

pEngine

Generated by Doxygen 1.8.17

1 Hierarchical Index	1
1.1 Class Hierarchy	1
2 Class Index	2
2.1 Class List	2
3 File Index	3
3.1 File List	3
4 Class Documentation	4
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	40
4.5.1 Detailed Description	41
4.5.2 Member Function Documentation	41
4.6 File_Reader Class Reference	48
4.6.1 Detailed Description	48
4.6.2 Constructor & Destructor Documentation	49
4.6.3 Member Function Documentation	50
4.7 File_Writer Class Reference	56
4.7.1 Detailed Description	57
4.7.2 Constructor & Destructor Documentation	57
4.7.3 Member Function Documentation	57
4.8 Graphics Class Reference	61
4.8.1 Detailed Description	62
4.8.2 Constructor & Destructor Documentation	62

4.8.3 Member Function Documentation	62
4.9 Model Class Reference	67
4.9.1 Detailed Description	69
4.9.2 Constructor & Destructor Documentation	69
4.9.3 Member Function Documentation	70
4.10 Model_Data Class Reference	74
4.10.1 Detailed Description	75
4.10.2 Constructor & Destructor Documentation	75
4.10.3 Member Function Documentation	76
4.11 Model_Data_Manager Class Reference	81
4.11.1 Detailed Description	82
4.11.2 Member Function Documentation	82
4.12 Object Class Reference	84
4.12.1 Detailed Description	86
4.12.2 Constructor & Destructor Documentation	86
4.12.3 Member Function Documentation	87
4.13 Object_Manager Class Reference	95
4.13.1 Detailed Description	96
4.13.2 Member Function Documentation	96
4.14 Physics Class Reference	102
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	112
4.15.1 Detailed Description	113
4.15.2 Member Function Documentation	113
4.16 Shader Class Reference	115
4.16.1 Detailed Description	116
4.16.2 Member Function Documentation	116
4.17 Texture Class Reference	121
4.17.1 Detailed Description	122
4.17.2 Constructor & Destructor Documentation	122
4.17.3 Member Function Documentation	123
4.18 Texture_Manager Class Reference	125
4.18.1 Detailed Description	126
4.18.2 Member Function Documentation	126
4.19 Trace Class Reference	129
4.19.1 Detailed Description	129
4.19.2 Member Function Documentation	129

4.20 Transform Class Reference	131
4.20.1 Detailed Description	132
4.20.2 Constructor & Destructor Documentation	132
4.20.3 Member Function Documentation	133
4.21 Vector3_Func Class Reference	139
4.21.1 Detailed Description	140
4.21.2 Member Function Documentation	140
5 File Documentation	144
5.1 behavior.cpp File Reference	144
5.1.1 Detailed Description	144
5.2 behavior.hpp File Reference	144
5.2.1 Detailed Description	145
5.3 camera.cpp File Reference	145
5.3.1 Detailed Description	145
5.4 camera.hpp File Reference	146
5.4.1 Detailed Description	146
5.5 component.cpp File Reference	146
5.5.1 Detailed Description	146
5.6 component.hpp File Reference	147
5.6.1 Detailed Description	147
5.7 editor.cpp File Reference	147
5.7.1 Detailed Description	148
5.8 editor.hpp File Reference	148
5.8.1 Detailed Description	148
5.9 engine.cpp File Reference	149
5.9.1 Detailed Description	149
5.10 engine.hpp File Reference	149
5.10.1 Detailed Description	150
5.11 file_reader.cpp File Reference	150
5.11.1 Detailed Description	150
5.12 file_reader.hpp File Reference	151
5.12.1 Detailed Description	151
5.13 file_writer.cpp File Reference	151
5.13.1 Detailed Description	151
5.14 file_writer.hpp File Reference	152
5.14.1 Detailed Description	152
5.15 graphics.cpp File Reference	152
5.15.1 Detailed Description	153

5.16 graphics.hpp File Reference	153
5.16.1 Detailed Description	153
5.17 main.cpp File Reference	154
5.17.1 Detailed Description	154
5.17.2 Function Documentation	154
5.18 model.cpp File Reference	155
5.18.1 Detailed Description	155
5.19 model.hpp File Reference	155
5.19.1 Detailed Description	156
5.20 model_data.cpp File Reference	156
5.20.1 Detailed Description	156
5.21 model_data.hpp File Reference	157
5.21.1 Detailed Description	157
5.22 model_data_manager.cpp File Reference	157
5.22.1 Detailed Description	158
5.23 model_data_manager.hpp File Reference	158
5.23.1 Detailed Description	158
5.24 object.cpp File Reference	159
5.24.1 Detailed Description	159
5.25 object.hpp File Reference	159
5.25.1 Detailed Description	160
5.26 object_manager.cpp File Reference	160
5.26.1 Detailed Description	160
5.27 object_manager.hpp File Reference	161
5.27.1 Detailed Description	161
5.28 physics.cpp File Reference	161
5.28.1 Detailed Description	162
5.29 physics.hpp File Reference	162
5.29.1 Detailed Description	162
5.30 random.cpp File Reference	163
5.30.1 Detailed Description	163
5.31 random.hpp File Reference	163
5.31.1 Detailed Description	164
5.32 shader.cpp File Reference	164
5.32.1 Detailed Description	164
5.33 shader.hpp File Reference	165
5.33.1 Detailed Description	165
5.34 texture.cpp File Reference	165
5.34.1 Detailed Description	166

5.35 texture.hpp File Reference	166
5.35.1 Detailed Description	166
5.36 texture_manager.cpp File Reference	167
5.36.1 Detailed Description	167
5.37 texture_manager.hpp File Reference	167
5.37.1 Detailed Description	168
5.38 trace.cpp File Reference	168
5.38.1 Detailed Description	168
5.39 trace.hpp File Reference	169
5.39.1 Detailed Description	169
5.40 transform.cpp File Reference	169
5.40.1 Detailed Description	169
5.41 transform.hpp File Reference	170
5.41.1 Detailed Description	170
5.42 vector3_func.cpp File Reference	170
5.42.1 Detailed Description	170
5.43 vector3_func.hpp File Reference	171
5.43.1 Detailed Description	171

Index	173
--------------	------------

1 Hierarchical Index

1.1 Class Hierarchy

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

Camera	13
Component	23
Behavior	4
Model	67
Physics	102
Transform	131
Editor	25
Engine	40
File_Reader	48

File_Writer	56
Graphics	61
Model_Data	74
Model_Data_Manager	81
Object	84
Object_Manager	95
Random	112
Shader	115
Texture	121
Texture_Manager	125
Trace	129
Vector3_Func	139

2 Class Index

2.1 Class List

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

Behavior	4
Camera	13
Component	23
Editor	25
Engine	40
File_Reader	48
File_Writer	56
Graphics	61
Model	67
Model_Data	74
Model_Data_Manager	81
Object	84
Object_Manager	95

Physics	102
Random	112
Shader	115
Texture	121
Texture_Manager	125
Trace	129
Transform	131
Vector3_Func	139

3 File Index

3.1 File List

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

behavior.cpp	144
behavior.hpp	144
camera.cpp	145
camera.hpp	146
component.cpp	146
component.hpp	147
editor.cpp	147
editor.hpp	148
engine.cpp	149
engine.hpp	149
file_reader.cpp	150
file_reader.hpp	151
file_writer.cpp	151
file_writer.hpp	152
graphics.cpp	152
graphics.hpp	153
main.cpp	154

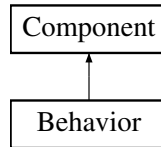
model.cpp	155
model.hpp	155
model_data.cpp	156
model_data.hpp	157
model_data_manager.cpp	157
model_data_manager.hpp	158
object.cpp	159
object.hpp	159
object_manager.cpp	160
object_manager.hpp	161
physics.cpp	161
physics.hpp	162
random.cpp	163
random.hpp	163
shader.cpp	164
shader.hpp	165
texture.cpp	165
texture.hpp	166
texture_manager.cpp	167
texture_manager.hpp	167
trace.cpp	168
trace.hpp	169
transform.cpp	169
transform.hpp	170
vector3_func.cpp	170
vector3_func.hpp	171

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>	Behavior 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 233 of file behavior.cpp.

```

233                                     {
234     // Checking if this script is already attached
235     if (CheckIfCopy(newScriptName)) return false;
236     // Setting up new lua state
237     sol::state* state = new sol::state;
238     state->open_libraries(sol::lib::base, sol::lib::math, sol::lib::io, sol::lib::string);
239     states.emplace_back(state);
240     // Adding new script filename to list
241     scripts.emplace_back(newScriptName);
242     ClassSetup(state);
243     // Setting up lua script to run
244     states.back()->script_file(std::string("data/scripts/" + scripts.back()).c_str());
245     (*states.back())["Start"]();
246
247     return true;
248 }

```

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 257 of file behavior.cpp.

```

257                                     {
258     // Checking if script is the same as an existing one
259     for (std::string scriptName : scripts) {
260         if (scriptName.compare(newScriptName) == 0) return true;
261     }
262
263     // Script is not a copy
264     return false;
265 }

```

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", &Object_Manager::FindObject);
174
175     // Giving lua physics class
176     sol::usertype<Physics> physics_type = state->new_usertype<Physics>("Physics",
177     sol::constructors<Physics(), Physics(const Physics)>());
178     // Giving lua physics class variables
179     physics_type.set("acceleration", sol::property(Physics::GetAccelerationRef, &Physics::SetAcceleration));
180     physics_type.set("forces", sol::property(Physics::GetForcesRef, &Physics::SetForces));
181     physics_type.set("velocity", sol::property(Physics::GetVelocityRef, &Physics::SetVelocity));
182     // Giving lua physics class functions
183     physics_type.set_function("ApplyForce", &Physics::ApplyForce);
184     physics_type.set_function("UpdateGravity", &Physics::UpdateGravity);
185
186     // Giving lua transform class
187     sol::usertype<Transform> transform_type = state->new_usertype<Transform>("Transform",
188     sol::constructors<Transform(), Transform(const Transform)>());
189     // Giving lua transform class variables
190     transform_type.set("position", sol::property(Transform::GetPositionRef,
191     &Transform::SetPosition));
192     transform_type.set("rotation", sol::property(Transform::GetRotationRef,
193     &Transform::SetRotation));
194     transform_type.set("scale", sol::property(Transform::GetScaleRef,
195     &Transform::SetScale));
196     transform_type.set("startPosition", sol::property(Transform::GetStartPositionRef,
197     &Transform::SetStartPosition));
198
199     // Giving lua object class
200     state->set("object", GetParent());
201     sol::usertype<Object> object_type = state->new_usertype<Object>("Object",
202     sol::constructors<Object(), Object(const Object)>());
203     // Giving lua object class variables
204     object_type.set("name", sol::property(Object::GetNameRef, &Object::SetName));
205     object_type.set("id", sol::readonly_property(Object::GetId));
206     object_type.set_function("GetPhysics", &Object::GetComponent<Physics>);
207     object_type.set_function("GetTransform", &Object::GetComponent<Transform>);
208 }

```

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 271 of file `behavior.cpp`.

