# 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 . . . . . 24

4.3.2 Member Enumeration Documentation . . . . . 24

4.3.3 Constructor & Destructor Documentation . . . . . 25

4.3.4 Member Function Documentation . . . . . 25

4.4 Editor Class Reference . . . . . 26

4.4.1 Detailed Description . . . . . 27

4.4.2 Member Function Documentation . . . . . 28

4.5 Engine Class Reference . . . . . 42

4.5.1 Detailed Description . . . . . 43

4.5.2 Member Function Documentation . . . . . 43

4.6 File_Reader Class Reference . . . . . 50

4.6.1 Detailed Description . . . . . 51

4.6.2 Member Function Documentation . . . . . 51

4.7 File_Writer Class Reference . . . . . 58

4.7.1 Detailed Description . . . . . 59

4.7.2 Constructor & Destructor Documentation . . . . . 59

4.7.3 Member Function Documentation . . . . . 59

4.8 Graphics Class Reference . . . . . 63

4.8.1 Detailed Description . . . . . 64

4.8.2 Constructor & Destructor Documentation . . . . . 64

4.8.3 Member Function Documentation . . . . . 64

4.9 Model Class Reference . . . . . 70
  4.9.1 Detailed Description . . . . . 72
  4.9.2 Constructor & Destructor Documentation . . . . . 72
  4.9.3 Member Function Documentation . . . . . 73
4.10 Model_Data Class Reference . . . . . 77
  4.10.1 Detailed Description . . . . . 78
  4.10.2 Constructor & Destructor Documentation . . . . . 78
  4.10.3 Member Function Documentation . . . . . 79
4.11 Model_Data_Manager Class Reference . . . . . 84
  4.11.1 Detailed Description . . . . . 85
  4.11.2 Member Function Documentation . . . . . 85
4.12 Object Class Reference . . . . . 87
  4.12.1 Detailed Description . . . . . 89
  4.12.2 Constructor & Destructor Documentation . . . . . 89
  4.12.3 Member Function Documentation . . . . . 90
4.13 Object_Manager Class Reference . . . . . 98
  4.13.1 Detailed Description . . . . . 99
  4.13.2 Member Function Documentation . . . . . 99
4.14 Physics Class Reference . . . . . 106
  4.14.1 Detailed Description . . . . . 108
  4.14.2 Constructor & Destructor Documentation . . . . . 108
  4.14.3 Member Function Documentation . . . . . 109
4.15 Random Class Reference . . . . . 118
  4.15.1 Detailed Description . . . . . 118
  4.15.2 Member Function Documentation . . . . . 118
4.16 Shader Class Reference . . . . . 121
  4.16.1 Detailed Description . . . . . 122
  4.16.2 Member Function Documentation . . . . . 122
4.17 Texture Class Reference . . . . . 127
  4.17.1 Detailed Description . . . . . 128
  4.17.2 Constructor & Destructor Documentation . . . . . 128
  4.17.3 Member Function Documentation . . . . . 129
4.18 Texture_Manager Class Reference . . . . . 132
  4.18.1 Detailed Description . . . . . 132
  4.18.2 Member Function Documentation . . . . . 132
4.19 Trace Class Reference . . . . . 135
  4.19.1 Detailed Description . . . . . 135
  4.19.2 Member Function Documentation . . . . . 136
4.20 Transform Class Reference . . . . . 137

4.20.1 Detailed Description . . . 139
4.20.2 Constructor & Destructor Documentation . . . 139
4.20.3 Member Function Documentation . . . 140
4.21 Vector3_Func Class Reference . . . 146
4.21.1 Detailed Description . . . 147
4.21.2 Member Function Documentation . . . 147

**5 File Documentation 151**

5.1 behavior.cpp File Reference . . . 151
5.1.1 Detailed Description . . . 151
5.2 behavior.hpp File Reference . . . 151
5.2.1 Detailed Description . . . 152
5.3 camera.cpp File Reference . . . 152
5.3.1 Detailed Description . . . 152
5.4 camera.hpp File Reference . . . 153
5.4.1 Detailed Description . . . 153
5.5 component.cpp File Reference . . . 153
5.5.1 Detailed Description . . . 153
5.6 component.hpp File Reference . . . 154
5.6.1 Detailed Description . . . 154
5.7 editor.cpp File Reference . . . 154
5.7.1 Detailed Description . . . 155
5.8 editor.hpp File Reference . . . 155
5.8.1 Detailed Description . . . 155
5.9 engine.cpp File Reference . . . 156
5.9.1 Detailed Description . . . 156
5.10 engine.hpp File Reference . . . 156
5.10.1 Detailed Description . . . 157
5.11 file_reader.cpp File Reference . . . 157
5.11.1 Detailed Description . . . 157
5.12 file_reader.hpp File Reference . . . 158
5.12.1 Detailed Description . . . 158
5.13 file_writer.cpp File Reference . . . 158
5.13.1 Detailed Description . . . 158
5.14 file_writer.hpp File Reference . . . 159
5.14.1 Detailed Description . . . 159
5.15 graphics.cpp File Reference . . . 159
5.15.1 Detailed Description . . . 160
5.16 graphics.hpp File Reference . . . 160

5.16.1 Detailed Description . . . . . 160
5.17 main.cpp File Reference . . . . . 161
5.17.1 Detailed Description . . . . . 161
5.17.2 Function Documentation . . . . . 161
5.18 model.cpp File Reference . . . . . 162
5.18.1 Detailed Description . . . . . 162
5.19 model.hpp File Reference . . . . . 162
5.19.1 Detailed Description . . . . . 163
5.20 model_data.cpp File Reference . . . . . 163
5.20.1 Detailed Description . . . . . 163
5.21 model_data.hpp File Reference . . . . . 164
5.21.1 Detailed Description . . . . . 164
5.22 model_data_manager.cpp File Reference . . . . . 164
5.22.1 Detailed Description . . . . . 165
5.23 model_data_manager.hpp File Reference . . . . . 165
5.23.1 Detailed Description . . . . . 165
5.24 object.cpp File Reference . . . . . 166
5.24.1 Detailed Description . . . . . 166
5.25 object.hpp File Reference . . . . . 166
5.25.1 Detailed Description . . . . . 167
5.26 object_manager.cpp File Reference . . . . . 167
5.26.1 Detailed Description . . . . . 167
5.27 object_manager.hpp File Reference . . . . . 168
5.27.1 Detailed Description . . . . . 168
5.28 physics.cpp File Reference . . . . . 168
5.28.1 Detailed Description . . . . . 169
5.29 physics.hpp File Reference . . . . . 169
5.29.1 Detailed Description . . . . . 169
5.30 random.cpp File Reference . . . . . 170
5.30.1 Detailed Description . . . . . 170
5.31 random.hpp File Reference . . . . . 170
5.31.1 Detailed Description . . . . . 171
5.32 shader.cpp File Reference . . . . . 171
5.32.1 Detailed Description . . . . . 171
5.33 shader.hpp File Reference . . . . . 172
5.33.1 Detailed Description . . . . . 172
5.34 texture.cpp File Reference . . . . . 172
5.34.1 Detailed Description . . . . . 173
5.35 texture.hpp File Reference . . . . . 173

