

pEngine

Generated by Doxygen 1.8.17

---

<b>1 Hierarchical Index</b>	<b>1</b>
1.1 Class Hierarchy	1
<b>2 Class Index</b>	<b>2</b>
2.1 Class List	2
<b>3 File Index</b>	<b>3</b>
3.1 File List	3
<b>4 Class Documentation</b>	<b>4</b>
4.1 Behavior Class Reference	4
4.1.1 Detailed Description	6
4.1.2 Constructor & Destructor Documentation	6
4.1.3 Member Function Documentation	7
4.2 Camera Class Reference	13
4.2.1 Detailed Description	15
4.2.2 Constructor & Destructor Documentation	15
4.2.3 Member Function Documentation	16
4.3 Component Class Reference	23
4.3.1 Detailed Description	23
4.3.2 Member Enumeration Documentation	23
4.3.3 Constructor & Destructor Documentation	24
4.3.4 Member Function Documentation	24
4.4 Editor Class Reference	25
4.4.1 Detailed Description	27
4.4.2 Member Function Documentation	27
4.5 Engine Class Reference	39
4.5.1 Detailed Description	41
4.5.2 Member Function Documentation	41
4.6 File_Reader Class Reference	47
4.6.1 Detailed Description	48
4.6.2 Member Function Documentation	48
4.7 File_Writer Class Reference	55
4.7.1 Detailed Description	56
4.7.2 Constructor & Destructor Documentation	56
4.7.3 Member Function Documentation	56
4.8 Graphics Class Reference	60
4.8.1 Detailed Description	61
4.8.2 Constructor & Destructor Documentation	61
4.8.3 Member Function Documentation	61

4.9 Model Class Reference	66
4.9.1 Detailed Description	68
4.9.2 Constructor & Destructor Documentation	68
4.9.3 Member Function Documentation	69
4.10 Model_Data Class Reference	73
4.10.1 Detailed Description	74
4.10.2 Constructor & Destructor Documentation	74
4.10.3 Member Function Documentation	75
4.11 Model_Data_Manager Class Reference	80
4.11.1 Detailed Description	81
4.11.2 Member Function Documentation	81
4.12 Object Class Reference	83
4.12.1 Detailed Description	85
4.12.2 Constructor & Destructor Documentation	85
4.12.3 Member Function Documentation	86
4.13 Object_Manager Class Reference	94
4.13.1 Detailed Description	95
4.13.2 Member Function Documentation	95
4.14 Physics Class Reference	101
4.14.1 Detailed Description	103
4.14.2 Constructor & Destructor Documentation	103
4.14.3 Member Function Documentation	104
4.15 Random Class Reference	113
4.15.1 Detailed Description	113
4.15.2 Member Function Documentation	113
4.16 Shader Class Reference	116
4.16.1 Detailed Description	117
4.16.2 Member Function Documentation	117
4.17 Texture Class Reference	122
4.17.1 Detailed Description	123
4.17.2 Constructor & Destructor Documentation	123
4.17.3 Member Function Documentation	123
4.18 Texture_Manager Class Reference	126
4.18.1 Detailed Description	127
4.18.2 Member Function Documentation	127
4.19 Trace Class Reference	130
4.19.1 Detailed Description	130
4.19.2 Member Function Documentation	130
4.20 Transform Class Reference	132

4.20.1 Detailed Description . . . . .	133
4.20.2 Constructor & Destructor Documentation . . . . .	133
4.20.3 Member Function Documentation . . . . .	134
4.21 Vector3_Func Class Reference . . . . .	140
4.21.1 Detailed Description . . . . .	141
4.21.2 Member Function Documentation . . . . .	141
<b>5 File Documentation</b>	<b>145</b>
5.1 behavior.cpp File Reference . . . . .	145
5.1.1 Detailed Description . . . . .	145
5.2 behavior.hpp File Reference . . . . .	145
5.2.1 Detailed Description . . . . .	146
5.3 camera.cpp File Reference . . . . .	146
5.3.1 Detailed Description . . . . .	146
5.4 camera.hpp File Reference . . . . .	147
5.4.1 Detailed Description . . . . .	147
5.5 component.cpp File Reference . . . . .	147
5.5.1 Detailed Description . . . . .	147
5.6 component.hpp File Reference . . . . .	148
5.6.1 Detailed Description . . . . .	148
5.7 editor.cpp File Reference . . . . .	148
5.7.1 Detailed Description . . . . .	149
5.8 editor.hpp File Reference . . . . .	149
5.8.1 Detailed Description . . . . .	149
5.9 engine.cpp File Reference . . . . .	150
5.9.1 Detailed Description . . . . .	150
5.10 engine.hpp File Reference . . . . .	150
5.10.1 Detailed Description . . . . .	151
5.11 file_reader.cpp File Reference . . . . .	151
5.11.1 Detailed Description . . . . .	151
5.12 file_reader.hpp File Reference . . . . .	152
5.12.1 Detailed Description . . . . .	152
5.13 file_writer.cpp File Reference . . . . .	152
5.13.1 Detailed Description . . . . .	152
5.14 file_writer.hpp File Reference . . . . .	153
5.14.1 Detailed Description . . . . .	153
5.15 graphics.cpp File Reference . . . . .	153
5.15.1 Detailed Description . . . . .	154
5.16 graphics.hpp File Reference . . . . .	154

---

5.16.1 Detailed Description . . . . .	154
5.17 main.cpp File Reference . . . . .	155
5.17.1 Detailed Description . . . . .	155
5.17.2 Function Documentation . . . . .	155
5.18 model.cpp File Reference . . . . .	156
5.18.1 Detailed Description . . . . .	156
5.19 model.hpp File Reference . . . . .	156
5.19.1 Detailed Description . . . . .	157
5.20 model_data.cpp File Reference . . . . .	157
5.20.1 Detailed Description . . . . .	157
5.21 model_data.hpp File Reference . . . . .	158
5.21.1 Detailed Description . . . . .	158
5.22 model_data_manager.cpp File Reference . . . . .	158
5.22.1 Detailed Description . . . . .	159
5.23 model_data_manager.hpp File Reference . . . . .	159
5.23.1 Detailed Description . . . . .	159
5.24 object.cpp File Reference . . . . .	160
5.24.1 Detailed Description . . . . .	160
5.25 object.hpp File Reference . . . . .	160
5.25.1 Detailed Description . . . . .	161
5.26 object_manager.cpp File Reference . . . . .	161
5.26.1 Detailed Description . . . . .	161
5.27 object_manager.hpp File Reference . . . . .	162
5.27.1 Detailed Description . . . . .	162
5.28 physics.cpp File Reference . . . . .	162
5.28.1 Detailed Description . . . . .	163
5.29 physics.hpp File Reference . . . . .	163
5.29.1 Detailed Description . . . . .	163
5.30 random.cpp File Reference . . . . .	164
5.30.1 Detailed Description . . . . .	164
5.31 random.hpp File Reference . . . . .	164
5.31.1 Detailed Description . . . . .	165
5.32 shader.cpp File Reference . . . . .	165
5.32.1 Detailed Description . . . . .	165
5.33 shader.hpp File Reference . . . . .	166
5.33.1 Detailed Description . . . . .	166
5.34 texture.cpp File Reference . . . . .	166
5.34.1 Detailed Description . . . . .	167
5.35 texture.hpp File Reference . . . . .	167

---

5.35.1 Detailed Description . . . . .	167
5.36 texture_manager.cpp File Reference . . . . .	168
5.36.1 Detailed Description . . . . .	168
5.37 texture_manager.hpp File Reference . . . . .	168
5.37.1 Detailed Description . . . . .	169
5.38 trace.cpp File Reference . . . . .	169
5.38.1 Detailed Description . . . . .	169
5.39 trace.hpp File Reference . . . . .	170
5.39.1 Detailed Description . . . . .	170
5.40 transform.cpp File Reference . . . . .	170
5.40.1 Detailed Description . . . . .	170
5.41 transform.hpp File Reference . . . . .	171
5.41.1 Detailed Description . . . . .	171
5.42 vector3_func.cpp File Reference . . . . .	171
5.42.1 Detailed Description . . . . .	171
5.43 vector3_func.hpp File Reference . . . . .	172
5.43.1 Detailed Description . . . . .	172

<b>Index</b>	<b>173</b>
--------------	------------

## 1 Hierarchical Index

### 1.1 Class Hierarchy

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

<b>Camera</b>	<b>13</b>
<b>Component</b>	<b>23</b>
<b>Behavior</b>	<b>4</b>
<b>Model</b>	<b>66</b>
<b>Physics</b>	<b>101</b>
<b>Transform</b>	<b>132</b>
<b>Editor</b>	<b>25</b>
<b>Engine</b>	<b>39</b>
<b>File_Reader</b>	<b>47</b>
<b>File_Writer</b>	<b>55</b>

<b>Graphics</b>	<b>60</b>
<b>Model_Data</b>	<b>73</b>
<b>Model_Data_Manager</b>	<b>80</b>
<b>Object</b>	<b>83</b>
<b>Object_Manager</b>	<b>94</b>
<b>Random</b>	<b>113</b>
<b>Shader</b>	<b>116</b>
<b>Texture</b>	<b>122</b>
<b>Texture_Manager</b>	<b>126</b>
<b>Trace</b>	<b>130</b>
<b>Vector3_Func</b>	<b>140</b>

## 2 Class Index

### 2.1 Class List

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

<b>Behavior</b>	<b>4</b>
<b>Camera</b>	<b>13</b>
<b>Component</b>	<b>23</b>
<b>Editor</b>	<b>25</b>
<b>Engine</b>	<b>39</b>
<b>File_Reader</b>	<b>47</b>
<b>File_Writer</b>	<b>55</b>
<b>Graphics</b>	<b>60</b>
<b>Model</b>	<b>66</b>
<b>Model_Data</b>	<b>73</b>
<b>Model_Data_Manager</b>	<b>80</b>
<b>Object</b>	<b>83</b>
<b>Object_Manager</b>	<b>94</b>
<b>Physics</b>	<b>101</b>

<a href="#">Random</a>	113
<a href="#">Shader</a>	116
<a href="#">Texture</a>	122
<a href="#">Texture_Manager</a>	126
<a href="#">Trace</a>	130
<a href="#">Transform</a>	132
<a href="#">Vector3_Func</a>	140

## 3 File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">behavior.cpp</a>	145
<a href="#">behavior.hpp</a>	145
<a href="#">camera.cpp</a>	146
<a href="#">camera.hpp</a>	147
<a href="#">component.cpp</a>	147
<a href="#">component.hpp</a>	148
<a href="#">editor.cpp</a>	148
<a href="#">editor.hpp</a>	149
<a href="#">engine.cpp</a>	150
<a href="#">engine.hpp</a>	150
<a href="#">file_reader.cpp</a>	151
<a href="#">file_reader.hpp</a>	152
<a href="#">file_writer.cpp</a>	152
<a href="#">file_writer.hpp</a>	153
<a href="#">graphics.cpp</a>	153
<a href="#">graphics.hpp</a>	154
<a href="#">main.cpp</a>	155
<a href="#">model.cpp</a>	156



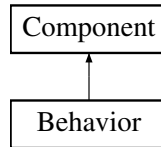
<a href="#">model.hpp</a>	156
<a href="#">model_data.cpp</a>	157
<a href="#">model_data.hpp</a>	158
<a href="#">model_data_manager.cpp</a>	158
<a href="#">model_data_manager.hpp</a>	159
<a href="#">object.cpp</a>	160
<a href="#">object.hpp</a>	160
<a href="#">object_manager.cpp</a>	161
<a href="#">object_manager.hpp</a>	162
<a href="#">physics.cpp</a>	162
<a href="#">physics.hpp</a>	163
<a href="#">random.cpp</a>	164
<a href="#">random.hpp</a>	164
<a href="#">shader.cpp</a>	165
<a href="#">shader.hpp</a>	166
<a href="#">texture.cpp</a>	166
<a href="#">texture.hpp</a>	167
<a href="#">texture_manager.cpp</a>	168
<a href="#">texture_manager.hpp</a>	168
<a href="#">trace.cpp</a>	169
<a href="#">trace.hpp</a>	170
<a href="#">transform.cpp</a>	170
<a href="#">transform.hpp</a>	171
<a href="#">vector3_func.cpp</a>	171
<a href="#">vector3_func.hpp</a>	172

## 4 Class Documentation

### 4.1 Behavior Class Reference

```
#include <behavior.hpp>
```

Inheritance diagram for Behavior:



### Public Member Functions

- [Behavior](#) ()  
*Creates an empty [Behavior](#) object.*
- [Behavior](#) (const [Behavior](#) &other)  
*Copy constructor.*
- [Behavior](#) ([File\\_Reader](#) &reader)  
*Creates [Behavior](#) object using file.*
- [Behavior](#) \* [Clone](#) () const  
*Clones current [Behavior](#) object.*
- [~Behavior](#) ()  
*Deletes all of the lua states.*
- void [Update](#) ()  
*Update for [Behavior](#) object. Calls [Behavior](#) manager giving list of its behaviors.*
- void [Read](#) ([File\\_Reader](#) &reader)  
*Reads in the behaviors to be used.*
- void [Write](#) ([File\\_Writer](#) &writer)  
*Gives the names of each lua file to the writer.*
- void [SetupClassesForLua](#) ()  
*Setups up the interface between the engine and the lua files.*
- std::vector< std::string > & [GetScripts](#) ()  
*Returns list of lua filenames.*
- void [ClassSetup](#) (sol::state \*state)  
*Sends engine variables and functions to lua.*
- bool [SwitchScript](#) (unsigned scriptNum, std::string newScriptName)  
*Switches one script to another (replace)*
- bool [AddScript](#) (std::string newScriptName)  
*Attaching new script to the object.*
- bool [CheckIfCopy](#) (std::string newScriptName)  
*Checks if the script is already attached to the object.*
- void [Clear](#) ()  
*Clears states and state filenames from object.*

### Static Public Member Functions

- static [CType](#) [GetCType](#) ()  
*Gets the CType of [Behavior](#) (used in [Object::GetComponent<>\(\)](#))*

## Private Attributes

- `std::vector< std::string > scripts`  
*Names of the lua scripts being used.*
- `std::vector< sol::state * > states`  
*States of each lua script.*

## Additional Inherited Members

### 4.1.1 Detailed Description

[Behavior](#) class

Definition at line 30 of file `behavior.hpp`.

### 4.1.2 Constructor & Destructor Documentation

#### 4.1.2.1 [Behavior\(\)](#) [1/3] `Behavior::Behavior ( )`

Creates an empty [Behavior](#) object.

Definition at line 29 of file `behavior.cpp`.

```
29 : Component(CType::CBehavior) {}
```

Referenced by `Clone()`.

#### 4.1.2.2 [Behavior\(\)](#) [2/3] `Behavior::Behavior (` `const Behavior & other )`

Copy constructor.

##### Parameters

<i>other</i>	<a href="#">Behavior</a> object to copy
--------------	---

Definition at line 36 of file `behavior.cpp`.

```
36                                     : Component(CType::CBehavior) {  
37     *this = other;  
38 }
```

#### 4.1.2.3 Behavior() [3/3] Behavior::Behavior ( File\_Reader & reader )

Creates Behavior object using file.

##### Parameters

<i>reader</i>	Data from file
---------------	----------------

Definition at line 45 of file behavior.cpp.

```
45                                     : Component(CType::CBehavior) {
46     Read(reader);
47 }
```

References Read().

#### 4.1.2.4 ~Behavior() Behavior::~~Behavior ( )

Deletes all of the lua states.

Definition at line 62 of file behavior.cpp.

```
62     {
63     Clear();
64 }
```

References Clear().

### 4.1.3 Member Function Documentation

#### 4.1.3.1 AddScript() bool Behavior::AddScript ( std::string newScriptName )

Attaching new script to the object.

##### Parameters

<i>newScriptName</i>	
----------------------	--

##### Returns

true

false

Definition at line 235 of file behavior.cpp.

```

235                                     {
236     // Checking if this script is already attached
237     if (CheckIfCopy(newScriptName)) return false;
238     // Setting up new lua state
239     sol::state* state = new sol::state;
240     state->open_libraries(sol::lib::base, sol::lib::math, sol::lib::io, sol::lib::string);
241     states.emplace_back(state);
242     // Adding new script filename to list
243     scripts.emplace_back(newScriptName);
244     ClassSetup(state);
245     // Setting up lua script to run
246     states.back()->script_file(std::string(std::string(getenv("USERPROFILE")) +
"/Documents/pEngine/scripts/" + scripts.back().c_str());
247     (*states.back())["Start"]();
248
249     return true;
250 }

```

References CheckIfCopy(), ClassSetup(), scripts, and states.

Referenced by Editor::Display\_Scripts().

**4.1.3.2 CheckIfCopy()** bool Behavior::CheckIfCopy (   
 std::string newScriptName )

Checks if the script is already attached to the object.

#### Parameters

<i>newScriptName</i>	Name of script being checked
----------------------	------------------------------

#### Returns

true

false

Definition at line 259 of file behavior.cpp.

```

259                                     {
260     // Checking if script is the same as an existing one
261     for (std::string scriptName : scripts) {
262         if (scriptName.compare(newScriptName) == 0) return true;
263     }
264
265     // Script is not a copy
266     return false;
267 }

```

References scripts.

Referenced by AddScript(), and SwitchScript().

**4.1.3.3 ClassSetup()** void Behavior::ClassSetup (   
 sol::state \* state )

Sends engine variables and functions to lua.

## Parameters

state
-------

Definition at line 148 of file behavior.cpp.

```

148                                     {
149     // Getting objects components
150     //Physics* physics = GetParent()->GetComponent<Physics>();
151     //Transform* transform = GetParent()->GetComponent<Transform>();
152
153     // Giving lua random functions
154     state->set_function("random_vec3", Random::random_vec3);
155     state->set_function("random_float", Random::random_float);
156
157     // Giving lua glm::vec3 wrapper class
158     sol::usertype<glm::vec3> vec3_type = state->new_usertype<glm::vec3>("vec3",
159     sol::constructors<glm::vec3(float, float, float), glm::vec3(float)>());
160     // Giving lua glm::vec3 wrapper class variables
161     vec3_type.set("x", &glm::vec3::x);
162     vec3_type.set("y", &glm::vec3::y);
163     vec3_type.set("z", &glm::vec3::z);
164     // Giving lua glm::vec3 wrapper class functions
165     state->set_function("normalize", Vector3_Func::normalize);
166     state->set_function("distance", Vector3_Func::distance);
167     state->set_function("get_direction", Vector3_Func::get_direction);
168     state->set_function("zero_vec3", Vector3_Func::zero_vec3);
169     state->set_function("length", Vector3_Func::length);
170     state->set_function("add_float", Vector3_Func::add_float);
171     state->set_function("add_vec3", Vector3_Func::add_vec3);
172
173     state->set_function("FindObject", sol::overload(sol::resolve<Object*>(int)>(&Object_Manager::FindObject),
174
175     sol::resolve<Object*>(std::string)>(&Object_Manager::FindObject)));
176
177     // Giving lua physics class
178     sol::usertype<Physics> physics_type = state->new_usertype<Physics>("Physics",
179     sol::constructors<Physics(), Physics(const Physics)>());
180     // Giving lua physics class variables
181     physics_type.set("acceleration", sol::property(Physics::GetAccelerationRef, &Physics::SetAcceleration));
182     physics_type.set("forces", sol::property(Physics::GetForcesRef, &Physics::SetForces));
183     physics_type.set("velocity", sol::property(Physics::GetVelocityRef, &Physics::SetVelocity));
184     // Giving lua physics class functions
185     physics_type.set_function("ApplyForce", &Physics::ApplyForce);
186     physics_type.set_function("UpdateGravity", &Physics::UpdateGravity);
187
188     // Giving lua transform class
189     sol::usertype<Transform> transform_type = state->new_usertype<Transform>("Transform",
190     sol::constructors<Transform(), Transform(const Transform)>());
191     // Giving lua transform class variables
192     transform_type.set("position", sol::property(Transform::GetPositionRef,
193     &Transform::SetPosition));
194     transform_type.set("rotation", sol::property(Transform::GetRotationRef,
195     &Transform::SetRotation));
196     transform_type.set("scale", sol::property(Transform::GetScaleRef,
197     &Transform::SetScale));
198     transform_type.set("startPosition", sol::property(Transform::GetStartPositionRef,
199     &Transform::SetStartPosition));
200
201     // Giving lua object class
202     state->set("object", GetParent());
203     sol::usertype<Object> object_type = state->new_usertype<Object>("Object",
204     sol::constructors<Object(), Object(const Object)>());
205     // Giving lua object class variables
206     object_type.set("name", sol::property(Object::GetNameRef, &Object::SetName));
207     object_type.set("id", sol::readonly_property(Object::GetId));
208     object_type.set_function("GetPhysics", &Object::GetComponent<Physics>);
209     object_type.set_function("GetTransform", &Object::GetComponent<Transform>);
210 }

```

References Vector3\_Func::add\_float(), Vector3\_Func::add\_vec3(), Physics::ApplyForce(), Vector3\_Func::distance(), Object\_Manager::FindObject(), Vector3\_Func::get\_direction(), Physics::GetAccelerationRef(), Physics::GetForcesRef(), Object::GetId(), Object::GetNameRef(), Component::GetParent(), Transform::GetPositionRef(), Transform::GetRotationRef(), Transform::GetScaleRef(), Transform::GetStartPositionRef(), Physics::GetVelocityRef(), Vector3\_Func::length(), Vector3\_Func::normalize(), Random::random\_float(), Random::random\_vec3(), Physics::SetAcceleration(),

Physics::SetForces(), Object::SetName(), Transform::SetPosition(), Transform::SetRotation(), Transform::SetScale(), Transform::SetStartPosition(), Physics::SetVelocity(), Physics::UpdateGravity(), and Vector3\_Func::zero\_vec3().

Referenced by AddScript(), and SetupClassesForLua().

#### 4.1.3.4 Clear() `void Behavior::Clear ( )`

Clears states and state filenames from object.

Definition at line 273 of file behavior.cpp.

```
273     {
274         for (sol::state* state : states) {
275             if (!state) continue;
276             delete state;
277             state = nullptr;
278         }
279         states.clear();
280         scripts.clear();
281     }
282 }
```

References scripts, and states.

Referenced by Object::ReRead(), and ~Behavior().

#### 4.1.3.5 Clone() `Behavior * Behavior::Clone ( ) const`

Clones current [Behavior](#) object.

##### Returns

[Behavior](#)\*

Definition at line 54 of file behavior.cpp.

```
54     {
55         return new Behavior(*this);
56     }
```

References Behavior().

#### 4.1.3.6 GetCType() `CType Behavior::GetCType ( ) [static]`

Gets the CType of [Behavior](#) (used in [Object::GetComponent<>\(\)](#))

##### Returns

CType

Definition at line 117 of file behavior.cpp.

```
117     {
118         return CType::CBehavior;
119     }
```

#### 4.1.3.7 GetScripts() `std::vector< std::string > & Behavior::GetScripts ( )`

Returns list of lua filenames.

Returns

`std::vector<std::string>&`

Definition at line 141 of file behavior.cpp.

```
141 { return scripts; }
```

References scripts.

Referenced by Editor::Display\_Scripts().

#### 4.1.3.8 Read() `void Behavior::Read ( File_Reader & reader )`

Reads in the behaviors to be used.

Parameters

<i>reader</i>	Data from file
---------------	----------------

Definition at line 83 of file behavior.cpp.

```
83      {
84      unsigned behavior_num = 0;
85
86      // Reads the name of the lua files
87      while (true) {
88          // Getting the name of the next lua file
89          std::string behavior_name = reader.Read_Behavior_Name("behavior_" + std::to_string(behavior_num));
90          if (behavior_name.compare("") == 0) break;
91          // Adding lua filename to list
92          scripts.emplace_back(behavior_name);
93          ++behavior_num;
94      }
95      // Creating lua state for each of the scripts that were read in
96      for (unsigned i = 0; i < scripts.size(); ++i) {
97          sol::state* state = new sol::state;
98          state->open_libraries(sol::lib::base, sol::lib::math, sol::lib::io, sol::lib::string);
99          states.emplace_back(state);
100      }
101 }
```

References File\_Reader::Read\_Behavior\_Name(), scripts, and states.

Referenced by Behavior(), and Object::ReRead().

#### 4.1.3.9 SetupClassesForLua() `void Behavior::SetupClassesForLua ( )`

Setups up the interface between the engine and the lua files.



Definition at line 125 of file behavior.cpp.

```
125     {
126         for (sol::state* state : states) {
127             ClassSetup(state);
128         }
129
130         for (unsigned i = 0; i < states.size(); ++i) {
131             states[i]->script_file(std::string(std::string(getenv("USERPROFILE")) +
132             "/Documents/pEngine/scripts/" + scripts[i]).c_str());
133             (*states[i])["Start"]();
134         }
135     }
```

References ClassSetup(), scripts, and states.

Referenced by Object\_Manager::ReadList(), and Object::ReRead().

**4.1.3.10 SwitchScript()** bool Behavior::SwitchScript (  
 unsigned scriptNum,  
 std::string newScriptName )

Switches one script to another (replace)

Parameters

<i>scriptNum</i>	
<i>newScriptName</i>	

Returns

true

false

Definition at line 215 of file behavior.cpp.

```
215     {
216         // Checking if this script is already attached
217         if (CheckIfCopy(newScriptName)) return false;
218         if (newScriptName.compare(".lua") == 0) return false;
219         sol::state* state = states[scriptNum];
220         scripts[scriptNum] = newScriptName;
221         // Setting up new lua script
222         state->script_file(std::string(std::string(getenv("USERPROFILE")) + "/Documents/pEngine/scripts/" +
223         scripts[scriptNum]).c_str());
224         (*state)["Start"]();
225         return true;
226     }
```

References CheckIfCopy(), scripts, and states.

Referenced by Editor::Display\_Scripts().

**4.1.3.11 Update()** void Behavior::Update ( )

Update for [Behavior](#) object. Calls [Behavior](#) manager giving list of its behaviors.

Definition at line 71 of file behavior.cpp.

```

71         {
72     for (sol::state* state : states) {
73         if (!state) continue;
74         (*state)["FixedUpdate"] (Engine::GetDt());
75     }
76 }
```

References Engine::GetDt(), and states.

Referenced by Object::Update().

**4.1.3.12 Write()** void Behavior::Write (   
File\_Writer & writer )

Gives the names of each lua file to the writer.

**Parameters**

<i>writer</i>	
---------------	--

Definition at line 108 of file behavior.cpp.

```

108     {
109         writer.Write_Behavior_Name(scripts);
110 }
```

References scripts, and File\_Writer::Write\_Behavior\_Name().

Referenced by Object::Write().

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

- [behavior.hpp](#)
- [behavior.cpp](#)

**4.2 Camera Class Reference**

```
#include <camera.hpp>
```

**Public Member Functions**

- [Camera](#) (int width, int height)  
*Creates a new camera with default values.*

### Static Public Member Functions

- static bool [Initialize](#) ([File\\_Reader](#) &settings)  
*Initializes the camera.*
- static void [Update](#) ()  
*Moves the camera and checks for some other inputs.*
- static void [MouseUpdate](#) (GLFWwindow \*, double xpos, double ypos)  
*Moves the camera using the mouse.*
- static void [Shutdown](#) ()  
*Deletes the camera object if it exists.*
- static glm::vec3 & [GetPosition](#) ()  
*Returns the position of the camera.*
- static glm::vec3 & [GetFront](#) ()  
*Returns the direction of the camera.*
- static glm::vec3 & [GetUp](#) ()  
*Returns the upward direction of the camera.*
- static float [GetFov](#) ()  
*Returns the field of view of the camera.*
- static float [GetNear](#) ()  
*Returns the near view distance of the camera.*
- static float [GetFar](#) ()  
*Returns the far view distance of the camera.*
- static float [GetYaw](#) ()  
*Returns the x rotation of the camera.*
- static float [GetPitch](#) ()  
*Returns the y rotation of the camera.*
- static float & [GetOriginalMoveSpeed](#) ()  
*Returns reference to originalMoveSpeed.*
- static float & [GetOriginalSprintSpeed](#) ()  
*Returns reference to originalSprintSpeed.*
- static float & [GetOriginalSensitivity](#) ()  
*Returns reference to originalSensitivity.*

### Private Attributes

- glm::vec3 [position](#)  
*Position of camera.*
- glm::vec3 [front](#)  
*Direction of camera.*
- glm::vec3 [up](#)  
*90 degree upwards direction of camera*
- float [yaw](#)  
*x rotation*
- float [pitch](#)  
*y rotation*
- std::pair< float, float > [last](#)  
*Last position of mouse on screen.*

- float `fov`  
*Field of view.*
- float `speed`  
*Move speed.*
- float `nearV`  
*Near view distance.*
- float `farV`  
*Far view distance.*
- float `sensitivity`  
*Mouse sensitivity.*
- float `originalMoveSpeed`  
*Initial move speed (speed gets change by delta time)*
- float `originalSprintSpeed`  
*Initial sprint speed.*
- float `originalSensitivity`  
*Original mouse sensitivity.*
- bool `canMoveMouse`  
*Whether the user can move the camera using the mouse.*

### 4.2.1 Detailed Description

`Camera` class ?

Definition at line 26 of file camera.hpp.

### 4.2.2 Constructor & Destructor Documentation

**4.2.2.1 Camera()** `Camera::Camera (`  
     `int width,`  
     `int height )`

Creates a new camera with default values.

Parameters

<i>width</i>	Width of screen
<i>height</i>	Height of screen

Definition at line 33 of file camera.cpp.

```

33         : position(0.f, 0.f, 0.f), front(0.f, 0.f, -1.f),
34         up(0.f, 1.f, 0.f), yaw(-90.f), pitch(0.f), last({ width / 2.f, height / 2.f }),
35         fov(45.f), speed(1), nearV(0.1f), farV(10000.f), sensitivity(1), canMoveMouse(true) {}

```

Referenced by `Initialize()`.

### 4.2.3 Member Function Documentation

#### 4.2.3.1 GetFar() `float Camera::GetFar ( ) [static]`

Returns the far view distance of the camera.

Returns

float

Definition at line 221 of file camera.cpp.

```
221 { return camera->farV; }
```

References camera, and farV.

Referenced by Graphics::Render().

#### 4.2.3.2 GetFov() `float Camera::GetFov ( ) [static]`

Returns the field of view of the camera.

Returns

float

Definition at line 207 of file camera.cpp.

```
207 { return camera->fov; }
```

References camera, and fov.

Referenced by Graphics::Render().

#### 4.2.3.3 GetFront() `glm::vec3 & Camera::GetFront ( ) [static]`

Returns the direction of the camera.

Returns

vec3&

Definition at line 193 of file camera.cpp.

```
193 { return camera->front; }
```

References camera, and front.

Referenced by Graphics::Render().

#### 4.2.3.4 GetNear() `float Camera::GetNear ( ) [static]`

Returns the near view distance of the camera.

##### Returns

float

Definition at line 214 of file camera.cpp.

```
214 { return camera->nearV; }
```

References camera, and nearV.

Referenced by Graphics::Render().

#### 4.2.3.5 GetOriginalMoveSpeed() `float & Camera::GetOriginalMoveSpeed ( ) [static]`

Returns reference to originalMoveSpeed.

##### Returns

float&

Definition at line 242 of file camera.cpp.

```
242 { return camera->originalMoveSpeed; }
```

References camera, and originalMoveSpeed.

Referenced by Editor::Display\_Camera\_Settings().

#### 4.2.3.6 GetOriginalSensitivity() `float & Camera::GetOriginalSensitivity ( ) [static]`

Returns reference to originalSensitivity.

##### Returns

float&

Definition at line 256 of file camera.cpp.

```
256 { return camera->originalSensitivity; }
```

References camera, and originalSensitivity.

Referenced by Editor::Display\_Camera\_Settings().