```
271         {
272     for (sol::state* state : states) {
273         if (!state) continue;
274         delete state;
275         state = nullptr;
276     }
277
278     states.clear();
279     scripts.clear();
280 }
```

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 `GetType()` `CType Behavior::GetType () [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("data/scripts/" + scripts[i]).c_str());
132         (*states[i])["Start"]();
133     }
134 }
```

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 214 of file behavior.cpp.

```

214     {
215         // Checking if this script is already attached
216         if (CheckIfCopy(newScriptName)) return false;
217         sol::state* state = states[scriptNum];
218         scripts[scriptNum] = newScriptName;
219         // Setting up new lua script
220         state->script_file(std::string("data/scripts/" + scripts[scriptNum]).c_str());
221         (*state)["Start"]();
222     }
223     return true;
224 }
```

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) {
96                 camera->position += camera->speed * camera->up;
97             }
98             if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_LEFT_CONTROL) == GLFW_PRESS) {
99                 camera->position -= camera->speed * camera->up;
100             }
101
102             if (glfwGetMouseButton(Graphics::GetWindow(), GLFW_MOUSE_BUTTON_RIGHT) == GLFW_PRESS &&
103                 Editor::GetTakeKeyboardInput()) {
104                 camera->canMoveMouse = true;
105             }
106             if (glfwGetMouseButton(Graphics::GetWindow(), GLFW_MOUSE_BUTTON_RIGHT) == GLFW_RELEASE) {
107                 camera->canMoveMouse = false;
108             }
109         }

```

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

Referenced by Engine::Update().

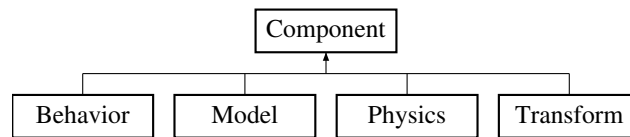
The documentation for this class was generated from the following files:

- [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 404 of file editor.cpp.

```

404     {
405         ImGui::Begin("Camera Settings");
406
407         ImGui::PushItemWidth(137);
408
409         // Default move speed
410         ImGui::Text("Move Speed");
411         ImGui::SameLine(100); ImGui::InputFloat("##2", &Camera::GetOriginalMoveSpeed());
412
413         // Move speed when holding shift
414         ImGui::Text("Sprint Speed");
415         ImGui::SameLine(100); ImGui::InputFloat("##3", &Camera::GetOriginalSprintSpeed());
416
417         // Mouse sensitivity when looking around
418         ImGui::Text("Sensitivity");
419         ImGui::SameLine(100); ImGui::InputFloat("##4", &Camera::GetOriginalSensitivity());
420
421         ImGui::PopItemWidth();
422
423         ImGui::End();
424     }
```

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 266 of file editor.cpp.

```

266     {
267         ImGui::Begin("Components##1");
268
269         if (selected_object == -1) { ImGui::End(); return; }
270         Object* object = Object_Manager::FindObject(selected_object);
271         std::string objectName = object->GetName();
272
273         // Display name box (allows changing the name of an object)
274         static char nameBuf[128] = "";
275         sprintf(nameBuf, objectName.c_str());
276
277         if (ImGui::InputText("Name", nameBuf, 128, ImGuiInputTextFlags_EnterReturnsTrue)) {
```



```

278     object->SetName(std::string(nameBuf));
279 }
280
281 if (ImGui::IsItemDeactivatedAfterEdit()) {
282     object->SetName(std::string(nameBuf));
283 }
284
285 // Template used by the selected object
286 ImGui::Text("Template:");
287 ImGui::SameLine(100);
288 std::string templateName = object->GetTemplateName();
289 if (templateName.empty()) templateName = "No template##1";
290 if (ImGui::Button(templateName.c_str())) {
291     ImGuiFileDialog::Instance()->OpenDialog("ChooseTemplate##1", "Choose File", ".json",
292     "./data/json/objects/");
293 }
294
295 ImGui::SameLine();
296 if (ImGui::Button("New Template")) {
297     object->Write();
298 }
299
300 if (ImGuiFileDialog::Instance()->Display("ChooseTemplate##1")) {
301     if (ImGuiFileDialog::Instance()->IsOk()) {
302         std::string filePathName = ImGuiFileDialog::Instance()->GetCurrentFileName();
303         object->ReRead(filePathName);
304     }
305     ImGuiFileDialog::Instance()->Close();
306 }
307
308 // Getting all of the components
309 Behavior* behavior = object->GetComponent<Behavior>();
310 Model* model = object->GetComponent<Model>();
311 Physics* physics = object->GetComponent<Physics>();
312 Transform* transform = object->GetComponent<Transform>();
313
314 // Display all of the components of the selected_object
315 Display_Transform(transform);
316 Display_Physics(physics);
317 Display_Model(model);
318 Display_Scripts(behavior);
319
320 ImGui::Separator();
321
322 // Button to add new components to the selected_object
323 if (ImGui::Button("Add Component##1")) {
324     ImGui::OpenPopup("New Component##1");
325 }
326
327 // Add new components to object (only ones that the object doesn't already have)
328 if (ImGui::BeginPopup("New Component##1")) {
329     if (!physics) {
330         if (ImGui::Selectable("Physics##1")) {
331             physics = new Physics;
332             object->AddComponent(physics);
333         }
334     }
335     if (!model) {
336         if (ImGui::Selectable("Model##1")) {
337             model = new Model;
338             object->AddComponent(model);
339         }
340     }
341     if (!behavior) {
342         if (ImGui::Selectable("Scripts##1")) {
343             behavior = new Behavior;
344             object->AddComponent(behavior);
345         }
346     }
347     ImGui::EndPopup();
348 }
349
350 ImGui::End();
351 }

```

References `Display_Model()`, `Display_Physics()`, `Display_Scripts()`, `Display_Transform()`, `Object_Manager::FindObject()`, 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 170 of file editor.cpp.

```

170     {
171         // Setting up viewport
172         ImGuiViewport* viewport = ImGui::GetMainViewport();
173         ImGui::SetNextWindowPos(viewport->Pos);
174         ImGui::SetNextWindowSize(viewport->Size);
175         ImGui::SetNextWindowViewport(viewport->ID);
176         ImGui::SetNextWindowBgAlpha(0.0f);
177
178         // Setting up window flags
179         ImGuiWindowFlags window_flags = ImGuiWindowFlags_MenuBar | ImGuiWindowFlags_NoDocking;
180         window_flags |= ImGuiWindowFlags_NoTitleBar | ImGuiWindowFlags_NoCollapse | ImGuiWindowFlags_NoResize |
181         ImGuiWindowFlags_NoMove;
182         window_flags |= ImGuiWindowFlags_NoBringToFrontOnFocus | ImGuiWindowFlags_NoNavFocus;
183
184         // Setting up window style
185         ImGui::PushStyleVar(ImGuiStyleVar_WindowRounding, 0.0f);
186         ImGui::PushStyleVar(ImGuiStyleVar_WindowBorderSize, 0.0f);
187         ImGui::PushStyleVar(ImGuiStyleVar_WindowPadding, ImVec2(0.0f, 0.0f));
188
189         // Making the window
190         ImGui::SetNextWindowBgAlpha(0.0f);
191         ImGui::Begin("Editor Window", &editor->isOpen, window_flags);
192         ImGui::PopStyleVar(3);
193
194         // Setting up window settings
195         ImGuiID dockspace_id = ImGui::GetID("Editor");
196         ImGuiDockNodeFlags dockspace_flags = ImGuiDockNodeFlags_PassthruCentralNode |
197         ImGuiDockNodeFlags_NoDockingInCentralNode;
198         ImGui::DockSpace(dockspace_id, ImVec2(0.0f, 0.0f), dockspace_flags);
199         editor->Display_Menu_Bar();
200         ImGui::End();
201     }

```

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 677 of file editor.cpp.

```

677     {
678         static bool saveAs = false;
679         if (ImGui::BeginMenuBar()) {
680             if (ImGui::BeginMenu("File##1")) {
681                 if (ImGui::MenuItem("Save##1")) {
682                     Engine::Write();
683                 }
684                 if (ImGui::MenuItem("Save As..##1")) {
685                     saveAs = true;
686                 }
687             }
688             ImGui::EndMenu();
689         }
690         if (saveAs) {
691             static char nameBuf[128] = "";
692             sprintf(nameBuf, Engine::GetPresetName().c_str());
693             if (ImGui::InputText("Name", nameBuf, 128, ImGuiInputTextFlags_EnterReturnsTrue)) {
694                 Engine::SetPresetName(std::string(nameBuf));
695             }
696         }
697     }

```

```

695         Engine::Write();
696         saveAs = false;
697     }
698
699     if (ImGui::IsItemDeactivatedAfterEdit()) {
700         Engine::SetPresetName(std::string(nameBuf));
701         Engine::Write();
702         saveAs = false;
703     }
704 }
705
706 ImGui::EndMenuBar();
707 }
708 }

```

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 511 of file editor.cpp.

```

511                                     {
512     if (!model) return;
513
514     std::string modelName = model->GetModelName();
515     std::string textureName = model->GetTextureName();
516
517     // Setting up tree flags
518     ImGuiTreeNodeFlags node_flags = ImGuiTreeNodeFlags_SpanAvailWidth | ImGuiTreeNodeFlags_OpenOnDoubleClick
| ImGuiTreeNodeFlags_OpenOnArrow;
519     if (selected_component == CType::CModel) node_flags |= ImGuiTreeNodeFlags_Selected;
520
521     const bool model_open = ImGui::TreeNodeEx((void*)(intptr_t)CType::CModel, node_flags, "Model");
522     if (ImGui::IsItemClicked()) selected_component = CType::CModel;
523
524     // Right click behavior to delete model component from selected object
525     if (ImGui::IsItemClicked(ImGuiMouseButton_Right)) {
526         selected_component = CType::CModel;
527         ImGui::OpenPopup("DeleteModel##1");
528     }
529
530     if (ImGui::BeginPopup("DeleteModel##1")) {
531         if (ImGui::Selectable("Delete##3")) {
532             model->GetParent()->RemoveComponent<Model>();
533             selected_component = -1;
534         }
535         ImGui::EndPopup();
536     }
537
538     if (model_open) {
539         // Model that is being used
540         ImGui::Text("Model"); ImGui::SameLine(100);
541         if (ImGui::Button(modelName.c_str())) {
542             ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDialogKey##1", "Choose File", ".obj",
"./data/models/");
543         }
544
545         if (ImGuiFileDialog::Instance()->Display("ChooseFileDialogKey##1")) {

```

```

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

```

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 578 of file `editor.cpp`.

```

578                                     {
579     if (!physics) return;
580
581     glm::vec3 velocity = physics->GetVelocity();
582
583     ImGuiTreeNodeFlags node_flags = ImGuiTreeNodeFlags_SpanAvailWidth | ImGuiTreeNodeFlags_OpenOnDoubleClick
584     | ImGuiTreeNodeFlags_OpenOnArrow;
585     if (selected_component == CType::CPhysics) node_flags |= ImGuiTreeNodeFlags_Selected;
586
587     const bool physics_open = ImGui::TreeNodeEx((void*)(intptr_t)CType::CPhysics, node_flags, "Physics");
588     if (ImGui::IsItemClicked()) selected_component = CType::CPhysics;
589
590     if (ImGui::IsItemClicked(ImGuiMouseButton_Right)) {
591         selected_component = CType::CPhysics;
592         ImGui::OpenPopup("DeletePhysics##1");
593     }
594
595     if (ImGui::BeginPopup("DeletePhysics##1")) {
596         if (ImGui::Selectable("Delete##4")) {
597             physics->GetParent()->RemoveComponent<Physics>();
598             selected_component = -1;
599         }
600     }
601     ImGui::EndPopup();

```

```

600     }
601
602     if (physics_open) {
603         ImGui::Text("Velocity");
604
605         ImGui::PushItemWidth(50);
606         ImGui::SameLine(100); ImGui::InputFloat("x##1", &velocity.x);
607         ImGui::SameLine(175); ImGui::InputFloat("y##1", &velocity.y);
608         ImGui::SameLine(250); ImGui::InputFloat("z##1", &velocity.z);
609
610         ImGui::InputFloat("Mass##1", &physics->GetMassRef());
611         ImGui::PopItemWidth();
612
613         ImGui::TreePop();
614     }
615 }

```

References `Physics::GetMassRef()`, `Component::GetParent()`, `Physics::GetVelocity()`, `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 205 of file `editor.cpp`.