5.35.1 Detailed Description . . . . . 173
5.36 texture_manager.cpp File Reference . . . . . 174
5.36.1 Detailed Description . . . . . 174
5.37 texture_manager.hpp File Reference . . . . . 174
5.37.1 Detailed Description . . . . . 175
5.38 trace.cpp File Reference . . . . . 175
5.38.1 Detailed Description . . . . . 175
5.39 trace.hpp File Reference . . . . . 176
5.39.1 Detailed Description . . . . . 176
5.40 transform.cpp File Reference . . . . . 176
5.40.1 Detailed Description . . . . . 176
5.41 transform.hpp File Reference . . . . . 177
5.41.1 Detailed Description . . . . . 177
5.42 vector3_func.cpp File Reference . . . . . 177
5.42.1 Detailed Description . . . . . 177
5.43 vector3_func.hpp File Reference . . . . . 178
5.43.1 Detailed Description . . . . . 178

**Index** **179**

# 1 Hierarchical Index

## 1.1 Class Hierarchy

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

**Camera** **13**
**Component** **23**
  **Behavior** **4**
  **Model** **70**
  **Physics** **106**
  **Transform** **137**
**Editor** **26**
**Engine** **42**
**File_Reader** **50**
**File_Writer** **58**

| | |
|---|---|
| Graphics | 63 |
| Model_Data | 77 |
| Model_Data_Manager | 84 |
| Object | 87 |
| Object_Manager | 98 |
| Random | 118 |
| Shader | 121 |
| Texture | 127 |
| Texture_Manager | 132 |
| Trace | 135 |
| Vector3_Func | 146 |

# 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 | 26 |
| Engine | 42 |
| File_Reader | 50 |
| File_Writer | 58 |
| Graphics | 63 |
| Model | 70 |
| Model_Data | 77 |
| Model_Data_Manager | 84 |
| Object | 87 |
| Object_Manager | 98 |
| Physics | 106 |

Random 118
Shader 121
Texture 127
Texture_Manager 132
Trace 135
Transform 137
Vector3_Func 146

# 3 File Index

## 3.1 File List

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

behavior.cpp 151
behavior.hpp 151
camera.cpp 152
camera.hpp 153
component.cpp 153
component.hpp 154
editor.cpp 154
editor.hpp 155
engine.cpp 156
engine.hpp 156
file_reader.cpp 157
file_reader.hpp 158
file_writer.cpp 158
file_writer.hpp 159
graphics.cpp 159
graphics.hpp 160
main.cpp 161
model.cpp 162

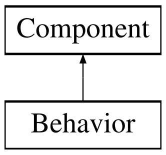
model.hpp 162
model_data.cpp 163
model_data.hpp 164
model_data_manager.cpp 164
model_data_manager.hpp 165
object.cpp 166
object.hpp 166
object_manager.cpp 167
object_manager.hpp 168
physics.cpp 168
physics.hpp 169
random.cpp 170
random.hpp 170
shader.cpp 171
shader.hpp 172
texture.cpp 172
texture.hpp 173
texture_manager.cpp 174
texture_manager.hpp 174
trace.cpp 175
trace.hpp 176
transform.cpp 176
transform.hpp 177
vector3_func.cpp 177
vector3_func.hpp 178

# 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.

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

Referenced by Clone().

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

Copy constructor.

**Parameters**

| | |
|---|---|
| *other* | Behavior object to copy |

Definition at line 36 of file behavior.cpp.

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

#### 4.1.2.3 Behavior() [3/3]

```
Behavior::Behavior (
        File_Reader & reader )
```

Creates Behavior object using file.

**Parameters**

| | |
|---|---|
| *reader* | Data from file |

Definition at line 45 of file behavior.cpp.

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

References Read().

#### 4.1.2.4 ∼Behavior()

```
Behavior::∼Behavior ( )
```

Deletes all of the lua states.

Definition at line 62 of file behavior.cpp.

```
                       {
    Clear();
}
```

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**

| | |
|---|---|
| *newScriptName* | |

**Returns**

true

false

Definition at line 233 of file behavior.cpp.

```
                                             {
      // Checking if this script is already attached
    if (newScriptName.find(".lua") == std::string::npos) return false;
    if (CheckIfCopy(newScriptName)) return false;
      // Setting up new lua state
    sol::state* state = new sol::state;
    state->open_libraries(sol::lib::base, sol::lib::math, sol::lib::io, sol::lib::string);
    states.emplace_back(state);
      // Adding new script filename to list
    scripts.emplace_back(newScriptName);
    ClassSetup(state);
      // Setting up lua script to run
    states.back()->script_file(scripts.back());
    (*states.back())["Start"]();

    return true;
}
```

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**

| | |
|---|---|
| *newScriptName* | Name of script being checked |

**Returns**

true

false

Definition at line 258 of file behavior.cpp.

```
                                               {
      // Checking if script is the same as an existing one
    for (std::string scriptName : scripts) {
        if (scriptName.compare(newScriptName) == 0) return true;
    }

      // Script is not a copy
    return false;
}
```

