOAT_NEGATIVE_EXPONENTIAL (in module bge.logic), 255
KX_RAY_AXIS_POS_X (in module bge.logic), 250
KX_RAY_AXIS_NEG_X (in module bge.logic), 250
KX_RAY_AXIS_NEG_Y (in module bge.logic), 250
KX_RAY_AXIS_NEG_Z (in module bge.logic), 250
KX_RAY_AXIS_POS_X (in module bge.logic), 250
KX_RAY_AXIS_POS_Y (in module bge.logic), 250
KX_RAY_AXIS_POS_Z (in module bge.logic), 250
KX_RaySensor (built-in class), 206
KX_SCAD_RemoveObjectActuator (built-in class), 207
KX_SCAD_DynamicActuator (built-in class), 208
KX_SCAD_EndObjectActuator (built-in class), 208
KX_SCAD_ReplaceMeshActuator (built-in class), 208
KX_Scene (built-in class), 209
KX_SCENE_ADD_BACK_SCENE (in module bge.logic), 256
KX_SCENE_ADD_FRONT_SCENE (in module bge.logic), 256
KX_SCENE_REMOVE_SCENE (in module bge.logic), 256
KX_SCENE_RESTART (in module bge.logic), 256
KX_SCENE_RESUME (in module bge.logic), 256
KX_SCENE_SET_CAMERA (in module bge.logic), 256
KX_SCENE_SET_SCENE (in module bge.logic), 256
KX_SCENE_SUSPEND (in module bge.logic), 256
KX_SceneActuator (built-in class), 212
KX_SENSOR_ACTIVE (in module bge.logic), 248
KX_SENSOR_INACTIVE (in module bge.logic), 248
KX_SENSOR JUST_ACTIVATED (in module bge.logic), 248
KX_SENSOR JUST_DEACTIVATED (in module bge.logic), 248
lights (KX_Scene attribute), 210
limit (KX_ConstraintActuator attribute), 165
limit_x (KX_MouseActuator attribute), 195
limit_y (KX_MouseActuator attribute), 195
limits (bge.texture.FilterBlueScreen attribute), 296
lin_attenuation (KX_LightObject attribute), 189
lin_error (BL_ArmatureConstraint attribute), 139
linearDamping (KX_GameObject attribute), 171
linearVelocity (KX_SCA_AddObjectActuator attribute), 207
LINEFEEDKEY (in module bge.events), 310
LINEHINGE_CONSTRAINT (in module bge.constraints), 316
Linspace() (mathutils.NVector class method), 357
linV (KX_ObjectActuator attribute), 199
linVelocityMax (KX_GameObject attribute), 172
linVelocityMin (KX_GameObject attribute), 171
listener_location (aud.Device attribute), 321
listener_orientation (aud.Device attribute), 321
listener_velocity (aud.Device attribute), 321
LKEY (in module bge.events), 305
load() (bge.texture.ImageBuff method), 282
load() (in module bfl), 417
loadGlobalDict() (in module bge.logic), 243
local_x (KX_MouseActuator attribute), 194
local_y (KX_MouseActuator attribute), 194
localAngularVelocity (KX_GameObject attribute), 175
localInertia (KX_GameObject attribute), 172
localLinearVelocity (KX_GameObject attribute), 175
localOrientation (KX_GameObject attribute), 175
localPointA (KX_CollisionContactPoint attribute), 163
localPointB (KX_CollisionContactPoint attribute), 163
localPosition (KX_GameObject attribute), 175
localScale (KX_GameObject attribute), 175
localTransform (KX_GameObject attribute), 175
location (aud.Handle attribute), 326
location (KX_GameObject attribute), 175
dodBias (BL_ArmatureChannel attribute), 136
dodBias (BL_Texture attribute), 146
dodDistanceFactor (KX_Camera attribute), 156
dodDistanceFactor (KX_TextureRenderer attribute), 216
dodManager (KX_GameObject attribute), 177
logicCulling (KX_GameObject attribute), 174
logicCullingRadius (KX_GameObject attribute), 174
loop_count (aud.Handle attribute), 326
M
magnitude (mathutils.Quaternion attribute), 356
magnitude (mathutils.Vector attribute), 361
make_compatible() (mathutils.Euler method), 345
makeScreenshot() (in module bge.render), 267
mask (KX_MouseFocusSensor attribute), 196
mask (KX_RaySensor attribute), 206
mask (KX_StateActuator attribute), 214
mass (KX_GameObject attribute), 171
mass (KX_SCA_DynamicActuator attribute), 208
material (KX_PolyProxy attribute), 201
material_id (KX_PolyProxy attribute), 201
material_name (KX_PolyProxy attribute), 201
materialID() (in module bge.texture), 300
materials (KX_Mesh attribute), 192
mathutils (module), 329
mathutils.bvhtree (module), 329
mathutils.geometry (module), 331
mathutils.interpolate (module), 336
mathutils.kdtree (module), 336
mathutils.noise (module), 338
matrix (bge.texture.FilterColor attribute), 297
Matrix (class in mathutils), 346
Matrix.freeze() (in module mathutils), 349
Matrix.lerp() (in module mathutils), 350
max (KX_BoundingBox attribute), 156
max (KX_CameraActor attribute), 161
max (KX_ConstraintActuator attribute), 165
max (SCA_PropertySensor attribute), 235
maxJumps (KX_CharacterWrapper attribute), 162
maxSlope (KX_CharacterWrapper attribute), 162
median_scale (mathutils.Matrix attribute), 352
merge() (KX_BatchGroup method), 152
mesh (KX_LodLevel attribute), 190
mesh (KX_SCA_ReplaceMeshActuator attribute), 209
meshes (KX_GameObject attribute), 176
MIDDLEMOUSE (in module bge.events), 304
min (KX_BoundingBox attribute), 156
min (KX_CameraActor attribute), 161
min (KX_ConstraintActuator attribute), 165
min (SCA_PropertySensor attribute), 235
MINUSKEY (in module bge.events), 310
mipmap (bge.texture.Texture attribute), 294
mipmap (KX_2DFilter attribute), 148