```

205     {
206         ImGui::Begin("Scene");
207
208         // Display all objects
209         for (int i = 0; i < (int)Object_Manager::GetSize(); ++i) {
210             if (ImGui::Selectable(Object_Manager::FindObject(i)->GetName().c_str(), selected_object == i,
211                 ImGuiSelectableFlags_AllowDoubleClick)) {
212                 if (selected_object != i) editor->selected_component = -1;
213                 selected_object = i;
214                 selected_component = -1;
215             }
216
217             // Checking for right click behavior
218             if (ImGui::IsItemClicked(ImGuiMouseButton_Right)) {
219                 if (selected_object != i) editor->selected_component = -1;
220                 selected_object = i;
221                 selected_component = -1;
222                 ImGui::OpenPopup("ObjectSettings##1");
223             }
224
225             if (ImGui::BeginPopup("ObjectSettings##1")) {
226                 // Removes selected object from scene
227                 if (ImGui::Selectable("Delete##1")) {
228                     Object_Manager::RemoveObject(selected_object);
229                     selected_object = -1;
230                     selected_component = -1;
231                 }
232
233                 // Copies selected object
234                 if (ImGui::Selectable("Copy##1")) {
235                     editor->object_to_copy = editor->selected_object;
236                 }
237
238                 // Pastes copied object into scene
239                 if (ImGui::Selectable("Paste##1")) {
240                     if (editor->object_to_copy != -1) {
241                         Object* object = new Object(*Object_Manager::FindObject(editor->selected_object));
242                         Object_Manager::AddObject(object);
243                     }
244                 }
245                 ImGui::EndPopup();
246             }
247
248             ImGui::Separator();
249         }
250     }

```

```

248     // Button to add new object to the scene
249     if (ImGui::Button("Add Object")) {
250         Object* newObject = new Object;
251         Transform* transform = new Transform;
252         transform->SetStartPosition(glm::vec3(0.f));
253         newObject->SetName("New_Object");
254         newObject->AddComponent(transform);
255
256         Object_Manager::AddObject(newObject);
257     }
258
259     ImGui::End();
260 }

```

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

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 431 of file `editor.cpp`.

```

431                                     {
432     if (!behavior) return;
433
434     // Setting up tree flags
435     ImGuiTreeNodeFlags node_flags = ImGuiTreeNodeFlags_SpanAvailWidth | ImGuiTreeNodeFlags_OpenOnDoubleClick
| ImGuiTreeNodeFlags_OpenOnArrow;
436     if (selected_component == CType::CBehavior) node_flags |= ImGuiTreeNodeFlags_Selected;
437
438     const bool scripts_open = ImGui::TreeNodeEx((void*)(intptr_t)CType::CBehavior, node_flags, "Scripts");
439     if (ImGui::IsItemClicked()) selected_component = CType::CBehavior;
440
441     // Right click behavior to delete script component from object
442     if (ImGui::IsItemClicked(ImGuiMouseButton_Right)) {
443         selected_component = CType::CBehavior;
444         ImGui::OpenPopup("DeleteScripts##1");
445     }
446
447     if (ImGui::BeginPopup("DeleteScripts##1")) {
448         if (ImGui::Selectable("Delete##2")) {
449             behavior->GetParent()->RemoveComponent<Behavior>();
450             selected_component = -1;
451         }
452         ImGui::EndPopup();
453     }
454
455     // Displays the currently attached scripts
456     if (scripts_open) {
457         std::vector<std::string>& scripts = behavior->GetScripts();
458         unsigned scriptNum = 1;
459         for (std::string& script : scripts) {
460             ImGui::Text(std::string("Script " + std::to_string(scriptNum) + ":").c_str());
461             ImGui::SameLine(100);
462             if (ImGui::Button(script.c_str())) {
463                 ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDialogKey##3", "Choose File", ".lua",
"./data/scripts/");

```

```

464         }
465
466         if (ImGuiFileDialog::Instance()->Display("ChooseFileDlgKey##3")) {
467             if (ImGuiFileDialog::Instance()->IsOk()) {
468                 std::string filePathName = ImGuiFileDialog::Instance()->GetCurrentFileName();
469                 if (!behavior->SwitchScript(scriptNum - 1, filePathName))
470                     ImGui::OpenPopup("ExistingScript##1");
471             }
472
473             ImGuiFileDialog::Instance()->Close();
474         }
475         ++scriptNum;
476     }
477
478     // Add new script to the object
479     ImGui::Indent(71);
480     if (ImGui::Button("New Script##1")) {
481         ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDlgKey##4", "Choose File", ".lua",
482         "./data/scripts/");
483
484         if (ImGuiFileDialog::Instance()->Display("ChooseFileDlgKey##4")) {
485             if (ImGuiFileDialog::Instance()->IsOk()) {
486                 std::string filePathName = ImGuiFileDialog::Instance()->GetCurrentFileName();
487                 if (!behavior->AddScript(filePathName))
488                     ImGui::OpenPopup("ExistingScript##1");
489             }
490
491             ImGuiFileDialog::Instance()->Close();
492         }
493
494         // Popup to say that the selected script to add is already attached to the object
495         if (ImGui::BeginPopup("ExistingScript##1")) {
496             ImGui::Text(std::string("Script already attached to " +
497             Object_Manager::FindObject(editor->selected_object)->GetName()).c_str(),
498             ImGui::GetFontSize() * 2);
499             ImGui::EndPopup();
500         }
501
502         ImGui::TreePop();
503     }
504 }

```

References Behavior::AddScript(), editor, Object_Manager::FindObject(), Object::GetName(), Component::GetParent(), Behavior::GetScripts(), Object::RemoveComponent(), selected_component, selected_object, 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 622 of file editor.cpp.

```

622         {
623             if (!transform) return;
624
625             glm::vec3& position = transform->GetPositionRef();
626             glm::vec3& scale = transform->GetScaleRef();
627             glm::vec3& rotation = transform->GetRotationRef();

```

```

628     glm::vec3& startPos = transform->GetStartPositionRef();
629
630     ImGuiTreeNodeFlags node_flags = ImGuiTreeNodeFlags_SpanAvailWidth | ImGuiTreeNodeFlags_OpenOnDoubleClick
| ImGuiTreeNodeFlags_OpenOnArrow;
631     if (selected_component == CType::CTransform) node_flags |= ImGuiTreeNodeFlags_Selected;
632
633     const bool transform_open = ImGui::TreeNodeEx((void*)(intptr_t)CType::CTransform, node_flags,
"Transform");
634     if (ImGui::IsItemClicked()) selected_component = CType::CTransform;
635
636     if (transform_open) {
637         ImGui::Text("Position");
638
639         ImGui::PushItemWidth(50);
640         ImGui::SameLine(100); ImGui::InputFloat("x##1", &position.x);
641         ImGui::SameLine(175); ImGui::InputFloat("y##1", &position.y);
642         ImGui::SameLine(250); ImGui::InputFloat("z##1", &position.z);
643         ImGui::PopItemWidth();
644
645         ImGui::Text("Scale");
646
647         ImGui::PushItemWidth(50);
648         ImGui::SameLine(100); ImGui::InputFloat("x##2", &scale.x);
649         ImGui::SameLine(175); ImGui::InputFloat("y##2", &scale.y);
650         ImGui::SameLine(250); ImGui::InputFloat("z##2", &scale.z);
651         ImGui::PopItemWidth();
652
653         ImGui::Text("Rotation");
654
655         ImGui::PushItemWidth(50);
656         ImGui::SameLine(100); ImGui::InputFloat("x##3", &rotation.x);
657         ImGui::SameLine(175); ImGui::InputFloat("y##3", &rotation.y);
658         ImGui::SameLine(250); ImGui::InputFloat("z##3", &rotation.z);
659         ImGui::PopItemWidth();
660
661         ImGui::Text("Start Pos");
662
663         ImGui::PushItemWidth(50);
664         ImGui::SameLine(100); ImGui::InputFloat("x##5", &startPos.x);
665         ImGui::SameLine(175); ImGui::InputFloat("y##5", &startPos.y);
666         ImGui::SameLine(250); ImGui::InputFloat("z##5", &startPos.z);
667         ImGui::PopItemWidth();
668
669         ImGui::TreePop();
670     }
671 }

```

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 357 of file `editor.cpp`.

```

357     {
358         ImGui::Begin("World Settings");
359         std::string presetName = Engine::GetPresetName();
360
361         // Allows user to change the preset that is loaded
362         ImGui::Text("Presets"); ImGui::SameLine(120);
363         if (ImGui::Button(presetName.c_str())) {
364             ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDlgKey##3", "Choose File", ".json",
"./data/json/preset/");
365         }
366
367         if (ImGuiFileDialog::Instance()->Display("ChooseFileDlgKey##3")) {
368             if (ImGuiFileDialog::Instance()->IsOk()) {
369                 std::string filePathName = ImGuiFileDialog::Instance()->GetCurrentFileName();

```



```

370         selected_object = -1;
371         Engine::Restart(filePathName);
372     }
373
374     ImGuiFileDialog::Instance()->Close();
375 }
376
377 ImGui::PushItemWidth(141);
378
379     // Strength of the light being used
380     ImGui::Text("Light Power");
381     ImGui::SameLine(120); ImGui::InputFloat("##1", &Engine::GetLightPower());
382
383     // Position of the light being used
384     ImGui::Text("Light Position");
385     ImGui::PushItemWidth(50);
386     ImGui::SameLine(120); ImGui::InputFloat("x##4", &Engine::GetLightPos().x);
387     ImGui::SameLine(195); ImGui::InputFloat("y##4", &Engine::GetLightPos().y);
388     ImGui::SameLine(270); ImGui::InputFloat("z##4", &Engine::GetLightPos().z);
389     ImGui::PopItemWidth();
390
391     // Grav const of the engine
392     ImGui::Text("Grav Const");
393     ImGui::SameLine(120); ImGui::InputDouble("##5", &Engine::GetGravConst());
394
395     ImGui::PopItemWidth();
396
397     ImGui::End();
398 }

```

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 716 of file editor.cpp.

```
716 { 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 129 of file editor.cpp.

```

129         {
130             ImGui::Render();
131             ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
132
133             if (ImGui::GetIO().ConfigFlags & ImGuiConfigFlags_ViewportsEnable) {
134                 GLFWwindow* backup_current_context = glfwGetCurrentContext();
135                 ImGui::UpdatePlatformWindows();
136                 ImGui::RenderPlatformWindowsDefault();
137                 glfwMakeContextCurrent(backup_current_context);
138             }
139     }
```

Referenced by Graphics::Render().

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

Sets selected object to invalid value.