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 149 of file behavior.cpp.

```
                                        {
      // Giving lua random functions
    state->set_function("random_vec3", Random::random_vec3);
    state->set_function("random_float", Random::random_float);

      // Giving lua glm::vec3 wrapper class
    sol::usertype<glm::vec3> vec3_type = state->new_usertype<glm::vec3>("vec3",
        sol::constructors<glm::vec3(float, float, float), glm::vec3(float)>());
      // Giving lua glm::vec3 wrapper class variables
    vec3_type.set("x", &glm::vec3::x);
    vec3_type.set("y", &glm::vec3::y);
    vec3_type.set("z", &glm::vec3::z);
      // Giving lua glm::vec3 wrapper class functions
    state->set_function("normalize", Vector3_Func::normalize);
    state->set_function("distance", Vector3_Func::distance);
    state->set_function("get_direction", Vector3_Func::get_direction);
    state->set_function("zero_vec3", Vector3_Func::zero_vec3);
    state->set_function("length", Vector3_Func::length);
    state->set_function("add_float", Vector3_Func::add_float);
    state->set_function("add_vec3", Vector3_Func::add_vec3);

    state->set_function("FindObject", sol::overload(sol::resolve<Object*(int)>(&Object_Manager::FindObject),

        sol::resolve<Object*(std::string)>(&Object_Manager::FindObject)));

      // Giving lua physics class
    sol::usertype<Physics> physics_type = state->new_usertype<Physics>("Physics",
        sol::constructors<Physics(), Physics(const Physics)>());
      // Giving lua physics class variables
    physics_type.set("acceleration", sol::property(Physics::GetAccelerationRef, &Physics::SetAcceleration));
    physics_type.set("forces",       sol::property(Physics::GetForcesRef,       &Physics::SetForces));
    physics_type.set("velocity",     sol::property(Physics::GetVelocityRef,     &Physics::SetVelocity));
      // Giving lua physics class functions
    physics_type.set_function("ApplyForce",    &Physics::ApplyForce);
    physics_type.set_function("UpdateGravity", &Physics::UpdateGravity);

      // Giving lua transform class
    sol::usertype<Transform> transform_type = state->new_usertype<Transform>("Transform",
        sol::constructors<Transform(), Transform(const Transform)>());
      // Giving lua transform class variables
    transform_type.set("position",      sol::property(Transform::GetPositionRef,
   &Transform::SetPosition));
    transform_type.set("rotation",      sol::property(Transform::GetRotationRef,
   &Transform::SetRotation));
    transform_type.set("scale",         sol::property(Transform::GetScaleRef,
   &Transform::SetScale));
    transform_type.set("startPosition", sol::property(Transform::GetStartPositionRef,
   &Transform::SetStartPosition));

      // Giving lua object class
    state->set("object", GetParent());
    sol::usertype<Object> object_type = state->new_usertype<Object>("Object",
        sol::constructors<Object(), Object(const Object)>());
      // Giving lua object class variables
    object_type.set("name", sol::property(Object::GetNameRef, &Object::SetName));
    object_type.set("id",   sol::readonly_property(Object::GetId));
    object_type.set_function("GetPhysics", &Object::GetComponent<Physics>);
    object_type.set_function("GetTransform", &Object::GetComponent<Transform>);
}
```

References Vector3_Func::add_float(), Vector3_Func::add_vec3(), Physics::ApplyForce(), Vector3_Func::distance(), Object_Manager::FindObject(), Vector3_Func::get_direction(), Physics::GetAccelerationRef(), Physics::GetForces↩Ref(), Object::GetId(), Object::GetNameRef(), Component::GetParent(), Transform::GetPositionRef(), Transform::Get↩RotationRef(), 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 272 of file behavior.cpp.

```
                        {
    for (sol::state* state : states) {
        if (!state) continue;
        delete state;
        state = nullptr;
    }

    states.clear();
    scripts.clear();
}
```

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.

```
                                {
    return new Behavior(*this);
}
```

References Behavior().

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

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

**Returns**

CType

Definition at line 118 of file behavior.cpp.

```
                            {
    return CType::CBehavior;
}
```

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

Returns list of lua filenames.

**Returns**

std::vector<std::string>&

Definition at line 142 of file behavior.cpp.

```
{ 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**

| | |
|---|---|
| *reader* | Data from file |

Definition at line 83 of file behavior.cpp.

```
                                      {
   unsigned behavior_num = 0;

     // Reads the name of the lua files
   while (true) {
         // Getting the name of the next lua file
       std::string behavior_name = reader.Read_Behavior_Name("behavior_" + std::to_string(behavior_num));
       if (behavior_name.compare("") == 0) break;
       if (behavior_name.find(".lua") == std::string::npos) continue;
         // Adding lua filename to list
       scripts.emplace_back(std::string(getenv("USERPROFILE")) + "/Documents/pEngine/scripts/" +
   behavior_name);
       ++behavior_num;
   }
     // Creating lua state for each of the scripts that were read in
   for (unsigned i = 0; i < scripts.size(); ++i) {
       sol::state* state = new sol::state;
       state->open_libraries(sol::lib::base, sol::lib::math, sol::lib::io, sol::lib::string);
        states.emplace_back(state);
    }
}
```

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 126 of file behavior.cpp.

```
                                    {
    for (sol::state* state : states) {
        ClassSetup(state);
    }

    for (unsigned i = 0; i < states.size(); ++i) {
        states[i]->script_file(scripts[i]);
        (*states[i])["Start"]();
    }
}
```

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**

| | |
|---|---|
| *scriptNum* | |
| *newScriptName* | |

**Returns**

true

false

Definition at line 212 of file behavior.cpp.

```
                                                                                {
      // Checking if this script is already attached
    if (CheckIfCopy(newScriptName)) return false;
    if (newScriptName.compare(".lua") == 0) return false;
    if (newScriptName.find(".lua") == std::string::npos) return false;
    sol::state* state = states[scriptNum];
    scripts[scriptNum] = newScriptName;
      // Setting up new lua script
    state->script_file(scripts[scriptNum]);
    (*state)["Start"]();

    return true;
}
```

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.

```
                     {
    for (sol::state* state : states) {
        if (!state) continue;
        (*state)["FixedUpdate"](Engine::GetDt());
    }
}
```

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**

| | |
|---|---|
| *writer* | |

Definition at line 109 of file behavior.cpp.