mirror (BL_Texture attribute), 145
mistColor (KX_WorldInfo attribute), 224
mistDistance (KX_WorldInfo attribute), 223
mistEnable (KX_WorldInfo attribute), 223
mistIntensity (KX_WorldInfo attribute), 224
mistStart (KX_WorldInfo attribute), 223
mistType (KX_WorldInfo attribute), 224
MKEY (in module bge.events), 305
mode (BL_ArmatureChannel attribute), 132
mode (KX_GameActor attribute), 169
mode (KX_ParentActuator attribute), 200
mode (KX_SCA_DynamicActuator attribute), 208
mode (KX_SceneActuator attribute), 213
mode (KX_SoundActuator attribute), 213
mode (SCA_2DFilterActuator attribute), 225
mode (SCA_MouseSensor attribute), 233
mode (SCA_PropertyActuator attribute), 234
mode (SCA_PropertySensor attribute), 234
mode (SCA_PythonController attribute), 235
MODELMATRIX (in module bge.logic), 263
MODELMATRIX_INVERSE (in module bge.logic), 263
MODELMATRIX_INVERSETRANSPOSE (in module bge.logic), 263
MODELMATRIX_TRANSPOSE (in module bge.logic), 263
modelview_matrix (KX_Camera attribute), 157
MODELVIEWMATRIX (in module bge.logic), 263
MODELVIEWMATRIX_INVERSE (in module bge.logic), 263
MODELVIEWMATRIX_INVERSETRANSPOSE (in module bge.logic), 263
MODELVIEWMATRIX_TRANSPOSE (in module bge.logic), 263
mouse (in module bge.logic), 242
MOUSEX (in module bge.events), 304
MOUSEY (in module bge.events), 304
multi_fractal() (in module mathutils.noise), 339

N
name (BL_ArmatureBone attribute), 133
name (BL_ArmatureChannel attribute), 135
name (BL_ArmatureConstraint attribute), 139
name (EXP_Value attribute), 148
name (KX_GameObject attribute), 171
name (KX_Scene attribute), 210
name (SCA_ILogicBrick attribute), 227
name (SCA_PythonJoystick attribute), 236
navmesh (KX_SteeringActuator attribute), 215
near (KX_Camera attribute), 157
neg_ticks (SCA_JSensor attribute), 229
negate() (mathutils.Vector method), 359
NextFrame() (in module bge.logic), 246
NINEKEY (in module bge.events), 306
NKEY (in module bge.events), 305
noise() (in module mathutils.noise), 339
noise_vector() (in module mathutils.noise), 340
normal (BL_Texture attribute), 145
normal (KX_CollisionContactPoint attribute), 163
normal (KX_PlanarMap attribute), 201
normal (KX_VertexProxy attribute), 220
normal() (in module mathutils.geometry), 335
normalize() (mathutils.Matrix method), 350
normalize() (mathutils.Vector method), 359
normalized() (mathutils.Matrix method), 350
normalized() (mathutils.Vector method), 359
numAxis (SCA_JoystickSensor attribute), 231
numAxis (SCA_PythonJoystick attribute), 237
numButtons (SCA_JoystickSensor attribute), 231
numButtons (SCA_PythonJoystick attribute), 237
numHats (SCA_JoystickSensor attribute), 231
numHats (SCA_PythonJoystick attribute), 237
numMaterials (KX_Mesh attribute), 192
numPolygons (KX_Mesh attribute), 192

O
object (KX_CameraActuator attribute), 161
object (KX_ParentActuator attribute), 200
object (KX_PythonComponent attribute), 205
object (KX_SCA_AddObjectActuator attribute), 207
object (KX_TrackToActuator attribute), 217
object_axis (KX_MouseActuator attribute), 194
objectCallbacks (BL_BakeShader attribute), 141
objectLastCreated (KX_SCA_AddObjectActuator attribute), 207
objects (KX_BatchGroup attribute), 152
objects (KX_Scene attribute), 210
objectsInactive (KX_Scene attribute), 210
occlusion (KX_GameObject attribute), 174
offScreen (KX_2DFilter attribute), 149
OKEY (in module bge.events), 305
ONEKEY (in module bge.events), 306
onFinish (KX_LibLoadStatus attribute), 187
onGround (KX_CharacterWrapper attribute), 161
operation (KX_StateActuator attribute), 214
option (KX_ConstraintActuator attribute), 164
order (mathutils.Euler attribute), 346
orientation (aud.Handle attribute), 326
orientation (KX_GameObject attribute), 174
ortho_scale (KX_Camera attribute), 157
orthogonal() (mathutils.Vector method), 359
OrthoProjection() (mathutils.Matrix class method), 347
overlap() (mathutils.bvhtree.BVHTree method), 330
overrideCullingCamera (KX_Scene attribute), 211
owner (mathutils.Color attribute), 344
owner (mathutils.Euler attribute), 346
owner (mathutils.Matrix attribute), 352
owner (mathutils.Quaternion attribute), 356
owner (mathutils.Vector attribute), 361
owner (SCA_ILogicBrick attribute), 227
owner (SCA_PythonController attribute), 235

P
PAD0 (in module bge.events), 307
PAD1 (in module bge.events), 307
PAD2 (in module bge.events), 307
PAD3 (in module bge.events), 307
PAD4 (in module bge.events), 307
PAD5 (in module bge.events), 307
PAD6 (in module bge.events), 307
PAD7 (in module bge.events), 307
PAD8 (in module bge.events), 307
PAD9 (in module bge.events), 307
PADASTERKEY (in module bge.events), 308
PADENTER (in module bge.events), 308
PADMINUS (in module bge.events), 308
PADPERIOD (in module bge.events), 307
PADPLUSKEY (in module bge.events), 308
PADS And (module bge.events), 307
PAGEROWNKEY (in module bge.events), 310
PAGEDOWNKEY (in module bge.events), 310
pad1 (SCA_RandomActuator attribute), 238