Returns

void

Definition at line 162 of file editor.cpp.

```
162     {  
163         editor->selected_object = -1;  
164     }
```

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 146 of file editor.cpp.

```
146     {  
147         if (!editor) return;  
148  
149         ImGui_ImplOpenGL3_Shutdown();  
150         ImGui_ImplGlfw_Shutdown();  
151         ImGui::DestroyContext();  
152  
153         delete editor;  
154         editor = nullptr;  
155     }
```

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                 // Copy current selected object
99                 if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_C) == GLFW_PRESS) {
100                     if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_C) == GLFW_RELEASE) {
101                         editor->object_to_copy = editor->selected_object;
102                     }
103                 }
104                 // Paste current selected object
105                 if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_V) == GLFW_PRESS) {
106                     if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_V) == GLFW_RELEASE) {
107                         if (editor->object_to_copy != -1) {
108                             Object* object = new Object(*Object_Manager::FindObject(editor->selected_object));
109                             Object_Manager::AddObject(object);
110                         }
111                     }
112                 }
113             }
114         }
115
116         // Display the different windows
117         editor->Display_Dockspace();
118         editor->Display_Scene();
119         editor->Display_Components();
120         editor->Display_World_Settings();
121         editor->Display_Camera_Settings();
122     }

```

References Object_Manager::AddObject(), Display_Camera_Settings(), Display_Components(), Display_Dockspace(), Display_Scene(), Display_World_Settings(), editor, Object_Manager::FindObject(), Graphics::GetWindow(), object_to_copy, selected_object, 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 void [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 void [Restart](#) ()
Resets the objects in the engine.
- static void [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 171 of file `engine.cpp`.

```
171 { 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 178 of file engine.cpp.

```
178 { 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 185 of file engine.cpp.

```
185 { 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 206 of file engine.cpp.

```
206 { 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 199 of file engine.cpp.

```
199 { 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 192 of file engine.cpp.

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

References engine, and presetName.

Referenced by Editor::Display_Menu_Bar(), and Editor::Display_World_Settings().

4.5.2.7 Initialize() `void Engine::Initialize () [static]`

Initializes the engine and the systems in the engine.

Returns

void

Definition at line 41 of file engine.cpp.

```

41     {
42         // Initializing engine
43         engine = new Engine;
44         if (!engine) {
45             Trace::Message("Engine was not initialized.\n");
46             return;
47         }
48
49         // Reading settings from json
50         File_Reader settings("settings.json");
51         engine->presetName = settings.Read_String("preset");
52
53         File_Reader preset("preset/" + engine->presetName);
54         engine->gravConst = preset.Read_Double("gravConst");
55
56         engine->lightPower = 1000.f;
57         engine->lightPos = preset.Read_Vec3("lightPos");
58         if (engine->lightPos == glm::vec3(0.f)) {
59             engine->lightPos = glm::vec3(4, 4, 0);
60         }
61
62         // Initializing sub systems
63         if (!Model_Data_Manager::Initialize()) return;
64         if (!Texture_Manager::Initialize()) return;
65         if (!Camera::Initialize(settings)) return;
66         if (!Graphics::Initialize(settings)) return;
67         if (!Object_Manager::Initialize(preset)) return;
68         if (!Random::Initialize()) return;
69         if (!Editor::Initialize()) return;
70
71         // Setting up variables used for dt
72         engine->currentTime = std::chrono::steady_clock::now();
73         engine->accumulator = 0.f;
74         engine->time = 0.f;
75         engine->isRunning = true;
76     }

```

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_String(), File_Reader::Read_Vec3(), and time.

Referenced by main().

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

Resets the objects in the engine.

Returns

void

Definition at line 132 of file engine.cpp.

```

132     {
133         // Removing all current objects
134         Object_Manager::Shutdown();
135         Editor::Reset();
136
137         // Initializing object manager
138         File_Reader settings("settings.json");

```

```

139     engine->presetName = settings.Read_String("preset");
140
141     File_Reader preset("preset/" + engine->presetName);
142     engine->gravConst = preset.Read_Double("gravConst");
143     if (!Object_Manager::Initialize(preset)) return;
144 }

```

References engine, gravConst, Object_Manager::Initialize(), presetName, File_Reader::Read_Double(), 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] void Engine::Restart (
 std::string presetName) [static]

Resets the engine to the given preset.

Parameters

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

Returns

void

Definition at line 152 of file engine.cpp.

```

152 {
153     // Removing all current objects
154     Object_Manager::Shutdown();
155     Editor::Reset();
156
157     // Initializing object manager
158     File_Reader settings("settings.json");
159     engine->presetName = presetName;
160
161     File_Reader preset("preset/" + engine->presetName);
162     engine->gravConst = preset.Read_Double("gravConst");
163     if (!Object_Manager::Initialize(preset)) return;
164 }

```

References engine, gravConst, Object_Manager::Initialize(), presetName, File_Reader::Read_Double(), 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 230 of file engine.cpp.

```
230 {  
231     engine->presetName = presetName_;  
232 }
```

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 110 of file engine.cpp.

```
110 {  
111     if (!engine) return;  
112  
113     // Shutdown sub systems  
114     Editor::Shutdown();  
115     Random::Shutdown();  
116     Object_Manager::Shutdown();  
117     Graphics::Shutdown();  
118     Camera::Shutdown();  
119     Texture_Manager::Shutdown();  
120     Model_Data_Manager::Shutdown();  
121  
122     // Delete engine object  
123     delete engine;  
124     engine = nullptr;  
125 }
```

References engine, Random::Shutdown(), Editor::Shutdown(), Model_Data_Manager::Shutdown(), Texture_Manager::Shutdown(), Object_Manager::Shutdown(), Camera::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 84 of file engine.cpp.

```

84     {
85         // Calculating dt
86         engine->newTime = std::chrono::steady_clock::now();
87         engine->timeTaken = engine->newTime - engine->currentTime;
88         engine->deltaTime = float(engine->timeTaken.count()) *
89             std::chrono::steady_clock::period::num / std::chrono::steady_clock::period::den;
90         engine->currentTime = engine->newTime;
91         engine->accumulator += engine->deltaTime;
92
93         Editor::Update();
94         Camera::Update();
95         // Only called when it is time (fixed time step)
96         while (engine->accumulator >= engine->dt) {
97             // Update objects
98             Object_Manager::Update();
99             // Update dt related variables
100             engine->accumulator -= engine->dt;
101             engine->time += engine->dt;
102         }
103     }

```

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 214 of file engine.cpp.

```

214     {
215         File_Writer writer;
216
217         writer.Write_Value("gravConst", engine->gravConst);
218         writer.Write_Vec3("lightPos", engine->lightPos);
219         Object_Manager::Write(writer);
220
221         writer.Write_File(std::string ("preset/" + engine->presetName));
222     }

```

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

- [File_Reader](#) (std::string filename)
Creates [File_Reader](#) object and reads given file.
- void [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 Constructor & Destructor Documentation

4.6.2.1 File_Reader() `File_Reader::File_Reader (std::string filename)`

Creates [File_Reader](#) object and reads given file.

Parameters

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

Definition at line 30 of file file_reader.cpp.

```
30                                     {
31     Read_File(filename);
32 }
```

4.6.3 Member Function Documentation

4.6.3.1 Read_Behavior_Name() `std::string File_Reader::Read_Behavior_Name (std::string valueName)`

Reads the name of the behavior.

Parameters

<i>valueName</i>	Behavior to read
------------------	------------------

Returns

`std::string` Name of the behavior

Definition at line 210 of file file_reader.cpp.

```
210                                     {
211     // Checking if value exists
212     if (!root["behaviors"].HasMember(valueName.c_str())) {
213         return std::string("");
214     }
215
216     return root["behaviors"][valueName.c_str()].GetString();
217 }
```

Referenced by Behavior::Read().

4.6.3.2 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 101 of file file_reader.cpp.

```

101     {
102         // Checking if the value is a bool
103         if (!root.HasMember(valueName.c_str())) {
104             return false;
105         }
106         return root[valueName.c_str()].GetBool();
107     }

```

4.6.3.3 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 129 of file file_reader.cpp.

```

129     {
130         // Checking if the value is a double (has decimal)
131         if (!root.HasMember(valueName.c_str())) {
132             return false;
133         }
134         return root[valueName.c_str()].GetDouble();
135     }

```

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

4.6.3.4 Read_File() void 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
-----------------	-----------------------------

Definition at line 39 of file file_reader.cpp.


```

39                                     {
40     // Opening the json file
41     std::string fileToOpen = "data/json/" + filename;
42     FILE* file = fopen(fileToOpen.c_str(), "r");
43
44     char buffer[65536];
45     FileReadStream stream(file, buffer, sizeof(buffer));
46     root.ParseStream<0, UTF8<>, FileReadStream>(stream);
47
48     fclose(file);
49 }

```

4.6.3.5 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 115 of file file_reader.cpp.

```

115                                     {
116     // Checking if the value is a double (has decimal)
117     if (!root.HasMember(valueName.c_str())) {
118         return 0.f;
119     }
120     return root[valueName.c_str()].GetFloat();
121 }

```

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

4.6.3.6 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 57 of file file_reader.cpp.

```

57                                     {
58     // Checking if the value is an int
59     if (!root.HasMember(valueName.c_str())) {
60         return 0;
61     }
62     return root[valueName.c_str()].GetInt();
63 }

```

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

4.6.3.7 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 143 of file `file_reader.cpp`.

```

143                                     {
144     // Checking if the value exists
145     if (!root.HasMember(valueName.c_str())) {
146         return std::string("");
147     }
148     if (!root[valueName.c_str()].HasMember("objectName")) {
149         return std::string("");
150     }
151     return root[valueName.c_str()]["objectName"].GetString();
152 }
153 }

```

Referenced by Object_Manager::ReadList().

4.6.3.8 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 179 of file file_reader.cpp.

```
179                                     {
180     if (!root[valueName.c_str()].HasMember("position")) {
181         return glm::vec3(0.f, 0.f, 0.f);
182     }
183
184     Value& array = root[valueName.c_str()]["position"];
185     return glm::vec3(array[0].GetFloat(), array[1].GetFloat(), array[2].GetFloat());
186 }
```

Referenced by Object_Manager::ReadList().

4.6.3.9 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 194 of file file_reader.cpp.

```
194                                     {
195     // Checking if value exists
196     if (!root[valueName.c_str()].HasMember("scale")) {
197         return glm::vec3(0.f, 0.f, 0.f);
198     }
199
200     Value& array = root[valueName.c_str()]["scale"];
201     return glm::vec3(array[0].GetFloat(), array[1].GetFloat(), array[2].GetFloat());
202 }
```

Referenced by Object_Manager::ReadList().

4.6.3.10 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 161 of file file_reader.cpp.

```

161                                     {
162     // Checking if the value exists
163     if (!root.HasMember(valueName.c_str())) {
164         return std::string("");
165     }
166     if (!root[valueName.c_str()].HasMember("templateName")) {
167         return std::string("");
168     }
169     return root[valueName.c_str()]["templateName"].GetString();
171 }

```

Referenced by Object_Manager::ReadList().

4.6.3.11 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 71 of file file_reader.cpp.

```

71                                     {
72     // Checking if the value is a std::string
73     if (!root.HasMember(valueName.c_str())) {
74         return std::string("");
75     }
76     return root[valueName.c_str()].GetString();
77 }

```

Referenced by Model_Data_Manager::Get(), Texture_Manager::Get(), Engine::Initialize(), Shader::Initialize(), Model_Data::Load(), Object::ReRead(), and Engine::Restart().