```
                                       {
    writer.Write_Behavior_Name(scripts);
}
```

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 bool Initialize ()

  *Initialize the camera with default values.*
- 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**

| | |
|---|---|
| *width* | Width of screen |
| *height* | Height of screen |

Definition at line 33 of file camera.cpp.

```
                                      : position(0.f, 0.f, 0.f), front(0.f, 0.f, -1.f),
      up(0.f, 1.f, 0.f), yaw(-90.f), pitch(0.f), last({ width / 2.f, height / 2.f }),
    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 243 of file camera.cpp.

```
243 { 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 229 of file camera.cpp.

```
229 { 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 215 of file camera.cpp.
```
215 { 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 236 of file camera.cpp.
```
236 { 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 264 of file camera.cpp.
```
264 { 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 278 of file camera.cpp.

```
278 { 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 271 of file camera.cpp.

```
271 { 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 257 of file camera.cpp.

```
257 { 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 208 of file camera.cpp.

```
{ 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 222 of file camera.cpp.

```
{ 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 250 of file camera.cpp.

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

References camera, and yaw.

### 4.2.3.12 Initialize() [1/2] `bool Camera::Initialize ( ) [static]`

Initialize the camera with default values.

**Returns**

true

false

Definition at line 66 of file camera.cpp.

```
                          {
        // Initializing the camera
      camera = new Camera(1920, 1080);
      if (!camera) {
          Trace::Message("Camera was not initialized.");
          return false;
      }

        // Getting data from settings file
      camera->originalMoveSpeed = 10.f;
      camera->originalSprintSpeed = 30.f;
      camera->originalSensitivity = 150.f;

      return true;
  }
```

References camera, Camera(), Trace::Message(), originalMoveSpeed, originalSensitivity, and originalSprintSpeed.

Referenced by Engine::Initialize().

### 4.2.3.13 Initialize() [2/2] `bool Camera::Initialize ( File_Reader & settings ) [static]`

Initializes the camera.

**Parameters**

| | |
|---|---|
| *settings* | File that contains settings for the camera |

**Returns**

true

false

Definition at line 44 of file camera.cpp.

```
                                              {
        // Initializing the camera
      camera = new Camera(settings.Read_Int("windowWidth"), settings.Read_Int("windowHeight"));
      if (!camera) {
          Trace::Message("Camera was not initialized.");
          return false;
      }

        // Getting data from settings file
      camera->originalMoveSpeed = settings.Read_Float("moveSpeed");
      camera->originalSprintSpeed = settings.Read_Float("sprintSpeed");
```

```
    camera->originalSensitivity = settings.Read_Float("sensitivity");

    return true;
}
```

References camera, Camera(), Trace::Message(), originalMoveSpeed, originalSensitivity, originalSprintSpeed, File_↩ Reader::Read_Float(), and File_Reader::Read_Int().

#### 4.2.3.14 MouseUpdate()

```
void Camera::MouseUpdate (
            GLFWwindow * ,
            double xpos,
            double ypos )  [static]
```

Moves the camera using the mouse.

**Parameters**

| | |
|---|---|
| *xpos* | x position of the mouse |
| *ypos* | y position of the mouse |

**Returns**

void

Definition at line 138 of file camera.cpp.

```
                                                              {
    if (!camera->canMoveMouse) {
        camera->last = { xpos, ypos };
        return;
    }
      // Setting up variables
    static bool firstMouse = true;
    std::pair<double, double> mousePos = { xpos, ypos };

      // Setting the camera sens using delta time
    camera->sensitivity = camera->originalSensitivity * Engine::GetDeltaTime();

      // Checking if this is the first time the function was called
    if (firstMouse) {
        camera->last = { mousePos.first, mousePos.second };
        firstMouse = false;
    }

      // Finding how far the mouse is from its last position
    std::pair<float, float> offset = {
        mousePos.first - camera->last.first,
        camera->last.second - mousePos.second
    };
      // Setting new last position
    camera->last = { mousePos.first, mousePos.second };

      // Updating offsets to use the sensitivity of the camera
    offset.first *= camera->sensitivity;
    offset.second *= camera->sensitivity;

      // Applying the offset to the camera's direction
    camera->yaw += offset.first;
    camera->pitch += offset.second;

      // Stops the camera from circling completely in the y direction
    if (camera->pitch > 89.f) camera->pitch = 89.f;
    if (camera->pitch < -89.f) camera->pitch = -89.f;
```

```

      // Finding the direction of the camera
    glm::vec3 tempFront = {
        std::cos(glm::radians(camera->yaw)) * std::cos(glm::radians(camera->pitch)),
        std::sin(glm::radians(camera->pitch)),
        std::sin(glm::radians(camera->yaw)) * std::cos(glm::radians(camera->pitch))
    };
    camera->front = glm::normalize(tempFront);

      // Finding the upward direction of the camera
    glm::vec3 tempUp = { 0.f, 1.f, 0.f };
    glm::vec3 right = glm::normalize(glm::cross(tempUp, camera->front));
    glm::vec3 up = glm::cross(camera->front, right);
    camera->up = up;
}
```

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

Referenced by Graphics::Initialize().

#### 4.2.3.15 Shutdown() `void Camera::Shutdown ( ) [static]`

Deletes the camera object if it exists.

**Returns**

void

Definition at line 196 of file camera.cpp.

```
                       {
    if (camera) {
        delete camera;
        camera = nullptr;
    }
}
```

References camera.

Referenced by Engine::Shutdown().

#### 4.2.3.16 Update() `void Camera::Update ( ) [static]`

Moves the camera and checks for some other inputs.