#### 4.2.3.7 GetOriginalSprintSpeed() `float & Camera::GetOriginalSprintSpeed ( ) [static]`

Returns reference to originalSprintSpeed.

##### Returns

`float&`

Definition at line 249 of file camera.cpp.

```
249 { return camera->originalSprintSpeed; }
```

References camera, and originalSprintSpeed.

Referenced by Editor::Display\_Camera\_Settings().

#### 4.2.3.8 GetPitch() `float Camera::GetPitch ( ) [static]`

Returns the y rotation of the camera.

##### Returns

`float`

Definition at line 235 of file camera.cpp.

```
235 { return camera->pitch; }
```

References camera, and pitch.

#### 4.2.3.9 GetPosition() `glm::vec3 & Camera::GetPosition ( ) [static]`

Returns the position of the camera.

##### Returns

`vec3&`

Definition at line 186 of file camera.cpp.

```
186 { return camera->position; }
```

References camera, and position.

Referenced by Graphics::Render().

#### 4.2.3.10 GetUp() `glm::vec3 & Camera::GetUp ( ) [static]`

Returns the upward direction of the camera.

##### Returns

`vec3&`

Definition at line 200 of file camera.cpp.

```
200 { return camera->up; }
```

References camera, and up.

Referenced by Graphics::Render().

#### 4.2.3.11 GetYaw() `float Camera::GetYaw ( ) [static]`

Returns the x rotation of the camera.

##### Returns

`float`

Definition at line 228 of file camera.cpp.

```
228 { return camera->yaw; }
```

References camera, and yaw.

#### 4.2.3.12 Initialize() `bool Camera::Initialize ( File_Reader & settings ) [static]`

Initializes the camera.

##### Parameters

<i>settings</i>	File that contains settings for the camera
-----------------	--

##### Returns

`true`

`false`

Definition at line 44 of file camera.cpp.

```
44 {
```



```

45     // Initializing the camera
46     camera = new Camera(settings.Read_Int("windowWidth"), settings.Read_Int("windowHeight"));
47     if (!camera) {
48         Trace::Message("Camera was not initialized.");
49         return false;
50     }
51
52     // Getting data from settings file
53     camera->originalMoveSpeed = settings.Read_Float("moveSpeed");
54     camera->originalSprintSpeed = settings.Read_Float("sprintSpeed");
55     camera->originalSensitivity = settings.Read_Float("sensitivity");
56
57     return true;
58 }

```

References camera, Camera(), Trace::Message(), originalMoveSpeed, originalSensitivity, originalSprintSpeed, File\_Reader::Read\_Float(), and File\_Reader::Read\_Int().

Referenced by Engine::Initialize().

**4.2.3.13 MouseUpdate()** void Camera::MouseUpdate (

```

    GLFWwindow * ,
    double xpos,
    double ypos ) [static]

```

Moves the camera using the mouse.

#### Parameters

<i>xpos</i>	x position of the mouse
<i>ypos</i>	y position of the mouse

#### Returns

void

Definition at line 116 of file camera.cpp.

```

116
117     if (!camera->canMoveMouse) {
118         camera->last = { xpos, ypos };
119         return;
120     }
121     // Setting up variables
122     static bool firstMouse = true;
123     std::pair<double, double> mousePos = { xpos, ypos };
124
125     // Setting the camera sens using delta time
126     camera->sensitivity = camera->originalSensitivity * Engine::GetDeltaTime();
127
128     // Checking if this is the first time the function was called
129     if (firstMouse) {
130         camera->last = { mousePos.first, mousePos.second };
131         firstMouse = false;
132     }
133
134     // Finding how far the mouse is from its last position
135     std::pair<float, float> offset = {
136         mousePos.first - camera->last.first,
137         camera->last.second - mousePos.second
138     };
139     // Setting new last position

```

```

140     camera->last = { mousePos.first, mousePos.second };
141
142     // Updating offsets to use the sensitivity of the camera
143     offset.first *= camera->sensitivity;
144     offset.second *= camera->sensitivity;
145
146     // Applying the offset to the camera's direction
147     camera->yaw += offset.first;
148     camera->pitch += offset.second;
149
150     // Stops the camera from circling completely in the y direction
151     if (camera->pitch > 89.f) camera->pitch = 89.f;
152     if (camera->pitch < -89.f) camera->pitch = -89.f;
153
154     // Finding the direction of the camera
155     glm::vec3 tempFront = {
156         std::cos(glm::radians(camera->yaw)) * std::cos(glm::radians(camera->pitch)),
157         std::sin(glm::radians(camera->pitch)),
158         std::sin(glm::radians(camera->yaw)) * std::cos(glm::radians(camera->pitch))
159     };
160     camera->front = glm::normalize(tempFront);
161
162     // Finding the upward direction of the camera
163     glm::vec3 tempUp = { 0.f, 1.f, 0.f };
164     glm::vec3 right = glm::normalize(glm::cross(tempUp, camera->front));
165     glm::vec3 up = glm::cross(camera->front, right);
166     camera->up = up;
167 }

```

References camera, canMoveMouse, front, Engine::GetDeltaTime(), last, originalSensitivity, pitch, sensitivity, up, and yaw.

Referenced by Graphics::Initialize().

#### 4.2.3.14 Shutdown() void Camera::Shutdown ( ) [static]

Deletes the camera object if it exists.

##### Returns

void

Definition at line 174 of file camera.cpp.

```

174     {
175     if (camera) {
176         delete camera;
177         camera = nullptr;
178     }
179 }

```

References camera.

Referenced by Engine::Shutdown().

#### 4.2.3.15 Update() void Camera::Update ( ) [static]

Moves the camera and checks for some other inputs.

##### Returns

void

Definition at line 65 of file camera.cpp.

```

65         {
66             // Checking if the engine should be closed
67             if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_ESCAPE) == GLFW_PRESS) {
68                 if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_ESCAPE) == GLFW_RELEASE) {
69                     glfwSetWindowShouldClose(Graphics::GetWindow(), true);
70                 }
71             }
72
73             // Checking if sprint is being used
74             if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_LEFT_SHIFT) == GLFW_PRESS &&
75                 Editor::GetTakeKeyboardInput()) {
76                 camera->speed = camera->originalSprintSpeed * Engine::GetDeltaTime();
77             }
78             else {
79                 camera->speed = camera->originalMoveSpeed * Engine::GetDeltaTime();
80             }
81
82             // Checking for movement using W, A, S, D, SPACE, and CTRL
83             if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_W) == GLFW_PRESS && Editor::GetTakeKeyboardInput()) {
84                 camera->position += camera->speed * camera->front;
85             }
86             if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_S) == GLFW_PRESS && Editor::GetTakeKeyboardInput()) {
87                 camera->position -= camera->speed * camera->front;
88             }
89             if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_A) == GLFW_PRESS && Editor::GetTakeKeyboardInput()) {
90                 camera->position -= glm::normalize(glm::cross(camera->front, camera->up)) * camera->speed;
91             }
92             if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_D) == GLFW_PRESS && Editor::GetTakeKeyboardInput()) {
93                 camera->position += glm::normalize(glm::cross(camera->front, camera->up)) * camera->speed;
94             }
95             if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_SPACE) == GLFW_PRESS && Editor::GetTakeKeyboardInput()) {
96                 camera->position += camera->speed * camera->up;
97             }
98             if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_LEFT_CONTROL) == GLFW_PRESS &&
99                 Editor::GetTakeKeyboardInput()) {
100                 camera->position -= camera->speed * camera->up;
101             }
102
103             if (glfwGetMouseButton(Graphics::GetWindow(), GLFW_MOUSE_BUTTON_RIGHT) == GLFW_PRESS &&
104                 Editor::GetTakeKeyboardInput()) {
105                 camera->canMoveMouse = true;
106             }
107             if (glfwGetMouseButton(Graphics::GetWindow(), GLFW_MOUSE_BUTTON_RIGHT) == GLFW_RELEASE) {
108                 camera->canMoveMouse = false;
109             }
110         }

```

References camera, canMoveMouse, front, Engine::GetDeltaTime(), Editor::GetTakeKeyboardInput(), Graphics::GetWindow(), originalMoveSpeed, originalSprintSpeed, position, speed, and up.

Referenced by Engine::Update().

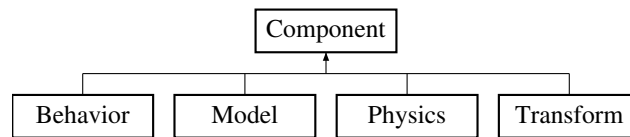
The documentation for this class was generated from the following files:

- [camera.hpp](#)
- [camera.cpp](#)

## 4.3 Component Class Reference

```
#include <component.hpp>
```

Inheritance diagram for Component:



### Public Types

- enum [CType](#) { [CBehavior](#), [CModel](#), [CPhysics](#), [CTransform](#) }

### Public Member Functions

- [Component](#) ([CType](#) type\_)  
*Creates a new component of given type.*
- void [SetParent](#) ([Object](#) \*object)  
*Sets the parent of the component.*
- [Object](#) \* [GetParent](#) () const  
*Gets the parent of the component.*
- [CType](#) [GetCType](#) () const  
*Gets the type of the component.*

### Private Attributes

- [CType](#) type  
*Type of component.*
- [Object](#) \* parent  
*Object that this component is attached to.*

#### 4.3.1 Detailed Description

[Component](#) class

Definition at line 20 of file `component.hpp`.

#### 4.3.2 Member Enumeration Documentation

#### 4.3.2.1 CType enum [Component::CType](#)

Types of components

Definition at line 23 of file component.hpp.

```
23      {  
24          CBehavior,  
25          CModel,  
26          CPhysics,  
27          CTransform,  
28      };
```

### 4.3.3 Constructor & Destructor Documentation

#### 4.3.3.1 Component() [Component::Component](#) ( [CType](#) *type\_* )

Creates a new component of given type.

Parameters

<i>type_↔</i>	Type of component
—	

Definition at line 20 of file component.cpp.

```
20 : type(type_) {}
```

### 4.3.4 Member Function Documentation

#### 4.3.4.1 GetCType() [CType](#) [Component::GetCType](#) ( ) const

Gets the type of the component.

Returns

CType Type of the component

Definition at line 41 of file component.cpp.

```
41 { return type; }
```

References type.

Referenced by [Object::AddComponent\(\)](#).

#### 4.3.4.2 GetParent() `Object * Component::GetParent ( ) const`

Gets the parent of the component.

##### Returns

`Object*` The parent

Definition at line 34 of file `component.cpp`.

```
34 { return parent; }
```

References parent.

Referenced by `Behavior::ClassSetup()`, `Editor::Display_Model()`, `Editor::Display_Physics()`, `Editor::Display_Scripts()`, `Model::Draw()`, `Physics::Update()`, and `Physics::UpdateGravity()`.

#### 4.3.4.3 SetParent() `void Component::SetParent ( Object * object )`

Sets the parent of the component.

##### Parameters

<i>object</i>	The object that is the parent
---------------	-------------------------------

Definition at line 27 of file `component.cpp`.

```
27 { parent = object; }
```

References parent.

Referenced by `Object::AddComponent()`.

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

- [component.hpp](#)
- [component.cpp](#)

## 4.4 Editor Class Reference

```
#include <editor.hpp>
```

### Static Public Member Functions

- static bool [Initialize](#) ()  
*Sets up the config and style of the editor.*
- static void [Update](#) ()  
*Updates the editor content and calls display functions.*
- static void [Render](#) ()  
*Render the editor.*
- static void [Shutdown](#) ()  
*Destroy editor windows and systems.*
- static void [Reset](#) ()  
*Sets selected object to invalid value.*
- static bool [GetTakeKeyboardInput](#) ()  
*Returns whether the program should ignore keyboard input.*

### Private Member Functions

- void [Display\\_Dockspace](#) ()  
*Setup and display the editor's dockspace.*
- void [Display\\_Scene](#) ()  
*Display the scene window.*
- void [Display\\_Components](#) ()  
*Display all of the components of the current selected\_object.*
- void [Display\\_World\\_Settings](#) ()  
*Shows all of the settings of the engine itself.*
- void [Display\\_Camera\\_Settings](#) ()  
*Displays the different camera settings, allows user to change them as needed.*
- void [Display\\_Scripts](#) ([Behavior](#) \*behavior)  
*Displays the different lua scripts attached to the selected object.*
- void [Display\\_Model](#) ([Model](#) \*model)  
*Displays the data of the model being used.*
- void [Display\\_Physics](#) ([Physics](#) \*physics)  
*Shows the [Physics](#) component.*
- void [Display\\_Transform](#) ([Transform](#) \*transform)  
*Display transform data, users can change any of it.*
- void [Display\\_Menu\\_Bar](#) ()  
*Displays menu bar that can be used to save the scene.*

### Private Attributes

- bool [isOpen](#)  
*Whether the editor window is open or not.*
- int [selected\\_object](#)  
*Current object selected in the scene window.*
- int [selected\\_component](#)  
*Current component selected.*
- bool [takeKeyboardInput](#)  
*Whether the program should take keyboard input.*
- int [object\\_to\\_copy](#)  
*[Object](#) that will be copied if paste is used (doesn't need to be the same as selected\_object)*

### 4.4.1 Detailed Description

[Editor](#) class

Definition at line 25 of file editor.hpp.

### 4.4.2 Member Function Documentation

#### 4.4.2.1 Display\_Camera\_Settings() `void Editor::Display_Camera_Settings ( ) [private]`

Displays the different camera settings, allows user to change them as needed.

Definition at line 410 of file editor.cpp.

```

410     {
411         ImGui::Begin("Camera Settings");
412
413         ImGui::PushItemWidth(137);
414
415         // Default move speed
416         ImGui::Text("Move Speed");
417         ImGui::SameLine(100); ImGui::InputFloat("##2", &Camera::GetOriginalMoveSpeed());
418
419         // Move speed when holding shift
420         ImGui::Text("Sprint Speed");
421         ImGui::SameLine(100); ImGui::InputFloat("##3", &Camera::GetOriginalSprintSpeed());
422
423         // Mouse sensitivity when looking around
424         ImGui::Text("Sensitivity");
425         ImGui::SameLine(100); ImGui::InputFloat("##4", &Camera::GetOriginalSensitivity());
426
427         ImGui::PopItemWidth();
428
429         ImGui::End();
430     }
```

References [Camera::GetOriginalMoveSpeed\(\)](#), [Camera::GetOriginalSensitivity\(\)](#), and [Camera::GetOriginalSprintSpeed\(\)](#).

Referenced by [Update\(\)](#).

#### 4.4.2.2 Display\_Components() `void Editor::Display_Components ( ) [private]`

Display all of the components of the current selected\_object.

Definition at line 271 of file editor.cpp.

```

271     {
272         ImGui::Begin("Components##1");
273
274         if (selected_object == -1) { ImGui::End(); return; }
275         Object* object = Object_Manager::FindObject(selected_object);
276         std::string objectName = object->GetName();
277
278         ImGui::Text("Id: %d", object->GetId());
279
280         // Display name box (allows changing the name of an object)
281         static char nameBuf[128] = "";
282         sprintf(nameBuf, objectName.c_str());
```



```

283
284     if (ImGui::InputText("Name", nameBuf, 128, ImGuiInputTextFlags_EnterReturnsTrue)) {
285         object->SetName(std::string(nameBuf));
286     }
287
288     if (ImGui::IsItemDeactivatedAfterEdit()) {
289         object->SetName(std::string(nameBuf));
290     }
291
292     // Template used by the selected object
293     ImGui::Text("Template:");
294     ImGui::SameLine(100);
295     std::string templateName = object->GetTemplateName();
296     if (templateName.empty()) templateName = "No template##1";
297     if (ImGui::Button(templateName.c_str())) {
298         ImGuiFileDialog::Instance()->OpenDialog("ChooseTemplate##1", "Choose File", ".json",
299         std::string(getenv("USERPROFILE")) + "/Documents/pEngine/json/objects/");
300     }
301
302     ImGui::SameLine();
303     if (ImGui::Button("New Template")) {
304         object->Write();
305     }
306
307     if (ImGuiFileDialog::Instance()->Display("ChooseTemplate##1")) {
308         if (ImGuiFileDialog::Instance()->IsOk()) {
309             std::string filePathName = ImGuiFileDialog::Instance()->GetCurrentFileName();
310             object->ReRead(filePathName);
311         }
312
313         ImGuiFileDialog::Instance()->Close();
314     }
315
316     // Getting all of the components
317     Behavior* behavior = object->GetComponent<Behavior>();
318     Model* model = object->GetComponent<Model>();
319     Physics* physics = object->GetComponent<Physics>();
320     Transform* transform = object->GetComponent<Transform>();
321
322     // Display all of the components of the selected_object
323     Display_Transform(transform);
324     Display_Physics(physics);
325     Display_Model(model);
326     Display_Scripts(behavior);
327
328     ImGui::Separator();
329
330     // Button to add new components to the selected_object
331     if (ImGui::Button("Add Component##1")) {
332         ImGui::OpenPopup("New Component##1");
333     }
334
335     // Add new components to object (only ones that the object doesn't already have)
336     if (ImGui::BeginPopup("New Component##1")) {
337         if (!physics) {
338             if (ImGui::Selectable("Physics##1")) {
339                 physics = new Physics;
340                 object->AddComponent(physics);
341             }
342         }
343         if (!model) {
344             if (ImGui::Selectable("Model##1")) {
345                 model = new Model;
346                 object->AddComponent(model);
347             }
348         }
349         if (!behavior) {
350             if (ImGui::Selectable("Scripts##1")) {
351                 behavior = new Behavior;
352                 object->AddComponent(behavior);
353             }
354         }
355         ImGui::EndPopup();
356     }
357
358     ImGui::End();

```

References Display\_Model(), Display\_Physics(), Display\_Scripts(), Display\_Transform(), Object\_Manager::FindObject(), Object::GetId(), and selected\_object.

Referenced by Update().

#### 4.4.2.3 Display\_Dockspace() void Editor::Display\_Dockspace ( ) [private]

Setup and display the editor's dockspace.

Definition at line 155 of file editor.cpp.

```

155     {
156         // Setting up viewport
157         ImGuiViewport* viewport = ImGui::GetMainViewport();
158         ImGui::SetNextWindowPos(viewport->Pos);
159         ImGui::SetNextWindowSize(viewport->Size);
160         ImGui::SetNextWindowViewport(viewport->ID);
161         ImGui::SetNextWindowBgAlpha(0.0f);
162
163         // Setting up window flags
164         ImGuiWindowFlags window_flags = ImGuiWindowFlags_MenuBar | ImGuiWindowFlags_NoDocking;
165         window_flags |= ImGuiWindowFlags_NoTitleBar | ImGuiWindowFlags_NoCollapse | ImGuiWindowFlags_NoResize |
        ImGuiWindowFlags_NoMove;
166         window_flags |= ImGuiWindowFlags_NoBringToFrontOnFocus | ImGuiWindowFlags_NoNavFocus;
167
168         // Setting up window style
169         ImGui::PushStyleVar(ImGuiStyleVar_WindowRounding, 0.0f);
170         ImGui::PushStyleVar(ImGuiStyleVar_WindowBorderSize, 0.0f);
171         ImGui::PushStyleVar(ImGuiStyleVar_WindowPadding, ImVec2(0.0f, 0.0f));
172
173         // Making the window
174         ImGui::SetNextWindowBgAlpha(0.0f);
175         ImGui::Begin("Editor Window", &editor->isOpen, window_flags);
176         ImGui::PopStyleVar(3);
177
178         // Setting up window settings
179         ImGuiID dockspace_id = ImGui::GetID("Editor");
180         ImGuiDockNodeFlags dockspace_flags = ImGuiDockNodeFlags_PassthruCentralNode |
        ImGuiDockNodeFlags_NoDockingInCentralNode;
181         ImGui::DockSpace(dockspace_id, ImVec2(0.0f, 0.0f), dockspace_flags);
182         editor->Display_Menu_Bar();
183         ImGui::End();
184     }

```

References Display\_Menu\_Bar(), editor, and isOpen.

Referenced by Update().

#### 4.4.2.4 Display\_Menu\_Bar() void Editor::Display\_Menu\_Bar ( ) [private]

Displays menu bar that can be used to save the scene.

Definition at line 692 of file editor.cpp.

```

692     {
693         static bool saveAs = false;
694         if (ImGui::BeginMenuBar()) {
695             if (ImGui::BeginMenu("File##1")) {
696                 if (ImGui::MenuItem("Save##1")) {
697                     Engine::Write();
698                 }
699                 if (ImGui::MenuItem("Save As..##1")) {
700                     saveAs = true;
701                 }
702             }
703             ImGui::EndMenu();
704         }
705         if (saveAs) {
706             static char nameBuf[128] = "";

```

```

707         sprintf(nameBuf, Engine::GetPresetName().c_str());
708         if (ImGui::InputText("Name", nameBuf, 128, ImGuiInputTextFlags_EnterReturnsTrue)) {
709             Engine::SetPresetName(std::string(nameBuf));
710             Engine::Write();
711             saveAs = false;
712         }
713
714         if (ImGui::IsItemDeactivatedAfterEdit()) {
715             Engine::SetPresetName(std::string(nameBuf));
716             Engine::Write();
717             saveAs = false;
718         }
719     }
720
721     ImGui::EndMenuBar();
722 }
723 }

```

References Engine::GetPresetName(), Engine::SetPresetName(), and Engine::Write().

Referenced by Display\_Dockspace().

**4.4.2.5 Display\_Model()** void Editor::Display\_Model (  
     Model \* model ) [private]

Displays the data of the model being used.

#### Parameters

<i>model</i>	
--------------	--

Definition at line 514 of file editor.cpp.

```

514                                     {
515         if (!model) return;
516
517         std::string modelName = model->GetModelName();
518         std::string textureName = model->GetTextureName();
519
520         if (modelName.compare("") == 0) modelName = "no model";
521         if (textureName.compare("") == 0) textureName = "no texture";
522
523         // Setting up tree flags
524         ImGuiTreeNodeFlags node_flags = ImGuiTreeNodeFlags_SpanAvailWidth | ImGuiTreeNodeFlags_OpenOnDoubleClick
| ImGuiTreeNodeFlags_OpenOnArrow;
525         if (selected_component == CType::CModel) node_flags |= ImGuiTreeNodeFlags_Selected;
526
527         const bool model_open = ImGui::TreeNodeEx((void*)(intptr_t)CType::CModel, node_flags, "Model");
528         if (ImGui::IsItemClicked()) selected_component = CType::CModel;
529
530         // Right click behavior to delete model component from selected object
531         if (ImGui::IsItemClicked(ImGuiMouseButton_Right)) {
532             selected_component = CType::CModel;
533             ImGui::OpenPopup("DeleteModel##1");
534         }
535
536         if (ImGui::BeginPopup("DeleteModel##1")) {
537             if (ImGui::Selectable("Delete##3")) {
538                 model->GetParent()->RemoveComponent<Model>();
539                 selected_component = -1;
540             }
541             ImGui::EndPopup();
542         }
543
544         if (model_open) {
545             // Model that is being used
546             ImGui::Text("Model"); ImGui::SameLine(100);

```

```

547         if (ImGui::Button(modelName.c_str())) {
548             ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDlgKey##1", "Choose File", ".obj",
std::string(getenv("USERPROFILE")) + "/Documents/pEngine/models/");
549         }
550
551         if (ImGuiFileDialog::Instance()->Display("ChooseFileDlgKey##1")) {
552             if (ImGuiFileDialog::Instance()->IsOk()) {
553                 std::string filePathName = ImGuiFileDialog::Instance()->GetCurrentFileName();
554                 model->SwitchModel(filePathName);
555             }
556
557             ImGuiFileDialog::Instance()->Close();
558         }
559
560         // Texture that is being used
561         ImGui::Text("Texture"); ImGui::SameLine(100);
562         if (ImGui::Button(textureName.c_str())) {
563             ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDlgKey##2", "Choose File", ".dds, .DDS",
std::string(getenv("USERPROFILE")) + "/Documents/pEngine/textures/");
564         }
565
566         if (ImGuiFileDialog::Instance()->Display("ChooseFileDlgKey##2")) {
567             if (ImGuiFileDialog::Instance()->IsOk()) {
568                 std::string filePathName = ImGuiFileDialog::Instance()->GetCurrentFileName();
569                 model->SwitchTexture(filePathName);
570             }
571
572             ImGuiFileDialog::Instance()->Close();
573         }
574
575         ImGui::TreePop();
576     }
577 }

```

References `Model::GetModelName()`, `Component::GetParent()`, `Model::GetTextureName()`, `Object::RemoveComponent()`, `selected_component`, `Model::SwitchModel()`, and `Model::SwitchTexture()`.

Referenced by `Display_Components()`.

**4.4.2.6 Display\_Physics()** `void Editor::Display_Physics ( Physics * physics ) [private]`

Shows the `Physics` component.

#### Parameters

<code>physics</code>	
----------------------	--

Definition at line 584 of file `editor.cpp`.

```

584                                     {
585         if (!physics) return;
586
587         glm::vec3& velocity = physics->GetVelocityRef();
588         glm::vec3& rotVel = physics->GetRotationalVelocityRef();
589
590         ImGuiTreeNodeFlags node_flags = ImGuiTreeNodeFlags_SpanAvailWidth | ImGuiTreeNodeFlags_OpenOnDoubleClick
| ImGuiTreeNodeFlags_OpenOnArrow;
591         if (selected_component == CType::CPhysics) node_flags |= ImGuiTreeNodeFlags_Selected;
592
593         const bool physics_open = ImGui::TreeNodeEx((void*)(intptr_t)CType::CPhysics, node_flags, "Physics");
594         if (ImGui::IsItemClicked()) selected_component = CType::CPhysics;
595
596         if (ImGui::IsItemClicked(ImGuiMouseButton_Right)) {
597             selected_component = CType::CPhysics;
598             ImGui::OpenPopup("DeletePhysics##1");
599         }

```

```

600
601     if (ImGui::BeginPopup("DeletePhysics##1")) {
602         if (ImGui::Selectable("Delete##4")) {
603             physics->GetParent() -> RemoveComponent<Physics>();
604             selected_component = -1;
605         }
606         ImGui::EndPopup();
607     }
608
609     if (physics_open) {
610         ImGui::Text("Velocity");
611
612         ImGui::PushItemWidth(65);
613         ImGui::SameLine(100); ImGui::InputFloat("x##1", &velocity.x);
614         ImGui::SameLine(185); ImGui::InputFloat("y##1", &velocity.y);
615         ImGui::SameLine(270); ImGui::InputFloat("z##1", &velocity.z);
616
617         ImGui::Text("RotVel");
618
619         ImGui::PushItemWidth(65);
620         ImGui::SameLine(100); ImGui::InputFloat("x##6", &rotVel.x);
621         ImGui::SameLine(185); ImGui::InputFloat("y##6", &rotVel.y);
622         ImGui::SameLine(270); ImGui::InputFloat("z##6", &rotVel.z);
623
624         ImGui::Text("Mass");
625         ImGui::SameLine(100); ImGui::InputFloat("##6", &physics->GetMassRef());
626         ImGui::PopItemWidth();
627
628         ImGui::TreePop();
629     }
630 }

```

References `Physics::GetMassRef()`, `Component::GetParent()`, `Physics::GetRotationalVelocityRef()`, `Physics::GetVelocityRef()`, `Object::RemoveComponent()`, and `selected_component`.

Referenced by `Display_Components()`.

#### 4.4.2.7 Display\_Scene() `void Editor::Display_Scene ( ) [private]`

Display the scene window.

Definition at line 190 of file `editor.cpp`.

```

190     {
191         ImGui::Begin("Scene");
192
193         if (!takeKeyboardInput && ImGui::IsWindowFocused()) {
194             if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_LEFT_CONTROL) == GLFW_PRESS) {
195                 // Copy current selected object
196                 if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_C) == GLFW_PRESS) {
197                     if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_C) == GLFW_RELEASE) {
198                         editor->object_to_copy = editor->selected_object;
199                     }
200                 }
201                 // Paste current selected object
202                 if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_V) == GLFW_PRESS) {
203                     if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_V) == GLFW_RELEASE) {
204                         if (editor->object_to_copy != -1) {
205                             Object* object = new Object(*Object_Manager::FindObject(editor->selected_object));
206                             Object_Manager::AddObject(object);
207                         }
208                     }
209                 }
210             }
211         }
212
213         // Display all objects
214         for (int i = 0; i < (int)Object_Manager::GetSize(); ++i) {
215             if (ImGui::Selectable(Object_Manager::FindObject(i)->GetName().c_str(), selected_object == i,
216                 ImGuiSelectableFlags_AllowDoubleClick)) {
217                 if (selected_object != i) editor->selected_component = -1;
218                 selected_object = i;
219             }
220         }
221     }

```

```

218         selected_component = -1;
219     }
220
221     // Checking for right click behavior
222     if (ImGui::IsItemClicked(ImGuiMouseButton_Right)) {
223         if (selected_object != i) editor->selected_component = -1;
224         selected_object = i;
225         selected_component = -1;
226         ImGui::OpenPopup("ObjectSettings##1");
227     }
228 }
229
230 if (ImGui::BeginPopup("ObjectSettings##1")) {
231     // Removes selected object from scene
232     if (ImGui::Selectable("Delete##1")) {
233         Object_Manager::RemoveObject(selected_object);
234         selected_object = -1;
235         selected_component = -1;
236     }
237     // Copies selected object
238     if (ImGui::Selectable("Copy##1")) {
239         editor->object_to_copy = editor->selected_object;
240     }
241     // Pastes copied object into scene
242     if (ImGui::Selectable("Paste##1")) {
243         if (editor->object_to_copy != -1) {
244             Object* object = new Object(*Object_Manager::FindObject(editor->selected_object));
245             Object_Manager::AddObject(object);
246         }
247     }
248     ImGui::EndPopup();
249 }
250
251 ImGui::Separator();
252
253 // Button to add new object to the scene
254 if (ImGui::Button("Add Object")) {
255     Object* newObject = new Object;
256     Transform* transform = new Transform;
257     transform->SetStartPosition(glm::vec3(0.f));
258     newObject->SetName("New_Object");
259     newObject->AddComponent(transform);
260
261     Object_Manager::AddObject(newObject);
262 }
263
264 ImGui::End();
265 }

```

References `Object::AddComponent()`, `Object_Manager::AddObject()`, `editor`, `Object_Manager::FindObject()`, `Object_Manager::GetSize()`, `Graphics::GetWindow()`, `object_to_copy`, `Object_Manager::RemoveObject()`, `selected_component`, `selected_object`, `Object::SetName()`, `Transform::SetStartPosition()`, and `takeKeyboardInput`.

Referenced by `Update()`.

**4.4.2.8 Display\_Scripts()** `void Editor::Display_Scripts ( Behavior * behavior ) [private]`

Displays the different lua scripts attached to the selected object.