4.6.3.12 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 86 of file file_reader.cpp.

```

86                                     {
87     // Checking if the value is an array
88     if (!root.HasMember(valueName.c_str())) {
89         return glm::vec3(0.f, 0.f, 0.f);
90     }
91     return glm::vec3(root[valueName.c_str()][0].GetFloat(), root[valueName.c_str()][1].GetFloat(),
92                     root[valueName.c_str()][2].GetFloat());

```

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 = "data/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::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

<code>settings</code>	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         Editor::Render();
193         glfwSwapBuffers(graphics->window);
194     }
195
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()`.

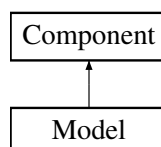
The documentation for this class was generated from the following files:

- [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 Model 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 148 of file model.cpp.

```
148     {
149     return CType::CModel;
150 }
```

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

Returns the filename of the current model.

Returns

std::string

Definition at line 121 of file model.cpp.

```
121     {
122     if (!data) return "no model";
123     return data->GetModelName();
124 }
```

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 141 of file model.cpp.

```
141 { 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 131 of file model.cpp.

```
131 {
132     if (!texture) return "no texture";
133     return texture->GetTextureName();
134 }
```

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

<code>reader</code>	File_reader object that contains Model 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 { data = Model_Data_Manager::Get(modelName); }
```

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 114 of file model.cpp.

```
114 { texture = Texture_Manager::Get(textureName); }
```

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 219 of file model_data.cpp.

```

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

```

```

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

```

References Texture::Display(), Shader::GetLightId(), Engine::GetLightPos(), Engine::GetLightPower(), Shader::GetLightPowerId(), Shader::GetMatrixId(), Shader::GetModelMatrixId(), Transform::GetPosition(), 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 291 of file model_data.cpp.

```
291 { 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         // Setting the name of the file (used in model_data_manager)
97         modelName = modelName_;
98
99         // Creating variables for reading
100         std::vector<unsigned> vertex_indices, uv_indices, normal_indices;
101         std::vector<glm::vec3> temp_vertices;
102         std::vector<glm::vec2> temp_uvs;
103         std::vector<glm::vec3> temp_normals;
104
105         // Opening the file
106         std::string fileToOpen = "data/models/" + modelName;
107         FILE* file = fopen(fileToOpen.c_str(), "r");
108         if (!file) {
109             Trace::Message("File '" + modelName + "' was not successfully opened.\n");
110             return false;
111         }
112
113         // Until the whole file is read
114         while (true) {
115             char line_header[256];
116
117             // Getting next line of the file
118             int res = fscanf(file, "%s", line_header);
119             if (res == EOF) break;
120
121             // Checking for which data needs to be read in
122             if (strcmp(line_header, "v") == 0) {
123                 glm::vec3 vertex;
124                 fscanf(file, "%f %f %f\n", &vertex.x, &vertex.y, &vertex.z);
125                 temp_vertices.emplace_back(vertex);
126                 continue;
127             }
128
129             if (strcmp(line_header, "vt") == 0) {
130                 glm::vec2 uv;
131                 fscanf(file, "%f %f\n", &uv.x, &uv.y);
132                 temp_uvs.emplace_back(uv);
133                 continue;
134             }
135
136             if (strcmp(line_header, "vn") == 0) {
137                 glm::vec3 normal;
138                 fscanf(file, "%f %f %f\n", &normal.x, &normal.y, &normal.z);
139                 temp_normals.emplace_back(normal);
140                 continue;
141             }
142
143             if (strcmp(line_header, "f") == 0) {
144                 // Connecting face to previous read vertices, uvs, and normals
145                 unsigned vertex_index[3], uv_index[3], normal_index[3];
146                 int matches = fscanf(file, "%d/%d/%d %d/%d/%d %d/%d/%d\n", &vertex_index[0], &uv_index[0],
147                                     &normal_index[0],
148                                     &vertex_index[1], &uv_index[1], &normal_index[1], &vertex_index[2], &uv_index[2],
149                                     &normal_index[2]);
150
151                 // Expects models split into triangles
152                 if (matches != 9) {
153                     Trace::Message("File is incompatible with this parser. Export using different settings.");
154                     return false;
155                 }
156
157                 // Setting vertices for current face
158                 vertices.emplace_back((temp_vertices[vertex_index[0] - 1]).x);
159                 vertices.emplace_back((temp_vertices[vertex_index[0] - 1]).y);
160                 vertices.emplace_back((temp_vertices[vertex_index[0] - 1]).z);
161
162                 vertices.emplace_back((temp_vertices[vertex_index[1] - 1]).x);
163                 vertices.emplace_back((temp_vertices[vertex_index[1] - 1]).y);
164                 vertices.emplace_back((temp_vertices[vertex_index[1] - 1]).z);
165
166                 vertices.emplace_back((temp_vertices[vertex_index[2] - 1]).x);
167                 vertices.emplace_back((temp_vertices[vertex_index[2] - 1]).y);
168                 vertices.emplace_back((temp_vertices[vertex_index[2] - 1]).z);
169
170                 // Setting uvs for current face
171                 uvs.emplace_back((temp_uvs[uv_index[0] - 1]).x);
172                 uvs.emplace_back((temp_uvs[uv_index[0] - 1]).y);
173
174                 uvs.emplace_back((temp_uvs[uv_index[1] - 1]).x);
175                 uvs.emplace_back((temp_uvs[uv_index[1] - 1]).y);
176
177                 uvs.emplace_back((temp_uvs[uv_index[2] - 1]).x);
178                 uvs.emplace_back((temp_uvs[uv_index[2] - 1]).y);
179
180                 // Setting normals for current face
181                 normals.emplace_back((temp_normals[normal_index[0] - 1]).x);
182                 normals.emplace_back((temp_normals[normal_index[0] - 1]).y);
183                 normals.emplace_back((temp_normals[normal_index[0] - 1]).z);
184
185                 normals.emplace_back((temp_normals[normal_index[1] - 1]).x);
186                 normals.emplace_back((temp_normals[normal_index[1] - 1]).y);
187                 normals.emplace_back((temp_normals[normal_index[1] - 1]).z);
188
189                 normals.emplace_back((temp_normals[normal_index[2] - 1]).x);
190                 normals.emplace_back((temp_normals[normal_index[2] - 1]).y);
191                 normals.emplace_back((temp_normals[normal_index[2] - 1]).z);
192
193                 // Creating the face
194                 faces.emplace_back(vertex_indices, uv_indices, normal_indices);
195             }
196         }
197     }
198 }

```

```

173         uvs.emplace_back((temp_uvs[uv_index[1] - 1]).y);
174
175         uvs.emplace_back((temp_uvs[uv_index[2] - 1]).x);
176         uvs.emplace_back((temp_uvs[uv_index[2] - 1]).y);
177
178         // Setting normals for current face
179         normals.emplace_back((temp_normals[normal_index[0] - 1]).x);
180         normals.emplace_back((temp_normals[normal_index[0] - 1]).y);
181         normals.emplace_back((temp_normals[normal_index[0] - 1]).z);
182
183         normals.emplace_back((temp_normals[normal_index[1] - 1]).x);
184         normals.emplace_back((temp_normals[normal_index[1] - 1]).y);
185         normals.emplace_back((temp_normals[normal_index[1] - 1]).z);
186
187         normals.emplace_back((temp_normals[normal_index[2] - 1]).x);
188         normals.emplace_back((temp_normals[normal_index[2] - 1]).y);
189         normals.emplace_back((temp_normals[normal_index[2] - 1]).z);
190     }
191 }
192
193 // Bind vertex data to buffers
194 glGenBuffers(1, &vertexbuffer);
195 glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
196 glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(float), &vertices[0], GL_STATIC_DRAW);
197
198 // Bind uv data to buffers
199 glGenBuffers(1, &uvbuffer);
200 glBindBuffer(GL_ARRAY_BUFFER, uvbuffer);
201 glBufferData(GL_ARRAY_BUFFER, uvs.size() * sizeof(float), &uvs[0], GL_STATIC_DRAW);
202
203 // Bind normals data to buffers
204 glGenBuffers(1, &normalbuffer);
205 glBindBuffer(GL_ARRAY_BUFFER, normalbuffer);
206 glBufferData(GL_ARRAY_BUFFER, normals.size() * sizeof(float), &normals[0], GL_STATIC_DRAW);
207
208 return true;
209 }

```

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

<i>reader</i>	File_Reader 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         data->Load(modelName);
80         model_data_manager->models.emplace_back(data);
81
82         return data;
83     }

```

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 91 of file model_data_manager.cpp.

```

91     {
92     if (!model_data_manager) return;
93
94     // Deleting all of the Model_Data objects
95     for (Model_Data* model_data : model_data_manager->models) {
96         if (!model_data) continue;
97
98         delete model_data;
99         model_data = nullptr;
100     }
101
102     delete model_data_manager;
103     model_data_manager = nullptr;
104 }
```

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](#) (std::string filename)
Creating object from a file.
- [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.
- `void Read (std::string objectFilename)`

Reads object from file. This includes the components of an object.
- `void 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()` [1/3] `Object::Object ()`

Default constructor.

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

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

Referenced by `Clone()`.

4.12.2.2 `Object()` [2/3] `Object::Object (` `const Object & other)`

Copy constructor.

Parameters

<i>other</i>	Object 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()`.

4.12.2.3 Object() [3/3] `Object::Object (std::string filename)`

Creating object from a file.

Parameters

<i>filename</i>	Name of file used to create object
-----------------	------------------------------------

Definition at line 73 of file `object.cpp`.

```

73     {
74         Read(filename);
75     }

```

References `Read()`.

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 104 of file `object.cpp`.

```

104     {
105         component->SetParent(this);
106         components.emplace(component->GetCType(), component);
107     }

```

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 272 of file object.cpp.

```

272     {
273         Behavior* behavior = GetComponent<Behavior>();
274         Model* model = GetComponent<Model>();
275         Physics* physics = GetComponent<Physics>();
276
277         if (behavior) {
278             delete behavior;
279             behavior = nullptr;
280         }
281         if (model) {
282             delete model;
283             model = nullptr;
284         }
285         if (physics) {
286             delete physics;
287             physics = nullptr;
288         }
289     }

```

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

Clones this object.

Returns

Object*

Definition at line 82 of file object.cpp.

```

82     {
83         return new Object(*this);
84     }

```

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 45 of file object.hpp.

```

45         {
46             // Searching for component using the type (enum as int)
47             auto found = components.find(T::GetCType());
48             if (found == components.end()) {
49                 return nullptr;
50             }
51             // Cast found component into correct type
52             return (T*) found->second;
53         }

```

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>	Component class to return
----------	---------------------------

Parameters

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

Returns

T* Pointer to the component

Definition at line 97 of file object.hpp.

```

97         {
98             // Searching for component using the type (enum as int)
99             auto found = components.find(T::GetCType());
100             if (found == components.end()) {
101                 return nullptr;
102             }
103             // Cast found component into correct type
104             return (T*) found->second;
105         }

```

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 264 of file object.cpp.

```
264                                     {
265     return components;
266 }
```

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 121 of file object.cpp.

```
121 { return id; }
```

References id.

Referenced by [Object_Manager::CheckName\(\)](#), [Behavior::ClassSetup\(\)](#), 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 137 of file object.cpp.