**Returns**

void

Definition at line 87 of file camera.cpp.

```
                       {
    // Checking if the engine should be closed
  if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_ESCAPE) == GLFW_PRESS) {
      if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_ESCAPE) == GLFW_RELEASE) {
          glfwSetWindowShouldClose(Graphics::GetWindow(), true);
      }
  }

    // Checking if sprint is being used
  if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_LEFT_SHIFT) == GLFW_PRESS &&
  Editor::GetTakeKeyboardInput()) {
      camera->speed = camera->originalSprintSpeed * Engine::GetDeltaTime();
  }
  else {
       camera->speed = camera->originalMoveSpeed * Engine::GetDeltaTime();
   }

     // Checking for movement using W, A, S, D, SPACE, and CTRL
   if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_W) == GLFW_PRESS && Editor::GetTakeKeyboardInput()) {
       camera->position += camera->speed * camera->front;
   }
   if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_S) == GLFW_PRESS && Editor::GetTakeKeyboardInput()) {
       camera->position -= camera->speed * camera->front;
   }
   if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_A) == GLFW_PRESS && Editor::GetTakeKeyboardInput()) {
       camera->position -= glm::normalize(glm::cross(camera->front, camera->up)) * camera->speed;
   }
   if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_D) == GLFW_PRESS && Editor::GetTakeKeyboardInput()) {
       camera->position += glm::normalize(glm::cross(camera->front, camera->up)) * camera->speed;
   }
   if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_SPACE) == GLFW_PRESS && Editor::GetTakeKeyboardInput()) {
       camera->position += camera->speed * camera->up;
   }
   if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_LEFT_CONTROL) == GLFW_PRESS &&
  Editor::GetTakeKeyboardInput()) {
       camera->position -= camera->speed * camera->up;
   }

   if (glfwGetMouseButton(Graphics::GetWindow(), GLFW_MOUSE_BUTTON_RIGHT) == GLFW_PRESS &&
  Editor::GetTakeKeyboardInput()) {
       camera->canMoveMouse = true;
   }
   if (glfwGetMouseButton(Graphics::GetWindow(), GLFW_MOUSE_BUTTON_RIGHT) == GLFW_RELEASE) {
       camera->canMoveMouse = false;
   }
}
```

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

Referenced by Engine::Update().

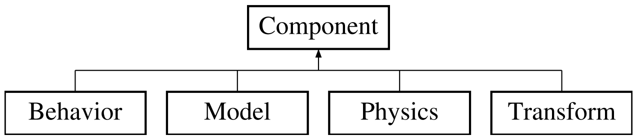
The documentation for this class was generated from the following files:

- 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**

| | |
|---|---|
| *type_* | 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**

| | |
|---|---|
| *object* | 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.*

### Static Private Member Functions

- static std::string Make_Display_String (std::string inputString)

  *Removes the filepath from a filename.*

### 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 431 of file editor.cpp.

```
                                        {
    ImGui::Begin("Camera Settings");

    ImGui::PushItemWidth(137);

      // Default move speed
    ImGui::Text("Move Speed");
    ImGui::SameLine(100); ImGui::InputFloat("##2", &Camera::GetOriginalMoveSpeed());

      // Move speed when holding shift
    ImGui::Text("Sprint Speed");
    ImGui::SameLine(100); ImGui::InputFloat("##3", &Camera::GetOriginalSprintSpeed());

      // Mouse sensitivity when looking around
    ImGui::Text("Sensitivity");
    ImGui::SameLine(100); ImGui::InputFloat("##4", &Camera::GetOriginalSensitivity());

    ImGui::PopItemWidth();

    ImGui::End();
}
```

References Camera::GetOriginalMoveSpeed(), Camera::GetOriginalSensitivity(), and Camera::GetOriginalSprint↩ Speed().

Referenced by Update().

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

Display all of the components of the current selected_object.

Definition at line 271 of file editor.cpp.

```
                                   {
    ImGui::Begin("Components##1");

    if (selected_object == -1) { ImGui::End(); return; }
    Object* object = Object_Manager::FindObject(selected_object);
    std::string objectName = object->GetName();

    ImGui::Text("Id: %d", object->GetId());

      // Display name box (allows changing the name of an object)
    static char nameBuf[128] = "";
    sprintf(nameBuf, objectName.c_str());

    if (ImGui::InputText("Name", nameBuf, 128, ImGuiInputTextFlags_EnterReturnsTrue)) {
        object->SetName(std::string(nameBuf));
    }

    if (ImGui::IsItemDeactivatedAfterEdit()) {
        object->SetName(std::string(nameBuf));
    }

      // Template used by the selected object
    ImGui::Text("Template:");
    ImGui::SameLine(100);
    std::string templateName = object->GetTemplateName();
    if (templateName.empty()) templateName = "No template##1";
    else templateName = Editor::Make_Display_String(templateName);

    if (ImGui::Button(templateName.c_str())) {
        ImGuiFileDialog::Instance()->OpenDialog("ChooseTemplate##1", "Choose File", ".json",
   std::string(getenv("USERPROFILE")) + "/Documents/pEngine/json/objects/");
    }

    ImGui::SameLine();
    if (ImGui::Button("New Template")) {
        ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDlgKey##6", "Choose File", ".json",
   std::string(getenv("USERPROFILE")) + "/Documents/pEngine/json/objects/");
    }


    if (ImGuiFileDialog::Instance()->Display("ChooseFileDlgKey##6")) {
        if (ImGuiFileDialog::Instance()->IsOk()) {
            std::string filePath = ImGuiFileDialog::Instance()->GetCurrentPath();
            object->Write(filePath);
        }

       ImGuiFileDialog::Instance()->Close();
    }

    if (ImGuiFileDialog::Instance()->Display("ChooseTemplate##1")) {
        if (ImGuiFileDialog::Instance()->IsOk()) {
            std::string filePath = ImGuiFileDialog::Instance()->GetCurrentPath();
            filePath += "/" + ImGuiFileDialog::Instance()->GetCurrentFileName();
            object->ReRead(filePath);
        }

        ImGuiFileDialog::Instance()->Close();
    }

      // Getting all of the components
    Behavior* behavior = object->GetComponent<Behavior>();
    Model* model = object->GetComponent<Model>();
    Physics* physics = object->GetComponent<Physics>();
    Transform* transform = object->GetComponent<Transform>();

      // Display all of the components of the selected_object
    Display_Transform(transform);
    Display_Physics(physics);
    Display_Model(model);
    Display_Scripts(behavior);

    ImGui::Separator();

      // Button to add new components to the selected_object
    if (ImGui::Button("Add Component##1")) {
        ImGui::OpenPopup("New Component##1");
    }

      // Add new components to object (only ones that the object doesn't already have)
    if (ImGui::BeginPopup("New Component##1")) {
        if (!physics) {
            if (ImGui::Selectable("Physics##1")) {
                physics = new Physics;
                object->AddComponent(physics);
            }
        }
        if (!model) {
            if (ImGui::Selectable("Model##1")) {
                model = new Model;
                object->AddComponent(model);
            }
        }
        if (!behavior) {
            if (ImGui::Selectable("Scripts##1")) {
                behavior = new Behavior;
                object->AddComponent(behavior);
            }
        }
        ImGui::EndPopup();
    }

    ImGui::End();
}
```

References Display_Model(), Display_Physics(), Display_Scripts(), Display_Transform(), Object_Manager::Find↩ Object(), Object::GetId(), Make_Display_String(), and selected_object.