pad2 (SCA_RandomActuator attribute), 238
parallaxStep (BL_Texture attribute), 146
parallaxStep (BL_Texture attribute), 146
parent (BL_ArmatureBone attribute), 134
parent (BL_ArmatureChannel attribute), 135
parent (KX_GameObject attribute), 172
passNumber (SCA_2DFilterActuator attribute), 225
path (KX_SteeringActuator attribute), 216
pathUpdatePeriod (KX_SteeringActuator attribute), 215
pause() (bge.texture.VideoDeckLink method), 293
pause() (bge.texture.VideoFFmpeg method), 280
PAUSEKEY (in module bge.events), 310
pauseSound() (KX_SoundActuator method), 214
PERIODKEY (in module bge.events), 310
perspective (KX_Camera attribute), 157
physicsCulling (KX_GameObject attribute), 174
physicsCullingRadius (KX_GameObject attribute), 174
pid (KX_ObjectActuator attribute), 200
pitch (aud.Handle attribute), 326
pitch (KX_SoundActuator attribute), 213
PKKEY (in module bge.events), 305
play() (bge.texture.VideoDeckLink method), 293
play() (bge.texture.VideoFFmpeg method), 280
playAction() (KX_GameObject method), 186
plot() (bge.texture.ImageBuff method), 282
pointInsideFrustum() (KX_Camera method), 159
points_in_planes (mathutils.geometry), 335
POINTTOPPOINT_CONSTRAINT (in module bge.constraints), 316
poly_3d_calc() (in module mathutils.interpolate), 336
polygons (KX_Mesh attribute), 192
pos_ticks (SCA_I Sensor attribute), 229
pose_head (BL_ArmatureChannel attribute), 137
pose_matrix (BL_ArmatureChannel attribute), 137
pose_tail (BL_ArmatureChannel attribute), 137
position (aud.Handle attribute), 327
position (bge.texture.ImageViewport attribute), 290
position (KX_GameObject attribute), 174
position (SCA_MouseSensor attribute), 233
position (SCA_PythonMouse attribute), 238
position() (in module bfl), 417
positive (SCA_I Sensor attribute), 228
post_draw (KX_Scene attribute), 211
pre_draw (KX_Scene attribute), 211
pre_draw_setup (KX_Scene attribute), 211
preseek (bge.texture.VideoFFmpeg attribute), 279
previous (bge.texture.FilterBlueScreen attribute), 296
previous (bge.texture.FilterColor attribute), 297
previous (bge.texture.FilterGray attribute), 297
previous (bge.texture.FilterLevel attribute), 298
previous (bge.texture.FilterNormal attribute), 298
PrintGLInfo () (in module bge.logic), 247
PrintMemInfo () (in module bge.logic), 247
priority (BL_ActionActuator attribute), 132
progress (KX_LibLoadStatus attribute), 188
projection_matrix (KX_Camera attribute), 157
propName (BL_ActionActuator attribute), 132
propName (KX_CollisionSensor attribute), 164
propName (KX_ConstraintActuator attribute), 165
propName (KX_MouseFocusSensor attribute), 196
propName (KX_NetworkMessageActuator attribute), 198
propName (KX_RaySensor attribute), 206
propName (SCA_PropertyActuator attribute), 234
propName (SCA_PropertySensor attribute), 235
propName (SCA_RandomActuator attribute), 239
Q
QKEY (in module bge.events), 305
quad_attenuation (KX_LightObject attribute), 190
Quaternion (class in mathutils), 352
Quaternion.conjugate() (in module mathutils), 353
Quaternion.conjugated() (in module mathutils), 353
Quaternion.copy() (in module mathutils), 353
Quaternion.freeze() (in module mathutils), 354
Quaternion.identity() (in module mathutils), 354
Quaternion.invert() (in module mathutils), 354
Quaternion.inverted() (in module mathutils), 354
Quaternion.negate() (in module mathutils), 354
Quaternion.normalize() (in module mathutils), 354
Quaternion.normalized() (in module mathutils), 354
Quaternion.rotation_difference() (in module mathutils), 354
Quaternion.slerp() (in module mathutils), 354
queue (SCA_InputEvent attribute), 229
QUOTEKEY (in module bge.events), 310
r (KX_VertexProxy attribute), 221
r (mathutils.Color attribute), 344
radius (KX_BoundingBox attribute), 156
random() (in module mathutils.noise), 340
random_unit_vector() (in module mathutils.noise), 340
range (bge.texture.VideoFFmpeg attribute), 278
range (KX_RaySensor attribute), 206
range (KX_WorldInfo attribute), 224
Range() (mathutils.Vector class method), 357
RAS_2DFILTER_BLUR (in module bge.logic), 258
RAS_2DFILTER_CUSTOMFILTER (in module bge.logic), 258
RAS_2DFILTER_DILATION (in module bge.logic), 258
RAS_2DFILTER_DISABLED (in module bge.logic), 258
RAS_2DFILTER_ENABLED (in module bge.logic), 258
RAS_2DFILTER_EROSION (in module bge.logic), 258
RAS_2DFILTER_GRAYSCALE (in module bge.logic), 258
RAS_2DFILTER_INVERT (in module bge.logic), 258
RAS_2DFILTER_LAPLACIAN (in module bge.logic), 258
RAS_2DFILTER_MOTIONBLUR (in module bge.logic), 258
RAS_2DFILTER_NOFILTER (in module bge.logic), 258
RAS_2DFILTER_PREWITT (in module bge.logic), 258
RAS_2DFILTER_SEPIA (in module bge.logic), 258
RAS_2DFILTER_SOBEL (in module bge.logic), 258
RAS_2DFILTER_SHARPEN (in module bge.logic), 258
RAS_2DFILTER_TEXTURE (in module bge.logic), 258
RAS_2DFILTER_TEXTURES (in module bge.logic), 258
RAS_2DFILTER_UV (in module bge.logic), 258
refresh() (bge.texture.ImageMirror method), 284