```
137 { return name; }
```

References name.

Referenced by [Object_Manager::CheckName\(\)](#), [Editor::Display_Scripts\(\)](#), [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 144 of file `object.cpp`.

```
144 { 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 158 of file `object.cpp`.

```
158 { return templateName; }
```

References `templateName`.

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

4.12.3.11 Read() `void Object::Read (std::string objectFilename)`

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

Parameters

<i>objectFilename</i>	
-----------------------	--

Definition at line 165 of file `object.cpp`.

```
165                                     {
166     // Getting data from file
167     File_Reader object_reader("objects/" + objectFilename);
168
169     // Reading Behavior component form file
170     Behavior* object_behavior = new Behavior(object_reader);
171     AddComponent(object_behavior);
```



```

172
173     // Reading Model component from file
174     Model* object_model = new Model(object_reader);
175     AddComponent(object_model);
176
177     // Reading Physics component from file
178     Physics* object_physics = new Physics(object_reader);
179     AddComponent(object_physics);
180
181     // Reading Transform component from file
182     Transform* object_transform = new Transform(object_reader);
183     AddComponent(object_transform);
184 }

```

References AddComponent().

Referenced by Object().

4.12.3.12 RemoveComponent() `template<typename T >`
`void Object::RemoveComponent () [inline]`

Removes component from object.

Template Parameters

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

Definition at line 61 of file object.hpp.

```

61     {
62         // Searching for component using the type (enum as int)
63         auto found = components.find(T::GetCType());
64         if (found == components.end()) return;
65         // Delete component
66         delete found->second;
67         found->second = nullptr;
68         // Remove pointer from map
69         components.erase(found->first);
70     }

```

References components.

Referenced by Editor::Display_Model(), Editor::Display_Physics(), and Editor::Display_Scripts().

4.12.3.13 ReRead() `void Object::ReRead (`
`std::string objectFilename)`

Reading data into object that already exists.

Parameters

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

Definition at line 191 of file object.cpp.

```

191                                     {
192     // Getting data from file
193     File_Reader object_reader("objects/" + objectFilename);
194
195     if (name.compare("") == 0)
196         SetName(object_reader.Read_String("name"));
197
198     templateName = objectFilename;
199
200     // Reading Model component from file
201     Model* object_model = GetComponent<Model>();
202     if (!object_model) {
203         object_model = new Model;
204         AddComponent(object_model);
205     }
206     object_model->Read(object_reader);
207
208     // Reading Physics component from file
209     Physics* object_physics = GetComponent<Physics>();
210     if (!object_physics) {
211         object_physics = new Physics;
212         AddComponent(object_physics);
213     }
214     object_physics->Read(object_reader);
215
216     // Reading Transform component from file
217     Transform* object_transform = GetComponent<Transform>();
218     if (!object_transform) {
219         object_transform = new Transform;
220         AddComponent(object_transform);
221     }
222     object_transform->Read(object_reader);
223
224     // Reading Behavior component form file
225     Behavior* object_behavior = GetComponent<Behavior>();
226     if (object_behavior) object_behavior->Clear();
227     if (!object_behavior) {
228         object_behavior = new Behavior;
229         AddComponent(object_behavior);
230     }
231     object_behavior->Read(object_reader);
232     object_behavior->SetupClassesForLua();
233 }

```

References AddComponent(), Behavior::Clear(), name, Behavior::Read(), Model::Read(), Transform::Read(), Physics::Read(), 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

<i>id</i> ↔	Position in Object_Manager
<i>_</i> ↔	

Definition at line 114 of file object.cpp.

```
114 { 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 128 of file object.cpp.

```
128 {
129     name = Object_Manager::CheckName(name_, id);
130 }
```

References Object_Manager::CheckName(), and name.

Referenced by Behavior::ClassSetup(), Editor::Display_Scene(), Object(), 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 151 of file object.cpp.

```
151 { templateName = templateName_; }
```

References templateName.

Referenced by Object().

4.12.3.17 Update() `void Object::Update ()`

Updates object (only physics for now)

Definition at line 90 of file object.cpp.

```
90 {
91     Behavior* behavior = GetComponent<Behavior>();
92     if (behavior)
93         behavior->Update();
94     Physics* physics = GetComponent<Physics>();
95     if (physics)
96         physics->Update();
97 }
```

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 239 of file object.cpp.

```

239     {
240         File_Writer object_writer;
241         object_writer.Write_String("name", name);
242         templateName = name + ".json";
243
244         Model* object_model = GetComponent<Model>();
245         if (object_model) object_model->Write(object_writer);
246
247         Transform* object_transform = GetComponent<Transform>();
248         if (object_transform) object_transform->Write(object_writer);
249
250         Physics* object_physics = GetComponent<Physics>();
251         if (object_physics) object_physics->Write(object_writer);
252
253         Behavior* object_behavior = GetComponent<Behavior>();
254         if (object_behavior) object_behavior->Write(object_writer);
255
256         object_writer.Write_File(std::string("objects/" + name + ".json"));
257     }

```

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 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>	Object 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\(\)](#), [ReadList\(\)](#), and [Editor::Update\(\)](#).

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 152 of file `object_manager.cpp`.

```

152                                     {
153     // Checking if the name matches any other objects
154     int objWithName = 0;
155     for (Object* objToCheck : object_manager->objects) {
156         if (id != -1 && objToCheck->GetId() == id) continue;
157         if (objToCheck->GetName().find(objectName) != std::string::npos)
158             ++objWithName;
159     }
160
161     // Updating the name
162     if (objWithName > 0)
163         return objectName + "_" + std::to_string(objWithName);
164
165     return objectName;
166 }
```

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

Referenced by `Object::SetName()`.

4.13.2.3 FindObject() `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()`, `Editor::Display_↵Scripts()`, `Graphics::Render()`, `Shutdown()`, `Editor::Update()`, `Update()`, and `Physics::UpdateGravity()`.

4.13.2.4 `GetSize()` `unsigned Object_Manager::GetSize () [static]`

Gets the size of the `object_manager` object list.

Returns

unsigned Size of object list

Definition at line 74 of file `object_manager.cpp`.

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

References `object_manager`, and `objects`.

Referenced by `Editor::Display_Scene()`, `Graphics::Render()`, and `Physics::UpdateGravity()`.

4.13.2.5 `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()`.

4.13.2.6 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 112 of file `object_manager.cpp`.

```

112                                     {
113     // Track which object we are on
114     unsigned object_num = 0;
115
116     // Reads objects until there is a failed read
117     while (true) {
118         // Getting the name of the objects file
119         std::string object_name = preset.Read_Object_Name("object_" + std::to_string(object_num));
120         std::string template_name = preset.Read_Object_Template_Name("object_" +
121             std::to_string(object_num));
122         if (template_name.compare("") == 0) break;
123
124         // Constructing the object
125         Object* object = new Object(template_name);
126         object->SetName(object_name);
127         object->SetTemplateName(template_name);
128         // Reading in the objects position
129         glm::vec3 position = preset.Read_Object_Position("object_" + std::to_string(object_num));
130         glm::vec3 scale = preset.Read_Object_Scale("object_" + std::to_string(object_num));
131         Transform* transform = object->GetComponent<Transform>();
132         transform->SetPosition(position);
133         transform->SetStartPosition(position);
134         transform->SetScale(scale);
135         Behavior* behavior = object->GetComponent<Behavior>();
136         behavior->SetupClassesForLua();
137
138         // Adding the object to the manager
139         AddObject(object);
140         ++object_num;
141     }
142 }

```

References `AddObject()`, `File_Reader::Read_Object_Name()`, `File_Reader::Read_Object_Position()`, `File_Reader::Read_Object_Scale()`, `File_Reader::Read_Object_Template_Name()`, `Transform::SetPosition()`, `Transform::SetScale()`, `Transform::SetStartPosition()`, and `Behavior::SetupClassesForLua()`.

Referenced by `Initialize()`.

4.13.2.7 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 174 of file object_manager.cpp.

```

174         {
175             if (id >= (int)object_manager->objects.size()) return;
176             Object* objectToDelete = object_manager->objects[id];
177
178             // Moves all the objects to the right of one being deleted to the left
179             unsigned offset = 0;
180             for (unsigned objectNum = id + 1; objectNum < object_manager->objects.size(); ++objectNum) {
181                 Object* objectToSwitch = object_manager->objects[objectNum];
182                 object_manager->objects[id + offset] = objectToSwitch;
183                 objectToSwitch->SetId(id + offset++);
184             }
185
186             // Deleting the object
187             delete objectToDelete;
188             objectToDelete = nullptr;
189             object_manager->objects.pop_back();
190     }
```

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

Referenced by Editor::Display_Scene().

4.13.2.8 Shutdown() void Object_Manager::Shutdown () [static]

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

Returns

void

Definition at line 92 of file object_manager.cpp.

```

92         {
93             if (!object_manager) return; // If the object_manager doesn't exist
94
95             // Deleting each object in the manager
96             for (unsigned i = 0; i < object_manager->objects.size(); ++i) {
97                 Object* object = object_manager->FindObject(i);
98                 if (object)
99                     delete object;
100             }
101
102             // Deleting the manager
103             delete object_manager;
104             object_manager = nullptr;
105     }
```

References FindObject(), object_manager, and objects.

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

4.13.2.9 Update() `void Object_Manager::Update () [static]`

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

Returns

void

Definition at line 81 of file `object_manager.cpp`.

```
81     {  
82         for (unsigned i = 0; i < object_manager->objects.size(); ++i) {  
83             object_manager->FindObject(i)->Update();  
84         }  
85     }
```

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

Referenced by `Engine::Update()`.

4.13.2.10 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 198 of file `object_manager.cpp`.

```
198     {  
199         for (Object* object : object_manager->objects) {  
200             writer.Write_Object_Data(object);  
201         }  
202     }
```

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

Referenced by `Engine::Write()`.

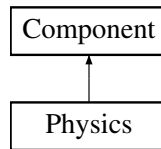
The documentation for this class was generated from the following files:

- [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 [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.
- 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)), 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)), 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 255 of file `physics.cpp`.

```
255 {  
256     return CType::CPhysics;  
257 }
```

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 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.

Referenced by Editor::Display_Physics().

4.14.3.12 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().

4.14.3.13 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 231 of file physics.cpp.

```
231 {
232     initialAcceleration = reader.Read_Vec3("acceleration");
233     initialVelocity = reader.Read_Vec3("velocity");
234     SetAcceleration(initialAcceleration);
235     SetVelocity(initialVelocity);
236     SetMass(reader.Read_Float("mass"));
237 }
```

References initialAcceleration, initialVelocity, File_Reader::Read_Float(), File_Reader::Read_Vec3(), SetAcceleration(), SetMass(), and SetVelocity().

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

4.14.3.14 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().

4.14.3.15 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().

4.14.3.16 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().

4.14.3.17 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().

4.14.3.18 Update() void Physics::Update ()

Updates the physics of the object.

Definition at line 173 of file physics.cpp.