Referenced by Update().

#### 4.4.2.3 Display_Dockspace() `void Editor::Display_Dockspace ( ) [private]`

Setup and display the editor's dockspace.

Definition at line 155 of file editor.cpp.

```
                                    {
      // Setting up viewport
    ImGuiViewport* viewport = ImGui::GetMainViewport();
    ImGui::SetNextWindowPos(viewport->Pos);
    ImGui::SetNextWindowSize(viewport->Size);
    ImGui::SetNextWindowViewport(viewport->ID);
    ImGui::SetNextWindowBgAlpha(0.0f);

      // Setting up window flags
    ImGuiWindowFlags window_flags = ImGuiWindowFlags_MenuBar | ImGuiWindowFlags_NoDocking;
    window_flags |= ImGuiWindowFlags_NoTitleBar | ImGuiWindowFlags_NoCollapse | ImGuiWindowFlags_NoResize |
   ImGuiWindowFlags_NoMove;
    window_flags |= ImGuiWindowFlags_NoBringToFrontOnFocus | ImGuiWindowFlags_NoNavFocus;

      // Setting up window style
    ImGui::PushStyleVar(ImGuiStyleVar_WindowRounding, 0.0f);
    ImGui::PushStyleVar(ImGuiStyleVar_WindowBorderSize, 0.0f);
    ImGui::PushStyleVar(ImGuiStyleVar_WindowPadding, ImVec2(0.0f, 0.0f));

      // Making the window
    ImGui::SetNextWindowBgAlpha(0.0f);
    ImGui::Begin("Editor Window", &editor->isOpen, window_flags);
    ImGui::PopStyleVar(3);

      // Setting up window settings
    ImGuiID dockspace_id = ImGui::GetID("Editor");
    ImGuiDockNodeFlags dockspace_flags = ImGuiDockNodeFlags_PassthruCentralNode |
   ImGuiDockNodeFlags_NoDockingInCentralNode;
    ImGui::DockSpace(dockspace_id, ImVec2(0.0f, 0.0f), dockspace_flags);
    editor->Display_Menu_Bar();
    ImGui::End();
}
```

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 717 of file editor.cpp.

```
                                  {
    if (ImGui::BeginMenuBar()) {
        if (ImGui::BeginMenu("File##1")) {
            if (ImGui::MenuItem("Save##1")) {
                Engine::Write();
            }
            if (ImGui::MenuItem("Save As..##1")) {
                ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDlgKey##7", "Choose File", ".json",
   std::string(getenv("USERPROFILE")) + "/Documents/pEngine/json/preset");
            }
            ImGui::EndMenu();
        }

        if (ImGuiFileDialog::Instance()->Display("ChooseFileDlgKey##7")) {
            if (ImGuiFileDialog::Instance()->IsOk()) {
                std::string filePath = ImGuiFileDialog::Instance()->GetCurrentPath();
                filePath += "/" + ImGuiFileDialog::Instance()->GetCurrentFileName() + ".json";
                Engine::SetPresetName(std::string(filePath));
                Engine::Write();
            }
            ImGuiFileDialog::Instance()->Close();
        }

        ImGui::EndMenuBar();
    }
}
```

References 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**

| | |
|---|---|
| *model* | |

Definition at line 537 of file editor.cpp.

```
                                             {
    if (!model) return;

    std::string modelName = Editor::Make_Display_String(model->GetModelName());
    std::string textureName = Editor::Make_Display_String(model->GetTextureName());

    if (modelName.compare("") == 0) modelName = "no model";
    if (textureName.compare("") == 0) textureName = "no texture";

      // Setting up tree flags
    ImGuiTreeNodeFlags node_flags = ImGuiTreeNodeFlags_SpanAvailWidth | ImGuiTreeNodeFlags_OpenOnDoubleClick
   | ImGuiTreeNodeFlags_OpenOnArrow;
    if (selected_component == CType::CModel) node_flags |= ImGuiTreeNodeFlags_Selected;

    const bool model_open = ImGui::TreeNodeEx((void*)(intptr_t)CType::CModel, node_flags, "Model");
    if (ImGui::IsItemClicked()) selected_component = CType::CModel;

      // Right click behavior to delete model component from selected object
    if (ImGui::IsItemClicked(ImGuiMouseButton_Right)) {
        selected_component = CType::CModel;
        ImGui::OpenPopup("DeleteModel##1");
    }

    if (ImGui::BeginPopup("DeleteModel##1")) {
        if (ImGui::Selectable("Delete##3")) {
            model->GetParent()->RemoveComponent<Model>();
            selected_component = -1;
        }
        ImGui::EndPopup();
    }

    if (model_open) {
          // Model that is being used
        ImGui::Text("Model"); ImGui::SameLine(100);
        if (ImGui::Button(modelName.c_str())) {
            ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDlgKey##1", "Choose File", ".obj",
    std::string(getenv("USERPROFILE")) + "/Documents/pEngine/models/");
        }

        if (ImGuiFileDialog::Instance()->Display("ChooseFileDlgKey##1")) {
            if (ImGuiFileDialog::Instance()->IsOk()) {
                std::string filePath = ImGuiFileDialog::Instance()->GetCurrentPath();
                filePath += "/" + ImGuiFileDialog::Instance()->GetCurrentFileName();
                model->SwitchModel(filePath);
            }

            ImGuiFileDialog::Instance()->Close();
        }

          // Texture that is being used
        ImGui::Text("Texture"); ImGui::SameLine(100);
        if (ImGui::Button(textureName.c_str())) {
            ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDlgKey##2", "Choose File", ".dds,.DDS",
    std::string(getenv("USERPROFILE")) + "/Documents/pEngine/textures/");
        }

        if (ImGuiFileDialog::Instance()->Display("ChooseFileDlgKey##2")) {
            if (ImGuiFileDialog::Instance()->IsOk()) {
                std::string filePath = ImGuiFileDialog::Instance()->GetCurrentPath();
                filePath += "/" + ImGuiFileDialog::Instance()->GetCurrentFileName();
                model->SwitchTexture(filePath);
            }

            ImGuiFileDialog::Instance()->Close();
        }

        ImGui::TreePop();
    }
}
```