refresh() (bge.texture.ImageMix method), 286
refresh() (bge.texture.ImageRender method), 288
refresh() (bge.texture.ImageViewport method), 290
refresh() (bge.texture.Textured method), 294
refresh() (bge.texture.VideoFFmpeg method), 280
reinstancePhysicsMesh() (KX_GameObject method), 185
relative (aud.Device attribute), 327
released (SCA_InputEvent attribute), 230
reload() (bge.texture.ImageFFmpeg method), 281
removeConstraint() (in module bge.constraints), 313
removeFilter() (KX_2DFilterManager method), 150
removeOffScreen() (KX_2DFilter method), 150
removeParent() (KX_GameObject method), 182
render() (bge.texture.ImageRender method), 288
renderer (BL_Texture attribute), 146
repeat (bge.texture.VideoFFmpeg attribute), 279
repeat (SCA_DelaySensor attribute), 226
Repeat() (mathutils.Vector class method), 358
replace() (KX_Scene method), 212
replaceMaterial() (KX_Mesh method), 193
replaceMesh() (KX_GameObject method), 177
replacePhysicsShape() (KX_GameObject method), 186
reset() (KX_CharacterWrapper method), 162
reset() (KX_MouseActuator method), 194
reset() (SCA_Sensor method), 229
reset_y (KX_MouseActuator attribute), 194
reset_y (KX_MouseActuator method), 194
resetDistance (KX_NearSensor attribute), 198
resize() (mathutils.Vector method), 360
resize_2d() (mathutils.Vector method), 360
resize_3d() (mathutils.Vector method), 360
resize_4d() (mathutils.Vector method), 360
resize_4x4() (mathutils.Matrix method), 350
resized() (mathutils.Vector method), 360
resolution (KX_FontObject attribute), 169
restart() (KX_Scene method), 212
restartGame() (in module bge.logic), 243
restoreDynamics() (KX_GameObject method), 181
restorePhysics() (KX_GameObject method), 181
resume() (KX_Scene method), 212
RETKEY (in module bge.events), 310
reverse() (EXP_ListValue method), 147
ridged_multi_fractal() (in module mathutils.noise), 340
right (bge.texture.DeckLink attribute), 295
RIGHT_EYE (in module bge.render), 271
RIGHTALTKEY (in module bge.events), 306
RIGHTARROWKEY (in module bge.events), 307
setCollisionMargin() (KX_GameObject method), 185
setContactBreakingThreshold() (in module bge.constraints), 313
setCubeMap() (KX_2DFilter method), 149
setDamping() (KX_GameObject method), 181
setDeactivationAngularThreshold() (in module bge.constraints), 313
setDeactivationLinearThreshold() (in module bge.constraints), 313
setDeactivationTime() (in module bge.constraints), 313
setDebugMode() (in module bge.constraints), 313
setEyeSeparation() (in module bge.render), 268
setFloatConst() (SCA_RandomActuator method), 239
setFloatNegativeExponential() (SCA_RandomActuator method), 240
setFloatNormal() (SCA_RandomActuator method), 239
setFloatUniform() (SCA_RandomActuator method), 239
setFocalLength() (in module bge.render), 268
setFullScreen() (in module bge.render), 267
setGLSLMaterialSetting() (in module bge.render), 269
setGravity() (in module bge.logic), 244
setIntConst() (SCA_RandomActuator method), 239
setIntPoisson() (SCA_RandomActuator method), 239
setIntUniform() (SCA_RandomActuator method), 239
setLinearAirDamping() (in module bge.constraints), 314
setLinearVelocity() (KX_GameObject method), 180
setLogFile() (in module bge.texture), 300
setLogisticTicRate() (in module bge.logic), 245
setMaterialMode() (in module bge.render), 268
setMaxLogicFrame() (in module bge.logic), 245
setMaxPhysicsFrame() (in module bge.logic), 245
setMipmapping() (in module bge.render), 269
setMousePosition() (in module bge.render), 268
setNormal() (KX_VertexProxy method), 222
setNumIterations() (in module bge.constraints), 314
setNumTimeSubSteps() (in module bge.constraints), 314
setOcclusion() (KX_GameObject method), 178
setOnTop() (KX_Camera method), 178
setParam() (KX_ConstraintWrapper method), 165
setParent() (KX_GameObject method), 182
setPhysicsTicRate() (in module bge.logic), 245
setRender() (in module bge.logic), 246
setRGBA() (KX_VertexProxy method), 222
setRollInfluence() (KX_VehicleWrapper method), 219
setSampler() (BL_Shader method), 142
setSolverDamping() (in module bge.constraints), 314
setSolverTau() (in module bge.constraints), 314
setSolverType() (in module bge.constraints), 314
setSource() (bge.texture.ImageMix method), 286
setSource() (BL_Shader method), 142
setSourceList() (BL_Shader method), 142
setSteeringValue() (KX_VehicleWrapper method), 219
setSuspensionCompression() (KX_VehicleWrapper method), 219
setSuspensionDamping() (KX_VehicleWrapper method), 219
setSuspensionStiffness() (KX_VehicleWrapper method), 219
setTexture() (KX_2DFilter method), 149
setTimeScale() (in module bge.logic), 246
setTyreFriction() (KX_VehicleWrapper method), 219
setUniform1f() (BL_Shader method), 143
setUniform1i() (BL_Shader method), 143
setUniform2f() (BL_Shader method), 143
setUniform2i() (BL_Shader method), 143
setUniform3f() (BL_Shader method), 143
setUniform3i() (BL_Shader method), 143
setUniform4f() (BL_Shader method), 144