#### Parameters

<i>behavior</i>	Contains the script data
-----------------	--------------------------

Definition at line 437 of file `editor.cpp`.

```

437                                     {
438     if (!behavior) return;
439
440     // Setting up tree flags
441     ImGuiTreeNodeFlags node_flags = ImGuiTreeNodeFlags_SpanAvailWidth | ImGuiTreeNodeFlags_OpenOnDoubleClick
| ImGuiTreeNodeFlags_OpenOnArrow;
442     if (selected_component == CType::CBehavior) node_flags |= ImGuiTreeNodeFlags_Selected;
443
444     const bool scripts_open = ImGui::TreeNodeEx((void*)(intptr_t)CType::CBehavior, node_flags, "Scripts");
445     if (ImGui::IsItemClicked()) selected_component = CType::CBehavior;
446
447     // Right click behavior to delete script component from object
448     if (ImGui::IsItemClicked(ImGuiMouseButton_Right)) {
449         selected_component = CType::CBehavior;
450         ImGui::OpenPopup("DeleteScripts##1");
451     }
452
453     if (ImGui::BeginPopup("DeleteScripts##1")) {
454         if (ImGui::Selectable("Delete##2")) {
455             behavior->GetParent()->RemoveComponent<Behavior>();
456             selected_component = -1;
457         }
458         ImGui::EndPopup();
459     }
460
461     // Displays the currently attached scripts
462     if (scripts_open) {
463         std::vector<std::string>& scripts = behavior->GetScripts();
464         unsigned scriptNum = 1;
465         for (std::string& script : scripts) {
466             ImGui::Text(std::string("Script " + std::to_string(scriptNum) + ":").c_str());
467             ImGui::SameLine(100);
468             if (ImGui::Button(script.c_str())) {
469                 ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDlgKey##3", "Choose File", ".lua",
std::string(getenv("USERPROFILE")) + "/Documents/pEngine/scripts/");
470             }
471
472             if (ImGuiFileDialog::Instance()->Display("ChooseFileDlgKey##3")) {
473                 if (ImGuiFileDialog::Instance()->IsOk()) {
474                     std::string filePathName = ImGuiFileDialog::Instance()->GetCurrentFileName();
475                     behavior->SwitchScript(scriptNum - 1, filePathName);
476                 }
477
478                 ImGuiFileDialog::Instance()->Close();
479             }
480             ++scriptNum;
481         }
482
483         // Add new script to the object
484         ImGui::Indent(71);
485         if (ImGui::Button("New Script##1")) {
486             ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDlgKey##4", "Choose File", ".lua",
std::string(getenv("USERPROFILE")) + "/Documents/pEngine/scripts/");
487         }
488
489         if (ImGuiFileDialog::Instance()->Display("ChooseFileDlgKey##4")) {
490             if (ImGuiFileDialog::Instance()->IsOk()) {
491                 std::string filePathName = ImGuiFileDialog::Instance()->GetCurrentFileName();
492                 behavior->AddScript(filePathName);
493             }
494
495             ImGuiFileDialog::Instance()->Close();
496         }
497
498         // Popup to say that the selected script to add is already attached to the object
499         if (ImGui::BeginPopup("ExistingScript##1")) {
500             ImGui::Text(std::string("Script already being used or doesn't exist").c_str(),
501                 ImGui::GetFontSize() * 2);
502             ImGui::EndPopup();
503         }
504
505         ImGui::TreePop();
506     }
507 }

```

References Behavior::AddScript(), Component::GetParent(), Behavior::GetScripts(), Object::RemoveComponent(), selected\_component, and Behavior::SwitchScript().

Referenced by Display\_Components().

#### 4.4.2.9 Display\_Transform() void Editor::Display\_Transform ( Transform \* transform ) [private]

Display transform data, users can change any of it.

##### Parameters

<i>transform</i>	
------------------	--

Definition at line 637 of file editor.cpp.

```

637                                     {
638     if (!transform) return;
639
640     glm::vec3& position = transform->GetPositionRef();
641     glm::vec3& scale = transform->GetScaleRef();
642     glm::vec3& rotation = transform->GetRotationRef();
643     glm::vec3& startPos = transform->GetStartPositionRef();
644
645     ImGuiTreeNodeFlags node_flags = ImGuiTreeNodeFlags_SpanAvailWidth | ImGuiTreeNodeFlags_OpenOnDoubleClick
| ImGuiTreeNodeFlags_OpenOnArrow;
646     if (selected_component == CType::CTransform) node_flags |= ImGuiTreeNodeFlags_Selected;
647
648     const bool transform_open = ImGui::TreeNodeEx((void*)(intptr_t)CType::CTransform, node_flags,
"Transform");
649     if (ImGui::IsItemClicked()) selected_component = CType::CTransform;
650
651     if (transform_open) {
652         ImGui::Text("Position");
653
654         ImGui::PushItemWidth(65);
655         ImGui::SameLine(100); ImGui::InputFloat("x##1", &position.x);
656         ImGui::SameLine(185); ImGui::InputFloat("y##1", &position.y);
657         ImGui::SameLine(270); ImGui::InputFloat("z##1", &position.z);
658         ImGui::PopItemWidth();
659
660         ImGui::Text("Scale");
661
662         ImGui::PushItemWidth(65);
663         ImGui::SameLine(100); ImGui::InputFloat("x##2", &scale.x);
664         ImGui::SameLine(185); ImGui::InputFloat("y##2", &scale.y);
665         ImGui::SameLine(270); ImGui::InputFloat("z##2", &scale.z);
666         ImGui::PopItemWidth();
667
668         ImGui::Text("Rotation");
669
670         ImGui::PushItemWidth(65);
671         ImGui::SameLine(100); ImGui::InputFloat("x##3", &rotation.x);
672         ImGui::SameLine(185); ImGui::InputFloat("y##3", &rotation.y);
673         ImGui::SameLine(270); ImGui::InputFloat("z##3", &rotation.z);
674         ImGui::PopItemWidth();
675
676         ImGui::Text("Start Pos");
677
678         ImGui::PushItemWidth(65);
679         ImGui::SameLine(100); ImGui::InputFloat("x##5", &startPos.x);
680         ImGui::SameLine(185); ImGui::InputFloat("y##5", &startPos.y);
681         ImGui::SameLine(270); ImGui::InputFloat("z##5", &startPos.z);
682         ImGui::PopItemWidth();
683
684         ImGui::TreePop();
685     }
686 }
```

References Transform::GetPositionRef(), Transform::GetRotationRef(), Transform::GetScaleRef(), Transform::GetStartPositionRef(), and selected\_component.

Referenced by Display\_Components().



#### 4.4.2.10 Display\_World\_Settings() void Editor::Display\_World\_Settings ( ) [private]

Shows all of the settings of the engine itself.

Definition at line 364 of file editor.cpp.

```

364 {
365     ImGui::Begin("World Settings");
366     std::string presetName = Engine::GetPresetName();
367
368     // Allows user to change the preset that is loaded
369     ImGui::Text("Presets"); ImGui::SameLine(120);
370     if (ImGui::Button(presetName.c_str())) {
371         ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDialogKey##3", "Choose File", ".json",
372         std::string(getenv("USERPROFILE")) + "/Documents/pEngine/json/preset/");
373     }
374     if (ImGuiFileDialog::Instance()->Display("ChooseFileDialogKey##3")) {
375         if (ImGuiFileDialog::Instance()->IsOk()) {
376             std::string filePathName = ImGuiFileDialog::Instance()->GetCurrentFileName();
377             if (Engine::Restart(filePathName)) selected_object = -1;
378         }
379         ImGuiFileDialog::Instance()->Close();
380     }
381
382     ImGui::PushItemWidth(141);
383
384     // Strength of the light being used
385     ImGui::Text("Light Power");
386     ImGui::SameLine(120); ImGui::InputFloat("##1", &Engine::GetLightPower());
387
388     // Position of the light being used
389     ImGui::Text("Light Position");
390     ImGui::PushItemWidth(65);
391     ImGui::SameLine(120); ImGui::InputFloat("x##4", &Engine::GetLightPos().x);
392     ImGui::SameLine(205); ImGui::InputFloat("y##4", &Engine::GetLightPos().y);
393     ImGui::SameLine(290); ImGui::InputFloat("z##4", &Engine::GetLightPos().z);
394     ImGui::PopItemWidth();
395
396     // Grav const of the engine
397     ImGui::Text("Grav Const");
398     ImGui::SameLine(120); ImGui::InputDouble("##5", &Engine::GetGravConst());
399
400     ImGui::PopItemWidth();
401
402     ImGui::End();
403 }
404 }
```

References Engine::GetGravConst(), Engine::GetLightPos(), Engine::GetLightPower(), Engine::GetPresetName(), Engine::Restart(), and selected\_object.

Referenced by Update().

#### 4.4.2.11 GetTakeKeyboardInput() bool Editor::GetTakeKeyboardInput ( ) [static]

Returns whether the program should ignore keyboard input.

Returns

true  
false

Definition at line 731 of file editor.cpp.

```
731 { return editor->takeKeyboardInput; }
```

References editor, and takeKeyboardInput.

Referenced by Camera::Update(), and Graphics::Update().

**4.4.2.12 Initialize()** `bool Editor::Initialize ( ) [static]`

Sets up the config and style of the editor.

Returns

true  
false

Definition at line 35 of file editor.cpp.

```

35         {
36             // Initializing the editor
37             editor = new Editor;
38             if (!editor) {
39                 Trace::Message("Editor failed to initialize.\n");
40                 return false;
41             }
42             editor->selected_object = -1;
43             editor->selected_component = -1;
44             editor->object_to_copy = -1;
45
46             ImGui::CHECKVERSION();
47             ImGui::CreateContext();
48
49             // Setting up ImGui flags
50             ImGui::GetIO().ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard;
51             ImGui::GetIO().ConfigFlags |= ImGuiConfigFlags_DockingEnable;
52             ImGui::GetIO().ConfigFlags |= ImGuiConfigFlags_ViewportsEnable;
53
54             // Setting style for ImGui
55             ImGui::StyleColorsDark();
56             if (ImGui::GetIO().ConfigFlags & ImGuiConfigFlags_ViewportsEnable) {
57                 ImGui::GetStyle().WindowRounding = 0.f;
58                 ImGui::GetStyle().Colors[ImGuiCol_WindowBg].w = 1.f;
59             }
60
61             // Setting up ImGui
62             ImGui_ImplGlfw_InitForOpenGL(Graphics::GetWindow(), true);
63             ImGui_ImplOpenGL3_Init("#version 330");
64
65             return true;
66     }
```

References editor, Graphics::GetWindow(), Trace::Message(), object\_to\_copy, selected\_component, and selected\_↵ object.

Referenced by Engine::Initialize().

**4.4.2.13 Render()** `void Editor::Render ( ) [static]`

Render the editor.

Returns

void

Definition at line 114 of file editor.cpp.

```

114         {
115             ImGui::Render();
116             ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
117
118             if (ImGui::GetIO().ConfigFlags & ImGuiConfigFlags_ViewportsEnable) {
119                 GLFWwindow* backup_current_context = glfwGetCurrentContext();
120                 ImGui::UpdatePlatformWindows();
121                 ImGui::RenderPlatformWindowsDefault();
122                 glfwMakeContextCurrent(backup_current_context);
123             }
124     }
```

Referenced by Graphics::Render().

**4.4.2.14 Reset()** `void Editor::Reset ( ) [static]`

Sets selected object to invalid value.

**Returns**

void

Definition at line 147 of file editor.cpp.

```
147     {  
148         editor->selected_object = -1;  
149     }
```

References editor, and selected\_object.

Referenced by Engine::Restart().

**4.4.2.15 Shutdown()** `void Editor::Shutdown ( ) [static]`

Destroy editor windows and systems.

**Returns**

void

Definition at line 131 of file editor.cpp.

```
131     {  
132         if (!editor) return;  
133  
134         ImGui_ImplOpenGL3_Shutdown();  
135         ImGui_ImplGlfw_Shutdown();  
136         ImGui::DestroyContext();  
137  
138         delete editor;  
139         editor = nullptr;  
140     }
```

References editor.

Referenced by Engine::Shutdown().

**4.4.2.16 Update()** void Editor::Update ( ) [static]

Updates the editor content and calls display functions.

**Returns**

void

Definition at line 73 of file editor.cpp.

```

73     {
74         // ImGui update functions
75         ImGui_ImplOpenGL3_NewFrame();
76         ImGui_ImplGlfw_NewFrame();
77         ImGui::NewFrame();
78
79         //ImGui::ShowDemoWindow();
80
81         // Updating whether program should ignore keyboard input
82         if (!ImGui::GetIO().WantCaptureKeyboard) {
83             editor->takeKeyboardInput = true;
84         }
85         else {
86             editor->takeKeyboardInput = false;
87         }
88
89         // Keyboard shortcuts
90         if (!editor->takeKeyboardInput) {
91             // Save current settings as preset
92             if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_LEFT_CONTROL) == GLFW_PRESS) {
93                 if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_S) == GLFW_PRESS) {
94                     if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_S) == GLFW_RELEASE) {
95                         Engine::Write();
96                     }
97                 }
98             }
99         }
100
101         // Display the different windows
102         editor->Display_Dockspace();
103         editor->Display_Scene();
104         editor->Display_Components();
105         editor->Display_World_Settings();
106         editor->Display_Camera_Settings();
107     }

```

References Display\_Camera\_Settings(), Display\_Components(), Display\_Dockspace(), Display\_Scene(), Display\_World\_Settings(), editor, Graphics::GetWindow(), takeKeyboardInput, and Engine::Write().

Referenced by Engine::Update().

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

- [editor.hpp](#)
- [editor.cpp](#)

**4.5 Engine Class Reference**

```
#include <engine.hpp>
```

## Static Public Member Functions

- static bool `Initialize ()`  
*Initializes the engine and the systems in the engine.*
- static void `Update ()`  
*Updates object and camera. `Object` updates have a fixed time step, camera updates have variable time step.*
- static void `Shutdown ()`  
*Shutdown systems and then engine.*
- static bool `Restart ()`  
*Resets the objects in the engine.*
- static bool `Restart (std::string presetName)`  
*Resets the engine to the given preset.*
- static float `GetDeltaTime ()`  
*Returns delta time (variable)*
- static float `GetDt ()`  
*Returns delta time (fixed)*
- static double & `GetGravConst ()`  
*Returns gravitational constant.*
- static std::string `GetPresetName ()`  
*Returns the name of the current preset.*
- static float & `GetLightPower ()`  
*Returns reference to power of the light in the scene.*
- static glm::vec3 & `GetLightPos ()`  
*Returns reference to the position of the light in the scene.*
- static void `Write ()`  
*Writes the engine data to a preset file (creates new one if it doesn't already exist)*
- static void `SetPresetName (std::string presetName_)`  
*Sets the name of the preset file.*

## Private Attributes

- bool `isRunning`  
*state of the main loop*
- float `deltaTime`  
*time between frames*
- float `accumulator`  
*amount of unused time for physics updates*
- float `time`  
*total time*
- const float `dt = 0.01f`  
*fixed delta time for physics updates*
- std::chrono::steady\_clock::time\_point `currentTime`  
*current read time*
- std::chrono::steady\_clock::time\_point `newTime`  
*newest read time*
- std::chrono::steady\_clock::duration `timeTaken`  
*time between frames*

- double [gravConst](#)  
*gravitational constant (used in physics)*
- std::string [presetName](#)  
*name of the preset being used*
- float [lightPower](#)  
*Power of the light in the scene.*
- glm::vec3 [lightPos](#)  
*Position of the light in the scene.*

### 4.5.1 Detailed Description

[Engine](#) class

Definition at line 24 of file engine.hpp.

### 4.5.2 Member Function Documentation

#### 4.5.2.1 GetDeltaTime() `float Engine::GetDeltaTime ( ) [static]`

Returns delta time (variable)

Returns

float Variable delta time

Definition at line 189 of file engine.cpp.

```
189 { return engine->deltaTime; }
```

References `deltaTime`, and `engine`.

Referenced by `Camera::MouseUpdate()`, and `Camera::Update()`.

#### 4.5.2.2 GetDt() `float Engine::GetDt ( ) [static]`

Returns delta time (fixed)

Returns

float Fixed delta time

Definition at line 196 of file engine.cpp.

```
196 { return engine->dt; }
```

References `dt`, and `engine`.

Referenced by `Behavior::Update()`, and `Physics::Update()`.

#### 4.5.2.3 GetGravConst() `double & Engine::GetGravConst ( ) [static]`

Returns gravitational constant.

##### Returns

double Gravitational constant

Definition at line 203 of file engine.cpp.

```
203 { return engine->gravConst; }
```

References engine, and gravConst.

Referenced by Editor::Display\_World\_Settings(), and Physics::UpdateGravity().

#### 4.5.2.4 GetLightPos() `glm::vec3 & Engine::GetLightPos ( ) [static]`

Returns reference to the position of the light in the scene.

##### Returns

glm::vec3&

Definition at line 224 of file engine.cpp.

```
224 { return engine->lightPos; }
```

References engine, and lightPos.

Referenced by Editor::Display\_World\_Settings(), and Model\_Data::Draw().

#### 4.5.2.5 GetLightPower() `float & Engine::GetLightPower ( ) [static]`

Returns reference to power of the light in the scene.

##### Returns

float&

Definition at line 217 of file engine.cpp.

```
217 { return engine->lightPower; }
```

References engine, and lightPower.

Referenced by Editor::Display\_World\_Settings(), and Model\_Data::Draw().

#### 4.5.2.6 GetPresetName() `std::string Engine::GetPresetName ( ) [static]`

Returns the name of the current preset.

##### Returns

`std::string`

Definition at line 210 of file engine.cpp.

```
210 { return engine->presetName; }
```

References engine, and presetName.

Referenced by Editor::Display\_Menu\_Bar(), and Editor::Display\_World\_Settings().

#### 4.5.2.7 Initialize() `bool Engine::Initialize ( ) [static]`

Initializes the engine and the systems in the engine.

##### Returns

`true`

`false`

Definition at line 42 of file engine.cpp.

```
42 {
43     // Initializing engine
44     engine = new Engine;
45     if (!engine) {
46         Trace::Message("Engine was not initialized.\n");
47         return false;
48     }
49
50     // Reading settings from json
51     File_Reader settings;
52     if (!settings.Read_File("settings.json")) return false;
53
54     File_Reader preset;
55     if (!preset.Read_File("preset/" + settings.Read_String("preset"))) return false;
56
57     engine->presetName = settings.Read_String("preset");
58     engine->gravConst = preset.Read_Double("gravConst");
59
60     engine->lightPower = 1000.f;
61     engine->lightPos = preset.Read_Vec3("lightPos");
62     if (engine->lightPos == glm::vec3(0.f)) {
63         engine->lightPos = glm::vec3(4, 4, 0);
64     }
65
66     // Initializing sub systems
67     if (!Model_Data_Manager::Initialize()) return false;
68     if (!Texture_Manager::Initialize()) return false;
69     if (!Camera::Initialize(settings)) return false;
70     if (!Graphics::Initialize(settings)) return false;
71     if (!Object_Manager::Initialize(preset)) return false;
72     if (!Random::Initialize()) return false;
73     if (!Editor::Initialize()) return false;
74
75     // Setting up variables used for dt
76     engine->currentTime = std::chrono::steady_clock::now();
77     engine->accumulator = 0.f;
78     engine->time = 0.f;
```



```
79     engine->isRunning = true;
80
81     return true;
82 }
```

References accumulator, currentTime, engine, gravConst, Random::Initialize(), Editor::Initialize(), Model\_Data\_↵ Manager::Initialize(), Texture\_Manager::Initialize(), Object\_Manager::Initialize(), Camera::Initialize(), Graphics::↵ Initialize(), isRunning, lightPos, lightPower, Trace::Message(), presetName, File\_Reader::Read\_Double(), File\_↵ Reader::Read\_File(), File\_Reader::Read\_String(), File\_Reader::Read\_Vec3(), and time.

Referenced by main().

#### 4.5.2.8 Restart() [1/2] bool Engine::Restart ( ) [static]

Resets the objects in the engine.

##### Returns

true  
false

Definition at line 139 of file engine.cpp.

```
139     {
140         // Initializing object manager
141         File_Reader settings;
142         if (!settings.Read_File("settings.json")) return false;
143
144         File_Reader preset;
145         if (!preset.Read_File("preset/" + engine->presetName)) return false;
146
147         // Removing all current objects
148         Object_Manager::Shutdown();
149         Editor::Reset();
150
151         engine->presetName = settings.Read_String("preset");
152         engine->gravConst = preset.Read_Double("gravConst");
153         if (!Object_Manager::Initialize(preset)) return false;
154
155         return true;
156 }
```

References engine, gravConst, Object\_Manager::Initialize(), presetName, File\_Reader::Read\_Double(), File\_Reader\_↵ ::Read\_File(), File\_Reader::Read\_String(), Editor::Reset(), and Object\_Manager::Shutdown().

Referenced by Editor::Display\_World\_Settings(), and Graphics::Update().

#### 4.5.2.9 Restart() [2/2] bool Engine::Restart ( std::string presetName ) [static]

Resets the engine to the given preset.

## Parameters

<i>presetName</i>	Given preset
-------------------	--------------

## Returns

true  
false

Definition at line 165 of file engine.cpp.

```

165                                     {
166     // Initializing object manager
167     File_Reader settings;
168     if (!settings.Read_File("settings.json")) return false;
169
170     File_Reader preset;
171     if (!preset.Read_File("preset/" + presetName)) return false;
172
173     // Removing all current objects
174     Object_Manager::Shutdown();
175     Editor::Reset();
176
177     engine->presetName = presetName;
178     engine->gravConst = preset.Read_Double("gravConst");
179     if (!Object_Manager::Initialize(preset)) return false;
180
181     return true;
182 }
```

References engine, gravConst, Object\_Manager::Initialize(), presetName, File\_Reader::Read\_Double(), File\_Reader::Read\_File(), Editor::Reset(), and Object\_Manager::Shutdown().

**4.5.2.10 SetPresetName()** void Engine::SetPresetName (std::string presetName\_ ) [static]

Sets the name of the preset file.

## Parameters

<i>presetName_</i>	
--------------------	--

## Returns

void

Definition at line 248 of file engine.cpp.

```

248                                     {
249     engine->presetName = presetName_;
250 }
```

References engine, and presetName.

Referenced by Editor::Display\_Menu\_Bar().

#### 4.5.2.11 Shutdown() void Engine::Shutdown ( ) [static]

Shutdown systems and then engine.

##### Returns

void

Definition at line 116 of file engine.cpp.

```

116         {
117     if (!engine) return;
118
119     // Shutdown sub systems
120     Editor::Shutdown();
121     Random::Shutdown();
122     Object_Manager::Shutdown();
123     Graphics::Shutdown();
124     Camera::Shutdown();
125     Texture_Manager::Shutdown();
126     Model_Data_Manager::Shutdown();
127
128     // Delete engine object
129     delete engine;
130     engine = nullptr;
131 }
```

References engine, Random::Shutdown(), Editor::Shutdown(), Model\_Data\_Manager::Shutdown(), Texture\_Manager::Shutdown(), Camera::Shutdown(), Object\_Manager::Shutdown(), and Graphics::Shutdown().

Referenced by main().

#### 4.5.2.12 Update() void Engine::Update ( ) [static]

Updates object and camera. **Object** updates have a fixed time step, camera updates have variable time step.

##### Returns

void

Definition at line 90 of file engine.cpp.

```

90         {
91     // Calculating dt
92     engine->newTime = std::chrono::steady_clock::now();
93     engine->timeTaken = engine->newTime - engine->currentTime;
94     engine->deltaTime = float(engine->timeTaken.count()) *
95         std::chrono::steady_clock::period::num / std::chrono::steady_clock::period::den;
96     engine->currentTime = engine->newTime;
97     engine->accumulator += engine->deltaTime;
98
99     Editor::Update();
100    Camera::Update();
101    // Only called when it is time (fixed time step)
102    while (engine->accumulator >= engine->dt) {
103        // Update objects
104        Object_Manager::Update();
105        // Update dt related variables
106        engine->accumulator -= engine->dt;
107        engine->time += engine->dt;
108    }
109 }
```

References accumulator, currentTime, deltaTime, dt, engine, newTime, time, timeTaken, Editor::Update(), Camera::Update(), and Object\_Manager::Update().

Referenced by Graphics::Update().

**4.5.2.13 Write()** void Engine::Write ( ) [static]

Writes the engine data to a preset file (creates new one if it doesn't already exist)

Returns

void

Definition at line 232 of file engine.cpp.

```

232     {
233         File_Writer writer;
234
235         writer.Write_Value("gravConst", engine->gravConst);
236         writer.Write_Vec3("lightPos", engine->lightPos);
237         Object_Manager::Write(writer);
238
239         writer.Write_File(std::string ("preset/" + engine->presetName));
240     }
```

References engine, gravConst, lightPos, presetName, Object\_Manager::Write(), File\_Writer::Write\_File(), File\_Writer::Write\_Value(), and File\_Writer::Write\_Vec3().

Referenced by Editor::Display\_Menu\_Bar(), and Editor::Update().

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

- [engine.hpp](#)
- [engine.cpp](#)

**4.6 File\_Reader Class Reference**

```
#include <file_reader.hpp>
```

**Public Member Functions**

- bool [Read\\_File](#) (std::string filename)  
*Reads the json file data into the root variable.*
- int [Read\\_Int](#) (std::string valueName)  
*Reads int from the json file stored in root.*
- std::string [Read\\_String](#) (std::string valueName)  
*Reads std::string from the json file stored in root.*
- glm::vec3 [Read\\_Vec3](#) (std::string valueName)  
*Reads glm::vec3 from the json file stored in root. glm::vec3 is constructed from an array.*
- bool [Read\\_Bool](#) (std::string valueName)  
*Reads bool from the json file stored in root.*
- float [Read\\_Float](#) (std::string valueName)  
*Reads float from the json stored in root.*
- double [Read\\_Double](#) (std::string valueName)  
*Reads double from the json stored in root.*
- std::string [Read\\_Object\\_Name](#) (std::string valueName)  
*Reads the name of an object from an object list (preset folder)*
- std::string [Read\\_Object\\_Template\\_Name](#) (std::string valueName)  
*Reads the name of the template file for object.*
- glm::vec3 [Read\\_Object\\_Position](#) (std::string valueName)  
*Reads the position of an object from an object list (preset folder)*
- glm::vec3 [Read\\_Object\\_Scale](#) (std::string valueName)  
*Reads the scale of an object.*
- std::string [Read\\_Behavior\\_Name](#) (std::string valueName)  
*Reads the name of the behavior.*

## Private Attributes

- rapidjson::Document [root](#)  
*Holds the data of the json file.*

### 4.6.1 Detailed Description

[File\\_Reader](#) class

Definition at line 24 of file file\_reader.hpp.

### 4.6.2 Member Function Documentation

#### 4.6.2.1 Read\_Behavior\_Name() `std::string File_Reader::Read_Behavior_Name (std::string valueName )`

Reads the name of the behavior.

##### Parameters

<i>valueName</i>	<a href="#">Behavior</a> to read
------------------	----------------------------------

##### Returns

`std::string` Name of the behavior

Definition at line 205 of file file\_reader.cpp.

```
205                                     {
206     // Checking if value exists
207     if (!root["behaviors"].HasMember(valueName.c_str())) {
208         return std::string("");
209     }
210
211     return root["behaviors"][valueName.c_str()].GetString();
212 }
```

Referenced by [Behavior::Read\(\)](#).

#### 4.6.2.2 Read\_Bool() `bool File_Reader::Read_Bool (std::string valueName )`

Reads bool from the json file stored in root.

## Parameters

<i>valueName</i>	Name of the bool in the json file
------------------	-----------------------------------

## Returns

true  
false

Definition at line 96 of file file\_reader.cpp.

```
96                                     {  
97     // Checking if the value is a bool  
98     if (!root.HasMember(valueName.c_str())) {  
99         return false;  
100    }  
101    return root[valueName.c_str()].GetBool();  
102 }
```

#### 4.6.2.3 Read\_Double() double File\_Reader::Read\_Double ( std::string valueName )

Reads double from the json stored in root.

## Parameters

<i>valueName</i>	Name of the double in the json file
------------------	-------------------------------------

## Returns

double Value that was read

Definition at line 124 of file file\_reader.cpp.

```
124                                     {  
125     // Checking if the value is a double (has decimal)  
126     if (!root.HasMember(valueName.c_str())) {  
127         return false;  
128    }  
129    return root[valueName.c_str()].GetDouble();  
130 }
```

Referenced by Engine::Initialize(), and Engine::Restart().

#### 4.6.2.4 Read\_File() bool File\_Reader::Read\_File ( std::string filename )

Reads the json file data into the root variable.

## Parameters

<i>filename</i>	Name of the file to be read
-----------------	-----------------------------

## Returns

true  
false

Definition at line 32 of file file\_reader.cpp.

```
32                                     {
33     // Opening the json file
34     std::string fileToOpen = std::string(getenv("USERPROFILE")) + "/Documents/pEngine/json/" + filename;
35     FILE* file = fopen(fileToOpen.c_str(), "r");
36     if (!file) return false;
37
38     char buffer[65536];
39     FileReadStream stream(file, buffer, sizeof(buffer));
40     root.ParseStream<0, UTF8<>, FileReadStream>(stream);
41
42     fclose(file);
43     return true;
44 }
```

Referenced by Engine::Initialize(), Object::Read(), Object::ReRead(), and Engine::Restart().

**4.6.2.5 Read\_Float()** float File\_Reader::Read\_Float (   
std::string valueName )

Reads float from the json stored in root.

## Parameters

<i>valueName</i>	Name of the float in the json file
------------------	------------------------------------

## Returns

float Value that was read

Definition at line 110 of file file\_reader.cpp.

```
110                                     {
111     // Checking if the value is a double (has decimal)
112     if (!root.HasMember(valueName.c_str())) {
113         return 0.f;
114     }
115     return root[valueName.c_str()].GetFloat();
116 }
```

Referenced by Camera::Initialize(), and Physics::Read().

**4.6.2.6 Read\_Int()** `int File_Reader::Read_Int (`  
`std::string valueName )`

Reads int from the json file stored in root.



**Parameters**

<i>valueName</i>	Name of the int in the json file
------------------	----------------------------------

**Returns**

int Value that was read

Definition at line 52 of file file\_reader.cpp.

```
52                                     {
53     // Checking if the value is an int
54     if (!root.HasMember(valueName.c_str())) {
55         return 0;
56     }
57     return root[valueName.c_str()].GetInt();
58 }
```

Referenced by Camera::Initialize(), and Graphics::Initialize().

**4.6.2.7 Read\_Object\_Name()** std::string File\_Reader::Read\_Object\_Name (std::string valueName )

Reads the name of an object from an object list (preset folder)

**Parameters**

<i>valueName</i>	Specifies which object
------------------	------------------------

**Returns**

std::string Name of the object

Definition at line 138 of file file\_reader.cpp.

```
138                                     {
139     // Checking if the value exists
140     if (!root.HasMember(valueName.c_str())) {
141         return std::string("");
142     }
143     if (!root[valueName.c_str()].HasMember("objectName")) {
144         return std::string("");
145     }
146
147     return root[valueName.c_str()]["objectName"].GetString();
148 }
```

Referenced by Object\_Manager::ReadList().

**4.6.2.8 Read\_Object\_Position()** glm::vec3 File\_Reader::Read\_Object\_Position (std::string valueName )

Reads the position of an object from an object list (preset folder)

## Parameters

<i>valueName</i>	Specifies which object
------------------	------------------------

## Returns

glm::vec3 Position of object

Definition at line 174 of file file\_reader.cpp.

```

174                                     {
175     if (!root[valueName.c_str()].HasMember("position")) {
176         return glm::vec3(0.f, 0.f, 0.f);
177     }
178
179     Value& array = root[valueName.c_str()]["position"];
180     return glm::vec3(array[0].GetFloat(), array[1].GetFloat(), array[2].GetFloat());
181 }
```

Referenced by Object\_Manager::ReadList().

**4.6.2.9 Read\_Object\_Scale()** glm::vec3 File\_Reader::Read\_Object\_Scale (   
std::string valueName )

Reads the scale of an object.

## Parameters

<i>valueName</i>	
------------------	--

## Returns

glm::vec3

Definition at line 189 of file file\_reader.cpp.

```

189                                     {
190     // Checking if value exists
191     if (!root[valueName.c_str()].HasMember("scale")) {
192         return glm::vec3(0.f, 0.f, 0.f);
193     }
194
195     Value& array = root[valueName.c_str()]["scale"];
196     return glm::vec3(array[0].GetFloat(), array[1].GetFloat(), array[2].GetFloat());
197 }
```

Referenced by Object\_Manager::ReadList().

**4.6.2.10 Read\_Object\_Template\_Name()** std::string File\_Reader::Read\_Object\_Template\_Name (   
std::string valueName )

Reads the name of the template file for object.