References Model::GetModelName(), Component::GetParent(), Model::GetTextureName(), Make_Display_String(), 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**

| | |
|---|---|
| *physics* | |

Definition at line 609 of file editor.cpp.

```
                                              {
    if (!physics) return;

    glm::vec3& velocity = physics->GetVelocityRef();
    glm::vec3& rotVel = physics->GetRotationalVelocityRef();

    ImGuiTreeNodeFlags node_flags = ImGuiTreeNodeFlags_SpanAvailWidth | ImGuiTreeNodeFlags_OpenOnDoubleClick
   | ImGuiTreeNodeFlags_OpenOnArrow;
    if (selected_component == CType::CPhysics) node_flags |= ImGuiTreeNodeFlags_Selected;

    const bool physics_open = ImGui::TreeNodeEx((void*)(intptr_t)CType::CPhysics, node_flags, "Physics");
    if (ImGui::IsItemClicked()) selected_component = CType::CPhysics;

    if (ImGui::IsItemClicked(ImGuiMouseButton_Right)) {
        selected_component = CType::CPhysics;
```

```
            ImGui::OpenPopup("DeletePhysics##1");
        }

        if (ImGui::BeginPopup("DeletePhysics##1")) {
            if (ImGui::Selectable("Delete##4")) {
                physics->GetParent()->RemoveComponent<Physics>();
                selected_component = -1;
            }
            ImGui::EndPopup();
        }

        if (physics_open) {
            ImGui::Text("Velocity");

            ImGui::PushItemWidth(65);
            ImGui::SameLine(100); ImGui::InputFloat("x##1", &velocity.x);
            ImGui::SameLine(185); ImGui::InputFloat("y##1", &velocity.y);
            ImGui::SameLine(270); ImGui::InputFloat("z##1", &velocity.z);

            ImGui::Text("RotVel");

            ImGui::PushItemWidth(65);
            ImGui::SameLine(100); ImGui::InputFloat("x##6", &rotVel.x);
            ImGui::SameLine(185); ImGui::InputFloat("y##6", &rotVel.y);
            ImGui::SameLine(270); ImGui::InputFloat("z##6", &rotVel.z);

            ImGui::Text("Mass");
            ImGui::SameLine(100); ImGui::InputFloat("##6", &physics->GetMassRef());
            ImGui::PopItemWidth();

            ImGui::TreePop();
        }
    }
```

References Physics::GetMassRef(), Component::GetParent(), Physics::GetRotationalVelocityRef(), Physics::Get↩ VelocityRef(), Object::RemoveComponent(), and selected_component.

Referenced by Display_Components().

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

Display the scene window.

Definition at line 190 of file editor.cpp.

```
                                {
        ImGui::Begin("Scene");

        if (!takeKeyboardInput && ImGui::IsWindowFocused()) {
            if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_LEFT_CONTROL) == GLFW_PRESS) {
                // Copy current selected object
                if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_C) == GLFW_PRESS) {
                    if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_C) == GLFW_RELEASE) {
                        editor->object_to_copy = editor->selected_object;
                    }
                }
                // Paste current selected object
                if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_V) == GLFW_PRESS) {
                    if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_V) == GLFW_RELEASE) {
                        if (editor->object_to_copy != -1) {
                            Object* object = new Object(*Object_Manager::FindObject(editor->selected_object));
                            Object_Manager::AddObject(object);
                        }
                    }
                }
            }
        }

        // Display all objects
        for (int i = 0; i < (int)Object_Manager::GetSize(); ++i) {
            if (ImGui::Selectable(Object_Manager::FindObject(i)->GetName().c_str(), selected_object == i,
       ImGuiSelectableFlags_AllowDoubleClick)) {
                if (selected_object != i) editor->selected_component = -1;
                selected_object = i;
                selected_component = -1;
            }

              // Checking for right click behavior
            if (ImGui::IsItemClicked(ImGuiMouseButton_Right)) {
                if (selected_object != i) editor->selected_component = -1;
                selected_object = i;
                selected_component = -1;
                ImGui::OpenPopup("ObjectSettings##1");
            }
        }

        if (ImGui::BeginPopup("ObjectSettings##1")) {
              // Removes selected object from scene
            if (ImGui::Selectable("Delete##1")) {
                Object_Manager::RemoveObject(selected_object);
                selected_object = -1;
                selected_component = -1;
            }
              // Copies selected object
            if (ImGui::Selectable("Copy##1")) {
                editor->object_to_copy = editor->selected_object;
            }
              // Pastes copied object into scene
            if (ImGui::Selectable("Paste##1")) {
                if (editor->object_to_copy != -1) {
                    Object* object = new Object(*Object_Manager::FindObject(editor->selected_object));
                    Object_Manager::AddObject(object);
                }
            }
            ImGui::EndPopup();
        }

        ImGui::Separator();

        // Button to add new object to the scene
        if (ImGui::Button("Add Object")) {
            Object* newObject = new Object;
            Transform* transform = new Transform;
            transform->SetStartPosition(glm::vec3(0.f));
            newObject->SetName("New_Object");
            newObject->AddComponent(transform);

            Object_Manager::AddObject(newObject);
        }

        ImGui::End();
    }
```

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

Referenced by Update().

#### 4.4.2.8 Display_Scripts() `void Editor::Display_Scripts ( Behavior * behavior ) [private]`

Displays the different lua scripts attached to the selected object.