setUniform4i() (BL_Shader method), 144
setUniformDef() (BL_Shader method), 144
setUniformEye() (BL_Shader method), 145
setUniformfv() (BL_Shader method), 142
setUniformiv() (BL_Shader method), 144
setUniformMatrix3() (BL_Shader method), 144
setUniformMatrix4() (BL_Shader method), 144
setUseEpa() (in module bge.constraints), 315
setUseExternalClock() (in module bge.logic), 247
setUV() (KX_VertexProxy method), 221
setUV2() (KX_VertexProxy method), 222
setVelocity() (KX_CharacterWrapper method), 162
setViewport() (KX_Camera method), 160
setVisible() (KX_GameObject method), 178
setVsync() (in module bge.render), 270
setWeight() (bge.texture.ImageMix method), 286
setWidthSize() (in module bge.render), 267
setXYZ() (KX_VertexProxy method), 221
SEVENKEY (in module bge.events), 306
shader (KX_BlenderMaterial attribute), 153
shaderText (SCA_2DFilterActuator attribute), 225
SHADOW (in module bfl), 419
shadow() (in module bfl), 418
shadow_offset() (in module bfl), 418
shadowBias (KX_LightObject attribute), 189
shadowBindId (KX_LightObject attribute), 189
shadowBleedBias (KX_LightObject attribute), 189
shadowClipStart (KX_LightObject attribute), 189
shadowClipEnd (KX_LightObject attribute), 189
shadowColor (KX_LightObject attribute), 189
shadowFrustumSize (KX_LightObject attribute), 189
shadowMapType (KX_LightObject attribute), 189
shadowMatrix (KX_LightObject attribute), 189
SHD_TANGENT (in module bge.logic), 263
Shear() (mathutils.Matrix class method), 348
shift_x (KX_Camera attribute), 157
shift_y (KX_Camera attribute), 157
showFramerate() (in module bge.render), 270
showMouse() (in module bge.render), 268
showProfile() (in module bge.render), 270
showProperties() (in module bge.render), 270
SIXKEY (in module bge.events), 306
size (bge.texture.ImageBuf attribute), 283
size (bge.texture.ImageFFmpeg attribute), 281
size (bge.texture.ImageMirror attribute), 284
size (bge.texture.ImageMix attribute), 286
size (bge.texture.ImageRender attribute), 288
size (bge.texture.ImageViewport attribute), 290
size (bge.texture.VideoDeckLink attribute), 293
size (bge.texture.VideoFFmpeg attribute), 279
size (KX_FontObject attribute), 169
size() (in module bfl), 418
SKEY (in module bge.events), 305
skippedTicks (SCA_ISensor attribute), 228
SLASHKEY (in module bge.events), 310
sound (KX_SoundActuator attribute), 213
source (bge.texture.DeckLink attribute), 295
source (bge.texture.Texture attribute), 294
SOURCE_EMPTY (in module bge.texture), 300
SOURCE_ERROR (in module bge.texture), 300
SOURCE_PLAYING (in module bge.texture), 300
SOURCE_READY (in module bge.texture), 300
SOURCE_STOPPED (in module bge.texture), 301
SPACEKEY (in module bge.events), 310
specularAlpha (KX_BlenderMaterial attribute), 154
specularColor (KX_BlenderMaterial attribute), 154
specularFactor (BL_Texture attribute), 145
specularIntensity (BL_Texture attribute), 145
specularIntensity (KX_BlenderMaterial attribute), 145
speed_of_sound (aud.Device attribute), 321
sphereInsideFrustum() (KX_Camera method), 158
split() (KX_BatchGroup method), 152
spotblend (KX_LightObject attribute), 190
spotsize (KX_LightObject attribute), 190
start() (KX_PythonComponent method), 205
startGame() (in module bge.logic), 243
startSound() (KX_SoundActuator method), 214
startVibration() (SCA_VibrationActuator method), 241
state (KX_GameObject attribute), 176
state (SCA_IController attribute), 226
staticShadow (KX_LightObject attribute), 190
status (aud.Handle attribute), 327
status (bge.texture.ImageFFmpeg attribute), 280
status (bge.texture.VideoDeckLink attribute), 293
status (bge.texture.VideoFFmpeg attribute), 278
status (SCA_InputEvent attribute), 229
status (SCA_ISensor attribute), 229
stop() (bge.texture.VideoDeckLink method), 293
stop() (bge.texture.VideoFFmpeg method), 280
stopAction() (KX_GameObject method), 186
stopSound() (KX_SoundActuator method), 214
stopVibration() (SCA_VibrationActuator method), 241
strengthLeft (SCA_VibrationActuator attribute), 240
strengthRight (SCA_VibrationActuator attribute), 240
subject (KX_NetworkMessageActuator attribute), 198
subject (KX_NetworkMessageSensor attribute), 198
subjects (KX_NetworkMessageSensor attribute), 198
subtarget (BL_ArmatureActuator attribute), 133
subtarget (BL_ArmatureConstraint attribute), 140
suspend() (KX_Scene method), 212
suspendDynamics() (KX_GameObject method), 181
suspended (KX_Scene attribute), 211
suspendPhysics() (KX_GameObject method), 181

T

TABKEY (in module bge.events), 310
tail (BL_ArmatureBone attribute), 134
tap (SCA_ISensor attribute), 228
target (BL_ArmatureActuator attribute), 133
target (BL_ArmatureConstraint attribute), 140
target (KX_SteeringActuator attribute), 215
targetProperty (SCA_KeyboardSensor attribute), 232
tessellate_polygon() (in module mathutils.geometry), 335
text (KX_FontObject attribute), 169
text (SCA_PythonKeyboard attribute), 237