**Parameters**

<i>valueName</i>	
------------------	--

**Returns**

std::string

Definition at line 156 of file file\_reader.cpp.

```
156                                     {
157     // Checking if the value exists
158     if (!root.HasMember(valueName.c_str())) {
159         return std::string("");
160     }
161     if (!root[valueName.c_str()].HasMember("templateName")) {
162         return std::string("");
163     }
164
165     return root[valueName.c_str()]["templateName"].GetString();
166 }
```

Referenced by Object\_Manager::ReadList().

**4.6.2.11 Read\_String()** std::string File\_Reader::Read\_String (  
std::string valueName )

Reads std::string from the json file stored in root.

**Parameters**

<i>valueName</i>	Name of the std::string in the json file
------------------	--

**Returns**

std::string Value that was read

Definition at line 66 of file file\_reader.cpp.

```
66                                     {
67     // Checking if the value is a std::string
68     if (!root.HasMember(valueName.c_str())) {
69         return std::string("");
70     }
71     return root[valueName.c_str()].GetString();
72 }
```

Referenced by Model\_Data\_Manager::Get(), Texture\_Manager::Get(), Engine::Initialize(), Shader::Initialize(), Model\_↵Data::Load(), Object::ReRead(), and Engine::Restart().

**4.6.2.12 Read\_Vec3()** glm::vec3 File\_Reader::Read\_Vec3 (  
std::string valueName )

Reads glm::vec3 from the json file stored in root. glm::vec3 is constructed from an array.

## Parameters

<i>valueName</i>	Name of the glm::vec3 in the json file
------------------	--

## Returns

glm::vec3 Value that was read

Definition at line 81 of file file\_reader.cpp.

```

81         {
82             // Checking if the value is an array
83             if (!root.HasMember(valueName.c_str())) {
84                 return glm::vec3(0.f, 0.f, 0.f);
85             }
86             return glm::vec3(root[valueName.c_str()][0].GetFloat(), root[valueName.c_str()][1].GetFloat(),
87                             root[valueName.c_str()][2].GetFloat());
87     }
```

Referenced by Engine::Initialize(), and Physics::Read().

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

- [file\\_reader.hpp](#)
- [file\\_reader.cpp](#)

## 4.7 File\_Writer Class Reference

```
#include <file_writer.hpp>
```

### Public Member Functions

- [File\\_Writer](#) ()  
*Creates root object to write data into.*
- void [Write\\_File](#) (std::string filename)  
*Writes all the data stored in root to the given filename.*
- void [Write\\_Vec3](#) (std::string valueName, glm::vec3 value)  
*Write a glm::vec3 into root.*
- void [Write\\_String](#) (std::string valueName, std::string value)  
*Write a std::string into root.*
- template<typename T >  
void [Write\\_Value](#) (std::string valueName, T value)  
*Writes most values to root (can't do strings)*
- void [Write\\_Behavior\\_Name](#) (std::vector< std::string > &behaviorNames)  
*Writing behaviorNames into nested object and then into root.*
- void [Write\\_Object\\_Data](#) (Object \*object)  
*Writing data of an object into root.*

## Private Attributes

- rapidjson::Document [root](#)  
*Holds the data for the json file.*

### 4.7.1 Detailed Description

[File\\_Writer](#) class

Definition at line 30 of file file\_writer.hpp.

### 4.7.2 Constructor & Destructor Documentation

#### 4.7.2.1 File\_Writer() File\_Writer::File\_Writer ( )

Creates root object to write data into.

Definition at line 27 of file file\_writer.cpp.

```
27     {  
28         root.SetObject();  
29     }
```

### 4.7.3 Member Function Documentation

#### 4.7.3.1 Write\_Behavior\_Name() void File\_Writer::Write\_Behavior\_Name ( std::vector< std::string > & behaviorNames )

Writing behaviorNames into nested object and then into root.

##### Parameters

<i>behaviorNames</i>	
----------------------	--

Definition at line 88 of file file\_writer.cpp.

```
88     {  
89         Value behaviors(kObjectType);  
90  
91         // Filling object  
92         for (unsigned i = 0; i < behaviorNames.size(); ++i) {  
93             std::string behaviorName = std::string("behavior_" + std::to_string(i));  
94             Value name(behaviorName.c_str(), SizeType(behaviorName.size()), root.GetAllocator());  
95  
96             behaviors.AddMember(name, StringRef(behaviorNames[i].c_str()), root.GetAllocator());  
97         }
```

```

98
99     // Nesting object into root
100     root.AddMember("behaviors", behaviors, root.GetAllocator());
101 }

```

Referenced by Behavior::Write().

**4.7.3.2 Write\_File()** void File\_Writer::Write\_File (   
 std::string filename )

Writes all the data stored in root to the given filename.

Parameters

<i>filename</i>	
-----------------	--

Definition at line 36 of file file\_writer.cpp.

```

36                                     {
37     std::string fileToOpen = std::string(getenv("USERPROFILE")) + "/Documents/pEngine/json/" + filename;
38     FILE* file = fopen(fileToOpen.c_str(), "w");
39
40     char buffer[65536];
41     FileWriteStream stream(file, buffer, sizeof(buffer));
42
43     PrettyWriter<FileWriteStream> writer(stream);
44     writer.SetMaxDecimalPlaces(3);
45     writer.SetFormatOptions(kFormatSingleLineArray);
46     root.Accept(writer);
47
48     fclose(file);
49 }

```

Referenced by Engine::Write(), and Object::Write().

**4.7.3.3 Write\_Object\_Data()** void File\_Writer::Write\_Object\_Data (   
 Object \* object )

Writing data of an object into root.

Parameters

<i>object</i>	
---------------	--

Definition at line 108 of file file\_writer.cpp.

```

108                                     {
109     if (!object) return;
110
111     // Getting transform data from object
112     Transform* transform = object->GetComponent<Transform>();
113     glm::vec3 startPos = { 0.f, 0.f, 0.f };
114     glm::vec3 startScale = { 1.f, 1.f, 1.f };
115     if (transform) startPos = transform->GetStartPosition();
116     if (transform) startScale = transform->GetScale();

```

```

117
118 // Putting position into value rapidjson can use
119 Value pos(kArrayType);
120 pos.PushBack(startPos.x, root.GetAllocator());
121 pos.PushBack(startPos.y, root.GetAllocator());
122 pos.PushBack(startPos.z, root.GetAllocator());
123
124 // Putting scale into value rapidjson can use
125 Value scale(kArrayType);
126 scale.PushBack(startScale.x, root.GetAllocator());
127 scale.PushBack(startScale.y, root.GetAllocator());
128 scale.PushBack(startScale.z, root.GetAllocator());
129
130 // Creating and filling object
131 Value objectData(kObjectType);
132
133 Value objectName(object->GetName().c_str(), SizeType(object->GetName().size()), root.GetAllocator());
134 objectData.AddMember(StringRef("objectName"), objectName, root.GetAllocator());
135 Value templateName(object->GetTemplateName().c_str(), SizeType(object->GetTemplateName().size()),
136 root.GetAllocator());
137 objectData.AddMember(StringRef("templateName"), templateName, root.GetAllocator());
138 objectData.AddMember(StringRef("position"), pos, root.GetAllocator());
139 objectData.AddMember(StringRef("scale"), scale, root.GetAllocator());
140
141 // Nesting object into root
142 std::string objectIdName = "object_" + std::to_string(object->GetId());
143 Value name(objectIdName.c_str(), SizeType(objectIdName.size()), root.GetAllocator());
144 root.AddMember(name, objectData, root.GetAllocator());
145 }

```

References `Object::GetId()`, `Object::GetName()`, `Transform::GetScale()`, `Transform::GetStartPosition()`, and `Object::GetTemplateName()`.

Referenced by `Object_Manager::Write()`.

**4.7.3.4 Write\_String()** `void File_Writer::Write_String (`  
`std::string valueName,`  
`std::string value )`

Write a `std::string` into root.

#### Parameters

<i>valueName</i>	
<i>value</i>	

Definition at line 75 of file `file_writer.cpp`.

```

75
76 // Storing std::string in variable rapidjson can write
77 Value name(valueName.c_str(), SizeType(valueName.size()), root.GetAllocator());
78 Value newValue(value.c_str(), SizeType(value.size()), root.GetAllocator());
79
80 root.AddMember(name, newValue, root.GetAllocator());
81 }

```

Referenced by `Model::Write()`, and `Object::Write()`.

**4.7.3.5 Write\_Value()** `template<typename T >`

```
void File_Writer::Write_Value (
    std::string valueName,
    T value ) [inline]
```

Writes most values to root (can't do strings)

**Template Parameters**

<i>T</i>	
----------	--

**Parameters**

<i>valueName</i>	Name of value being written to root
<i>value</i>	Value being written to root

Definition at line 46 of file file\_writer.hpp.

```
46                                     {
47     rapidjson::Value name(valueName.c_str(), rapidjson::SizeType(valueName.size()),
48     root.GetAllocator());
49     root.AddMember(name, value, root.GetAllocator());
49 }
```

References root.

Referenced by Engine::Write(), and Physics::Write().

**4.7.3.6 Write\_Vec3()** `void File_Writer::Write_Vec3 (`

```
    std::string valueName,
    glm::vec3 value )
```

Write a glm::vec3 into root.

**Parameters**

<i>valueName</i>	Name of glm::vec3
<i>value</i>	glm::vec3 to write

Definition at line 57 of file file\_writer.cpp.

```
57                                     {
58     // Storing glm::vec3 in array that rapidjson can write
59     Value vector3(kArrayType);
60     vector3.PushBack(value.x, root.GetAllocator());
61     vector3.PushBack(value.y, root.GetAllocator());
62     vector3.PushBack(value.z, root.GetAllocator());
63
64     // Writing vector3 into root
65     Value name(valueName.c_str(), SizeType(valueName.size()), root.GetAllocator());
66     root.AddMember(name, vector3, root.GetAllocator());
67 }
```

Referenced by Engine::Write(), Transform::Write(), and Physics::Write().



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

- [file\\_writer.hpp](#)
- [file\\_writer.cpp](#)

## 4.8 Graphics Class Reference

```
#include <graphics.hpp>
```

### Public Member Functions

- [Graphics](#) (int width, int height)  
*Creates [Graphics](#) object with given window size.*

### Static Public Member Functions

- static bool [Initialize](#) ([File\\_Reader](#) &settings)  
*Initializes the [Graphics](#) system using the settings in the given data.*
- static bool [InitializeGL](#) ()  
*Initializes the settings of the graphics system.*
- static void [Update](#) ()  
*[Graphics](#) update loop. Calls other update functions for the engine, input, and rendering. This is the main update function for the engine.*
- static void [Render](#) ()  
*Renders all of the objects in the object\_manager.*
- static void [Shutdown](#) ()  
*Shutdown the graphics system.*
- static bool [ErrorCheck](#) (GLenum error)  
*Checking for error in given enum.*
- static void [ErrorCallback](#) (int error, const char \*description)  
*Error callback for when the graphics system has an issue.*
- static std::pair< int, int > [GetWindowSize](#) ()  
*Returns window size.*
- static GLFWwindow \* [GetWindow](#) ()  
*Return the graphics window.*

### Private Attributes

- std::pair< int, int > [windowSize](#)  
*Size of the window.*
- GLFWwindow \* [window](#)  
*Window for application.*
- GLuint [vertexArrayId](#)  
*Id of the VAO.*

### 4.8.1 Detailed Description

[Graphics](#) class

Definition at line 28 of file `graphics.hpp`.

### 4.8.2 Constructor & Destructor Documentation

**4.8.2.1 Graphics()** `Graphics::Graphics (`  
    `int width,`  
    `int height )`

Creates [Graphics](#) object with given window size.

#### Parameters

<i>width</i>	
<i>height</i>	

Definition at line 51 of file `graphics.cpp`.

```
51 {  
52     windowSize.first = width;  
53     windowSize.second = height;  
54 }
```

### 4.8.3 Member Function Documentation

**4.8.3.1 errorCallback()** `void Graphics::ErrorCallback (`  
    `int error,`  
    `const char * description ) [static]`

Error callback for when the graphics system has an issue.

#### Parameters

<i>error</i>	Error that occurred
<i>description</i>	Description of error

#### Returns

void

Definition at line 223 of file graphics.cpp.

```
223                                     {
224     Trace::Message("Error: " + std::string(description) + ": " + std::to_string(error) + "\n");
225 }
```

References Trace::Message().

**4.8.3.2 ErrorCheck()** bool Graphics::ErrorCheck (   
GLenum error ) [static]

Checking for error in given enum.

Parameters

<i>error</i>	Possible error
--------------	----------------

Returns

true

false

Definition at line 234 of file graphics.cpp.

```
234                                     {
235     error = glGetError();
236     if (error != GL_NO_ERROR) {
237         Trace::Message("Error initializing OpenGL. \n");
238         return false;
239     }
240
241     return true;
242 }
```

References Trace::Message().

Referenced by InitializeGL().

**4.8.3.3 GetWindow()** GLFWwindow \* Graphics::GetWindow ( ) [static]

Return the graphics window.

Returns

GLFWwindow\*

Definition at line 258 of file graphics.cpp.

```
258                                     {
259     return graphics->window;
260 }
```

References graphics, and window.

Referenced by Editor::Display\_Scene(), Editor::Initialize(), Editor::Update(), Camera::Update(), and Update().

**4.8.3.4 GetWindowSize()** `std::pair< int, int > Graphics::GetWindowSize ( ) [static]`

Returns window size.

**Returns**

`std::pair<int, int>`

Definition at line 249 of file `graphics.cpp`.

```
249                                     {
250     return graphics->windowSize;
251 }
```

References `graphics`, and `windowSize`.

**4.8.3.5 Initialize()** `bool Graphics::Initialize ( File_Reader & settings ) [static]`

Initializes the [Graphics](#) system using the settings in the given data.

**Parameters**

<i>settings</i>	Settings information
-----------------	----------------------

**Returns**

`true`

`false`

Definition at line 63 of file `graphics.cpp`.

```
63                                     {
64     // Initializing graphics
65     graphics = new Graphics(settings.Read_Int("windowWidth"), settings.Read_Int("windowHeight"));
66     if (!graphics) {
67         Trace::Message("Graphics was not initialized.");
68         return false;
69     }
70
71     // Setting up error recording with graphics
72     glfwSetErrorCallback(ErrorCallback);
73
74     if (!glfwInit()) {
75         Trace::Message("Could not initialize GLFW.\n");
76         return false;
77     }
78
79     // Setting up the graphics window
80     graphics->window = glfwCreateWindow(graphics->windowSize.first, graphics->windowSize.second,
81         "pEngine", nullptr, nullptr);
82     if (!graphics->window) {
83         Trace::Message("Error creating window.\n");
84         return false;
85     }
86
87     // Setting up callback functions
88     glfwSetCursorPosCallback(graphics->window, Camera::MouseUpdate);
89
90     glfwMakeContextCurrent(graphics->window);
```

```

91     //glfwSwapInterval(1);
92     InitializeGL();
93
94     glewExperimental = GL_TRUE;
95     glewInit();
96
97     // Setting up input for keyboard and mouse using glfw library
98     glfwSetInputMode(graphics->window, GLFW_STICKY_KEYS, GL_TRUE);
99     glfwSetInputMode(graphics->window, GLFW_CURSOR, GLFW_CURSOR_HIDDEN);
100
101     glGenVertexArrays(1, &graphics->vertexArrayId);
102     glBindVertexArray(graphics->vertexArrayId);
103
104     if (!Shader::Initialize(settings)) return false;
105
106     return true;
107 }

```

References `graphics`, `Shader::Initialize()`, `Trace::Message()`, `Camera::MouseUpdate()`, `File_Reader::Read_Int()`, `vertexArrayId`, `window`, and `windowSize`.

Referenced by `Engine::Initialize()`.

#### 4.8.3.6 InitializeGL() `bool Graphics::InitializeGL ( ) [static]`

Initializes the settings of the graphics system.

##### Returns

`true`  
`false`

Definition at line 115 of file `graphics.cpp`.

```

115     {
116         GLenum error = GL_NO_ERROR;
117
118         glClearColor(0.f, 0.f, 0.f, 1.f);
119         if (!Graphics::ErrorCheck(error)) return false;
120
121         glClearDepth(1.f);
122         if (!Graphics::ErrorCheck(error)) return false;
123
124         glEnable(GL_DEPTH_TEST);
125         if (!Graphics::ErrorCheck(error)) return false;
126
127         glDepthFunc(GL_LEQUAL);
128         if (!Graphics::ErrorCheck(error)) return false;
129
130         glShadeModel(GL_SMOOTH);
131         if (!Graphics::ErrorCheck(error)) return false;
132
133         glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
134         if (!Graphics::ErrorCheck(error)) return false;
135
136         glEnable(GL_CULL_FACE);
137         if (!Graphics::ErrorCheck(error)) return false;
138
139         return true;
140     }

```

References `ErrorCheck()`.

**4.8.3.7 Render()** void Graphics::Render ( ) [static]

Renders all of the objects in the object\_manager.

**Returns**

void

Definition at line 169 of file graphics.cpp.

```

169     {
170         // Setting up graphics system for rendering
171         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
172         Shader::Update();
173
174         glm::mat4 projection = perspective(radians(Camera::GetFov()), (float)graphics->windowSize.first /
175             (float)graphics->windowSize.second, Camera::GetNear(), Camera::GetFar());
176
177         // Getting the view matrix of the camera
178         glm::mat4 view = lookAt(
179             Camera::GetPosition(),
180             Camera::GetPosition() + Camera::GetFront(),
181             Camera::GetUp());
182
183         // Rendering all of the objects
184         for (unsigned i = 0; i < Object_Manager::GetSize(); ++i) {
185             Object* object = Object_Manager::FindObject(i);
186
187             Model* model = object->GetComponent<Model>();
188             if (!model) continue;
189
190             model->Draw(projection, view);
191         }
192
193         Editor::Render();
194
195         glfwSwapBuffers(graphics->window);
196     }

```

References Model::Draw(), Object\_Manager::FindObject(), Camera::GetFar(), Camera::GetFov(), Camera::GetFront(), Camera::GetNear(), Camera::GetPosition(), Object\_Manager::GetSize(), Camera::GetUp(), graphics, Editor::Render(), Shader::Update(), window, and windowSize.

**4.8.3.8 Shutdown()** void Graphics::Shutdown ( ) [static]

Shutdown the graphics system.

**Returns**

void

Definition at line 203 of file graphics.cpp.

```

203     {
204         if (!graphics) return;
205
206         Shader::Shutdown();
207         glDeleteVertexArrays(1, &graphics->vertexArrayId);
208         // Shutting down opengl
209         glfwDestroyWindow(graphics->window);
210         glfwTerminate();
211         // Deleting graphics object
212         delete graphics;
213         graphics = nullptr;
214     }

```

References graphics, Shader::Shutdown(), vertexArrayId, and window.

Referenced by Engine::Shutdown().

#### 4.8.3.9 Update() `void Graphics::Update ( ) [static]`

**Graphics** update loop. Calls other update functions for the engine, input, and rendering. This is the main update function for the engine.

##### Returns

void

Definition at line 148 of file `graphics.cpp`.

```

148     {
149         while(!glfwWindowShouldClose(graphics->window)) {
150             // Run updates
151             Engine::Update();
152             Render();
153             glfwPollEvents();
154
155             // Check for restart
156             if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_R) == GLFW_PRESS && Editor::GetTakeKeyboardInput()) {
157                 if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_R) == GLFW_RELEASE) {
158                     Engine::Restart();
159                 }
160             }
161         }
162     }
```

References `Editor::GetTakeKeyboardInput()`, `GetWindow()`, `graphics`, `Engine::Restart()`, `Engine::Update()`, and `window`.

Referenced by `main()`.

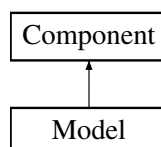
The documentation for this class was generated from the following files:

- [graphics.hpp](#)
- [graphics.cpp](#)

## 4.9 Model Class Reference

```
#include <model.hpp>
```

Inheritance diagram for `Model`:



## Public Member Functions

- [Model](#) (GLenum mode\_=GL\_TRIANGLES)  
*Creates a [Model](#) object with default values.*
- [Model](#) (const [Model](#) &other)  
*Copy constructor.*
- [Model](#) ([File\\_Reader](#) &reader, GLenum mode\_=GL\_TRIANGLES)  
*Creates a [Model](#) object using the data from a file.*
- [Model](#) \* [Clone](#) () const  
*Clones this [Model](#) object.*
- void [Load](#) ([File\\_Reader](#) &reader)  
*Load in the model data from a file (use model manager to not have multiple versions of the same model)*
- void [Draw](#) (glm::mat4 projection, glm::mat4 view)  
*Draw the model.*
- void [Read](#) ([File\\_Reader](#) &reader)  
*Reads name of model file and passes it to the Load function.*
- void [Write](#) ([File\\_Writer](#) &writer)  
*Gives name of model and texture to writer.*
- void [SwitchModel](#) (std::string modelName)  
*Switches the current model to that of the filename provided.*
- void [SwitchTexture](#) (std::string textureName)  
*Switches the current texture to that of the filename provided.*
- std::string [GetModelName](#) () const  
*Returns the filename of the current model.*
- std::string [GetTextureName](#) () const  
*Returns the filename of the current texture.*
- [Texture](#) \* [GetTexture](#) () const  
*Returns pointer to texture object.*

## Static Public Member Functions

- static [CType](#) [GetCType](#) ()  
*Gets the CType of [Model](#) (used in [Object::GetComponent<>\(\)](#))*

## Private Attributes

- GLenum [mode](#)  
*Draw mode (Default is GL\_TRIANGLES)*
- [Model\\_Data](#) \* [data](#)  
*Data about the faces of the model.*
- [Texture](#) \* [texture](#)  
*[Texture](#) object of model.*



## Additional Inherited Members

#### 4.9.1 Detailed Description

Model class

Definition at line 32 of file model.hpp.

### 4.9.2 Constructor & Destructor Documentation

```
4.9.2.1 Model() [1/3] Model::Model (
    GLenum mode_ = GL_TRIANGLES )
```

Creates a **Model** object with default values.

## Parameters

<i>mode</i> ←	Draw mode for opengl
—	

Definition at line 32 of file model.cpp.

```
32 : Component(CType::CModel), mode(mode_), data(nullptr), texture(nullptr) {}
```

Referenced by Clone().

#### 4.9.2.2 Model() [2/3] Model::Model ( const Model & other )

Copy constructor.

## Parameters

<i>other</i>	
--------------	--

Definition at line 39 of file model.cpp.

```
39 : Component(CType::CModel) { *this = other; }
```

```
4.9.2.3 Model() [3/3] Model::Model (
    File_Reader & reader,
    GLenum mode_ = GL_TRIANGLES )
```

Creates a [Model](#) object using the data from a file.

#### Parameters

<i>reader</i>	File with <a href="#">Model</a> data
<i>mode</i> ↔	Draw mode for opengl
—	

Definition at line 47 of file model.cpp.

```

47         : Component(CType::CModel), mode(mode_), data(nullptr),
      texture(nullptr) {
48     Read(reader);
49 }
```

References [Read\(\)](#).

### 4.9.3 Member Function Documentation

#### 4.9.3.1 Clone() [Model](#) \* [Model](#)::Clone ( ) const

Clones this [Model](#) object.

#### Returns

[Model](#)\* Cloned [Model](#)

Definition at line 56 of file model.cpp.

```
56 { return new Model(*this); }
```

References [Model\(\)](#).

#### 4.9.3.2 Draw() void [Model](#)::Draw ( glm::mat4 *projection*, glm::mat4 *view* )

Draw the model.

#### Parameters

<i>projection</i>	Projection matrix of the scene
<i>view</i>	View matrix of the scene

Definition at line 75 of file model.cpp.

```
75                                     {
76     Transform* transform = GetParent()->GetComponent<Transform>();
77     if (!data) return;
78
79     data->Draw(this, transform, projection, view);
80 }
```

References data, Model\_Data::Draw(), Object::GetComponent(), and Component::GetParent().

Referenced by Graphics::Render().

#### 4.9.3.3 GetCType() CType Model::GetCType ( ) [static]

Gets the CType of Model (used in Object::GetComponent<>())

##### Returns

CType

Definition at line 158 of file model.cpp.

```
158     {
159     return CType::CModel;
160 }
```

#### 4.9.3.4 GetModelName() std::string Model::GetModelName ( ) const

Returns the filename of the current model.

##### Returns

std::string

Definition at line 131 of file model.cpp.

```
131     {
132     if (!data) return "no model";
133     return data->GetModelName();
134 }
```

References data, and Model\_Data::GetModelName().

Referenced by Editor::Display\_Model().

**4.9.3.5 GetTexture()** `Texture * Model::GetTexture ( ) const`

Returns pointer to texture object.

**Returns**

Texture\*

Definition at line 151 of file model.cpp.

```
151 { return texture; }
```

References texture.

Referenced by Model\_Data::Draw().

**4.9.3.6 GetTextureName()** `std::string Model::GetTextureName ( ) const`

Returns the filename of the current texture.

**Returns**

std::string

Definition at line 141 of file model.cpp.

```
141 {
142     if (!texture) return "no texture";
143     return texture->GetTextureName();
144 }
```

References Texture::GetTextureName(), and texture.

Referenced by Editor::Display\_Model().

**4.9.3.7 Load()** `void Model::Load (
 File_Reader & reader )`

Load in the model data from a file (use model manager to not have multiple versions of the same model)

**Parameters**

<i>reader</i>	File_reader object that contains <a href="#">Model</a> info
---------------	---

Definition at line 64 of file model.cpp.

```
64 {
65     data = Model_Data_Manager::Get(reader);
66     texture = Texture_Manager::Get(reader);
67 }
```

References data, Texture\_Manager::Get(), Model\_Data\_Manager::Get(), and texture.

Referenced by Read().

**4.9.3.8 Read()** `void Model::Read (`  
    `File_Reader & reader )`

Reads name of model file and passes it to the Load function.

#### Parameters

<i>reader</i>	File that contains the name of the model's file
---------------	---

Definition at line 87 of file model.cpp.

```
87 { Load(reader); }
```

References Load().

Referenced by Model(), and Object::ReRead().

**4.9.3.9 SwitchModel()** `void Model::SwitchModel (`  
    `std::string modelName )`

Switches the current model to that of the filename provided.

#### Parameters

<i>modelName</i>	
------------------	--

Definition at line 107 of file model.cpp.

```
107                                     {
108     Model_Data* proxy = Model_Data_Manager::Get (modelName);
109     if (!proxy) return;
110
111     data = proxy;
112 }
```

References data, and Model\_Data\_Manager::Get().

Referenced by Editor::Display\_Model().

**4.9.3.10 SwitchTexture()** `void Model::SwitchTexture (`  
    `std::string textureName )`

Switches the current texture to that of the filename provided.

## Parameters

<i>textureName</i>	
--------------------	--

Definition at line 119 of file model.cpp.

```
119 {
120     Texture* proxy = Texture_Manager::Get(textureName);
121     if (!proxy) return;
122
123     texture = proxy;
124 }
```

References Texture\_Manager::Get(), and texture.

Referenced by Editor::Display\_Model().

**4.9.3.11 Write()** void Model::Write (  
File\_Writer & writer )

Gives name of model and texture to writer.

## Parameters

<i>writer</i>	
---------------	--

Definition at line 94 of file model.cpp.

```
94 {
95     std::string modelName = data->GetModelName();
96     std::string textureName = texture->GetTextureName();
97
98     writer.Write_String("modelToLoad", modelName.c_str());
99     writer.Write_String("textureToLoad", textureName.c_str());
100 }
```

References data, Model\_Data::GetModelName(), Texture::GetTextureName(), texture, and File\_Writer::Write\_String().

Referenced by Object::Write().

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

- [model.hpp](#)
- [model.cpp](#)

## 4.10 Model\_Data Class Reference

```
#include <model_data.hpp>
```

## Public Member Functions

- [Model\\_Data](#) ()  
*Default constructor.*
- [Model\\_Data](#) (const [Model\\_Data](#) &other)  
*Copy constructor.*
- [~Model\\_Data](#) ()  
*Deletes all buffers of the model.*
- bool [Load](#) ([File\\_Reader](#) &reader)  
*Loads data of a model from given file.*
- bool [Load](#) (std::string modelName\_)  
*Loads in model using given filename.*
- bool [Read](#) (std::string modelName\_)  
*Reads model data from file.*
- void [Draw](#) ([Model](#) \*parent, [Transform](#) \*transform, glm::mat4 projection, glm::mat4 view)  
*Draws the models.*
- std::string [GetModelName](#) () const  
*Returns the filename that the models data was gotten from.*

## Private Attributes

- std::vector< float > [vertices](#)  
*Contains vertices of model.*
- std::vector< float > [normals](#)  
*Contains normals of model.*
- std::vector< float > [uvs](#)  
*Contains uv data of model.*
- std::string [modelName](#)  
*Name of the file for the model.*
- GLuint [vertexbuffer](#)  
*Vertex buffer of model.*
- GLuint [normalbuffer](#)  
*Normal buffer of model.*
- GLuint [uvbuffer](#)  
*UV buffer of model.*

### 4.10.1 Detailed Description

[Model\\_Data](#) class

Definition at line 33 of file [model\\_data.hpp](#).

### 4.10.2 Constructor & Destructor Documentation

**4.10.2.1 Model\_Data()** [1/2] `Model_Data::Model_Data ( )`

Default constructor.

Definition at line 33 of file `model_data.cpp`.

```
33 {}
```

**4.10.2.2 Model\_Data()** [2/2] `Model_Data::Model_Data ( const Model\_Data & other )`

Copy constructor.

Parameters

<i>other</i>	
--------------	--

Definition at line 40 of file `model_data.cpp`.

```
40                                     {
41     for (float vert : other.vertices) {
42         vertices.emplace_back(vert);
43     }
44     for (float norm : other.normals) {
45         normals.emplace_back(norm);
46     }
47     for (float uv : other.uvs) {
48         uvs.emplace_back(uv);
49     }
50
51     vertexbuffer = other.vertexbuffer;
52     normalbuffer = other.normalbuffer;
53     uvbuffer = other.uvbuffer;
54 }
```

References `normalbuffer`, `normals`, `uvbuffer`, `uvs`, `vertexbuffer`, and `vertices`.

**4.10.2.3 ~Model\_Data()** `Model_Data::~~Model_Data ( )`

Deletes all buffers of the model.

Definition at line 60 of file `model_data.cpp`.

```
60     {
61         glDeleteBuffers(1, &vertexbuffer);
62         glDeleteBuffers(1, &uvbuffer);
63         glDeleteBuffers(1, &normalbuffer);
64     }
```

References `normalbuffer`, `uvbuffer`, and `vertexbuffer`.

**4.10.3 Member Function Documentation**



**4.10.3.1 Draw()** void Model\_Data::Draw (

```

    Model * parent,
    Transform * transform,
    glm::mat4 projection,
    glm::mat4 view )

```

Draws the models.