```
173 {
174     // Finding the acceleration of the object using F=ma
175     acceleration = forces / mass;
176
177     // Updating velocity
178     velocity += (acceleration * Engine::GetDt());
179
180     // Updating position
181     Transform* transform = GetParent()->GetComponent<Transform>();
182     glm::vec3 position = transform->GetPosition();
183     transform->SetOldPosition(position);
184     position = (velocity * Engine::GetDt()) + position;
185     transform->SetPosition(position);
186
187     // Resetting the forces acting on the object
188     forces = glm::vec3(0.f, 0.f, 0.f);
189 }
```

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

Referenced by Object::Update().

4.14.3.19 UpdateGravity() void Physics::UpdateGravity ()

Calculates the gravitational pull each object has on each other.

Definition at line 195 of file physics.cpp.

```
195 {
196     // Gets the needed components for the current object
197     Object* object = GetParent();
198     Transform* transform = object->GetComponent<Transform>();
199     Physics* physics = object->GetComponent<Physics>();
200     glm::vec3 position = transform->GetPosition();
201
202     // For each object
203     for (unsigned i = 0; i < Object_Manager::GetSize(); ++i) {
204         if ((int)i == object->GetId()) continue;
205         // Gets needed components for the object being checked
206         Object* other = Object_Manager::FindObject(i);
207         Physics* otherPhysics = other->GetComponent<Physics>();
208         Transform* otherTransform = other->GetComponent<Transform>();
```

```

209     glm::vec3 otherPosition = otherTransform->GetPosition();
210     // Finding the distance between the objects
211     double distance = sqrt(pow(double(otherPosition.x - position.x), 2.0) +
212         pow(double(otherPosition.y - position.y), 2.0) +
213         pow(double(otherPosition.z - position.z), 2.0));
214     // Calculating the force the objects apply on each other
215     double magnitude = Engine::GetGravConst() * ((physics->mass * otherPhysics->mass)) / pow(distance,
216         2.0);
217     // Getting the direction (normalized)
218     glm::vec3 direction = otherPosition - position;
219     glm::vec3 normDirection = glm::normalize(direction);
220     // Applying gravitational force to normalized direction
221     glm::vec3 force = normDirection * float(magnitude);
222     // Adding the gravitational force to the forces on object
223     physics->AddForce(force);
224 }

```

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

Referenced by `Behavior::ClassSetup()`.

4.14.3.20 Write() `void Physics::Write (`
`File_Writer & writer)`

Gives physics data to the writer object.

Parameters

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

Definition at line 244 of file `physics.cpp`.

```

244     {
245     writer.Write_Vec3("acceleration", initialAcceleration);
246     writer.Write_Vec3("velocity", initialVelocity);
247     writer.Write_Value("mass", mass);
248 }

```

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             return true;  
32         }  
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("src/shaders/" + settings.Read_String("vertexShader") + ".glsl",
40              "src/shaders/" + 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.
- void [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](#) (std::string filename)
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 23 of file texture.cpp.

```
23     {  
24         glDeleteTextures(1, &textureNum);  
25     }
```

References [textureNum](#).

4.17.3 Member Function Documentation

4.17.3.1 Display() `void Texture::Display ()`

Setup texture for drawing.

Definition at line 43 of file texture.cpp.

```
43     {  
44         if (!hasBeenSet) return;  
45     }  
46     glActiveTexture(GL_TEXTURE0);  
47     glBindTexture(GL_TEXTURE_2D, textureNum);  
48     glUniform1i(textureId, 0);  
49 }
```

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 56 of file texture.cpp.

```
56 { 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 63 of file texture.cpp.

```
63 { return textureNum; }
```

References `textureNum`.

4.17.3.4 Load() `void Texture::Load (std::string textureName_)`

Loads in texture with given filename.

Parameters

<i>textureName_</i>	Filename of texture
---------------------	---------------------

Definition at line 32 of file texture.cpp.

```

32                                     {
33     textureName = textureName_;
34     textureNum = Texture::LoadDDS("data/textures/" + textureName);
35     textureId = glGetUniformLocation(Shader::GetProgram(), "myTextureSampler");
36     hasBeenSet = true;
37 }
```

References Shader::GetProgram(), hasBeenSet, LoadDDS(), textureId, textureName, and textureNum.

Referenced by Texture_Manager::Get().

4.17.3.5 LoadDDS() GLuint Texture::LoadDDS (std::string imagepath) [static], [private]

Loads in the given dds file.

Parameters

<i>imagepath</i>	DDS filename
------------------	--------------

Returns

GLuint

Definition at line 74 of file texture.cpp.

```

74                                     {
75     unsigned char header[124];
76     FILE *fp;
77
78     // Opening the file
79     fp = fopen(imagepath.c_str(), "rb");
80     if (fp == nullptr)
81         return 0;
82
83     // Making sure it is a dds
84     char filecode[4];
85     fread(filecode, 1, 4, fp);
86     if (strncmp(filecode, "DDS ", 4) != 0) {
87         fclose(fp);
88         return 0;
89     }
90
91     // Getting the surface description
92     fread(&header, 124, 1, fp);
93
94     unsigned int height = *(unsigned int*)&(header[8]);
95     unsigned int width = *(unsigned int*)&(header[12]);
96     unsigned int linearSize = *(unsigned int*)&(header[16]);
97     unsigned int mipMapCount = *(unsigned int*)&(header[24]);
98     unsigned int fourCC = *(unsigned int*)&(header[80]);
99
100 }
```

```

101     unsigned char * buffer;
102     unsigned int bufsize;
103
104     bufsize = mipMapCount > 1 ? linearSize * 2 : linearSize;
105     buffer = (unsigned char*)malloc(bufsize * sizeof(unsigned char));
106     fread(buffer, 1, bufsize, fp);
107
108     // Close the file
109     fclose(fp);
110
111     unsigned int format;
112     switch(fourCC) {
113         case FOURCC_DXT1:
114             format = GL_COMPRESSED_RGBA_S3TC_DXT1_EXT;
115             break;
116         case FOURCC_DXT3:
117             format = GL_COMPRESSED_RGBA_S3TC_DXT3_EXT;
118             break;
119         case FOURCC_DXT5:
120             format = GL_COMPRESSED_RGBA_S3TC_DXT5_EXT;
121             break;
122         default:
123             free(buffer);
124             return 0;
125     }
126
127     GLuint textureID;
128     glGenTextures(1, &textureID);
129
130     glBindTexture(GL_TEXTURE_2D, textureID);
131     glPixelStorei(GL_UNPACK_ALIGNMENT,1);
132
133     unsigned int blockSize = (format == GL_COMPRESSED_RGBA_S3TC_DXT1_EXT) ? 8 : 16;
134     unsigned int offset = 0;
135
136     for (unsigned int level = 0; level < mipMapCount && (width || height); ++level) {
137         unsigned int size = ((width+3)/4)*((height+3)/4)*blockSize;
138         glCompressedTexImage2D(GL_TEXTURE_2D, level, format, width, height,
139                               0, size, buffer + offset);
140
141         offset += size;
142         width /= 2;
143         height /= 2;
144
145         if(width < 1) width = 1;
146         if(height < 1) height = 1;
147     }
148
149     free(buffer);
150
151     return textureID;
152 }
153 }

```

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>	File_Reader 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         texture->Load(textureName);
82         texture_manager->textures.emplace_back(texture);
83
84         return texture;
85     }

```

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 92 of file texture_manager.cpp.

```
92     {  
93         if (!texture_manager) return;  
94  
95         for (Texture* texture : texture_manager->textures) {  
96             if (!texture) continue;  
97  
98             delete texture;  
99             texture = nullptr;  
100         }  
101  
102         delete texture_manager;  
103         texture_manager = nullptr;  
104     }
```

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("trace.log", std::ofstream::out);  
31     if (!trace->trace_stream) std::cout << "Trace file wasn't opened successfully."  
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(), Model←_Data_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().

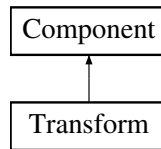
The documentation for this class was generated from the following files:

- [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 Transform 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.

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().

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
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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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             Engine::Initialize();
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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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"
```

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

Version

0.1

Date

2021-07-26

Copyright

Copyright (c) 2021

Index

- ~Behavior
 - Behavior, 7
- ~Model_Data
 - Model_Data, 76
- ~Texture
 - Texture, 122
- add_float
 - Vector3_Func, 140
- add_vec3
 - Vector3_Func, 141
- AddComponent
 - Object, 87
- AddForce
 - Physics, 104
- AddObject
 - Object_Manager, 96
- AddScript
 - Behavior, 7
- ApplyForce
 - Physics, 105
- 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, 144
- behavior.hpp, 144
- 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, 145
- camera.hpp, 146
- CheckIfCopy
 - Behavior, 8
- CheckName
 - Object_Manager, 96
- ClassSetup
 - Behavior, 8
- Clear
 - Behavior, 10
 - Object, 87
- Clone
 - Behavior, 10
 - Model, 70
 - Object, 88
 - Physics, 105
 - Transform, 133
- Component, 23
 - Component, 24
 - CType, 23
 - GetCType, 24
 - GetParent, 24
 - SetParent, 25
- component.cpp, 146
- component.hpp, 147
- 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, [34](#)
- Display_World_Settings
 - Editor, [35](#)
- distance
 - Vector3_Func, [141](#)
- Draw
 - Model, [70](#)
 - Model_Data, [76](#)
- 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, [34](#)
 - Display_World_Settings, [35](#)
 - GetTakeKeyboardInput, [36](#)
 - Initialize, [36](#)
 - Render, [37](#)
 - Reset, [37](#)
 - Shutdown, [38](#)
 - Update, [38](#)
- editor.cpp, [147](#)
- editor.hpp, [148](#)
- Engine, [40](#)
 - GetDeltaTime, [41](#)
 - GetDt, [41](#)
 - GetGravConst, [42](#)
 - GetLightPos, [42](#)
 - GetLightPower, [42](#)
 - GetPresetName, [43](#)
 - Initialize, [43](#)
 - Restart, [44](#), [45](#)
 - SetPresetName, [45](#)
 - Shutdown, [46](#)
 - Update, [46](#)
 - Write, [47](#)
- engine.cpp, [149](#)
- engine.hpp, [149](#)
- ErrorCallback
 - Graphics, [62](#)
- ErrorCheck
 - Graphics, [63](#)
- File_Reader, [48](#)
 - File_Reader, [49](#)
 - Read_Behavior_Name, [50](#)
 - Read_Bool, [50](#)
 - Read_Double, [51](#)
 - Read_File, [51](#)
 - Read_Float, [52](#)
 - Read_Int, [52](#)
 - Read_Object_Name, [53](#)
 - Read_Object_Position, [53](#)
 - Read_Object_Scale, [54](#)
 - Read_Object_Template_Name, [54](#)
 - Read_String, [55](#)
 - Read_Vec3, [55](#)
- file_reader.cpp, [150](#)
- file_reader.hpp, [151](#)
- File_Writer, [56](#)
 - File_Writer, [57](#)
 - Write_Behavior_Name, [57](#)
 - Write_File, [58](#)
 - Write_Object_Data, [58](#)
 - Write_String, [59](#)
 - Write_Value, [59](#)
 - Write_Vec3, [60](#)
- file_writer.cpp, [151](#)
- file_writer.hpp, [152](#)
- FindObject
 - Object_Manager, [97](#)
- Get
 - Model_Data_Manager, [82](#)
 - Texture_Manager, [126](#), [127](#)
- get_direction
 - Vector3_Func, [142](#)
- GetAcceleration
 - Physics, [106](#)
- GetAccelerationRef
 - Physics, [106](#)
- GetComponent
 - Object, [88](#)
- GetComponentConst
 - Object, [89](#)
- GetComponentList
 - Object, [89](#)
- GetCType
 - Behavior, [10](#)
 - Component, [24](#)
 - Model, [71](#)
 - Physics, [106](#)
 - Transform, [133](#)
- GetDeltaTime
 - Engine, [41](#)
- GetDt
 - Engine, [41](#)
- GetFar
 - Camera, [16](#)
- GetForces
 - Physics, [107](#)
- GetForcesRef
 - Physics, [107](#)