texts (KX_Scene attribute), 210
Texture (class in bge.texture), 294
texture_name (KX_PolyProxy attribute), 201
textures (KX_BlenderMaterial attribute), 154
THREEKEY (in module bge.events), 306
threshold (KX_MouseActuator attribute), 194
threshold (KX_MovementSensor attribute), 196
threshold (SCA_JoystickSensor attribute), 231
time (KX_ConstraintActuator attribute), 164
time (KX_SCA_AddObjectActuator attribute), 207
time (KX_SoundActuator attribute), 213
time (KX_TrackToActuator attribute), 217
timeOffset (KX_GameObject attribute), 176
timeTaken (KX_LibLoadStatus attribute), 188
TKEY (in module bge.events), 305
to_2d () (mathutils.Vector method), 360
to_3d () (mathutils.Vector method), 361
to_3x3 () (mathutils.Matrix method), 350
to_4d () (mathutils.Vector method), 361
to_4x4 () (mathutils.Matrix method), 350
to_axis_angle () (mathutils.Quaternion method), 355
to_euler () (mathutils.Matrix method), 350
to_quaternion () (mathutils.Quaternion method), 355
to_exponential_map () (mathutils.Quaternion method), 355
to_list () (bgl.Buffer method), 413
to_matrix () (mathutils.Euler method), 346
to_matrix () (mathutils.Quaternion method), 355
to_quaternion () (mathutils.Quaternion method), 355
to_quaternion () (mathutils.Matrix method), 351
to_scale () (mathutils.Matrix method), 351
to_track_quat () (mathutils.Vector method), 361
to_translation () (mathutils.Matrix method), 351
to_tuple () (mathutils.Vector method), 361
toggleProperty (SCA_KeyboardSensor attribute), 232
torque (KX_ObjectActuator attribute), 199
trackAxis (KX_TrackToActuator attribute), 217
transform () (KX_Mesh method), 193
transformUV () (KX_Mesh method), 193
translation (mathutils.Matrix attribute), 352
Translation () (mathutils.Matrix class method), 348
transpose () (mathutils.Matrix method), 351
transposed () (mathutils.Matrix method), 351
triggered (SCA_ISensor attribute), 228
turbulence () (in module mathutils.noise), 341
turbulence_vector () (in module mathutils.noise), 341
turnspeed (KX_SteeringActuator attribute), 215
TWOKEY (in module bge.events), 306
type (BL_ArmatureActuator attribute), 133
type (BL_ArmatureConstraint attribute), 139
type (KX_ArmatureSensor attribute), 151
type (KX_LightObject attribute), 188
type (SCA_InputEvent attribute), 230
U
u (KX_VertexProxy attribute), 220
u2 (KX_VertexProxy attribute), 221
UKEY (in module bge.events), 305
unload () (in module bgl), 418
UPARROWKEY (in module bge.events), 307
upAxis (KX_TrackToActuator attribute), 217
upbge_version (in module bge.app), 317
upbge_version_string (in module bge.app), 317
update () (BL_ArmatureObject method), 214
update () (KX_PythonComponent method), 205
update () (KX_TextureRenderer method), 216
updateShadow (bge.texture.ImageMirror attribute), 283
updateShadow (bge.texture.ImageRenderer attribute), 287
updateShadow () (KX_LightObject method), 190
use3D (KX_TrackToActuator attribute), 217
use_axis_x (KX_MouseActuator attribute), 194
use_axis_y (KX_MouseActuator attribute), 194
useAllKeys (SCA_KeyboardSensor attribute), 232
useContinue (BL_ActionActuator attribute), 132
useDisplayMesh (KX_SCA_ReplaceMeshActuator attribute), 209
useHighPriority (SCA_IController attribute), 227
useHysteresis (KX_LodLevel attribute), 191
useLocalAngV (KX_ObjectActuator attribute), 200
useLocalDLoc (KX_ObjectActuator attribute), 199
useLocalDRot (KX_ObjectActuator attribute), 199
useLocalForce (KX_ObjectActuator attribute), 199
useLocalLinV (KX_ObjectActuator attribute), 199
useLocalTorque (KX_ObjectActuator attribute), 199
useMaterial (KX_CollisionSensor attribute), 164
useMaterial (KX_LodLevel attribute), 190
useMaterial (KX_MouseFocusSensor attribute), 196
useMaterial (KX_RaySensor attribute), 206
useMesh (KX_LodLevel attribute), 190
useNegPulseMode (SCA_ISensor attribute), 228
useOcclusion (KX_VisibilityActuator attribute), 223
usePhysicsMesh (KX_SCA_ReplaceMeshActuator attribute), 209
usePosPulseMode (SCA_ISensor attribute), 228
usePropBody (KX_NetworkMessageActuator attribute), 198
usePulseCollision (KX_CollisionActuator attribute), 164
usePulseFocus (KX_MouseFocusSensor attribute), 196
useRecursion (KX_VisibilityActuator attribute), 223
useRestart (KX_SceneActuator attribute), 213
useShadow (KX_LightObject attribute), 189
useViewport (KX_Camera attribute), 158
useXRay (KX_MouseFocusSensor attribute), 196
useXRay (KX_RaySensor attribute), 206
UV (KX_VertexProxy attribute), 220
uvOffset (BL_Texture attribute), 146
uvRotation (BL_Texture attribute), 146
uvs (KX_VertexProxy attribute), 220
uvSize (BL_Texture attribute), 146

V
v (KX_VertexProxy attribute), 221
v (mathutils.Color attribute), 344
v1 (KX_PolyProxy attribute), 201
v2 (KX_PolyProxy attribute), 201
v2 (KX_VertexProxy attribute), 221
v3 (KX_PolyProxy attribute), 201
v4 (KX_PolyProxy attribute), 202
valid (bge.texture.ImageBuff attribute), 283
valid (bge.texture.ImageFFmpeg attribute), 280