Parameters

<i>parent</i>	Model component
<i>transform</i>	Transform component
<i>projection</i>	Projection matrix of the scene
<i>view</i>	View matrix of the scene

Definition at line 218 of file model\_data.cpp.

```

218
219     // Creating the MVP (Model * View * Projection) matrix
220     glm::mat4 model = glm::mat4(1.f);
221     model = glm::translate(model, transform->GetPosition());
222     model = glm::rotate(model, (transform->GetRotation().x / 180.f) * glm::pi<float>(), glm::vec3(1, 0, 0));
223     model = glm::rotate(model, (transform->GetRotation().y / 180.f) * glm::pi<float>(), glm::vec3(0, 1, 0));
224     model = glm::rotate(model, (transform->GetRotation().z / 180.f) * glm::pi<float>(), glm::vec3(0, 0, 1));
225     model = glm::scale(model, transform->GetScale());
226
227     // Sending data to the shaders
228     glm::mat4 MVP = projection * view * model;
229     glUniformMatrix4fv(Shader::GetMatrixId(), 1, GL_FALSE, &MVP[0][0]);
230     glUniformMatrix4fv(Shader::GetModelMatrixId(), 1, GL_FALSE, &model[0][0]);
231     glUniformMatrix4fv(Shader::GetViewMatrixId(), 1, GL_FALSE, &view[0][0]);
232
233     // Sending light data to the shaders
234     glm::vec3 lightPos = Engine::GetLightPos();
235     glUniform3f(Shader::GetLightId(), lightPos.x, lightPos.y, lightPos.z);
236     glUniform1f(Shader::GetLightPowerId(), Engine::GetLightPower());
237
238     // Setup texture for drawing if it exists
239     if (parent->GetTexture())
240         parent->GetTexture()->Display();
241
242     // Setup the model vertices
243     glEnableVertexAttribArray(0);
244     glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
245     glVertexAttribPointer(
246         0,
247         3,
248         GL_FLOAT,
249         GL_FALSE,
250         0,
251         (void*)0
252     );
253
254     // Setup the model uv
255     glEnableVertexAttribArray(1);
256     glBindBuffer(GL_ARRAY_BUFFER, uvbuffer);
257     glVertexAttribPointer(
258         1,
259         2,
260         GL_FLOAT,
261         GL_FALSE,
262         0,
263         (void*)0
264     );
265
266     // Setup the model normals
267     glEnableVertexAttribArray(2);
268     glBindBuffer(GL_ARRAY_BUFFER, normalbuffer);
269     glVertexAttribPointer(
270         2,
271         3,
272         GL_FLOAT,

```

```

273         GL_FALSE,
274         0,
275         (void*)0
276     );
277
278     // Draw the object
279     glDrawArrays(GL_TRIANGLES, 0, vertices.size());
280
281     // Disable data sent to shaders
282     glDisableVertexAttribArray(0);
283     glDisableVertexAttribArray(1);
284     glDisableVertexAttribArray(2);
285
286 }

```

References Texture::Display(), Shader::GetLightId(), Engine::GetLightPos(), Engine::GetLightPower(), Shader::GetLightPowerId(), Shader::GetMatrixId(), Shader::GetModelMatrixId(), Transform::GetPosition(), Transform::GetRotation(), Transform::GetScale(), Model::GetTexture(), Shader::GetViewMatrixId(), normalbuffer, uvbuffer, vertexbuffer, and vertices.

Referenced by Model::Draw().

#### 4.10.3.2 GetModelName() `std::string Model_Data::GetModelName ( ) const`

Returns the filename that the models data was gotten from.

##### Returns

string Name of the file that contains model data

Definition at line 293 of file model\_data.cpp.

```
293 { return modelName; }
```

References modelName.

Referenced by Model\_Data\_Manager::Get(), Model::GetModelName(), and Model::Write().

#### 4.10.3.3 Load() [1/2] `bool Model_Data::Load ( File_Reader & reader )`

Loads data of a model from given file.

##### Parameters

<i>reader</i>	<code>File_Reader</code> object containing the model data
---------------	---

##### Returns

true  
false

Definition at line 73 of file model\_data.cpp.

```
73 {  
74     std::string modelName_ = reader.Read_String("modelToLoad");  
75  
76     return Read(modelName_);  
77 }
```

References Read(), and File\_Reader::Read\_String().

Referenced by Model\_Data\_Manager::Get().

**4.10.3.4 Load()** [2/2] `bool Model_Data::Load (`  
`std::string modelName_ )`

Loads in model using given filename.

#### Parameters

<i>modelName_</i>	Model's filename
-------------------	------------------

#### Returns

true

false

Definition at line 86 of file model\_data.cpp.

```
86 { return Read(modelName_); }
```

References Read().

**4.10.3.5 Read()** `bool Model_Data::Read (`  
`std::string modelName_ )`

Reads model data from file.

#### Parameters

<i>modelName_</i>	Model's filename
-------------------	------------------

#### Returns

true

false

Definition at line 95 of file model\_data.cpp.

```

95                                     {
96     // Opening the file
97     std::string fileToOpen = std::string(getenv("USERPROFILE")) + "/Documents/pEngine/models/" + modelName_;
98     FILE* file = fopen(fileToOpen.c_str(), "r");
99     if (!file) {
100         return false;
101     }
102
103     // Setting the name of the file (used in model_data_manager)
104     modelName = modelName_;
105
106     // Creating variables for reading
107     std::vector<unsigned> vertex_indices, uv_indices, normal_indices;
108     std::vector<glm::vec3> temp_vertices;
109     std::vector<glm::vec2> temp_uvs;
110     std::vector<glm::vec3> temp_normals;
111
112     // Until the whole file is read
113     while (true) {
114         char line_header[256];
115
116         // Getting next line of the file
117         int res = fscanf(file, "%s", line_header);
118         if (res == EOF) break;
119
120         // Checking for which data needs to be read in
121         if (strcmp(line_header, "v") == 0) {
122             glm::vec3 vertex;
123             fscanf(file, "%f %f %f\n", &vertex.x, &vertex.y, &vertex.z);
124             temp_vertices.emplace_back(vertex);
125             continue;
126         }
127
128         if (strcmp(line_header, "vt") == 0) {
129             glm::vec2 uv;
130             fscanf(file, "%f %f\n", &uv.x, &uv.y);
131             temp_uvs.emplace_back(uv);
132             continue;
133         }
134
135         if (strcmp(line_header, "vn") == 0) {
136             glm::vec3 normal;
137             fscanf(file, "%f %f %f\n", &normal.x, &normal.y, &normal.z);
138             temp_normals.emplace_back(normal);
139             continue;
140         }
141
142         if (strcmp(line_header, "f") == 0) {
143             // Connecting face to previous read vertices, uvs, and normals
144             unsigned vertex_index[3], uv_index[3], normal_index[3];
145             int matches = fscanf(file, "%d/%d/%d %d/%d/%d %d/%d/%d\n", &vertex_index[0], &uv_index[0],
146 &normal_index[0],
147 &vertex_index[1], &uv_index[1], &normal_index[1], &vertex_index[2], &uv_index[2],
148 &normal_index[2]);
149
150             // Expects models split into triangles
151             if (matches != 9) {
152                 Trace::Message("File is incompatible with this parser. Export using different settings.");
153                 return false;
154             }
155
156             // Setting vertices for current face
157             vertices.emplace_back((temp_vertices[vertex_index[0] - 1]).x);
158             vertices.emplace_back((temp_vertices[vertex_index[0] - 1]).y);
159             vertices.emplace_back((temp_vertices[vertex_index[0] - 1]).z);
160
161             vertices.emplace_back((temp_vertices[vertex_index[1] - 1]).x);
162             vertices.emplace_back((temp_vertices[vertex_index[1] - 1]).y);
163             vertices.emplace_back((temp_vertices[vertex_index[1] - 1]).z);
164
165             vertices.emplace_back((temp_vertices[vertex_index[2] - 1]).x);
166             vertices.emplace_back((temp_vertices[vertex_index[2] - 1]).y);
167             vertices.emplace_back((temp_vertices[vertex_index[2] - 1]).z);
168
169             // Setting uvs for current face
170             uvs.emplace_back((temp_uvs[uv_index[0] - 1]).x);
171             uvs.emplace_back((temp_uvs[uv_index[0] - 1]).y);
172
173             uvs.emplace_back((temp_uvs[uv_index[1] - 1]).x);
174             uvs.emplace_back((temp_uvs[uv_index[1] - 1]).y);

```

```

173
174         uvs.emplace_back((temp_uvs[uv_index[2] - 1]).x);
175         uvs.emplace_back((temp_uvs[uv_index[2] - 1]).y);
176
177         // Setting normals for current face
178         normals.emplace_back((temp_normals[normal_index[0] - 1]).x);
179         normals.emplace_back((temp_normals[normal_index[0] - 1]).y);
180         normals.emplace_back((temp_normals[normal_index[0] - 1]).z);
181
182         normals.emplace_back((temp_normals[normal_index[1] - 1]).x);
183         normals.emplace_back((temp_normals[normal_index[1] - 1]).y);
184         normals.emplace_back((temp_normals[normal_index[1] - 1]).z);
185
186         normals.emplace_back((temp_normals[normal_index[2] - 1]).x);
187         normals.emplace_back((temp_normals[normal_index[2] - 1]).y);
188         normals.emplace_back((temp_normals[normal_index[2] - 1]).z);
189     }
190 }
191
192     // Bind vertex data to buffers
193     glGenBuffers(1, &vertexbuffer);
194     glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
195     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(float), &vertices[0], GL_STATIC_DRAW);
196
197     // Bind uv data to buffers
198     glGenBuffers(1, &uvbuffer);
199     glBindBuffer(GL_ARRAY_BUFFER, uvbuffer);
200     glBufferData(GL_ARRAY_BUFFER, uvs.size() * sizeof(float), &uvs[0], GL_STATIC_DRAW);
201
202     // Bind normals data to buffers
203     glGenBuffers(1, &normalbuffer);
204     glBindBuffer(GL_ARRAY_BUFFER, normalbuffer);
205     glBufferData(GL_ARRAY_BUFFER, normals.size() * sizeof(float), &normals[0], GL_STATIC_DRAW);
206
207     return true;
208 }

```

References `Trace::Message()`, `modelName`, `normalbuffer`, `normals`, `uvbuffer`, `uvs`, `vertexbuffer`, and `vertices`.

Referenced by `Load()`.

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

- [model\\_data.hpp](#)
- [model\\_data.cpp](#)

## 4.11 Model\_Data\_Manager Class Reference

```
#include <model_data_manager.hpp>
```

### Static Public Member Functions

- static bool [Initialize](#) ()  
*Initializes the model\_data\_manager.*
- static [Model\\_Data](#) \* [Get](#) ([File\\_Reader](#) &reader)  
*Checks if model data has already been read in. If yes then it returns a pointer to that data. If no it reads it in and adds it to the model list.*
- static [Model\\_Data](#) \* [Get](#) (std::string modelName)  
*Checks if model data has already been read in. If yes then it returns a pointer to that data. If no it reads it in and adds it to the model list.*
- static void [Shutdown](#) ()  
*Deletes all of the [Model\\_Data](#) objects in the models list then deletes model\_data\_manager.*

## Private Attributes

- `std::vector< Model\_Data * > models`  
List of the different [Model\\_Data](#) objects.

### 4.11.1 Detailed Description

[Model\\_Data\\_Manager](#) class

Definition at line 25 of file `model_data_manager.hpp`.

### 4.11.2 Member Function Documentation

#### 4.11.2.1 `Get()` [1/2] [Model\\_Data](#) \* `Model_Data_Manager::Get` ( [File\\_Reader](#) & `reader`) [static]

Checks if model data has already been read in. If yes then it returns a pointer to that data. If no it reads it in and adds it to the model list.

#### Parameters

<code>reader</code>	<a href="#">File_Reader</a> object containing model data
---------------------	--

#### Returns

[Model\\_Data](#)\* [Model](#) data either read or gotten from list

Definition at line 44 of file `model_data_manager.cpp`.

```

44         {
45     std::string filename = reader.Read_String("modelToLoad");
46     // Checks name of file against other model data objects
47     for (Model\_Data* model_data : model_data_manager->models) {
48         if (model_data->GetModelName().compare(filename) == 0) {
49             return model_data;
50         }
51     }
52
53     // Creates new Model\_Data object, then adds it to list
54     Model\_Data* data = new Model\_Data;
55     data->Load(reader);
56     model_data_manager->models.emplace_back(data);
57
58     return data;
59 }
```

References [Model\\_Data::GetModelName\(\)](#), [Model\\_Data::Load\(\)](#), [model\\_data\\_manager](#), [models](#), and [File\\_Reader::Read\\_String\(\)](#).

Referenced by [Model::Load\(\)](#), and [Model::SwitchModel\(\)](#).

**4.11.2.2 Get()** [2/2] `Model_Data * Model_Data_Manager::Get (`  
`std::string modelName ) [static]`

Checks if model data has already been read in. If yes then it returns a pointer to that data. If no it reads it in and adds it to the model list.

#### Parameters

<i>modelName</i>	Filename of the model to get
------------------	------------------------------

#### Returns

`Model_Data*` `Model` data either read or gotten from list

Definition at line 69 of file `model_data_manager.cpp`.

```

69         {
70         // Checks name of file against other model data objects
71         for (Model_Data* model_data : model_data_manager->models) {
72             if (model_data->GetModelName().compare(modelName) == 0) {
73                 return model_data;
74             }
75         }
76
77         // Creates new Model_Data object, then adds it to list
78         Model_Data* data = new Model_Data;
79         if (!data->Load(modelName)) {
80             delete data;
81             return nullptr;
82         }
83         model_data_manager->models.emplace_back(data);
84
85         return data;
86     }

```

References `Model_Data::GetModelName()`, `Model_Data::Load()`, `model_data_manager`, and `models`.

**4.11.2.3 Initialize()** `bool Model_Data_Manager::Initialize ( ) [static]`

Initializes the `model_data_manager`.

#### Returns

`true`  
`false`

Definition at line 24 of file `model_data_manager.cpp`.

```

24         {
25         // Initializing model_data_manager
26         model_data_manager = new Model_Data_Manager;
27         if (!model_data_manager) {
28             Trace::Message("Model Data Manager was not initialized.\n");
29             return false;
30         }
31
32         model_data_manager->models.reserve(10);
33         return true;
34     }

```

References `Trace::Message()`, `model_data_manager`, and `models`.

Referenced by `Engine::Initialize()`.

**4.11.2.4 Shutdown()** `void Model_Data_Manager::Shutdown ( ) [static]`

Deletes all of the [Model\\_Data](#) objects in the models list then deletes model\_data\_manager.

**Returns**

void

Definition at line 94 of file model\_data\_manager.cpp.

```

94     {
95         if (!model_data_manager) return;
96
97         // Deleting all of the Model_Data objects
98         for (Model_Data* model_data : model_data_manager->models) {
99             if (!model_data) continue;
100
101             delete model_data;
102             model_data = nullptr;
103         }
104
105         delete model_data_manager;
106         model_data_manager = nullptr;
107     }

```

References model\_data\_manager, and models.

Referenced by Engine::Shutdown().

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

- [model\\_data\\_manager.hpp](#)
- [model\\_data\\_manager.cpp](#)

**4.12 Object Class Reference**

```
#include <object.hpp>
```

**Public Member Functions**

- [Object](#) ()  
*Default constructor.*
- [Object](#) (const [Object](#) &other)  
*Copy constructor.*
- [Object](#) \* [Clone](#) () const  
*Clones this object.*
- void [Update](#) ()  
*Updates object (only physics for now)*
- void [AddComponent](#) ([Component](#) \*component)  
*Adds component to object. Only one of each type of component.*
- template<typename T >  
T \* [GetComponent](#) ()  
*Get a component of the object.*



- `template<typename T >`  
`void RemoveComponent ()`  
*Removes component from object.*
- `void SetId (int id_)`  
*Sets the id of object.*
- `int GetId () const`  
*Returns the id of object.*
- `void SetName (std::string name_)`  
*Sets name of object.*
- `std::string GetName () const`  
*Returns name of object.*
- `std::string & GetNameRef ()`  
*Returns reference to the name.*
- `void SetTemplateName (std::string templateName_)`  
*Sets the name of the template file.*
- `std::string GetTemplateName () const`  
*Returns the name of the template file.*
- `bool Read (std::string objectFilename)`  
*Reads object from file. This includes the components of an object.*
- `bool ReRead (std::string objectFilename)`  
*Reading data into object that already exists.*
- `void Write ()`  
*Writes the data of the object to a template file.*
- `std::unordered_map< CType, Component * > GetComponentList ()`  
*Returns the list of components.*
- `void Clear ()`  
*Clears the components from the object.*

## Private Member Functions

- `template<typename T >`  
`T * GetComponentConst () const`  
*Get a component of the object (const)*

## Private Attributes

- `std::unordered_map< CType, Component * > components`  
*List of components.*
- `std::string name`  
*Name of the object.*
- `std::string templateName`  
*Name of the template file used.*
- `int id`  
*Location of object in object\_manager.*

### 4.12.1 Detailed Description

[Object](#) class

Definition at line 25 of file `object.hpp`.

### 4.12.2 Constructor & Destructor Documentation

#### 4.12.2.1 `Object()` [1/2] `Object::Object ( )`

Default constructor.

Definition at line 28 of file `object.cpp`.

```
28 : id(-1) {}
```

Referenced by `Clone()`.

#### 4.12.2.2 `Object()` [2/2] `Object::Object (`                   `const Object & other )`

Copy constructor.

Parameters

<i>other</i>	<a href="#">Object</a> to be copied
--------------	-------------------------------------

Definition at line 35 of file `object.cpp`.

```
35     {
36         SetName(other.GetName());
37         SetTemplateName(other.GetTemplateName());
38
39         // Copying Behavior component
40         Behavior* behavior = other.GetComponentConst<Behavior>();
41         if (behavior) {
42             Behavior* newBehavior = new Behavior(*behavior);
43             AddComponent(newBehavior);
44         }
45
46         // Copying Model component
47         Model* model = other.GetComponentConst<Model>();
48         if (model) {
49             Model* newModel = new Model(*model);
50             AddComponent(newModel);
51         }
52
53         // Copying Physics component
54         Physics* physics = other.GetComponentConst<Physics>();
55         if (physics) {
56             Physics* newPhysics = new Physics(*physics);
57             AddComponent(newPhysics);
58         }
59     }
```

```

60     // Copying transform component
61     Transform* transform = other.GetComponentConst<Transform>();
62     if (transform) {
63         Transform* newTransform = new Transform(*transform);
64         AddComponent(newTransform);
65     }
66 }

```

References `AddComponent()`, `GetComponentConst()`, `GetName()`, `GetTemplateName()`, `SetName()`, and `SetTemplateName()`.

### 4.12.3 Member Function Documentation

#### 4.12.3.1 `AddComponent()` `void Object::AddComponent ( Component * component )`

Adds component to object. Only one of each type of component.

##### Parameters

<i>component</i>	<code>Component</code> to be added
------------------	------------------------------------

Definition at line 95 of file `object.cpp`.

```

95     {
96         component->SetParent(this);
97         components.emplace(component->GetCType(), component);
98     }

```

References `components`, `Component::GetCType()`, and `Component::SetParent()`.

Referenced by `Editor::Display_Scene()`, `Object()`, `Read()`, and `ReRead()`.

#### 4.12.3.2 `Clear()` `void Object::Clear ( )`

Clears the components from the object.

Definition at line 273 of file `object.cpp`.

```

273     {
274         Behavior* behavior = GetComponent<Behavior>();
275         Model* model = GetComponent<Model>();
276         Physics* physics = GetComponent<Physics>();
277
278         if (behavior) {
279             delete behavior;
280             behavior = nullptr;
281         }
282         if (model) {
283             delete model;
284             model = nullptr;
285         }
286         if (physics) {
287             delete physics;
288             physics = nullptr;
289         }
290     }

```

**4.12.3.3 Clone()** `Object * Object::Clone ( ) const`

Clones this object.

**Returns**

Object\*

Definition at line 73 of file object.cpp.

```
73     {
74         return new Object(*this);
75     }
```

References Object().

**4.12.3.4 GetComponent()** `template<typename T >`

`T* Object::GetComponent ( ) [inline]`

Get a component of the object.

**Template Parameters**

<i>T</i>	Component class to return
----------	---------------------------

**Parameters**

<i>type</i>	Type of component
-------------	-------------------

**Returns**

T\* Pointer to the component

Definition at line 44 of file object.hpp.

```
44     {
45         // Searching for component using the type (enum as int)
46         auto found = components.find(T::GetCType());
47         if (found == components.end()) {
48             return nullptr;
49         }
50         // Cast found component into correct type
51         return (T*) found->second;
52     }
```

References components.

Referenced by Model::Draw(), Physics::Update(), and Physics::UpdateGravity().

**4.12.3.5 GetComponentConst()** `template<typename T >`

`T* Object::GetComponentConst ( ) const [inline], [private]`

Get a component of the object (const)

### Template Parameters

<i>T</i>	<a href="#">Component</a> class to return
----------	---

### Parameters

<i>type</i>	Type of component
-------------	-------------------

### Returns

*T\** Pointer to the component

Definition at line 96 of file object.hpp.

```
96                                     {
97     // Searching for component using the type (enum as int)
98     auto found = components.find(T::GetCType());
99     if (found == components.end()) {
100         return nullptr;
101     }
102     // Cast found component into correct type
103     return (T*)found->second;
104 }
```

References components.

Referenced by Object().

**4.12.3.6 GetComponentList()** `std::unordered_map< CType, Component * > Object::GetComponentList ( )`

Returns the list of components.

### Returns

`std::unordered_map<CType, Component*>`

Definition at line 265 of file object.cpp.

```
265                                     {
266     return components;
267 }
```

References components.

#### 4.12.3.7 GetId() `int Object::GetId ( ) const`

Returns the id of object.

##### Returns

unsigned Position in [Object\\_Manager](#)

Definition at line 112 of file object.cpp.

```
112 { return id; }
```

References id.

Referenced by [Object\\_Manager::CheckName\(\)](#), [Behavior::ClassSetup\(\)](#), [Editor::Display\\_Components\(\)](#), and [File\\_Writer::Write\\_Object\\_Data\(\)](#).

#### 4.12.3.8 GetName() `std::string Object::GetName ( ) const`

Returns name of object.

##### Returns

string Name of object

Definition at line 128 of file object.cpp.

```
128 { return name; }
```

References name.

Referenced by [Object\\_Manager::CheckName\(\)](#), [Object\\_Manager::FindObject\(\)](#), [Object\(\)](#), and [File\\_Writer::Write\\_Object\\_Data\(\)](#).

#### 4.12.3.9 GetNameRef() `std::string & Object::GetNameRef ( )`

Returns reference to the name.

##### Returns

std::string&

Definition at line 135 of file object.cpp.

```
135 { return name; }
```

References name.

Referenced by [Behavior::ClassSetup\(\)](#).

#### 4.12.3.10 GetTemplateName() `std::string Object::GetTemplateName ( ) const`

Returns the name of the template file.

##### Returns

`std::string`

Definition at line 149 of file `object.cpp`.

```
149 { return templateName; }
```

References `templateName`.

Referenced by `Object()`, and `File_Writer::Write_Object_Data()`.

#### 4.12.3.11 Read() `bool Object::Read ( std::string objectFilename )`

Reads object from file. This includes the components of an object.

##### Parameters

<code>objectFilename</code>
-----------------------------

##### Returns

`true`

`false`

Definition at line 158 of file `object.cpp`.

```
158                                     {
159     // Getting data from file
160     File_Reader object_reader;
161     if (!object_reader.Read_File("objects/" + objectFilename)) return false;
162
163     // Reading Behavior component form file
164     Behavior* object_behavior = new Behavior(object_reader);
165     AddComponent(object_behavior);
166
167     // Reading Model component from file
168     Model* object_model = new Model(object_reader);
169     AddComponent(object_model);
170
171     // Reading Physics component from file
172     Physics* object_physics = new Physics(object_reader);
173     AddComponent(object_physics);
174
175     // Reading Transform component from file
176     Transform* object_transform = new Transform(object_reader);
177     AddComponent(object_transform);
178
179     return true;
180 }
```

References `AddComponent()`, and `File_Reader::Read_File()`.

Referenced by `Object_Manager::ReadList()`.

**4.12.3.12 RemoveComponent()** `template<typename T >`

```
void Object::RemoveComponent ( ) [inline]
```

Removes component from object.

**Template Parameters**

<i>T</i>	
----------	--

Definition at line 60 of file object.hpp.

```

60     {
61         // Searching for component using the type (enum as int)
62         auto found = components.find(T::GetType());
63         if (found == components.end()) return;
64         // Delete component
65         delete found->second;
66         found->second = nullptr;
67         // Remove pointer from map
68         components.erase(found->first);
69     }
```

References components.

Referenced by Editor::Display\_Model(), Editor::Display\_Physics(), and Editor::Display\_Scripts().

**4.12.3.13 ReRead()** `bool Object::ReRead (`  
`std::string objectFilename )`

Reading data into object that already exists.

**Parameters**

<i>objectFilename</i>	Name of template file
-----------------------	-----------------------

**Returns**

true

false

Definition at line 189 of file object.cpp.

```

189     {
190         // Getting data from file
191         File_Reader object_reader;
192         if (!object_reader.Read_File("objects/" + objectFilename)) return false;
193
194         if (name.compare("") == 0)
195             SetName(object_reader.Read_String("name"));
196
197         templateName = objectFilename;
198
199         // Reading Model component from file
200         Model* object_model = GetComponent<Model>();
201         if (!object_model) {
202             object_model = new Model;
203             AddComponent(object_model);
204         }
```



```

204     }
205     object_model->Read(object_reader);
206
207     // Reading Physics component from file
208     Physics* object_physics = GetComponent<Physics>();
209     if (!object_physics) {
210         object_physics = new Physics;
211         AddComponent(object_physics);
212     }
213     object_physics->Read(object_reader);
214
215     // Reading Transform component from file
216     Transform* object_transform = GetComponent<Transform>();
217     if (!object_transform) {
218         object_transform = new Transform;
219         AddComponent(object_transform);
220     }
221     object_transform->Read(object_reader);
222
223     // Reading Behavior component form file
224     Behavior* object_behavior = GetComponent<Behavior>();
225     if (object_behavior) object_behavior->Clear();
226     if (!object_behavior) {
227         object_behavior = new Behavior;
228         AddComponent(object_behavior);
229     }
230     object_behavior->Read(object_reader);
231     object_behavior->SetupClassesForLua();
232
233     return true;
234 }

```

References AddComponent(), Behavior::Clear(), name, Behavior::Read(), Model::Read(), Transform::Read(), Physics::Read(), File\_Reader::Read\_File(), File\_Reader::Read\_String(), SetName(), Behavior::SetupClassesForLua(), and templateName.

**4.12.3.14 SetId()** void Object::SetId (   
 int id\_ )

Sets the id of object.

Parameters

$id \leftrightarrow$	Position in <a href="#">Object_Manager</a>
$\_ \leftrightarrow$	

Definition at line 105 of file object.cpp.

```
105 { id = id_; }
```

Referenced by Object\_Manager::RemoveObject().

**4.12.3.15 SetName()** void Object::SetName (   
 std::string name\_ )

Sets name of object.

## Parameters

<i>name</i> ↔	Name of object
—	

Definition at line 119 of file object.cpp.

```
119 {
120     name = Object_Manager::CheckName(name_, id);
121 }
```

References Object\_Manager::CheckName(), and name.

Referenced by Behavior::ClassSetup(), Editor::Display\_Scene(), Object(), Object\_Manager::ReadList(), and ReRead().

**4.12.3.16 SetTemplateName()** void Object::SetTemplateName (   
std::string *templateName\_* )

Sets the name of the template file.

## Parameters

<i>template</i> ↔ <i>Name_</i>	Name of the template file
-----------------------------------	---------------------------

Definition at line 142 of file object.cpp.

```
142 { templateName = templateName_; }
```

References *templateName*.

Referenced by Object().

**4.12.3.17 Update()** void Object::Update ( )

Updates object (only physics for now)

Definition at line 81 of file object.cpp.

```
81 {
82     Behavior* behavior = GetComponent<Behavior>();
83     if (behavior)
84         behavior->Update();
85     Physics* physics = GetComponent<Physics>();
86     if (physics)
87         physics->Update();
88 }
```

References Behavior::Update(), and Physics::Update().

Referenced by Object\_Manager::Update().

#### 4.12.3.18 Write() void Object::Write ( )

Writes the data of the object to a template file.

Definition at line 240 of file object.cpp.

```

240     {
241         File_Writer object_writer;
242         object_writer.Write_String("name", name);
243         templateName = name + ".json";
244
245         Model* object_model = GetComponent<Model>();
246         if (object_model) object_model->Write(object_writer);
247
248         Transform* object_transform = GetComponent<Transform>();
249         if (object_transform) object_transform->Write(object_writer);
250
251         Physics* object_physics = GetComponent<Physics>();
252         if (object_physics) object_physics->Write(object_writer);
253
254         Behavior* object_behavior = GetComponent<Behavior>();
255         if (object_behavior) object_behavior->Write(object_writer);
256
257         object_writer.Write_File(std::string("objects/" + name + ".json"));
258     }

```

References name, templateName, Behavior::Write(), Model::Write(), Transform::Write(), Physics::Write(), File\_Writer::Write\_File(), and File\_Writer::Write\_String().

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

- [object.hpp](#)
- [object.cpp](#)

### 4.13 Object\_Manager Class Reference

```
#include <object_manager.hpp>
```

#### Public Member Functions

- void [ReadList](#) ([File\\_Reader](#) &preset)  
*Reads in objects from a preset list that is given.*

#### Static Public Member Functions

- static bool [Initialize](#) ([File\\_Reader](#) &preset)  
*Initializes the object\_manager object. Reads in objects for the given preset.*
- static void [AddObject](#) ([Object](#) \*object)  
*Adds object to object\_manager.*
- static [Object](#) \* [FindObject](#) (int id)  
*Finds a object using its id (location in object list) giving instant access.*
- static [Object](#) \* [FindObject](#) (std::string objectName)  
*Finds object with the matching name.*
- static unsigned [GetSize](#) ()  
*Gets the size of the object\_manager object list.*

- static void [Update](#) ()  
*Calls the update function for each object in the object list.*
- static void [Shutdown](#) ()  
*Deletes all objects in the manager and then the object manager.*
- static std::string [CheckName](#) (std::string objectName, int id)  
*Checks if the name of the given object is already being used. If it is being used it applies a number to the back.*
- static void [RemoveObject](#) (int id)  
*Removes an object from the object\_manager.*
- static void [Write](#) ([File\\_Writer](#) &writer)  
*Gives all of the object data to writer for output to file.*

### Private Attributes

- std::vector< [Object](#) \* > [objects](#)  
*Current objects being tracked by the engine.*

#### 4.13.1 Detailed Description

[Object\\_Manager](#) class

Definition at line 25 of file `object_manager.hpp`.

#### 4.13.2 Member Function Documentation

**4.13.2.1 AddObject()** void `Object_Manager::AddObject` (  
    [Object](#) \* *object* ) [static]

Adds object to object\_manager.

##### Parameters

<i>object</i>	<a href="#">Object</a> to be added
---------------	------------------------------------

##### Returns

void

Definition at line 52 of file `object_manager.cpp`.

```

52     {
53         // Tells object its location in object_manager object list
54         object->SetId(object_manager->objects.size());
55         object_manager->objects.emplace_back(object);
56     }
```

References `object_manager`, and `objects`.

Referenced by `Editor::Display_Scene()`, and `ReadList()`.

**4.13.2.2 CheckName()** `std::string Object_Manager::CheckName (`  
`std::string objectName,`  
`int id ) [static]`

Checks if the name of the given object is already being used. If it is being used it applies a number to the back.

#### Parameters

<i>objectName</i>	
<i>id</i>	

#### Returns

`std::string`

Definition at line 172 of file `object_manager.cpp`.

```

172 {
173     // Checking if the name matches any other objects
174     int objWithName = 0;
175     for (Object* objToCheck : object_manager->objects) {
176         if (id != -1 && objToCheck->GetId() == id) continue;
177         if (objToCheck->GetName().find(objectName) != std::string::npos)
178             ++objWithName;
179     }
180
181     // Updating the name
182     if (objWithName > 0)
183         return objectName + "_" + std::to_string(objWithName);
184
185     return objectName;
186 }
```

References `Object::GetId()`, `Object::GetName()`, `object_manager`, and `objects`.

Referenced by `Object::SetName()`.

**4.13.2.3 FindObject()** [1/2] `Object * Object_Manager::FindObject (`  
`int id ) [static]`

Finds a object using its `id` (location in object list) giving instant access.