GetFov
 Camera, [16](#)
GetFront
 Camera, [16](#)
GetGravConst
 Engine, [42](#)
GetId
 Object, [90](#)
GetLightId
 Shader, [116](#)
GetLightPos
 Engine, [42](#)
GetLightPower
 Engine, [42](#)
GetLightPowerId
 Shader, [116](#)
GetMass
 Physics, [107](#)
GetMassRef
 Physics, [108](#)
GetMatrixId
 Shader, [117](#)
GetModelMatrixId
 Shader, [117](#)
GetModelName
 Model, [71](#)
 Model_Data, [78](#)
GetName
 Object, [90](#)
GetNameRef
 Object, [90](#)
GetNear
 Camera, [16](#)
GetOldPosition
 Transform, [134](#)
GetOriginalMoveSpeed
 Camera, [17](#)
GetOriginalSensitivity
 Camera, [17](#)
GetOriginalSprintSpeed
 Camera, [17](#)
GetParent
 Component, [24](#)
GetPitch
 Camera, [18](#)
GetPosition
 Camera, [18](#)
 Transform, [134](#)
GetPositionRef
 Transform, [134](#)
GetPresetName
 Engine, [43](#)
GetProgram
 Shader, [117](#)
GetRotation
 Transform, [135](#)
GetRotationRef
 Transform, [135](#)
GetScale
 Transform, [135](#)
GetScaleRef
 Transform, [136](#)
GetScripts
 Behavior, [10](#)
GetSize
 Object_Manager, [98](#)
GetStartPosition
 Transform, [136](#)
GetStartPositionRef
 Transform, [136](#)
GetTakeKeyboardInput
 Editor, [36](#)
GetTemplateName
 Object, [91](#)
GetTexture
 Model, [71](#)
GetTextureName
 Model, [72](#)
 Texture, [123](#)
GetTextureNum
 Texture, [123](#)
GetUp
 Camera, [18](#)
GetVelocity
 Physics, [108](#)
GetVelocityRef
 Physics, [108](#)
GetViewMatrixId
 Shader, [118](#)
GetWindow
 Graphics, [63](#)
GetWindowSize
 Graphics, [63](#)
GetYaw
 Camera, [19](#)
Graphics, [61](#)
 ErrorCallback, [62](#)
 ErrorCheck, [63](#)
 GetWindow, [63](#)
 GetWindowSize, [63](#)
 Graphics, [62](#)
 Initialize, [64](#)
 InitializeGL, [65](#)
 Render, [65](#)
 Shutdown, [66](#)
 Update, [66](#)
graphics.cpp, [152](#)
graphics.hpp, [153](#)

- Initialize
 - Camera, 19
 - Editor, 36
 - Engine, 43
 - Graphics, 64
 - Model_Data_Manager, 83
 - Object_Manager, 98
 - Random, 113
 - Shader, 118
 - Texture_Manager, 127
 - Trace, 129
- InitializeGL
 - Graphics, 65
- length
 - Vector3_Func, 142
- Load
 - Model, 72
 - Model_Data, 78, 79
 - Texture, 123
- LoadDDS
 - Texture, 124
- LoadShader
 - Shader, 119
- main
 - main.cpp, 154
- main.cpp, 154
 - main, 154
- Message
 - Trace, 129
- Model, 67
 - Clone, 70
 - Draw, 70
 - GetCType, 71
 - GetModelName, 71
 - GetTexture, 71
 - GetTextureName, 72
 - Load, 72
 - Model, 69
 - Read, 73
 - SwitchModel, 73
 - SwitchTexture, 73
 - Write, 74
- model.cpp, 155
- model.hpp, 155
- Model_Data, 74
 - ~Model_Data, 76
 - Draw, 76
 - GetModelName, 78
 - Load, 78, 79
 - Model_Data, 75, 76
 - Read, 79
- model_data.cpp, 156
- model_data.hpp, 157
- Model_Data_Manager, 81
 - Get, 82
 - Initialize, 83
 - Shutdown, 83
- model_data_manager.cpp, 157
- model_data_manager.hpp, 158
- MouseUpdate
 - Camera, 20
- normalize
 - Vector3_Func, 143
- Object, 84
 - AddComponent, 87
 - Clear, 87
 - Clone, 88
 - GetComponent, 88
 - GetComponentConst, 89
 - GetComponentList, 89
 - GetId, 90
 - GetName, 90
 - GetNameRef, 90
 - GetTemplateName, 91
 - Object, 86, 87
 - Read, 91
 - RemoveComponent, 92
 - ReRead, 92
 - SetId, 93
 - SetName, 93
 - SetTemplateName, 94
 - Update, 94
 - Write, 94
- object.cpp, 159
- object.hpp, 159
- Object_Manager, 95
 - AddObject, 96
 - CheckName, 96
 - FindObject, 97
 - GetSize, 98
 - Initialize, 98
 - ReadList, 99
 - RemoveObject, 99
 - Shutdown, 100
 - Update, 100
 - Write, 101
- object_manager.cpp, 160
- object_manager.hpp, 161
- Physics, 102
 - AddForce, 104
 - ApplyForce, 105
 - Clone, 105
 - GetAcceleration, 106
 - GetAccelerationRef, 106
 - GetCType, 106

- GetForces, [107](#)
- GetForcesRef, [107](#)
- GetMass, [107](#)
- GetMassRef, [108](#)
- GetVelocity, [108](#)
- GetVelocityRef, [108](#)
- Physics, [103](#), [104](#)
- Read, [109](#)
- SetAcceleration, [109](#)
- SetForces, [110](#)
- SetMass, [110](#)
- SetVelocity, [110](#)
- Update, [111](#)
- UpdateGravity, [111](#)
- Write, [112](#)
- physics.cpp, [161](#)
- physics.hpp, [162](#)
- Random, [112](#)
 - Initialize, [113](#)
 - random_float, [113](#)
 - random_vec3, [114](#)
 - Shutdown, [114](#)
- random.cpp, [163](#)
- random.hpp, [163](#)
- random_float
 - Random, [113](#)
- random_vec3
 - Random, [114](#)
- Read
 - Behavior, [11](#)
 - Model, [73](#)
 - Model_Data, [79](#)
 - Object, [91](#)
 - Physics, [109](#)
 - Transform, [137](#)
- Read_Behavior_Name
 - File_Reader, [50](#)
- Read_Bool
 - File_Reader, [50](#)
- Read_Double
 - File_Reader, [51](#)
- Read_File
 - File_Reader, [51](#)
- Read_Float
 - File_Reader, [52](#)
- Read_Int
 - File_Reader, [52](#)
- Read_Object_Name
 - File_Reader, [53](#)
- Read_Object_Position
 - File_Reader, [53](#)
- Read_Object_Scale
 - File_Reader, [54](#)
- Read_Object_Template_Name
 - File_Reader, [54](#)
- Read_String
 - File_Reader, [55](#)
- Read_Vec3
 - File_Reader, [55](#)
- ReadFile
 - Shader, [120](#)
- ReadList
 - Object_Manager, [99](#)
- RemoveComponent
 - Object, [92](#)
- RemoveObject
 - Object_Manager, [99](#)
- Render
 - Editor, [37](#)
 - Graphics, [65](#)
- ReRead
 - Object, [92](#)
- Reset
 - Editor, [37](#)
- Restart
 - Engine, [44](#), [45](#)
- SetAcceleration
 - Physics, [109](#)
- SetForces
 - Physics, [110](#)
- SetId
 - Object, [93](#)
- SetMass
 - Physics, [110](#)
- SetName
 - Object, [93](#)
- SetOldPosition
 - Transform, [137](#)
- SetParent
 - Component, [25](#)
- SetPosition
 - Transform, [137](#)
- SetPresetName
 - Engine, [45](#)
- SetRotation
 - Transform, [138](#)
- SetScale
 - Transform, [138](#)
- SetStartPosition
 - Transform, [139](#)
- SetTemplateName
 - Object, [94](#)
- SetupClassesForLua
 - Behavior, [11](#)
- SetVelocity
 - Physics, [110](#)

- Shader, 115
 - GetLightId, 116
 - GetLightPowerId, 116
 - GetMatrixId, 117
 - GetModelMatrixId, 117
 - GetProgram, 117
 - GetViewMatrixId, 118
 - Initialize, 118
 - LoadShader, 119
 - ReadFile, 120
 - Shutdown, 120
 - Update, 121
- shader.cpp, 164
- shader.hpp, 165
- Shutdown
 - Camera, 21
 - Editor, 38
 - Engine, 46
 - Graphics, 66
 - Model_Data_Manager, 83
 - Object_Manager, 100
 - Random, 114
 - Shader, 120
 - Texture_Manager, 128
 - Trace, 130
- SwitchModel
 - Model, 73
- SwitchScript
 - Behavior, 12
- SwitchTexture
 - Model, 73
- Texture, 121
 - ~Texture, 122
 - Display, 123
 - GetTextureName, 123
 - GetTextureNum, 123
 - Load, 123
 - LoadDDS, 124
- texture.cpp, 165
- texture.hpp, 166
- Texture_Manager, 125
 - Get, 126, 127
 - Initialize, 127
 - Shutdown, 128
- texture_manager.cpp, 167
- texture_manager.hpp, 167
- Trace, 129
 - Initialize, 129
 - Message, 129
 - Shutdown, 130
- trace.cpp, 168
- trace.hpp, 169
- Transform, 131
 - Clone, 133
 - GetCType, 133
 - GetOldPosition, 134
 - GetPosition, 134
 - GetPositionRef, 134
 - GetRotation, 135
 - GetRotationRef, 135
 - GetScale, 135
 - GetScaleRef, 136
 - GetStartPosition, 136
 - GetStartPositionRef, 136
 - Read, 137
 - SetOldPosition, 137
 - SetPosition, 137
 - SetRotation, 138
 - SetScale, 138
 - SetStartPosition, 139
 - Transform, 132, 133
 - Write, 139
- transform.cpp, 169
- transform.hpp, 170
- Update
 - Behavior, 12
 - Camera, 21
 - Editor, 38
 - Engine, 46
 - Graphics, 66
 - Object, 94
 - Object_Manager, 100
 - Physics, 111
 - Shader, 121
- UpdateGravity
 - Physics, 111
- Vector3_Func, 139
 - add_float, 140
 - add_vec3, 141
 - distance, 141
 - get_direction, 142
 - length, 142
 - normalize, 143
 - zero_vec3, 143
- vector3_func.cpp, 170
- vector3_func.hpp, 171
- Write
 - Behavior, 13
 - Engine, 47
 - Model, 74
 - Object, 94
 - Object_Manager, 101
 - Physics, 112
 - Transform, 139
- Write_Behavior_Name

File_Writer, [57](#)
Write_File
File_Writer, [58](#)
Write_Object_Data
File_Writer, [58](#)
Write_String
File_Writer, [59](#)
Write_Value
File_Writer, [59](#)
Write_Vec3
File_Writer, [60](#)

zero_vec3
Vector3_Func, [143](#)