valid (bge.texture.ImageMirror attribute), 284
valid (bge.texture.ImageMix attribute), 286
valid (bge.texture.ImageRender attribute), 288
valid (bge.texture.ImageViewport attribute), 290
valid (bge.texture.VideoDeckLink attribute), 293
valid (bge.texture.VideoFFmpeg attribute), 279
validate() (BL_Shader method), 145
value (KX_ArmatureSensor attribute), 152
value (SCA_2DFilterActuator attribute), 225
value (SCA_PropertyActuator attribute), 234
value (SCA_PropertySensor attribute), 235
values (SCA_InputEvent attribute), 229
variable_lacunarity() (in module mathutils.noise), 341
Vector (class in mathutils), 356
Vector.angle() (in module mathutils), 358
Vector.angle_signed() (in module mathutils), 358
Vector.copy() (in module mathutils), 358
Vector.lerp() (in module mathutils), 359
Vector.project() (in module mathutils), 359
Vector.rotate() (in module mathutils), 360
Vector.rotation_difference() (in module mathutils), 360
Vector.slerp() (in module mathutils), 360
VEHICLE_CONSTRAINT (in module bge.constraints), 316
velocity (aud.Handle attribute), 327
velocity (KX_SteeringActuator attribute), 215
version (in module bge.app), 316
version_char (in module bge.app), 317
version_string (in module bge.app), 317
vertices (KX_PolyProxy attribute), 202
VideoDeckLink (class in bge.texture), 290
VideoFFmpeg (class in bge.texture), 278

VIEWMATRIX (in module bge.logic), 263
VIEWMATRIX_INVERSE (in module bge.logic), 263
VIEWMATRIX_INVERSETRANSPOSE (in module bge.logic), 263
VIEWMATRIX_TRANSPOSE (in module bge.logic), 263
viewpointObject (KX_TextureRenderer attribute), 216
visibility (KX_VisibilityActuator attribute), 223
visible (KX_GameObject attribute), 173
visible (KX_MouseActuator attribute), 194
visible (KX_PolyProxy attribute), 202
visible (SCA_PythonMouse attribute), 238
VKEY (in module bge.events), 305
volume (aud.Device attribute), 321
volume (aud.Handle attribute), 327
volume (KX_SoundActuator attribute), 213
volume_maximum (aud.Handle attribute), 327
volume_maximum (KX_SoundActuator attribute), 213
volume_minimum (aud.Handle attribute), 327
volume_minimum (KX_SoundActuator attribute), 213
volume_tetrahedron() (in module mathutils.geometry), 335
voronoi() (in module mathutils.noise), 342
VSYNC_ADAPTIVE (in module bge.render), 271
VSYNC_OFF (in module bge.render), 270
VSYNC_ON (in module bge.render), 270

W
w (mathutils.Quaternion attribute), 356
w (mathutils.Vector attribute), 362
walkDirection (KX_CharacterWrapper attribute), 162
weight (BL_ArmatureActuator attribute), 133
WHEELEDOWNMOUSE (in module bge.events), 304
WHEELUPMOUSE (in module bge.events), 304
whole (bge.texture.ImageMirror attribute), 285
whole (bge.texture.ImageRenderer attribute), 288
whole (bge.texture.ImageViewport attribute), 290
width (gpu.Offscreen.GPUOffscreen attribute), 422
width (KX_2DFilterOffScreen attribute), 151
WKEY (in module bge.events), 305
WORD_WRAP (in module bge.events), 419
word_wrap() (in module bge.events), 418
world (KX_Scene attribute), 211
world_to_camera (KX_Camera attribute), 158
worldAngularVelocity (KX_GameObject attribute), 176
worldLinearVelocity (KX_GameObject attribute), 175
worldOrientation (KX_GameObject attribute), 175
worldPosition (KX_GameObject attribute), 175
worldScale (KX_GameObject attribute), 175
UPBGE Manual Documentation, Release latest

worldTransform (KX_GameObject attribute), 175
ww (mathutils.Vector attribute), 362
www (mathutils.Vector attribute), 362
wwww (mathutils.Vector attribute), 362
wwwx (mathutils.Vector attribute), 362
wwwy (mathutils.Vector attribute), 362
wwwz (mathutils.Vector attribute), 362
wwx (mathutils.Vector attribute), 362
wwxw (mathutils.Vector attribute), 362
wwxx (mathutils.Vector attribute), 362
wwxy (mathutils.Vector attribute), 362
wwxz (mathutils.Vector attribute), 362
wwy (mathutils.Vector attribute), 362
wwyw (mathutils.Vector attribute), 362
wwyx (mathutils.Vector attribute), 362
wwyy (mathutils.Vector attribute), 362
wwyz (mathutils.Vector attribute), 362
wwz (mathutils.Vector attribute), 362
wwzw (mathutils.Vector attribute), 362
wwzx (mathutils.Vector attribute), 362
wwzy (mathutils.Vector attribute), 362
wwzz (mathutils.Vector attribute), 363
wx (mathutils.Vector attribute), 363
wxw (mathutils.Vector attribute), 363
wxww (mathutils.Vector attribute), 363
wxwx (mathutils.Vector attribute), 363
wxwy (mathutils.Vector attribute), 363
wxwz (mathutils.Vector attribute), 363
wxx (mathutils.Vector attribute), 363
wxxw (mathutils.Vector attribute), 363
wxxx (mathutils.Vector attribute), 363
wxxy (mathutils.Vector attribute), 363
wxxz (mathutils.Vector attribute), 363
wxy (mathutils.Vector attribute), 363
wxyw (mathutils.Vector attribute), 363
wxyx (mathutils.Vector attribute), 363
wxyy (mathutils.Vector attribute), 363
wxyz (mathutils.Vector attribute), 363
wxz (mathutils.Vector attribute), 363
wxzw (mathutils.Vector attribute), 363
wxzx (mathutils.Vector attribute), 363