#### Parameters

<i>id</i>	Location of object in <code>object_manager</code> object list
-----------	---

## Returns

Object\*

Definition at line 64 of file object\_manager.cpp.

```

64         {
65     if (id >= (int)object_manager->objects.size()) return nullptr;
66     return object_manager->objects[id];
67 }
```

References object\_manager, and objects.

Referenced by Behavior::ClassSetup(), Editor::Display\_Components(), Editor::Display\_Scene(), Graphics::Render(), Shutdown(), Update(), and Physics::UpdateGravity().

#### 4.13.2.4 FindObject() [2/2] Object \* Object\_Manager::FindObject (std::string objectName) [static]

Finds object with the matching name.

## Parameters

<i>objectName</i>	Name to look for
-------------------	------------------

## Returns

Object\*

Definition at line 75 of file object\_manager.cpp.

```

75         {
76     for (Object* object : object_manager->objects) {
77         if (objectName.compare(object->GetName()) == 0)
78             return object;
79     }
80     return nullptr;
81 }
82 }
```

References Object::GetName(), object\_manager, and objects.

#### 4.13.2.5 GetSize() unsigned Object\_Manager::GetSize ( ) [static]

Gets the size of the object\_manager object list.

## Returns

unsigned Size of object list

Definition at line 89 of file object\_manager.cpp.

```

89 { return object_manager->objects.size(); }
```

References object\_manager, and objects.

Referenced by Editor::Display\_Scene(), Graphics::Render(), and Physics::UpdateGravity().

**4.13.2.6 Initialize()** `bool Object_Manager::Initialize (`  
`File_Reader & preset ) [static]`

Initializes the object\_manager object. Reads in objects for the given preset.

#### Parameters

<i>preset</i>	List of objects for this preset
---------------	---------------------------------

#### Returns

true

false

Definition at line 31 of file object\_manager.cpp.

```

31                                     {
32     // Initializing object_manager
33     object_manager = new Object_Manager;
34     if (!object_manager) {
35         Trace::Message("Object Manager was not initialized.");
36         return false; // Failed to initialize
37     }
38
39     // Adding objects from preset into engine
40     object_manager->objects.reserve(10);
41     object_manager->ReadList(preset);
42
43     return true; // Successful initialization
44 }
```

References Trace::Message(), object\_manager, objects, and ReadList().

Referenced by Engine::Initialize(), and Engine::Restart().

**4.13.2.7 ReadList()** `void Object_Manager::ReadList (`  
`File_Reader & preset )`

Reads in objects from a preset list that is given.

#### Parameters

<i>preset</i>	List of objects to be added
---------------	-----------------------------

Definition at line 127 of file object\_manager.cpp.

```

127                                     {
128     // Track which object we are on
129     unsigned object_num = 0;
130
131     // Reads objects until there is a failed read
132     while (true) {
133         // Getting the name of the objects file
134         std::string object_name = preset.Read_Object_Name("object_" + std::to_string(object_num));
135         std::string template_name = preset.Read_Object_Template_Name("object_" +
136         std::to_string(object_num));
137         if (template_name.compare("") == 0) break;
```

```

137
138     // Constructing the object
139     Object* object = new Object;
140     if (!object->Read(template_name)) {
141         delete object;
142         continue;
143     }
144
145     object->SetName(object_name);
146     object->SetTemplateName(template_name);
147     // Reading in the objects position
148     glm::vec3 position = preset.Read_Object_Position("object_" + std::to_string(object_num));
149     glm::vec3 scale = preset.Read_Object_Scale("object_" + std::to_string(object_num));
150     Transform* transform = object->GetComponent<Transform>();
151     transform->SetPosition(position);
152     transform->SetStartPosition(position);
153     transform->SetScale(scale);
154     Behavior* behavior = object->GetComponent<Behavior>();
155     behavior->SetupClassesForLua();
156
157     // Adding the object to the manager
158     AddObject(object);
159
160     ++object_num;
161 }
162 }

```

References AddObject(), Object::Read(), File\_Reader::Read\_Object\_Name(), File\_Reader::Read\_Object\_Position(), File\_Reader::Read\_Object\_Scale(), File\_Reader::Read\_Object\_Template\_Name(), Object::SetName(), Transform::↵ SetPosition(), Transform::SetScale(), Transform::SetStartPosition(), and Behavior::SetupClassesForLua().

Referenced by Initialize().

**4.13.2.8 RemoveObject()** void Object\_Manager::RemoveObject (int id) [static]

Removes an object from the object\_manager.

#### Parameters

<i>id</i>	id of object to remove
-----------	------------------------

#### Returns

void

Definition at line 194 of file object\_manager.cpp.

```

194     {
195         if (id >= (int)object_manager->objects.size()) return;
196         Object* objectToDelete = object_manager->objects[id];
197
198         // Moves all the objects to the right of one being deleted to the left
199         unsigned offset = 0;
200         for (unsigned objectNum = id + 1; objectNum < object_manager->objects.size(); ++objectNum) {
201             Object* objectToSwitch = object_manager->objects[objectNum];
202             object_manager->objects[id + offset] = objectToSwitch;
203             objectToSwitch->SetId(id + offset++);
204         }
205
206         // Deleting the object
207         delete objectToDelete;
208         objectToDelete = nullptr;

```



```
209     object_manager->objects.pop_back();
210 }
```

References `object_manager`, `objects`, and `Object::SetId()`.

Referenced by `Editor::Display_Scene()`.

#### 4.13.2.9 Shutdown() `void Object_Manager::Shutdown ( ) [static]`

Deletes all objects in the manager and then the object manager.

##### Returns

`void`

Definition at line 107 of file `object_manager.cpp`.

```
107     {
108         if (!object_manager) return; // If the object_manager doesn't exist
109
110         // Deleting each object in the manager
111         for (unsigned i = 0; i < object_manager->objects.size(); ++i) {
112             Object* object = object_manager->FindObject(i);
113             if (object)
114                 delete object;
115         }
116
117         // Deleting the manager
118         delete object_manager;
119         object_manager = nullptr;
120     }
```

References `FindObject()`, `object_manager`, and `objects`.

Referenced by `Engine::Restart()`, and `Engine::Shutdown()`.

#### 4.13.2.10 Update() `void Object_Manager::Update ( ) [static]`

Calls the update function for each object in the object list.

##### Returns

`void`

Definition at line 96 of file `object_manager.cpp`.

```
96     {
97         for (unsigned i = 0; i < object_manager->objects.size(); ++i) {
98             object_manager->FindObject(i)->Update();
99         }
100     }
```

References `FindObject()`, `object_manager`, `objects`, and `Object::Update()`.

Referenced by `Engine::Update()`.

#### 4.13.2.11 Write() `void Object_Manager::Write ( File_Writer & writer ) [static]`

Gives all of the object data to writer for output to file.

## Parameters

<i>writer</i>	
---------------	--

## Returns

void

Definition at line 218 of file `object_manager.cpp`.

```

218 {
219     for (Object* object : object_manager->objects) {
220         writer.Write_Object_Data(object);
221     }
222 }
```

References `object_manager`, `objects`, and `File_Writer::Write_Object_Data()`.

Referenced by `Engine::Write()`.

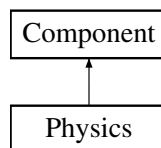
The documentation for this class was generated from the following files:

- [object\\_manager.hpp](#)
- [object\\_manager.cpp](#)

## 4.14 Physics Class Reference

```
#include <physics.hpp>
```

Inheritance diagram for `Physics`:



## Public Member Functions

- [Physics](#) ()  
*Creates [Physics](#) object with default values.*
- [Physics](#) (const [Physics](#) &other)  
*Copy constructor.*
- [Physics](#) ([File\\_Reader](#) &reader)  
*Creates [Physics](#) object using file.*
- [Physics](#) \* [Clone](#) () const  
*Clone [Physics](#) object.*
- void [SetAcceleration](#) (glm::vec3 accel)  
*Sets acceleration of object.*

- glm::vec3 [GetAcceleration](#) () const  
*Returns acceleration of object.*
- glm::vec3 & [GetAccelerationRef](#) ()  
*Returns reference to the acceleration of the object.*
- void [SetForces](#) (glm::vec3 force)  
*Sets forces acting on object.*
- void [AddForce](#) (glm::vec3 force)  
*Adds a force to the current forces acting on the object.*
- glm::vec3 [GetForces](#) () const  
*Returns the forces acting on the object.*
- glm::vec3 & [GetForcesRef](#) ()  
*Returns reference to the forces acting on the object.*
- void [ApplyForce](#) (glm::vec3 direction, float power)  
*Applies force in the given direction using the given power.*
- void [SetVelocity](#) (glm::vec3 vel)  
*Sets the velocity of the object.*
- glm::vec3 [GetVelocity](#) () const  
*Returns the current velocity of the object.*
- glm::vec3 & [GetVelocityRef](#) ()  
*Returns reference to velocity of the object.*
- void [SetRotationalVelocity](#) (glm::vec3 rotVel)  
*Sets rotational velocity.*
- glm::vec3 [GetRotationalVelocity](#) () const  
*Returns rotational velocity.*
- glm::vec3 & [GetRotationalVelocityRef](#) ()  
*Returns reference to rotational velocity.*
- void [SetMass](#) (float ma)  
*Sets the mass of the object.*
- float [GetMass](#) () const  
*Returns the mass of the object.*
- float & [GetMassRef](#) ()  
*Returns reference to mass of the object.*
- void [Update](#) ()  
*Updates the physics of the object.*
- void [UpdateGravity](#) ()  
*Calculates the gravitational pull each object has on each other.*
- void [Read](#) (File\_Reader &reader)  
*Reads data for [Physics](#) object from file.*
- void [Write](#) (File\_Writer &writer)  
*Gives physics data to the writer object.*

### Static Public Member Functions

- static [CType](#) [GetCType](#) ()  
*Gets the CType of [Physics](#) (used in [Object::GetComponent<>\(\)](#))*

## Private Attributes

- glm::vec3 [acceleration](#)  
*Acceleration of object.*
- glm::vec3 [forces](#)  
*Forces acting on object (reset at end of each update)*
- glm::vec3 [velocity](#)  
*Velocity of object.*
- glm::vec3 [initialVelocity](#)  
*Starting velocity.*
- glm::vec3 [initialAcceleration](#)  
*Starting acceleration.*
- glm::vec3 [rotationalVelocity](#)  
*How fast is the object rotating.*
- float [mass](#)  
*Mass of object.*

## Additional Inherited Members

### 4.14.1 Detailed Description

[Physics](#) class

Definition at line 25 of file physics.hpp.

### 4.14.2 Constructor & Destructor Documentation

#### 4.14.2.1 [Physics\(\)](#) [1/3] `Physics::Physics ( )`

Creates [Physics](#) object with default values.

Definition at line 32 of file physics.cpp.

```
32         : Component(CType::CPhysics),
33     acceleration(glm::vec3(0.f, 0.f, 0.f)), forces(glm::vec3(0.f, 0.f, 0.f)),
34     velocity(glm::vec3(0.f, 0.f, 0.f)), rotationalVelocity(glm::vec3(0.f, 0.f, 0.f)), mass(1.f) {}
```

Referenced by [Clone\(\)](#).

#### 4.14.2.2 [Physics\(\)](#) [2/3] `Physics::Physics (`     const [Physics](#) & *other* )

Copy constructor.

#### Parameters

<i>other</i>	Physics object to be copied
--------------	-----------------------------

Definition at line 41 of file physics.cpp.

```
41                                     : Component(CType::CPhysics) {
42     *this = other;
43 }
```

#### 4.14.2.3 Physics() [3/3] Physics::Physics (File\_Reader & reader )

Creates Physics object using file.

#### Parameters

<i>reader</i>	File to use for making physics object
---------------	---------------------------------------

Definition at line 50 of file physics.cpp.

```
50                                     : Component(CType::CPhysics),
51     acceleration(glm::vec3(0.f, 0.f, 0.f)), forces(glm::vec3(0.f, 0.f, 0.f)),
52     velocity(glm::vec3(0.f, 0.f, 0.f)), rotationalVelocity(glm::vec3(0.f, 0.f, 0.f)), mass(1.f) {
53     Read(reader);
54 }
```

References Read().

### 4.14.3 Member Function Documentation

#### 4.14.3.1 AddForce() void Physics::AddForce (glm::vec3 force )

Adds a force to the current forces acting on the object.

#### Parameters

<i>force</i>	
--------------	--

Definition at line 98 of file physics.cpp.

```
98 { forces += force; }
```

References forces.

Referenced by ApplyForce().

**4.14.3.2 ApplyForce()** `void Physics::ApplyForce (`  
`glm::vec3 direction,`  
`float power )`

Applies force in the given direction using the given power.

#### Parameters

<i>direction</i>	
<i>power</i>	

Definition at line 120 of file physics.cpp.

```
120
121     direction = glm::normalize(direction);
122     direction *= power;
123
124     AddForce(direction);
125 }
```

References AddForce().

Referenced by Behavior::ClassSetup().

**4.14.3.3 Clone()** `Physics * Physics::Clone ( ) const`

Clone [Physics](#) object.

#### Returns

Physics\* Cloned [Physics](#) object

Definition at line 61 of file physics.cpp.

```
61     {
62     return new Physics(*this);
63 }
```

References Physics().

**4.14.3.4 GetAcceleration()** `glm::vec3 Physics::GetAcceleration ( ) const`

Returns acceleration of object.

#### Returns

glm::vec3

Definition at line 77 of file physics.cpp.

```
77 { return acceleration; }
```

References acceleration.

#### 4.14.3.5 GetAccelerationRef() `glm::vec3 & Physics::GetAccelerationRef ( )`

Returns reference to the acceleration of the object.

##### Returns

`glm::vec3&`

Definition at line 84 of file physics.cpp.

```
84 { return acceleration; }
```

References acceleration.

Referenced by Behavior::ClassSetup().

#### 4.14.3.6 GetCType() `CType Physics::GetCType ( ) [static]`

Gets the CType of [Physics](#) (used in [Object::GetComponent<>\(\)](#))

##### Returns

`CType`

Definition at line 281 of file physics.cpp.

```
281 {  
282     return CType::CPhysics;  
283 }
```

#### 4.14.3.7 GetForces() `glm::vec3 Physics::GetForces ( ) const`

Returns the forces acting on the object.

##### Returns

`glm::vec3`

Definition at line 105 of file physics.cpp.

```
105 { return forces; }
```

References forces.

**4.14.3.8 GetForcesRef()** `glm::vec3 & Physics::GetForcesRef ( )`

Returns reference to the forces acting on the object.

**Returns**

`glm::vec3&`

Definition at line 112 of file physics.cpp.

```
112 { return forces; }
```

References forces.

Referenced by Behavior::ClassSetup().

**4.14.3.9 GetMass()** `float Physics::GetMass ( ) const`

Returns the mass of the object.

**Returns**

`float`

Definition at line 160 of file physics.cpp.

```
160 { return mass; }
```

References mass.

**4.14.3.10 GetMassRef()** `float & Physics::GetMassRef ( )`

Returns reference to mass of the object.

**Returns**

`float&`

Definition at line 167 of file physics.cpp.

```
167 { return mass; }
```

References mass.

Referenced by Editor::Display\_Physics().



**4.14.3.11 GetRotationalVelocity()** `glm::vec3 Physics::GetRotationalVelocity ( ) const`

Returns rotational velocity.

**Returns**

`glm::vec3`

Definition at line 181 of file physics.cpp.

```
181 { return rotationalVelocity; }
```

References rotationalVelocity.

**4.14.3.12 GetRotationalVelocityRef()** `glm::vec3 & Physics::GetRotationalVelocityRef ( )`

Returns reference to rotational velocity.

**Returns**

`glm::vec3&`

Definition at line 188 of file physics.cpp.

```
188 { return rotationalVelocity; }
```

References rotationalVelocity.

Referenced by Editor::Display\_Physics().

**4.14.3.13 GetVelocity()** `glm::vec3 Physics::GetVelocity ( ) const`

Returns the current velocity of the object.

**Returns**

`glm::vec3`

Definition at line 139 of file physics.cpp.

```
139 { return velocity; }
```

References velocity.

**4.14.3.14 GetVelocityRef()** `glm::vec3 & Physics::GetVelocityRef ( )`

Returns reference to velocity of the object.

**Returns**

`glm::vec3&`

Definition at line 146 of file physics.cpp.

```
146 { return velocity; }
```

References velocity.

Referenced by Behavior::ClassSetup(), and Editor::Display\_Physics().

**4.14.3.15 Read()** `void Physics::Read (   
File_Reader & reader )`

Reads data for [Physics](#) object from file.

**Parameters**

<i>reader</i>	File to be read from
---------------	----------------------

Definition at line 257 of file physics.cpp.

```
257 {
258     initialAcceleration = reader.Read_Vec3("acceleration");
259     initialVelocity = reader.Read_Vec3("velocity");
260     SetAcceleration(initialAcceleration);
261     SetVelocity(initialVelocity);
262     SetMass(reader.Read_Float("mass"));
263 }
```

References initialAcceleration, initialVelocity, File\_Reader::Read\_Float(), File\_Reader::Read\_Vec3(), SetAcceleration(), SetMass(), and SetVelocity().

Referenced by Physics(), and Object::ReRead().

**4.14.3.16 SetAcceleration()** `void Physics::SetAcceleration (   
glm::vec3 accel )`

Sets acceleration of object.

**Parameters**

<i>accel</i>	
--------------	--

Definition at line 70 of file physics.cpp.

```
70 { acceleration = accel; }
```

References acceleration.

Referenced by Behavior::ClassSetup(), and Read().

**4.14.3.17 SetForces()** `void Physics::SetForces (`  
`glm::vec3 force )`

Sets forces acting on object.

Parameters

<i>force</i>	
--------------	--

Definition at line 91 of file physics.cpp.

```
91 { forces = force; }
```

References forces.

Referenced by Behavior::ClassSetup().

**4.14.3.18 SetMass()** `void Physics::SetMass (`  
`float ma )`

Sets the mass of the object.

Parameters

<i>ma</i>	
-----------	--

Definition at line 153 of file physics.cpp.

```
153 { mass = ma; }
```

References mass.

Referenced by Read().

**4.14.3.19 SetRotationalVelocity()** `void Physics::SetRotationalVelocity (`  
`glm::vec3 rotVel )`

Sets rotational velocity.

## Parameters

<i>rotVel</i>	New rotational velocity
---------------	-------------------------

Definition at line 174 of file physics.cpp.

```
174 { rotationalVelocity = rotVel; }
```

References rotationalVelocity.

**4.14.3.20 SetVelocity()** void Physics::SetVelocity (   
 glm::vec3 vel )

Sets the velocity of the object.

## Parameters

<i>vel</i>	
------------	--

Definition at line 132 of file physics.cpp.

```
132 { velocity = vel; }
```

References velocity.

Referenced by Behavior::ClassSetup(), and Read().

**4.14.3.21 Update()** void Physics::Update ( )

Updates the physics of the object.

Definition at line 194 of file physics.cpp.

```
194 {
195     // Finding the acceleration of the object using F=ma
196     acceleration = forces / mass;
197
198     // Updating velocity
199     velocity += (acceleration * Engine::GetDt());
200
201     // Updating position
202     Transform* transform = GetParent()->GetComponent<Transform>();
203     glm::vec3 position = transform->GetPosition();
204     transform->SetOldPosition(position);
205     position = (velocity * Engine::GetDt()) + position;
206     transform->SetPosition(position);
207
208     // Updating rotation
209     glm::vec3 rotation = transform->GetRotation();
210     rotation = (rotationalVelocity * Engine::GetDt()) + rotation;
211     transform->SetRotation(rotation);
212
213     // Resetting the forces acting on the object
214     forces = glm::vec3(0.f, 0.f, 0.f);
215 }
```

References `acceleration`, `forces`, `Object::GetComponent()`, `Engine::GetDt()`, `Component::GetParent()`, `Transform::GetPosition()`, `Transform::GetRotation()`, `mass`, `rotationalVelocity`, `Transform::SetOldPosition()`, `Transform::SetPosition()`, `Transform::SetRotation()`, and `velocity`.

Referenced by `Object::Update()`.

#### 4.14.3.22 UpdateGravity() `void Physics::UpdateGravity ( )`

Calculates the gravitational pull each object has on each other.

Definition at line 221 of file `physics.cpp`.

```

221         {
222             // Gets the needed components for the current object
223             Object* object = GetParent();
224             Transform* transform = object->GetComponent<Transform>();
225             Physics* physics = object->GetComponent<Physics>();
226             glm::vec3 position = transform->GetPosition();
227
228             // For each object
229             for (unsigned i = 0; i < Object_Manager::GetSize(); ++i) {
230                 if ((int)i == object->GetId()) continue;
231                 // Gets needed components for the object being checked
232                 Object* other = Object_Manager::FindObject(i);
233                 Physics* otherPhysics = other->GetComponent<Physics>();
234                 Transform* otherTransform = other->GetComponent<Transform>();
235                 glm::vec3 otherPosition = otherTransform->GetPosition();
236                 // Finding the distance between the objects
237                 double distance = sqrt(pow(double(otherPosition.x - position.x), 2.0) +
238                                     pow(double(otherPosition.y - position.y), 2.0) +
239                                     pow(double(otherPosition.z - position.z), 2.0));
240                 // Calculating the force the objects apply on each other
241                 double magnitude = Engine::GetGravConst() * ((physics->mass * otherPhysics->mass)) / pow(distance,
242                 2.0);
243                 // Getting the direction (normalized)
244                 glm::vec3 direction = otherPosition - position;
245                 glm::vec3 normDirection = glm::normalize(direction);
246                 // Applying gravitational force to normalized direction
247                 glm::vec3 force = normDirection * float(magnitude);
248                 // Adding the gravitational force to the forces on object
249                 physics->AddForce(force);
250             }
251         }

```

References `Object_Manager::FindObject()`, `Object::GetComponent()`, `Engine::GetGravConst()`, `Component::GetParent()`, `Transform::GetPosition()`, `Object_Manager::GetSize()`, and `mass`.

Referenced by `Behavior::ClassSetup()`.

#### 4.14.3.23 Write() `void Physics::Write ( File_Writer & writer )`

Gives physics data to the writer object.

Parameters

<code>writer</code>	
---------------------	--

Definition at line 270 of file physics.cpp.

```
270     {
271     writer.Write_Vec3("acceleration", initialAcceleration);
272     writer.Write_Vec3("velocity", initialVelocity);
273     writer.Write_Value("mass", mass);
274 }
```

References `initialAcceleration`, `initialVelocity`, `mass`, `File_Writer::Write_Value()`, and `File_Writer::Write_Vec3()`.

Referenced by `Object::Write()`.

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

- [physics.hpp](#)
- [physics.cpp](#)

## 4.15 Random Class Reference

```
#include <random.hpp>
```

### Static Public Member Functions

- static bool [Initialize](#) ()  
*Initializes the random system.*
- static void [Shutdown](#) ()  
*Delete the random object.*
- static glm::vec3 [random\\_vec3](#) (float low, float high)  
*Creates a random vec3.*
- static float [random\\_float](#) (float low, float high)  
*Creates random float.*

### Private Attributes

- std::random\_device [rd](#)  
*Random device.*

#### 4.15.1 Detailed Description

[Random](#) class

Definition at line 23 of file random.hpp.

#### 4.15.2 Member Function Documentation

#### 4.15.2.1 Initialize() `bool Random::Initialize ( ) [static]`

Initializes the random system.

##### Returns

true  
false

Definition at line 24 of file random.cpp.

```

24         {
25         // Initializing random
26         random = new Random;
27         if (!random) {
28             Trace::Message("Random failed to initialize.");
29             return false;
30         }
31
32         return true;
33     }
```

References Trace::Message(), and random.

Referenced by Engine::Initialize().

#### 4.15.2.2 random\_float() `float Random::random_float ( float low, float high ) [static]`

Creates random float.

##### Parameters

<i>low</i>	Lower boundary in random gen
<i>high</i>	Upper boundary in random gen

##### Returns

float

Definition at line 70 of file random.cpp.

```

70         {
71         // Setup random gen
72         std::mt19937 gen(random->rd());
73         std::uniform_real_distribution<> dist(low, high);
74         // Gen random float
75         return dist(gen);
76     }
```

References random, and rd.

Referenced by Behavior::ClassSetup().

**4.15.2.3 random\_vec3()** glm::vec3 Random::random\_vec3 (   
float *low*,   
float *high* ) [static]

Creates a random vec3.

#### Parameters

<i>low</i>	Lower boundary in random gen
<i>high</i>	Upper boundary in random gen

#### Returns

vec3

Definition at line 54 of file random.cpp.

```

54                                     {
55     // Setup random gen
56     std::mt19937 gen(random->rd());
57     std::uniform_real_distribution<> dist(low, high);
58     // Gen random vec3
59     glm::vec3 result_vec3 = { dist(gen), dist(gen), dist(gen) };
60     return result_vec3;
61 }
```

References random, and rd.

Referenced by Behavior::ClassSetup().

**4.15.2.4 Shutdown()** void Random::Shutdown ( ) [static]

Delete the random object.

#### Returns

void

Definition at line 40 of file random.cpp.

```

40     {
41     if (!random) return;
42
43     delete random;
44     random = nullptr;
45 }
```

References random.

Referenced by Engine::Shutdown().

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

- [random.hpp](#)
- [random.cpp](#)



## 4.16 Shader Class Reference

```
#include <shader.hpp>
```

### Static Public Member Functions

- static bool [Initialize](#) ([File\\_Reader](#) &settings)  
*Initializes the shader object.*
- static void [Update](#) ()  
*Tells program to use shader.*
- static void [Shutdown](#) ()  
*Shutdown shader.*
- static std::string [ReadFile](#) (std::string filename)  
*Reads shader file into std::string.*
- static void [LoadShader](#) (std::string vertexPath, std::string fragmentPath)  
*Loads the vertex and fragment shader using given filepaths.*
- static GLuint [GetProgram](#) ()  
*Returns the program id.*
- static GLuint [GetMatrixId](#) ()  
*Returns the mvp buffer id.*
- static GLuint [GetViewMatrixId](#) ()  
*Returns the view matrix buffer id.*
- static GLuint [GetModelMatrixId](#) ()  
*Returns the model matrix buffer id.*
- static GLuint [GetLightId](#) ()  
*Returns the light pos buffer id.*
- static GLuint [GetLightPowerId](#) ()  
*Returns the light power buffer id.*

### Private Attributes

- GLuint [program](#)  
*Program id for the engine.*
- GLuint [matrixId](#)  
*MVP matrix id.*
- GLuint [viewMatrixId](#)  
*View matrix id.*
- GLuint [modelMatrixId](#)  
*Model matrix id.*
- GLuint [lightId](#)  
*Light id for world.*
- GLuint [lightPowerId](#)  
*Id for light power buffer.*

### 4.16.1 Detailed Description

[Shader](#) class

Definition at line 26 of file shader.hpp.

### 4.16.2 Member Function Documentation

#### 4.16.2.1 `GetLightId()` `GLuint Shader::GetLightId ( ) [static]`

Returns the light pos buffer id.

##### Returns

GLuint

Definition at line 173 of file shader.cpp.

```
173 { return shader->lightId; }
```

References `lightId`, and `shader`.

Referenced by `Model_Data::Draw()`.

#### 4.16.2.2 `GetLightPowerId()` `GLuint Shader::GetLightPowerId ( ) [static]`

Returns the light power buffer id.

##### Returns

GLuint

Definition at line 180 of file shader.cpp.

```
180 { return shader->lightPowerId; }
```

References `lightPowerId`, and `shader`.

Referenced by `Model_Data::Draw()`.

**4.16.2.3 GetMatrixId()** `GLuint Shader::GetMatrixId ( ) [static]`

Returns the mvp buffer id.

**Returns**

GLuint

Definition at line 152 of file shader.cpp.

```
152 { return shader->matrixId; }
```

References matrixId, and shader.

Referenced by Model\_Data::Draw().

**4.16.2.4 GetModelMatrixId()** `GLuint Shader::GetModelMatrixId ( ) [static]`

Returns the model matrix buffer id.

**Returns**

GLuint

Definition at line 166 of file shader.cpp.

```
166 { return shader->modelMatrixId; }
```

References modelMatrixId, and shader.

Referenced by Model\_Data::Draw().

**4.16.2.5 GetProgram()** `GLuint Shader::GetProgram ( ) [static]`

Returns the program id.

**Returns**

GLuint

Definition at line 145 of file shader.cpp.

```
145 { return shader->program; }
```

References program, and shader.

Referenced by Texture::Load().

#### 4.16.2.6 GetViewMatrixId() `GLuint Shader::GetViewMatrixId ( ) [static]`

Returns the view matrix buffer id.

Returns

GLuint

Definition at line 159 of file `shader.cpp`.

```
159 { return shader->viewMatrixId; }
```

References `shader`, and `viewMatrixId`.

Referenced by `Model_Data::Draw()`.

#### 4.16.2.7 Initialize() `bool Shader::Initialize ( File_Reader & settings ) [static]`

Initializes the shader object.

Parameters

<i>settings</i>	<code>File_Reader</code> object that contains name of shaders to use
-----------------	--

Returns

true

false

Definition at line 31 of file `shader.cpp`.

```
31 {
32     shader = new Shader;
33     if (!shader) {
34         Trace::Message("Shader failed to initialize.\n");
35         return false;
36     }
37
38     //LoadShader("src/shaders/vertex.glsl", "src/shaders/fragment.glsl");
39     LoadShader(std::string(getenv("USERPROFILE")) + "/Documents/pEngine/shaders/" +
40               settings.Read_String("vertexShader") + ".glsl",
41               std::string(getenv("USERPROFILE")) + "/Documents/pEngine/shaders/" +
42               settings.Read_String("fragShader") + ".glsl");
41     return true;
42 }
```

References `LoadShader()`, `Trace::Message()`, `File_Reader::Read_String()`, and `shader`.

Referenced by `Graphics::Initialize()`.

#### 4.16.2.8 LoadShader() `void Shader::LoadShader ( std::string vertexPath, std::string fragmentPath ) [static]`

Loads the vertex and fragment shader using given filepaths.

## Parameters

<i>vertexPath</i>	// Vertex shader filepath
<i>fragmentPath</i>	// Fragment shader filepath

## Returns

void

Definition at line 102 of file shader.cpp.

```

102
103     // Creating shaders
104     GLuint vertShader = glCreateShader(GL_VERTEX_SHADER);
105     GLuint fragShader = glCreateShader(GL_FRAGMENT_SHADER);
106
107     // Reading shaders
108     std::string vertShaderStr = ReadFile(vertexPath);
109     std::string fragShaderStr = ReadFile(fragmentPath);
110     const char *vertShaderSrc = vertShaderStr.c_str();
111     const char *fragShaderSrc = fragShaderStr.c_str();
112
113     // Compiling shaders
114     glShaderSource(vertShader, 1, &vertShaderSrc, nullptr);
115     glCompileShader(vertShader);
116
117     glShaderSource(fragShader, 1, &fragShaderSrc, nullptr);
118     glCompileShader(fragShader);
119
120     // Attaching shaders to engine
121     shader->program = glCreateProgram();
122     glAttachShader(shader->program, vertShader);
123     glAttachShader(shader->program, fragShader);
124
125     // Cleanup
126     glDeleteShader(vertShader);
127     glDeleteShader(fragShader);
128
129     // Setting up program
130     glLinkProgram(shader->program);
131     glUseProgram(shader->program);
132
133     shader->matrixId = glGetUniformLocation(shader->program, "MVP");
134     shader->viewMatrixId = glGetUniformLocation(shader->program, "V");
135     shader->modelMatrixId = glGetUniformLocation(shader->program, "M");
136     shader->lightId = glGetUniformLocation(shader->program, "LightPosition_worldspace");
137     shader->lightPowerId = glGetUniformLocation(shader->program, "LightPower");
138 }

```

References lightId, lightPowerId, matrixId, modelMatrixId, program, ReadFile(), shader, and viewMatrixId.