wxzy (mathutils.Vector attribute), 363
wxzz (mathutils.Vector attribute), 364
wy (mathutils.Vector attribute), 364
wyw (mathutils.Vector attribute), 364
wyww (mathutils.Vector attribute), 364
wywx (mathutils.Vector attribute), 364
wywy (mathutils.Vector attribute), 364
wywz (mathutils.Vector attribute), 364
wyx (mathutils.Vector attribute), 364
wyxw (mathutils.Vector attribute), 364
wyxx (mathutils.Vector attribute), 364
wyxy (mathutils.Vector attribute), 364
wyxz (mathutils.Vector attribute), 364

Index

wyy (mathutils.Vector attribute), 364
wyyw (mathutils.Vector attribute), 364
wyyx (mathutils.Vector attribute), 364
wyyy (mathutils.Vector attribute), 364
wyyz (mathutils.Vector attribute), 364
wyz (mathutils.Vector attribute), 364
wyzw (mathutils.Vector attribute), 364
wyzx (mathutils.Vector attribute), 364
wyzy (mathutils.Vector attribute), 364
wyzz (mathutils.Vector attribute), 365
wz (mathutils.Vector attribute), 365
wzw (mathutils.Vector attribute), 365
wzww (mathutils.Vector attribute), 365
wzwx (mathutils.Vector attribute), 365
wzwy (mathutils.Vector attribute), 365
wzwz (mathutils.Vector attribute), 365
wzx (mathutils.Vector attribute), 365
wzxw (mathutils.Vector attribute), 365
wzxx (mathutils.Vector attribute), 365
wzxy (mathutils.Vector attribute), 365
wzxz (mathutils.Vector attribute), 365
wzy (mathutils.Vector attribute), 365
wzyw (mathutils.Vector attribute), 365
wzyx (mathutils.Vector attribute), 365
wzyy (mathutils.Vector attribute), 365
wzyz (mathutils.Vector attribute), 365
wzz (mathutils.Vector attribute), 365
wzzw (mathutils.Vector attribute), 365
wzzx (mathutils.Vector attribute), 365
wzzy (mathutils.Vector attribute), 365
wzzz (mathutils.Vector attribute), 366

X
x (KX_VertexProxy attribute), 220
x (mathutils.Euler attribute), 346
x (mathutils.Quaternion attribute), 356
x (mathutils.Vector attribute), 366
XKEY (in module bge.events), 305
xw (mathutils.Vector attribute), 366
xww (mathutils.Vector attribute), 366
xwww (mathutils.Vector attribute), 366
xwwx (mathutils.Vector attribute), 366
xwwy (mathutils.Vector attribute), 366
xwwz (mathutils.Vector attribute), 366
xwx (mathutils.Vector attribute), 366
xwxw (mathutils.Vector attribute), 366
xwxx (mathutils.Vector attribute), 366
xwxy (mathutils.Vector attribute), 366
xwxz (mathutils.Vector attribute), 366
xwy (mathutils.Vector attribute), 366
xwyw (mathutils.Vector attribute), 366
xwyx (mathutils.Vector attribute), 366
xwyy (mathutils.Vector attribute), 366
xwyz (mathutils.Vector attribute), 366
457


Z

z (KX_VertexProxy attribute), 220
z (mathutils.Euler attribute), 346
z (mathutils.Quaternion attribute), 356
z (mathutils.Vector attribute), 374
zbuff (bge.texture.ImageRender attribute), 288
zbuff (bge.texture.ImageViewport attribute), 290
zenith (bge.texture.ImageMirror attribute), 283
zenith (bge.texture.ImageRender attribute), 287
zenithColor (KX_WorldInfo attribute), 224
zero() (mathutils.Euler method), 346
zero() (mathutils.Matrix method), 351
zero() (mathutils.Vector method), 361
ZEROKEY (in module bge.events), 306
ZKEY (in module bge.events), 305
zw (mathutils.Vector attribute), 374
zww (mathutils.Vector attribute), 374
zwwx (mathutils.Vector attribute), 374
zwwy (mathutils.Vector attribute), 374
zwwz (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwxh (mathutils.Vector attribute), 374
zwxh (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zwx (mathutils.Vector attribute), 374
zzxx (mathutils.Vector attribute), 376
zzxy (mathutils.Vector attribute), 376
zzxz (mathutils.Vector attribute), 376
zy (mathutils.Vector attribute), 376
zyw (mathutils.Vector attribute), 376
zyww (mathutils.Vector attribute), 376
zywx (mathutils.Vector attribute), 376
zywz (mathutils.Vector attribute), 376
zyx (mathutils.Vector attribute), 376
zyxw (mathutils.Vector attribute), 376
zyxx (mathutils.Vector attribute), 376
zyxy (mathutils.Vector attribute), 376
zyxz (mathutils.Vector attribute), 376
zyy (mathutils.Vector attribute), 376
zyyw (mathutils.Vector attribute), 376
zyyx (mathutils.Vector attribute), 376
zzy (mathutils.Vector attribute), 376
zzyw (mathutils.Vector attribute), 376
zzyx (mathutils.Vector attribute), 376
zzyy (mathutils.Vector attribute), 376
zzyz (mathutils.Vector attribute), 376
zzy (mathutils.Vector attribute), 377
zzyw (mathutils.Vector attribute), 377
zzyx (mathutils.Vector attribute), 377
zzyy (mathutils.Vector attribute), 377
zzyz (mathutils.Vector attribute), 377
zz (mathutils.Vector attribute), 377
zzw (mathutils.Vector attribute), 377
zzww (mathutils.Vector attribute), 377
zzwx (mathutils.Vector attribute), 377
zzwy (mathutils.Vector attribute), 377
zzxz (mathutils.Vector attribute), 377
zzz (mathutils.Vector attribute), 377
zzzw (mathutils.Vector attribute), 378
zzzx (mathutils.Vector attribute), 378
zzzy (mathutils.Vector attribute), 378
zzzz (mathutils.Vector attribute), 378