Referenced by Initialize().

**4.16.2.9 ReadFile()** std::string Shader::ReadFile (  
std::string filepath) [static]

Reads shader file into std::string.

## Parameters

<i>filepath</i>	Shader file
-----------------	-------------

**Returns**

std::string

Definition at line 73 of file shader.cpp.

```
73 {
74     std::string content;
75
76     // Opening the shader file
77     std::ifstream file(filepath.c_str(), std::ios::in);
78     if (!file.is_open()) {
79         Trace::Message("Failed to read file: " + filepath + "\n");
80         return "";
81     }
82
83     // Saving shader file into std::string
84     std::string line = "";
85     while (!file.eof()) {
86         getline(file, line);
87         content.append(line + "\n");
88     }
89
90     // Closing file and returning std::string
91     file.close();
92     return content;
93 }
```

References Trace::Message().

Referenced by LoadShader().

**4.16.2.10 Shutdown()** void Shader::Shutdown ( ) [static]

Shutdown shader.

**Returns**

void

Definition at line 58 of file shader.cpp.

```
58 {
59     if (!shader) return;
60
61     glDeleteProgram(shader->program);
62
63     delete shader;
64     shader = nullptr;
65 }
```

References program, and shader.

Referenced by Graphics::Shutdown().

#### 4.16.2.11 Update() `void Shader::Update ( ) [static]`

Tells program to use shader.

##### Returns

void

Definition at line 49 of file `shader.cpp`.

```
49      {  
50          glUseProgram(shader->program);  
51      }
```

References `program`, and `shader`.

Referenced by `Graphics::Render()`.

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

- [shader.hpp](#)
- [shader.cpp](#)

## 4.17 Texture Class Reference

```
#include <texture.hpp>
```

### Public Member Functions

- [~Texture](#) ()  
*Deletes texture data.*
- bool [Load](#) (std::string textureName\_)  
*Loads in texture with given filename.*
- void [Display](#) ()  
*Setup texture for drawing.*
- std::string [GetTextureName](#) () const  
*Returns texture name.*
- GLuint [GetTextureNum](#) () const  
*Returns texture data id.*

### Static Private Member Functions

- static GLuint [LoadDDS](#) (FILE \*fp)  
*Loads in the given dds file.*

### Private Attributes

- std::string [textureName](#)  
*Name of texture.*
- GLuint [textureNum](#)  
*Loaded texture data id.*
- GLuint [textureId](#)  
*Textures buffer id.*
- bool [hasBeenSet](#)  
*Whether there is a texture or not.*

### 4.17.1 Detailed Description

[Texture](#) class

Definition at line 23 of file texture.hpp.

### 4.17.2 Constructor & Destructor Documentation

#### 4.17.2.1 [~Texture\(\)](#) `Texture::~Texture ( )`

Deletes texture data.

Definition at line 24 of file texture.cpp.

```
24     {  
25         glDeleteTextures(1, &textureNum);  
26     }
```

References [textureNum](#).

### 4.17.3 Member Function Documentation

#### 4.17.3.1 [Display\(\)](#) `void Texture::Display ( )`

Setup texture for drawing.

Definition at line 55 of file texture.cpp.

```
55     {  
56         if (!hasBeenSet) return;  
57  
58         glActiveTexture(GL_TEXTURE0);  
59         glBindTexture(GL_TEXTURE_2D, textureNum);  
60         glUniform1i(textureId, 0);  
61     }
```

References [hasBeenSet](#), [textureId](#), and [textureNum](#).

Referenced by [Model\\_Data::Draw\(\)](#).



#### 4.17.3.2 GetTextureName() `std::string Texture::GetTextureName ( ) const`

Returns texture name.

##### Returns

`std::string`

Definition at line 68 of file texture.cpp.

```
68 { return textureName; }
```

References textureName.

Referenced by Texture\_Manager::Get(), Model::GetTextureName(), and Model::Write().

#### 4.17.3.3 GetTextureNum() `GLuint Texture::GetTextureNum ( ) const`

Returns texture data id.

##### Returns

`GLuint`

Definition at line 75 of file texture.cpp.

```
75 { return textureNum; }
```

References textureNum.

#### 4.17.3.4 Load() `bool Texture::Load ( std::string textureName_ )`

Loads in texture with given filename.

##### Parameters

<i>textureName_</i>	Filename of texture
---------------------	---------------------

##### Returns

`true`

`false`

Definition at line 35 of file texture.cpp.

```

35                                     {
36     FILE *fp;
37     std::string filename = std::string(getenv("USERPROFILE")) + "/Documents/pEngine/textures/" +
        textureName_;
38
39     // Opening the file
40     fp = fopen(filename.c_str(), "rb");
41     if (!fp) return false;
42
43     textureNum = Texture::LoadDDS(fp);
44     textureName = textureName_;
45     textureId = glGetUniformLocation(Shader::GetProgram(), "myTextureSampler");
46     hasBeenSet = true;
47
48     return true;
49 }

```

References Shader::GetProgram(), hasBeenSet, LoadDDS(), textureId, textureName, and textureNum.

Referenced by Texture\_Manager::Get().

**4.17.3.5 LoadDDS()** GLuint Texture::LoadDDS (   
 FILE \* *fp* ) [static], [private]

Loads in the given dds file.

#### Parameters

<i>fp</i>	The file stream
-----------	-----------------

#### Returns

GLuint

Definition at line 86 of file texture.cpp.

```

86     {
87         unsigned char header[124];
88
89         // Making sure it is a dds
90         char filecode[4];
91         fread(filecode, 1, 4, fp);
92         if (strncmp(filecode, "DDS ", 4) != 0) {
93             fclose(fp);
94             return 0;
95         }
96
97         // Getting the surface description
98         fread(&header, 124, 1, fp);
99
100        unsigned int height      = *(unsigned int*)&(header[8 ]);
101        unsigned int width       = *(unsigned int*)&(header[12]);
102        unsigned int linearSize  = *(unsigned int*)&(header[16]);
103        unsigned int mipMapCount = *(unsigned int*)&(header[24]);
104        unsigned int fourCC      = *(unsigned int*)&(header[80]);
105
106        unsigned char * buffer;
107        unsigned int bufsize;
108
109        bufsize = mipMapCount > 1 ? linearSize * 2 : linearSize;
110        buffer = (unsigned char*)malloc(bufsize * sizeof(unsigned char));
111        fread(buffer, 1, bufsize, fp);
112
113        // Close the file

```

```

114     fclose(fp);
115
116     unsigned int format;
117     switch(fourCC) {
118         case FOURCC_DXT1:
119             format = GL_COMPRESSED_RGBA_S3TC_DXT1_EXT;
120             break;
121         case FOURCC_DXT3:
122             format = GL_COMPRESSED_RGBA_S3TC_DXT3_EXT;
123             break;
124         case FOURCC_DXT5:
125             format = GL_COMPRESSED_RGBA_S3TC_DXT5_EXT;
126             break;
127         default:
128             free(buffer);
129             return 0;
130     }
131
132     GLuint textureID;
133     glGenTextures(1, &textureID);
134
135     glBindTexture(GL_TEXTURE_2D, textureID);
136     glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
137
138     unsigned int blockSize = (format == GL_COMPRESSED_RGBA_S3TC_DXT1_EXT) ? 8 : 16;
139     unsigned int offset = 0;
140
141     for (unsigned int level = 0; level < mipMapCount && (width || height); ++level) {
142         unsigned int size = ((width+3)/4)*((height+3)/4)*blockSize;
143         glCompressedTexImage2D(GL_TEXTURE_2D, level, format, width, height,
144             0, size, buffer + offset);
145
146         offset += size;
147         width /= 2;
148         height /= 2;
149
150         if(width < 1) width = 1;
151         if(height < 1) height = 1;
152     }
153
154     free(buffer);
155
156     return textureID;
157 }
158 }

```

References `FOURCC_DXT1`, `FOURCC_DXT3`, and `FOURCC_DXT5`.

Referenced by `Load()`.

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

- [texture.hpp](#)
- [texture.cpp](#)

## 4.18 Texture\_Manager Class Reference

```
#include <texture_manager.hpp>
```

### Static Public Member Functions

- static bool [Initialize](#) ()  
*Initializes the texture\_manager.*
- static [Texture](#) \* [Get](#) ([File\\_Reader](#) &reader)

*Looks for texture in list of loaded textures. If found it returns a pointer. If not found it creates texture, adds it to the list of textures and returns a pointer to it.*

- static [Texture](#) \* [Get](#) (std::string textureName)

*Looks for texture in list of loaded textures. If found it returns a pointer. If not found it creates texture, adds it to the list of textures and returns a pointer to it.*

- static void [Shutdown](#) ()

*Deletes all texture object and then the manager.*

### Private Attributes

- std::vector< [Texture](#) \* > [textures](#)

*List of loaded textures.*

### 4.18.1 Detailed Description

[Texture\\_Manager](#) class

Definition at line 25 of file texture\_manager.hpp.

### 4.18.2 Member Function Documentation

#### 4.18.2.1 [Get\(\)](#) [1/2] [Texture](#) \* [Texture\\_Manager::Get](#) ( [File\\_Reader](#) & *reader* ) [static]

Looks for texture in list of loaded textures. If found it returns a pointer. If not found it creates texture, adds it to the list of textures and returns a pointer to it.

#### Parameters

<i>reader</i>	<a href="#">File_Reader</a> object that contains name of texture
---------------	--

#### Returns

[Texture](#)\*

Definition at line 45 of file texture\_manager.cpp.

```

45                                     {
46     // Getting texture's filename
47     std::string filename = reader.Read_String("textureToLoad");
48     // Looking for texture in list of loaded textures
49     for (Texture* texture : texture_manager->textures) {
50         if (texture->GetTextureName().compare(filename) == 0) {
51             return texture;
52         }
53     }
54 }
```

```
55     // Creating new texture
56     Texture* texture = new Texture;
57     texture->Load(filename);
58     texture_manager->textures.emplace_back(texture);
59
60     return texture;
61 }
```

References Texture::GetTextureName(), Texture::Load(), File\_Reader::Read\_String(), texture\_manager, and textures.

Referenced by Model::Load(), and Model::SwitchTexture().

**4.18.2.2 Get()** [2/2] `Texture * Texture_Manager::Get (`  
`std::string textureName ) [static]`

Looks for texture in list of loaded textures. If found it returns a pointer. If not found it creates texture, adds it to the list of textures and returns a pointer to it.

#### Parameters

<i>textureName</i>	Name of texture
--------------------	-----------------

#### Returns

Texture\*

Definition at line 71 of file texture\_manager.cpp.

```
71     {
72     // Looking for texture in list of loaded textures
73     for (Texture* texture : texture_manager->textures) {
74         if (texture->GetTextureName().compare(textureName) == 0) {
75             return texture;
76         }
77     }
78
79     // Creating new texture
80     Texture* texture = new Texture;
81     if (!texture->Load(textureName)) {
82         delete texture;
83         return nullptr;
84     }
85     texture_manager->textures.emplace_back(texture);
86
87     return texture;
88 }
```

References Texture::GetTextureName(), Texture::Load(), texture\_manager, and textures.

**4.18.2.3 Initialize()** `bool Texture_Manager::Initialize ( ) [static]`

Initializes the texture\_manager.

**Returns**

true  
false

Definition at line 24 of file texture\_manager.cpp.

```
24     {  
25         // Initializing texture_manager  
26         texture_manager = new Texture_Manager;  
27         if (!texture_manager) {  
28             Trace::Message("Texture Manager was not initialized.\n");  
29             return false;  
30         }  
31  
32         // Reserving space in the texture_manager  
33         texture_manager->textures.reserve(10);  
34         return true;  
35     }
```

References `Trace::Message()`, `texture_manager`, and `textures`.

Referenced by `Engine::Initialize()`.

**4.18.2.4 Shutdown()** `void Texture_Manager::Shutdown ( ) [static]`

Deletes all texture object and then the manager.

**Returns**

void

Definition at line 95 of file texture\_manager.cpp.

```
95     {  
96         if (!texture_manager) return;  
97  
98         for (Texture* texture : texture_manager->textures) {  
99             if (!texture) continue;  
100  
101             delete texture;  
102             texture = nullptr;  
103         }  
104  
105         delete texture_manager;  
106         texture_manager = nullptr;  
107     }
```

References `texture_manager`, and `textures`.

Referenced by `Engine::Shutdown()`.

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

- [texture\\_manager.hpp](#)
- [texture\\_manager.cpp](#)

## 4.19 Trace Class Reference

```
#include <trace.hpp>
```

### Static Public Member Functions

- static void [Initialize](#) ()  
*Initializes the trace system.*
- static void [Message](#) (std::string message)  
*Prints a message into the output file.*
- static void [Shutdown](#) ()  
*Closes output file and deletes trace object.*

### Private Attributes

- std::fstream [trace\\_stream](#)  
*Output file.*

### 4.19.1 Detailed Description

[Trace](#) class

Definition at line 21 of file trace.hpp.

### 4.19.2 Member Function Documentation

#### 4.19.2.1 [Initialize\(\)](#) void Trace::Initialize ( ) [static]

Initializes the trace system.

#### Returns

void

Definition at line 26 of file trace.cpp.

```
26         {  
27     trace = new Trace;  
28  
29     // Opens output file  
30     trace->trace_stream.open(std::string(getenv("USERPROFILE")) + "/Documents/pEngine/trace.log",  
std::ofstream::out);  
31     if (!trace->trace_stream) std::cout << "Trace file wasn't opened successfully.\n";  
32 }
```

References [trace](#), and [trace\\_stream](#).

Referenced by [main\(\)](#).

#### 4.19.2.2 [Message\(\)](#) void Trace::Message (std::string message) [static]

Prints a message into the output file.

## Parameters

<i>message</i>	Message to be printed
----------------	-----------------------

## Returns

void

Definition at line 40 of file trace.cpp.

```
40 {  
41     if (!trace->trace_stream) return;  
42  
43     trace->trace_stream << message;  
44     std::cout << message;  
45 }
```

References trace, and trace\_stream.

Referenced by Graphics::ErrorCallback(), Graphics::ErrorCheck(), Random::Initialize(), Engine::Initialize(), ModelData\_Manager::Initialize(), Object\_Manager::Initialize(), Texture\_Manager::Initialize(), Editor::Initialize(), Shader::Initialize(), Camera::Initialize(), Graphics::Initialize(), Model\_Data::Read(), and Shader::ReadFile().

#### 4.19.2.3 Shutdown() void Trace::Shutdown ( ) [static]

Closes output file and deletes trace object.

## Returns

void

Definition at line 52 of file trace.cpp.

```
52 {  
53     // Closing output file  
54     if (trace->trace_stream) trace->trace_stream.close();  
55  
56     delete trace;  
57     trace = nullptr;  
58 }
```

References trace, and trace\_stream.

Referenced by main().

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

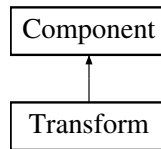
- [trace.hpp](#)
- [trace.cpp](#)



## 4.20 Transform Class Reference

```
#include <transform.hpp>
```

Inheritance diagram for Transform:



### Public Member Functions

- [Transform](#) ()  
*Creates [Transform](#) object with default values.*
- [Transform](#) (const [Transform](#) &other)  
*Copy constructor.*
- [Transform](#) ([File\\_Reader](#) &reader)  
*Creates [Transform](#) object using file.*
- [Transform](#) \* [Clone](#) () const  
*Clones current [Transform](#) object.*
- void [SetPosition](#) (glm::vec3 pos)  
*Sets position of object.*
- glm::vec3 [GetPosition](#) () const  
*Returns position of object.*
- glm::vec3 & [GetPositionRef](#) ()  
*Returns position reference of object.*
- void [SetOldPosition](#) (glm::vec3 oldPos)  
*Sets old position of object.*
- glm::vec3 [GetOldPosition](#) () const  
*Returns old position of object.*
- void [SetScale](#) (glm::vec3 sca)  
*Sets scale of object.*
- glm::vec3 [GetScale](#) () const  
*Returns scale of object.*
- glm::vec3 & [GetScaleRef](#) ()  
*Returns scale reference of object.*
- void [SetRotation](#) (glm::vec3 rot)  
*Sets rotation of object.*
- glm::vec3 [GetRotation](#) () const  
*Returns rotation of object.*
- glm::vec3 & [GetRotationRef](#) ()  
*Returns rotation reference of object.*
- void [SetStartPosition](#) (glm::vec3 startPosition\_)  
*Sets the start position of the object.*
- glm::vec3 [GetStartPosition](#) () const

- Returns the saved start position of the object.*
  - glm::vec3 & [GetStartPositionRef](#) ()
  - Returns a reference to the start position of the object.*
- void [Read](#) ([File\\_Reader](#) &reader)
- Reads data for [Transform](#) object from file.*
- void [Write](#) ([File\\_Writer](#) &writer)
- Gives transform data to writer object.*

### Static Public Member Functions

- static [CType](#) [GetCType](#) ()
- Gets the CType of [Transform](#) (used in [Object::GetComponent<>\(\)](#))*

### Private Attributes

- glm::vec3 [position](#)
- Position of object.*
- glm::vec3 [oldPosition](#)
- Previous position of object.*
- glm::vec3 [scale](#)
- Scale of object.*
- glm::vec3 [rotation](#)
- Rotation of object.*
- glm::vec3 [startPosition](#)
- Starting position of the object.*

### Additional Inherited Members

#### 4.20.1 Detailed Description

[Transform](#) class

Definition at line 25 of file [transform.hpp](#).

#### 4.20.2 Constructor & Destructor Documentation

##### 4.20.2.1 [Transform\(\)](#) [1/3] [Transform::Transform](#) ( )

Creates [Transform](#) object with default values.

Definition at line 19 of file [transform.cpp](#).

```
19         : Component(CType::CTransform),
20     position(glm::vec3(0.f, 0.f, 0.f)), scale(glm::vec3(1.f, 1.f, 1.f)), rotation(glm::vec3(0.f, 0.f, 0.f))
    {}
```

Referenced by [Clone\(\)](#).

##### 4.20.2.2 [Transform\(\)](#) [2/3] [Transform::Transform](#) (const [Transform](#) & other )

Copy constructor.

#### Parameters

<i>other</i>	
--------------	--

Definition at line 27 of file transform.cpp.

```
27                                     : Component (CType::CTransform) {
28     *this = other;
29 }
```

#### 4.20.2.3 Transform() [3/3] [Transform](#)::[Transform](#) ([File\\_Reader](#) & [reader](#) )

Creates [Transform](#) object using file.

#### Parameters

<i>reader</i>	File to use for making <a href="#">Transform</a> object
---------------	---

Definition at line 36 of file transform.cpp.

```
36                                     : Component (CType::CTransform),
37     position (glm::vec3(0.f, 0.f, 0.f)), scale (glm::vec3(1.f, 1.f, 1.f)), rotation (glm::vec3(0.f, 0.f, 0.f)) {
38     Read (reader);
39 }
```

References [Read](#)().

### 4.20.3 Member Function Documentation

#### 4.20.3.1 Clone() [Transform](#) \* [Transform](#)::[Clone](#) ( ) const

Clones current [Transform](#) object.

#### Returns

[Transform](#)\* Cloned [Transform](#)

Definition at line 46 of file transform.cpp.

```
46     {
47     return new Transform (*this);
48 }
```

References [Transform](#)().

#### 4.20.3.2 GetCType() `CType Transform::GetCType ( ) [static]`

Gets the CType of [Transform](#) (used in [Object::GetComponent<>\(\)](#))

##### Returns

CType

Definition at line 171 of file transform.cpp.

```
171 {  
172     return CType::CTransform;  
173 }
```

#### 4.20.3.3 GetOldPosition() `glm::vec3 Transform::GetOldPosition ( ) const`

Returns old position of object.

##### Returns

glm::vec3

Definition at line 83 of file transform.cpp.

```
83 { return oldPosition; }
```

References oldPosition.

#### 4.20.3.4 GetPosition() `glm::vec3 Transform::GetPosition ( ) const`

Returns position of object.

##### Returns

glm::vec3

Definition at line 62 of file transform.cpp.

```
62 { return position; }
```

References position.

Referenced by [Model\\_Data::Draw\(\)](#), [Physics::Update\(\)](#), and [Physics::UpdateGravity\(\)](#).

**4.20.3.5 GetPositionRef()** `glm::vec3 & Transform::GetPositionRef ( )`

Returns position reference of object.

**Returns**

`glm::vec3&`

Definition at line 69 of file transform.cpp.

```
69 { return position; }
```

References position.

Referenced by Behavior::ClassSetup(), and Editor::Display\_Transform().

**4.20.3.6 GetRotation()** `glm::vec3 Transform::GetRotation ( ) const`

Returns rotation of object.

**Returns**

`float`

Definition at line 118 of file transform.cpp.

```
118 { return rotation; }
```

References rotation.

Referenced by Model\_Data::Draw(), and Physics::Update().

**4.20.3.7 GetRotationRef()** `glm::vec3 & Transform::GetRotationRef ( )`

Returns rotation reference of object.

**Returns**

`glm::vec3&`

Definition at line 125 of file transform.cpp.

```
125 { return rotation; }
```

References rotation.

Referenced by Behavior::ClassSetup(), and Editor::Display\_Transform().

**4.20.3.8 GetScale()** `glm::vec3 Transform::GetScale ( ) const`

Returns scale of object.

Returns

`glm::vec3`

Definition at line 97 of file transform.cpp.

```
97 { return scale; }
```

References scale.

Referenced by `Model_Data::Draw()`, and `File_Writer::Write_Object_Data()`.

**4.20.3.9 GetScaleRef()** `glm::vec3 & Transform::GetScaleRef ( )`

Returns scale reference of object.

Returns

`glm::vec3&`

Definition at line 104 of file transform.cpp.

```
104 { return scale; }
```

References scale.

Referenced by `Behavior::ClassSetup()`, and `Editor::Display_Transform()`.

**4.20.3.10 GetStartPosition()** `glm::vec3 Transform::GetStartPosition ( ) const`

Returns the saved start position of the object.

Returns

`glm::vec3`

Definition at line 139 of file transform.cpp.

```
139 { return startPosition; }
```

References startPosition.

Referenced by `File_Writer::Write_Object_Data()`.

**4.20.3.11 GetStartPositionRef()** `glm::vec3 & Transform::GetStartPositionRef ( )`

Returns a reference to the start position of the object.

**Returns**

`glm::vec3&`

Definition at line 146 of file transform.cpp.

```
146 { return startPosition; }
```

References startPosition.

Referenced by Behavior::ClassSetup(), and Editor::Display\_Transform().

**4.20.3.12 Read()** `void Transform::Read (   
File_Reader & reader )`

Reads data for [Transform](#) object from file.

**Parameters**

<i>reader</i>	File to read from
---------------	-------------------

Definition at line 153 of file transform.cpp.

```
153 {  
154     //SetRotation(reader.Read_Float("rotation"));  
155 }
```

Referenced by Object::ReRead(), and Transform().

**4.20.3.13 SetOldPosition()** `void Transform::SetOldPosition (   
glm::vec3 oldPos )`

Sets old position of object.

**Parameters**

<i>oldPos</i>	
---------------	--

Definition at line 76 of file transform.cpp.

```
76 { oldPosition = oldPos; }
```

References oldPosition.

Referenced by Physics::Update().

**4.20.3.14 SetPosition()** `void Transform::SetPosition ( glm::vec3 pos )`

Sets position of object.

#### Parameters

<i>pos</i>	
------------	--

Definition at line 55 of file transform.cpp.

```
55 { position = pos; }
```

References position.

Referenced by Behavior::ClassSetup(), Object\_Manager::ReadList(), and Physics::Update().

**4.20.3.15 SetRotation()** `void Transform::SetRotation ( glm::vec3 rot )`

Sets rotation of object.

#### Parameters

<i>rot</i>	
------------	--

Definition at line 111 of file transform.cpp.

```
111 { rotation = rot; }
```

References rotation.

Referenced by Behavior::ClassSetup(), and Physics::Update().

**4.20.3.16 SetScale()** `void Transform::SetScale ( glm::vec3 sca )`

Sets scale of object.

#### Parameters

<i>sca</i>	
------------	--

Definition at line 90 of file transform.cpp.

```
90 { scale = sca; }
```

References scale.



Referenced by Behavior::ClassSetup(), and Object\_Manager::ReadList().

**4.20.3.17 SetStartPosition()** `void Transform::SetStartPosition ( glm::vec3 startPosition_ )`

Sets the start position of the object.

**Parameters**

<i>start</i> ↔ <i>Position_</i>	
------------------------------------	--

Definition at line 132 of file transform.cpp.

```
132 { startPosition = startPosition_; }
```

References startPosition.

Referenced by Behavior::ClassSetup(), Editor::Display\_Scene(), and Object\_Manager::ReadList().

**4.20.3.18 Write()** `void Transform::Write ( File_Writer & writer )`

Gives transform data to writer object.

**Parameters**

<i>writer</i>	
---------------	--

Definition at line 162 of file transform.cpp.

```
162 {  
163     writer.Write_Vec3("rotation", rotation);  
164 }
```

References rotation, and File\_Writer::Write\_Vec3().

Referenced by Object::Write().

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

- [transform.hpp](#)
- [transform.cpp](#)

## 4.21 Vector3\_Func Class Reference

```
#include <vector3_func.hpp>
```

## Static Public Member Functions

- static glm::vec3 [normalize](#) (const glm::vec3 vec)  
*Wrapper for the glm normalize function.*
- static float [distance](#) (const glm::vec3 vec1, const glm::vec3 vec2)  
*Wrapper for the glm distance function.*
- static glm::vec3 [get\\_direction](#) (const glm::vec3 vec1, const glm::vec3 vec2)  
*Wrapper for subtracting two glm vectors to make a new vector.*
- static glm::vec3 [zero\\_vec3](#) ()  
*Creates a glm::vec3 filled with zeroes.*
- static float [length](#) (const glm::vec3 vec3)  
*Wrapper for the glm length function.*
- static glm::vec3 [add\\_float](#) (const glm::vec3 vec, float num)  
*Adds float to each part of a glm::vec3.*
- static glm::vec3 [add\\_vec3](#) (const glm::vec3 vec1, const glm::vec3 vec2)  
*Add two glm::vec3 together.*

### 4.21.1 Detailed Description

[Vector3\\_Func](#) class

Definition at line 21 of file vector3\_func.hpp.

### 4.21.2 Member Function Documentation

**4.21.2.1 add\_float()** glm::vec3 Vector3\_Func::add\_float (const glm::vec3 vec, float num) [static]

Adds float to each part of a glm::vec3.

#### Parameters

<i>vec</i>	
<i>num</i>	

#### Returns

glm::vec3

Definition at line 73 of file vector3\_func.cpp.

```

73
74     glm::vec3 returnVec3;
75 }
```

```
75
76     returnVec3.x = vec.x + num;
77     returnVec3.y = vec.y + num;
78     returnVec3.z = vec.z + num;
79
80     return vec;
81 }
```

Referenced by Behavior::ClassSetup().

**4.21.2.2 add\_vec3()** glm::vec3 Vector3\_Func::add\_vec3 (  
    const glm::vec3 vec1,  
    const glm::vec3 vec2 ) [static]

Add two glm::vec3 together.

#### Parameters

<i>vec1</i>	
<i>vec2</i>	

#### Returns

glm::vec3

Definition at line 90 of file vector3\_func.cpp.

```
90
91     glm::vec3 returnVec3;
92
93     returnVec3.x = vec1.x + vec2.x;
94     returnVec3.y = vec1.y + vec2.y;
95     returnVec3.z = vec1.z + vec2.z;
96
97     return vec1;
98 }
```

Referenced by Behavior::ClassSetup().

**4.21.2.3 distance()** float Vector3\_Func::distance (  
    const glm::vec3 vec1,  
    const glm::vec3 vec2 ) [static]

Wrapper for the glm distance function.

#### Parameters

<i>vec1</i>	First input vec3
<i>vec2</i>	Second input vec3

**Returns**

float

Definition at line 32 of file vector3\_func.cpp.

```
32                                     {  
33     return glm::distance(vec1, vec2);  
34 }
```

Referenced by Behavior::ClassSetup().

**4.21.2.4 get\_direction()** glm::vec3 Vector3\_Func::get\_direction (  
const glm::vec3 vec1,  
const glm::vec3 vec2 ) [static]

Wrapper for subtracting two glm vectors to make a new vector.

**Parameters**

<i>vec1</i>	First input vec3
<i>vec2</i>	Second input vec3

**Returns**

glm::vec3

Definition at line 43 of file vector3\_func.cpp.

```
43                                     {  
44     return vec1 - vec2;  
45 }
```

Referenced by Behavior::ClassSetup().

**4.21.2.5 length()** float Vector3\_Func::length (  
const glm::vec3 vec ) [static]

Wrapper for the glm length function.

**Parameters**

<i>vec</i>	Input vec3
------------	------------

**Returns**

float

Definition at line 62 of file vector3\_func.cpp.

```
62                                     {
63     return glm::length(vec);
64 }
```

Referenced by Behavior::ClassSetup().

**4.21.2.6 normalize()** glm::vec3 Vector3\_Func::normalize ( const glm::vec3 vec ) [static]

Wrapper for the glm normalize function.

#### Parameters

vec	Input vec3
-----	------------

#### Returns

glm::vec3

Definition at line 21 of file vector3\_func.cpp.

```
21                                     {
22     return glm::normalize(vec);
23 }
```

Referenced by Behavior::ClassSetup().

**4.21.2.7 zero\_vec3()** glm::vec3 Vector3\_Func::zero\_vec3 ( ) [static]

Creates a glm::vec3 filled with zeroes.

#### Returns

glm::vec3

Definition at line 52 of file vector3\_func.cpp.

```
52                                     {
53     return glm::vec3(0.f, 0.f, 0.f);
54 }
```

Referenced by Behavior::ClassSetup().

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

- [vector3\\_func.hpp](#)
- [vector3\\_func.cpp](#)

## 5 File Documentation

### 5.1 behavior.cpp File Reference

```
#include <glm.hpp>
#include "behavior.hpp"
#include "engine.hpp"
#include "object.hpp"
#include "object_manager.hpp"
#include "physics.hpp"
#include "random.hpp"
#include "transform.hpp"
#include "vector3_func.hpp"
```

#### 5.1.1 Detailed Description

##### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

##### Version

0.1

##### Date

2021-06-22

##### Copyright

Copyright (c) 2021

### 5.2 behavior.hpp File Reference

```
#include <vector>
#include <vec3.hpp>
#include <lua.hpp>
#include <sol/sol.hpp>
#include "component.hpp"
#include "file_reader.hpp"
#include "file_writer.hpp"
```

#### Classes

- class [Behavior](#)

### 5.2.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-22

#### Copyright

Copyright (c) 2021

## 5.3 camera.cpp File Reference

```
#include <glfw3.h>
#include <glm.hpp>
#include "editor.hpp"
#include "engine.hpp"
#include "graphics.hpp"
#include "camera.hpp"
#include "trace.hpp"
```

#### Variables

- static [Camera](#) \* [camera](#) = nullptr  
*[Camera](#) object.*

### 5.3.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-05

#### Copyright

Copyright (c) 2021

## 5.4 camera.hpp File Reference

```
#include <utility>
#include <vec3.hpp>
#include "file_reader.hpp"
```

### Classes

- class [Camera](#)

### 5.4.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-05

#### Copyright

Copyright (c) 2021

## 5.5 component.cpp File Reference

```
#include "component.hpp"
```

### 5.5.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-05

#### Copyright

Copyright (c) 2021



## 5.6 component.hpp File Reference

### Classes

- class [Component](#)

### Typedefs

- typedef [Component::CType](#) CType  
*Typedef for CType (used in other files)*

### 5.6.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com))

#### Version

0.1

#### Date

2021-06-05

#### Copyright

Copyright (c) 2021

## 5.7 editor.cpp File Reference

```
#include <imgui.h>
#include "imgui_impl_glfw.h"
#include "imgui_impl_opengl3.h"
#include "imgui_internal.h"
#include "ImGuiFileDialog.h"
#include <vec3.hpp>
#include "camera.hpp"
#include "editor.hpp"
#include "engine.hpp"
#include "graphics.hpp"
#include "object_manager.hpp"
```

## Variables

- static `Editor * editor` = nullptr  
*Editor object.*

### 5.7.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-07-14

#### Copyright

Copyright (c) 2021

## 5.8 editor.hpp File Reference

```
#include "behavior.hpp"
#include "object.hpp"
#include "model.hpp"
#include "physics.hpp"
#include "trace.hpp"
#include "transform.hpp"
```

## Classes

- class `Editor`

### 5.8.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-07-14

#### Copyright

Copyright (c) 2021

## 5.9 engine.cpp File Reference

```
#include <cmath>
#include <string>
#include "engine.hpp"
#include "graphics.hpp"
#include "object_manager.hpp"
#include "object.hpp"
#include "component.hpp"
#include "model_data_manager.hpp"
#include "physics.hpp"
#include "camera.hpp"
#include "editor.hpp"
#include "file_reader.hpp"
#include "random.hpp"
#include "texture_manager.hpp"
```

### Variables

- static `Engine * engine` = nullptr  
*Engine object.*

### 5.9.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-04

#### Copyright

Copyright (c) 2021

## 5.10 engine.hpp File Reference

```
#include <chrono>
#include <string>
#include <vec3.hpp>
```

## Classes

- class [Engine](#)

### 5.10.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-04

#### Copyright

Copyright (c) 2021

## 5.11 file\_reader.cpp File Reference

```
#include <fstream>
#include <iostream>
#include <filereadstream.h>
#include "file_reader.hpp"
#include "trace.hpp"
```

### 5.11.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-04

#### Copyright

Copyright (c) 2021

## 5.12 file\_reader.hpp File Reference

```
#include <string>
#include <document.h>
#include <vec3.hpp>
```

### Classes

- class [File\\_Reader](#)

### 5.12.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-04

#### Copyright

Copyright (c) 2021

## 5.13 file\_writer.cpp File Reference

```
#include <fstream>
#include <iostream>
#include "file_writer.hpp"
#include "trace.hpp"
#include "transform.hpp"
```

### 5.13.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-07-27

#### Copyright

Copyright (c) 2021

## 5.14 file\_writer.hpp File Reference

```
#include <string>
#include <vector>
#include <document.h>
#include <filewritestream.h>
#include <prettywriter.h>
#include <vec3.hpp>
#include "object.hpp"
```

### Classes

- class [File\\_Writer](#)

#### 5.14.1 Detailed Description

##### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

##### Version

0.1

##### Date

2021-07-27

##### Copyright

Copyright (c) 2021

## 5.15 graphics.cpp File Reference

```
#include <string>
#include <vector>
#include <cmath>
#include <glew.h>
#include <vec3.hpp>
#include <vec2.hpp>
#include <mat4x4.hpp>
#include <glm.hpp>
#include <gtc/matrix_transform.hpp>
#include <gtx/transform.hpp>
#include "engine.hpp"
#include "graphics.hpp"
#include "object_manager.hpp"
#include "model.hpp"
#include "transform.hpp"
#include "camera.hpp"
#include "editor.hpp"
#include "trace.hpp"
#include "shader.hpp"
```

## Variables

- static [Graphics](#) \* [graphics](#) = nullptr  
*[Graphics](#) object.*

### 5.15.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-05

#### Copyright

Copyright (c) 2021

## 5.16 [graphics.hpp](#) File Reference

```
#include <utility>
#include <GL/gl.h>
#include <glfw3.h>
#include "file_reader.hpp"
```

## Classes

- class [Graphics](#)

### 5.16.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-05

#### Copyright

Copyright (c) 2021

## 5.17 main.cpp File Reference

```
#include "trace.hpp"
#include "engine.hpp"
#include "graphics.hpp"
```

### Functions

- `int main (int, char *[])`  
*Main function.*

#### 5.17.1 Detailed Description

##### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

##### Version

0.1

##### Date

2021-05-06

##### Copyright

Copyright (c) 2021

#### 5.17.2 Function Documentation

**5.17.2.1 main()** `int main (`  
    `int ,`  
    `char * [ ] )`

Main function.

##### Returns

`int`

Definition at line 22 of file main.cpp.

```
22         {
23             // Initializing systems
24             Trace::Initialize();
25             if (!Engine::Initialize()) return -1;
26
27             // Engine update loop
28             Graphics::Update();
29
30             // Shutting down systems
31             Engine::Shutdown();
32             Trace::Shutdown();
33
34             return 0;
35 }
```

References `Trace::Initialize()`, `Engine::Initialize()`, `Trace::Shutdown()`, `Engine::Shutdown()`, and `Graphics::Update()`.



## 5.18 model.cpp File Reference

```
#include <cstdio>
#include "object.hpp"
#include "model.hpp"
#include "model_data_manager.hpp"
#include "transform.hpp"
#include "texture.hpp"
#include "texture_manager.hpp"
#include "trace.hpp"
```

### 5.18.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com))

#### Version

0.1

#### Date

2021-06-06

#### Copyright

Copyright (c) 2021

## 5.19 model.hpp File Reference

```
#include <vector>
#include <array>
#include <string>
#include <GL/gl.h>
#include "component.hpp"
#include "file_reader.hpp"
#include "file_writer.hpp"
#include "model_data.hpp"
#include "texture.hpp"
```

#### Classes

- class [Model](#)

### 5.19.1 Detailed Description

**Author**

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

**Version**

0.1

**Date**

2021-06-06

**Copyright**

Copyright (c) 2021

## 5.20 model\_data.cpp File Reference

```
#include <cstdio>
#include <cstring>
#include <glew.h>
#include <glm.hpp>
#include <gtc/matrix_transform.hpp>
#include <gtx/transform.hpp>
#include "engine.hpp"
#include "model.hpp"
#include "model_data.hpp"
#include "trace.hpp"
#include "shader.hpp"
```

### 5.20.1 Detailed Description

**Author**

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

**Version**

0.1

**Date**

2021-06-06

**Copyright**

Copyright (c) 2021

## 5.21 model\_data.hpp File Reference

```
#include <vector>
#include <array>
#include <string>
#include <vec3.hpp>
#include <vec2.hpp>
#include <mat4x4.hpp>
#include <GL/gl.h>
#include "transform.hpp"
```

### Classes

- class [Model\\_Data](#)

### 5.21.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com))

#### Version

0.1

#### Date

2021-06-06

#### Copyright

Copyright (c) 2021

## 5.22 model\_data\_manager.cpp File Reference

```
#include "model_data_manager.hpp"
#include "trace.hpp"
```

### Variables

- static [Model\\_Data\\_Manager](#) \* [model\\_data\\_manager](#) = nullptr  
*Model\_Data\_Manager* object.

### 5.22.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-06

#### Copyright

Copyright (c) 2021

## 5.23 model\_data\_manager.hpp File Reference

```
#include <vector>
#include <string>
#include "model_data.hpp"
#include "file_reader.hpp"
```

### Classes

- class [Model\\_Data\\_Manager](#)

### 5.23.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-06

#### Copyright

Copyright (c) 2021

## 5.24 object.cpp File Reference

```
#include "object.hpp"  
#include "behavior.hpp"  
#include "model.hpp"  
#include "object_manager.hpp"  
#include "physics.hpp"  
#include "transform.hpp"  
#include "file_reader.hpp"
```

### 5.24.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-05

#### Copyright

Copyright (c) 2021

## 5.25 object.hpp File Reference

```
#include <unordered_map>  
#include <string>  
#include "component.hpp"  
#include "trace.hpp"
```

### Classes

- class [Object](#)

### 5.25.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-05

#### Copyright

Copyright (c) 2021

## 5.26 object\_manager.cpp File Reference

```
#include <string>
#include "behavior.hpp"
#include "object_manager.hpp"
#include "trace.hpp"
#include "transform.hpp"
```

### Variables

- static [Object\\_Manager](#) \* [object\\_manager](#) = nullptr  
*[Object\\_Manager](#) object.*

### 5.26.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-05

#### Copyright

Copyright (c) 2021

## 5.27 object\_manager.hpp File Reference

```
#include <vector>
#include "object.hpp"
#include "file_reader.hpp"
#include "file_writer.hpp"
```

### Classes

- class [Object\\_Manager](#)

### 5.27.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-05

#### Copyright

Copyright (c) 2021

## 5.28 physics.cpp File Reference

```
#include <cmath>
#include <glm.hpp>
#include "engine.hpp"
#include "object_manager.hpp"
#include "object.hpp"
#include "physics.hpp"
#include "transform.hpp"
```

### 5.28.1 Detailed Description

**Author**

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

**Version**

0.1

**Date**

2021-06-05

**Copyright**

Copyright (c) 2021

## 5.29 physics.hpp File Reference

```
#include <vec3.hpp>
#include "component.hpp"
#include "file_reader.hpp"
#include "file_writer.hpp"
```

**Classes**

- class [Physics](#)

### 5.29.1 Detailed Description

**Author**

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

**Version**

0.1

**Date**

2021-06-05

**Copyright**

Copyright (c) 2021



## 5.30 random.cpp File Reference

```
#include "random.hpp"  
#include "trace.hpp"
```

### Variables

- static [Random](#) \* [random](#) = nullptr  
*[Random](#) object.*

### 5.30.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-07-13

#### Copyright

Copyright (c) 2021

## 5.31 random.hpp File Reference

```
#include <random>  
#include <vec3.hpp>
```

### Classes

- class [Random](#)

### 5.31.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-07-13

#### Copyright

Copyright (c) 2021

## 5.32 shader.cpp File Reference

```
#include <fstream>
#include <glew.h>
#include "shader.hpp"
#include "trace.hpp"
```

### Variables

- static [Shader](#) \* [shader](#) = nullptr  
*[Shader](#) object.*

### 5.32.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-19

#### Copyright

Copyright (c) 2021

### 5.33 shader.hpp File Reference

```
#include <string>
#include <GL/gl.h>
#include "file_reader.hpp"
```

#### Classes

- class [Shader](#)

#### 5.33.1 Detailed Description

##### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

##### Version

0.1

##### Date

2021-06-19

##### Copyright

Copyright (c) 2021

### 5.34 texture.cpp File Reference

```
#include <glew.h>
#include "shader.hpp"
#include "texture.hpp"
#include "trace.hpp"
```

#### Macros

- #define [FOURCC\\_DXT1](#) 0x31545844  
*Equivalent to "DXT1" in ASCII.*
- #define [FOURCC\\_DXT3](#) 0x33545844  
*Equivalent to "DXT3" in ASCII.*
- #define [FOURCC\\_DXT5](#) 0x35545844  
*Equivalent to "DXT5" in ASCII.*

### 5.34.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-07-14

#### Copyright

Copyright (c) 2021

## 5.35 texture.hpp File Reference

```
#include <string>
#include <GL/gl.h>
```

### Classes

- class [Texture](#)

### 5.35.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-07-14

#### Copyright

Copyright (c) 2021

## 5.36 texture\_manager.cpp File Reference

```
#include "texture_manager.hpp"
#include "trace.hpp"
```

### Variables

- static [Texture\\_Manager](#) \* `texture_manager` = nullptr  
*Texture\_Manager* object.

### 5.36.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-07-14

#### Copyright

Copyright (c) 2021

## 5.37 texture\_manager.hpp File Reference

```
#include <string>
#include <vector>
#include "file_reader.hpp"
#include "texture.hpp"
```

### Classes

- class [Texture\\_Manager](#)

### 5.37.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-07-14

#### Copyright

Copyright (c) 2021

## 5.38 trace.cpp File Reference

```
#include <iostream>
#include <cstdint>
#include "trace.hpp"
```

### Variables

- static [Trace](#) \* [trace](#) = nullptr  
*Trace object.*

### 5.38.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-05

#### Copyright

Copyright (c) 2021

## 5.39 trace.hpp File Reference

```
#include <string>
#include <fstream>
```

### Classes

- class [Trace](#)

### 5.39.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-05

#### Copyright

Copyright (c) 2021

## 5.40 transform.cpp File Reference

```
#include "transform.hpp"
```

### 5.40.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-06-05

#### Copyright

Copyright (c) 2021

## 5.41 transform.hpp File Reference

```
#include <vec3.hpp>
#include "component.hpp"
#include "file_reader.hpp"
#include "file_writer.hpp"
```

### Classes

- class [Transform](#)

#### 5.41.1 Detailed Description

##### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

##### Version

0.1

##### Date

2021-06-05

##### Copyright

Copyright (c) 2021

## 5.42 vector3\_func.cpp File Reference

```
#include "vector3_func.hpp"
```

#### 5.42.1 Detailed Description

##### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

##### Version

0.1

##### Date

2021-07-26

##### Copyright

Copyright (c) 2021



## 5.43 vector3\_func.hpp File Reference

```
#include <glm.hpp>
#include <vec3.hpp>
```

### Classes

- class [Vector3\\_Func](#)

### 5.43.1 Detailed Description

#### Author

Kelson Wysocki ( [kelson.wysocki@gmail.com](mailto:kelson.wysocki@gmail.com) )

#### Version

0.1

#### Date

2021-07-26

#### Copyright

Copyright (c) 2021

## Index

- ~Behavior
  - Behavior, [7](#)
- ~Model\_Data
  - Model\_Data, [75](#)
- ~Texture
  - Texture, [123](#)
- add\_float
  - Vector3\_Func, [141](#)
- add\_vec3
  - Vector3\_Func, [142](#)
- AddComponent
  - Object, [86](#)
- AddForce
  - Physics, [104](#)
- AddObject
  - Object\_Manager, [95](#)
- AddScript
  - Behavior, [7](#)
- ApplyForce
  - Physics, [104](#)
- Behavior, [4](#)
  - ~Behavior, [7](#)
  - AddScript, [7](#)
  - Behavior, [6](#)
  - CheckIfCopy, [8](#)
  - ClassSetup, [8](#)
  - Clear, [10](#)
  - Clone, [10](#)
  - GetCType, [10](#)
  - GetScripts, [10](#)
  - Read, [11](#)
  - SetupClassesForLua, [11](#)
  - SwitchScript, [12](#)
  - Update, [12](#)
  - Write, [13](#)
- behavior.cpp, [145](#)
- behavior.hpp, [145](#)
- Camera, [13](#)
  - Camera, [15](#)
  - GetFar, [16](#)
  - GetFov, [16](#)
  - GetFront, [16](#)
  - GetNear, [16](#)
  - GetOriginalMoveSpeed, [17](#)
  - GetOriginalSensitivity, [17](#)
  - GetOriginalSprintSpeed, [17](#)
  - GetPitch, [18](#)
  - GetPosition, [18](#)
  - GetUp, [18](#)
  - GetYaw, [19](#)
  - Initialize, [19](#)
  - MouseUpdate, [20](#)
  - Shutdown, [21](#)
  - Update, [21](#)
- camera.cpp, [146](#)
- camera.hpp, [147](#)
- CheckIfCopy
  - Behavior, [8](#)
- CheckName
  - Object\_Manager, [96](#)
- ClassSetup
  - Behavior, [8](#)
- Clear
  - Behavior, [10](#)
  - Object, [86](#)
- Clone
  - Behavior, [10](#)
  - Model, [69](#)
  - Object, [86](#)
  - Physics, [105](#)
  - Transform, [134](#)
- Component, [23](#)
  - Component, [24](#)
  - CType, [23](#)
  - GetCType, [24](#)
  - GetParent, [24](#)
  - SetParent, [25](#)
- component.cpp, [147](#)
- component.hpp, [148](#)
- CType
  - Component, [23](#)
- Display
  - Texture, [123](#)
- Display\_Camera\_Settings
  - Editor, [27](#)
- Display\_Components
  - Editor, [27](#)
- Display\_Dockspace
  - Editor, [29](#)
- Display\_Menu\_Bar
  - Editor, [29](#)
- Display\_Model
  - Editor, [30](#)
- Display\_Physics
  - Editor, [31](#)
- Display\_Scene
  - Editor, [32](#)
- Display\_Scripts
  - Editor, [33](#)

- Display\_Transform
  - Editor, [35](#)
- Display\_World\_Settings
  - Editor, [35](#)
- distance
  - Vector3\_Func, [142](#)
- Draw
  - Model, [69](#)
  - Model\_Data, [75](#)
- Editor, [25](#)
  - Display\_Camera\_Settings, [27](#)
  - Display\_Components, [27](#)
  - Display\_Dockspace, [29](#)
  - Display\_Menu\_Bar, [29](#)
  - Display\_Model, [30](#)
  - Display\_Physics, [31](#)
  - Display\_Scene, [32](#)
  - Display\_Scripts, [33](#)
  - Display\_Transform, [35](#)
  - Display\_World\_Settings, [35](#)
  - GetTakeKeyboardInput, [36](#)
  - Initialize, [36](#)
  - Render, [37](#)
  - Reset, [37](#)
  - Shutdown, [38](#)
  - Update, [38](#)
- editor.cpp, [148](#)
- editor.hpp, [149](#)
- Engine, [39](#)
  - GetDeltaTime, [41](#)
  - GetDt, [41](#)
  - GetGravConst, [41](#)
  - GetLightPos, [42](#)
  - GetLightPower, [42](#)
  - GetPresetName, [42](#)
  - Initialize, [43](#)
  - Restart, [44](#)
  - SetPresetName, [45](#)
  - Shutdown, [45](#)
  - Update, [46](#)
  - Write, [46](#)
- engine.cpp, [150](#)
- engine.hpp, [150](#)
- ErrorCallback
  - Graphics, [61](#)
- ErrorCheck
  - Graphics, [62](#)
- File\_Reader, [47](#)
  - Read\_Behavior\_Name, [48](#)
  - Read\_Bool, [48](#)
  - Read\_Double, [49](#)
  - Read\_File, [49](#)
  - Read\_Float, [50](#)
  - Read\_Int, [50](#)
  - Read\_Object\_Name, [52](#)
  - Read\_Object\_Position, [52](#)
  - Read\_Object\_Scale, [53](#)
  - Read\_Object\_Template\_Name, [53](#)
  - Read\_String, [54](#)
  - Read\_Vec3, [54](#)
- file\_reader.cpp, [151](#)
- file\_reader.hpp, [152](#)
- File\_Writer, [55](#)
  - File\_Writer, [56](#)
  - Write\_Behavior\_Name, [56](#)
  - Write\_File, [57](#)
  - Write\_Object\_Data, [57](#)
  - Write\_String, [58](#)
  - Write\_Value, [58](#)
  - Write\_Vec3, [59](#)
- file\_writer.cpp, [152](#)
- file\_writer.hpp, [153](#)
- FindObject
  - Object\_Manager, [96, 97](#)
- Get
  - Model\_Data\_Manager, [81](#)
  - Texture\_Manager, [127, 128](#)
- get\_direction
  - Vector3\_Func, [143](#)
- GetAcceleration
  - Physics, [105](#)
- GetAccelerationRef
  - Physics, [105](#)
- GetComponent
  - Object, [87](#)
- GetComponentConst
  - Object, [87](#)
- GetComponentList
  - Object, [88](#)
- GetCType
  - Behavior, [10](#)
  - Component, [24](#)
  - Model, [70](#)
  - Physics, [106](#)
  - Transform, [134](#)
- GetDeltaTime
  - Engine, [41](#)
- GetDt
  - Engine, [41](#)
- GetFar
  - Camera, [16](#)
- GetForces
  - Physics, [106](#)
- GetForcesRef
  - Physics, [106](#)
- GetFov

- Camera, [16](#)
- GetFront
  - Camera, [16](#)
- GetGravConst
  - Engine, [41](#)
- GetId
  - Object, [88](#)
- GetLightId
  - Shader, [117](#)
- GetLightPos
  - Engine, [42](#)
- GetLightPower
  - Engine, [42](#)
- GetLightPowerId
  - Shader, [117](#)
- GetMass
  - Physics, [107](#)
- GetMassRef
  - Physics, [107](#)
- GetMatrixId
  - Shader, [117](#)
- GetModelMatrixId
  - Shader, [118](#)
- GetModelName
  - Model, [70](#)
  - Model\_Data, [77](#)
- GetName
  - Object, [89](#)
- GetNameRef
  - Object, [89](#)
- GetNear
  - Camera, [16](#)
- GetOldPosition
  - Transform, [135](#)
- GetOriginalMoveSpeed
  - Camera, [17](#)
- GetOriginalSensitivity
  - Camera, [17](#)
- GetOriginalSprintSpeed
  - Camera, [17](#)
- GetParent
  - Component, [24](#)
- GetPitch
  - Camera, [18](#)
- GetPosition
  - Camera, [18](#)
  - Transform, [135](#)
- GetPositionRef
  - Transform, [135](#)
- GetPresetName
  - Engine, [42](#)
- GetProgram
  - Shader, [118](#)
- GetRotation
  - Transform, [136](#)
- GetRotationalVelocity
  - Physics, [107](#)
- GetRotationalVelocityRef
  - Physics, [108](#)
- GetRotationRef
  - Transform, [136](#)
- GetScale
  - Transform, [136](#)
- GetScaleRef
  - Transform, [137](#)
- GetScripts
  - Behavior, [10](#)
- GetSize
  - Object\_Manager, [97](#)
- GetStartPosition
  - Transform, [137](#)
- GetStartPositionRef
  - Transform, [137](#)
- GetTakeKeyboardInput
  - Editor, [36](#)
- GetTemplateName
  - Object, [89](#)
- GetTexture
  - Model, [70](#)
- GetTextureName
  - Model, [71](#)
  - Texture, [123](#)
- GetTextureNum
  - Texture, [124](#)
- GetUp
  - Camera, [18](#)
- GetVelocity
  - Physics, [108](#)
- GetVelocityRef
  - Physics, [108](#)
- GetViewMatrixId
  - Shader, [118](#)
- GetWindow
  - Graphics, [62](#)
- GetWindowSize
  - Graphics, [62](#)
- GetYaw
  - Camera, [19](#)
- Graphics, [60](#)
  - ErrorCallback, [61](#)
  - ErrorCheck, [62](#)
  - GetWindow, [62](#)
  - GetWindowSize, [62](#)
  - Graphics, [61](#)
  - Initialize, [63](#)
  - InitializeGL, [64](#)
  - Render, [64](#)
  - Shutdown, [65](#)

- Update, 65
- graphics.cpp, 153
- graphics.hpp, 154
- Initialize
  - Camera, 19
  - Editor, 36
  - Engine, 43
  - Graphics, 63
  - Model\_Data\_Manager, 82
  - Object\_Manager, 97
  - Random, 113
  - Shader, 119
  - Texture\_Manager, 128
  - Trace, 130
- InitializeGL
  - Graphics, 64
- length
  - Vector3\_Func, 143
- Load
  - Model, 71
  - Model\_Data, 77, 78
  - Texture, 124
- LoadDDS
  - Texture, 125
- LoadShader
  - Shader, 119
- main
  - main.cpp, 155
- main.cpp, 155
- main, 155
- Message
  - Trace, 130
- Model, 66
  - Clone, 69
  - Draw, 69
  - GetType, 70
  - GetModelName, 70
  - GetTexture, 70
  - GetTextureName, 71
  - Load, 71
  - Model, 68
  - Read, 72
  - SwitchModel, 72
  - SwitchTexture, 72
  - Write, 73
- model.cpp, 156
- model.hpp, 156
- Model\_Data, 73
  - ~Model\_Data, 75
  - Draw, 75
  - GetModelName, 77
  - Load, 77, 78
  - Model\_Data, 74, 75
  - Read, 78
- model\_data.cpp, 157
- model\_data.hpp, 158
- Model\_Data\_Manager, 80
  - Get, 81
  - Initialize, 82
  - Shutdown, 82
- model\_data\_manager.cpp, 158
- model\_data\_manager.hpp, 159
- MouseUpdate
  - Camera, 20
- normalize
  - Vector3\_Func, 144
- Object, 83
  - AddComponent, 86
  - Clear, 86
  - Clone, 86
  - GetComponent, 87
  - GetComponentConst, 87
  - GetComponentList, 88
  - GetId, 88
  - GetName, 89
  - GetNameRef, 89
  - GetTemplateName, 89
  - Object, 85
  - Read, 90
  - RemoveComponent, 90
  - ReRead, 91
  - SetId, 92
  - SetName, 92
  - SetTemplateName, 93
  - Update, 93
  - Write, 93
- object.cpp, 160
- object.hpp, 160
- Object\_Manager, 94
  - AddObject, 95
  - CheckName, 96
  - FindObject, 96, 97
  - GetSize, 97
  - Initialize, 97
  - ReadList, 98
  - RemoveObject, 99
  - Shutdown, 100
  - Update, 100
  - Write, 100
- object\_manager.cpp, 161
- object\_manager.hpp, 162
- Physics, 101
  - AddForce, 104
  - ApplyForce, 104

- Clone, [105](#)
- GetAcceleration, [105](#)
- GetAccelerationRef, [105](#)
- GetCType, [106](#)
- GetForces, [106](#)
- GetForcesRef, [106](#)
- GetMass, [107](#)
- GetMassRef, [107](#)
- GetRotationalVelocity, [107](#)
- GetRotationalVelocityRef, [108](#)
- GetVelocity, [108](#)
- GetVelocityRef, [108](#)
- Physics, [103](#), [104](#)
- Read, [109](#)
- SetAcceleration, [109](#)
- SetForces, [110](#)
- SetMass, [110](#)
- SetRotationalVelocity, [110](#)
- SetVelocity, [111](#)
- Update, [111](#)
- UpdateGravity, [112](#)
- Write, [112](#)
- physics.cpp, [162](#)
- physics.hpp, [163](#)
- Random, [113](#)
  - Initialize, [113](#)
  - random\_float, [114](#)
  - random\_vec3, [114](#)
  - Shutdown, [115](#)
- random.cpp, [164](#)
- random.hpp, [164](#)
- random\_float
  - Random, [114](#)
- random\_vec3
  - Random, [114](#)
- Read
  - Behavior, [11](#)
  - Model, [72](#)
  - Model\_Data, [78](#)
  - Object, [90](#)
  - Physics, [109](#)
  - Transform, [138](#)
- Read\_Behavior\_Name
  - File\_Reader, [48](#)
- Read\_Bool
  - File\_Reader, [48](#)
- Read\_Double
  - File\_Reader, [49](#)
- Read\_File
  - File\_Reader, [49](#)
- Read\_Float
  - File\_Reader, [50](#)
- Read\_Int
  - File\_Reader, [50](#)
- Read\_Object\_Name
  - File\_Reader, [52](#)
- Read\_Object\_Position
  - File\_Reader, [52](#)
- Read\_Object\_Scale
  - File\_Reader, [53](#)
- Read\_Object\_Template\_Name
  - File\_Reader, [53](#)
- Read\_String
  - File\_Reader, [54](#)
- Read\_Vec3
  - File\_Reader, [54](#)
- ReadFile
  - Shader, [120](#)
- ReadList
  - Object\_Manager, [98](#)
- RemoveComponent
  - Object, [90](#)
- RemoveObject
  - Object\_Manager, [99](#)
- Render
  - Editor, [37](#)
  - Graphics, [64](#)
- ReRead
  - Object, [91](#)
- Reset
  - Editor, [37](#)
- Restart
  - Engine, [44](#)
- SetAcceleration
  - Physics, [109](#)
- SetForces
  - Physics, [110](#)
- SetId
  - Object, [92](#)
- SetMass
  - Physics, [110](#)
- SetName
  - Object, [92](#)
- SetOldPosition
  - Transform, [138](#)
- SetParent
  - Component, [25](#)
- SetPosition
  - Transform, [138](#)
- SetPresetName
  - Engine, [45](#)
- SetRotation
  - Transform, [139](#)
- SetRotationalVelocity
  - Physics, [110](#)
- SetScale

- Transform, [139](#)
- SetStartPosition
  - Transform, [140](#)
- SetTemplateName
  - Object, [93](#)
- SetupClassesForLua
  - Behavior, [11](#)
- SetVelocity
  - Physics, [111](#)
- Shader, [116](#)
  - GetLightId, [117](#)
  - GetLightPowerId, [117](#)
  - GetMatrixId, [117](#)
  - GetModelMatrixId, [118](#)
  - GetProgram, [118](#)
  - GetViewMatrixId, [118](#)
  - Initialize, [119](#)
  - LoadShader, [119](#)
  - ReadFile, [120](#)
  - Shutdown, [121](#)
  - Update, [121](#)
- shader.cpp, [165](#)
- shader.hpp, [166](#)
- Shutdown
  - Camera, [21](#)
  - Editor, [38](#)
  - Engine, [45](#)
  - Graphics, [65](#)
  - Model\_Data\_Manager, [82](#)
  - Object\_Manager, [100](#)
  - Random, [115](#)
  - Shader, [121](#)
  - Texture\_Manager, [129](#)
  - Trace, [131](#)
- SwitchModel
  - Model, [72](#)
- SwitchScript
  - Behavior, [12](#)
- SwitchTexture
  - Model, [72](#)
- Texture, [122](#)
  - ~Texture, [123](#)
  - Display, [123](#)
  - GetTextureName, [123](#)
  - GetTextureNum, [124](#)
  - Load, [124](#)
  - LoadDDS, [125](#)
- texture.cpp, [166](#)
- texture.hpp, [167](#)
- Texture\_Manager, [126](#)
  - Get, [127](#), [128](#)
  - Initialize, [128](#)
  - Shutdown, [129](#)
- texture\_manager.cpp, [168](#)
- texture\_manager.hpp, [168](#)
- Trace, [130](#)
  - Initialize, [130](#)
  - Message, [130](#)
  - Shutdown, [131](#)
- trace.cpp, [169](#)
- trace.hpp, [170](#)
- Transform, [132](#)
  - Clone, [134](#)
  - GetCType, [134](#)
  - GetOldPosition, [135](#)
  - GetPosition, [135](#)
  - GetPositionRef, [135](#)
  - GetRotation, [136](#)
  - GetRotationRef, [136](#)
  - GetScale, [136](#)
  - GetScaleRef, [137](#)
  - GetStartPosition, [137](#)
  - GetStartPositionRef, [137](#)
  - Read, [138](#)
  - SetOldPosition, [138](#)
  - SetPosition, [138](#)
  - SetRotation, [139](#)
  - SetScale, [139](#)
  - SetStartPosition, [140](#)
  - Transform, [133](#), [134](#)
  - Write, [140](#)
- transform.cpp, [170](#)
- transform.hpp, [171](#)
- Update
  - Behavior, [12](#)
  - Camera, [21](#)
  - Editor, [38](#)
  - Engine, [46](#)
  - Graphics, [65](#)
  - Object, [93](#)
  - Object\_Manager, [100](#)
  - Physics, [111](#)
  - Shader, [121](#)
- UpdateGravity
  - Physics, [112](#)
- Vector3\_Func, [140](#)
  - add\_float, [141](#)
  - add\_vec3, [142](#)
  - distance, [142](#)
  - get\_direction, [143](#)
  - length, [143](#)
  - normalize, [144](#)
  - zero\_vec3, [144](#)
- vector3\_func.cpp, [171](#)
- vector3\_func.hpp, [172](#)

## Write

- Behavior, [13](#)
- Engine, [46](#)
- Model, [73](#)
- Object, [93](#)
- Object\_Manager, [100](#)
- Physics, [112](#)
- Transform, [140](#)

## Write\_Behavior\_Name

- File\_Writer, [56](#)

## Write\_File

- File\_Writer, [57](#)

## Write\_Object\_Data

- File\_Writer, [57](#)

## Write\_String

- File\_Writer, [58](#)

## Write\_Value

- File\_Writer, [58](#)

## Write\_Vec3

- File\_Writer, [59](#)

## zero\_vec3

- Vector3\_Func, [144](#)