**Parameters**

| | |
|---|---|
| *behavior* | Contains the script data |

Definition at line 458 of file editor.cpp.

```
                                              {
    if (!behavior) return;

    // Setting up tree flags
    ImGuiTreeNodeFlags node_flags = ImGuiTreeNodeFlags_SpanAvailWidth | ImGuiTreeNodeFlags_OpenOnDoubleClick
   | ImGuiTreeNodeFlags_OpenOnArrow;
    if (selected_component == CType::CBehavior) node_flags |= ImGuiTreeNodeFlags_Selected;

    const bool scripts_open = ImGui::TreeNodeEx((void*)(intptr_t)CType::CBehavior, node_flags, "Scripts");
    if (ImGui::IsItemClicked()) selected_component = CType::CBehavior;

      // Right click behavior to delete script component from object
    if (ImGui::IsItemClicked(ImGuiMouseButton_Right)) {
        selected_component = CType::CBehavior;
        ImGui::OpenPopup("DeleteScripts##1");
    }

    if (ImGui::BeginPopup("DeleteScripts##1")) {
        if (ImGui::Selectable("Delete##2")) {
            behavior->GetParent()->RemoveComponent<Behavior>();
            selected_component = -1;
        }
        ImGui::EndPopup();
    }

      // Displays the currently attached scripts
    if (scripts_open) {
        std::vector<std::string>& scripts = behavior->GetScripts();
        unsigned scriptNum = 1;
        for (std::string& script : scripts) {
            ImGui::Text(std::string("Script " + std::to_string(scriptNum) + ":").c_str());
            ImGui::SameLine(100);
            if (ImGui::Button(Editor::Make_Display_String(script).c_str())) {
                ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDlgKey##3", "Choose File", ".lua",
   std::string(getenv("USERPROFILE")) + "/Documents/pEngine/scripts/");
            }

            if (ImGuiFileDialog::Instance()->Display("ChooseFileDlgKey##3")) {
                if (ImGuiFileDialog::Instance()->IsOk()) {
                    std::string filePath = ImGuiFileDialog::Instance()->GetCurrentPath();
                    filePath += "/" + ImGuiFileDialog::Instance()->GetCurrentFileName();
                    behavior->SwitchScript(scriptNum - 1, filePath);
                }

                ImGuiFileDialog::Instance()->Close();
            }
            ++scriptNum;
        }

          // Add new script to the object
        ImGui::Indent(71);
        if (ImGui::Button("New Script##1")) {
            ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDlgKey##4", "Choose File", ".lua",
   std::string(getenv("USERPROFILE")) + "/Documents/pEngine/scripts/");
        }

        if (ImGuiFileDialog::Instance()->Display("ChooseFileDlgKey##4")) {
            if (ImGuiFileDialog::Instance()->IsOk()) {
                std::string filePath = ImGuiFileDialog::Instance()->GetCurrentPath();
                filePath += "/" + ImGuiFileDialog::Instance()->GetCurrentFileName();
                behavior->AddScript(filePath);
            }

            ImGuiFileDialog::Instance()->Close();
        }

          // Popup to say that the selected script to add is already attached to the object
        if (ImGui::BeginPopup("ExistingScript##1")) {
            ImGui::Text(std::string("Script already being used or doesn't exist").c_str(),
                ImGui::GetFontSize() * 2);
            ImGui::EndPopup();
        }

        ImGui::TreePop();
    }
}
```

References Behavior::AddScript(), Component::GetParent(), Behavior::GetScripts(), Make_Display_String(), Object::↩ RemoveComponent(), selected_component, and Behavior::SwitchScript().

Referenced by Display_Components().

#### 4.4.2.9 Display_Transform() `void Editor::Display_Transform ( Transform * transform ) [private]`

Display transform data, users can change any of it.

**Parameters**

|   |   |
|---|---|
| *transform* |   |

Definition at line 662 of file editor.cpp.

```
                                                   {
    if (!transform) return;

    glm::vec3& position = transform->GetPositionRef();
    glm::vec3& scale = transform->GetScaleRef();
    glm::vec3& rotation = transform->GetRotationRef();
    glm::vec3& startPos = transform->GetStartPositionRef();

    ImGuiTreeNodeFlags node_flags = ImGuiTreeNodeFlags_SpanAvailWidth | ImGuiTreeNodeFlags_OpenOnDoubleClick
   | ImGuiTreeNodeFlags_OpenOnArrow;
    if (selected_component == CType::CTransform) node_flags |= ImGuiTreeNodeFlags_Selected;

    const bool transform_open = ImGui::TreeNodeEx((void*)(intptr_t)CType::CTransform, node_flags,
   "Transform");
    if (ImGui::IsItemClicked()) selected_component = CType::CTransform;

    if (transform_open) {
        ImGui::Text("Position");

        ImGui::PushItemWidth(65);
        ImGui::SameLine(100); ImGui::InputFloat("x##1", &position.x);
        ImGui::SameLine(185); ImGui::InputFloat("y##1", &position.y);
        ImGui::SameLine(270); ImGui::InputFloat("z##1", &position.z);
        ImGui::PopItemWidth();

        ImGui::Text("Scale");

        ImGui::PushItemWidth(65);
        ImGui::SameLine(100); ImGui::InputFloat("x##2", &scale.x);
        ImGui::SameLine(185); ImGui::InputFloat("y##2", &scale.y);
        ImGui::SameLine(270); ImGui::InputFloat("z##2", &scale.z);
        ImGui::PopItemWidth();

        ImGui::Text("Rotation");

        ImGui::PushItemWidth(65);
        ImGui::SameLine(100); ImGui::InputFloat("x##3", &rotation.x);
        ImGui::SameLine(185); ImGui::InputFloat("y##3", &rotation.y);
        ImGui::SameLine(270); ImGui::InputFloat("z##3", &rotation.z);
        ImGui::PopItemWidth();

        ImGui::Text("Start Pos");

        ImGui::PushItemWidth(65);
        ImGui::SameLine(100); ImGui::InputFloat("x##5", &startPos.x);
        ImGui::SameLine(185); ImGui::InputFloat("y##5", &startPos.y);
        ImGui::SameLine(270); ImGui::InputFloat("z##5", &startPos.z);
        ImGui::PopItemWidth();

        ImGui::TreePop();
    }
}
```

References Transform::GetPositionRef(), Transform::GetRotationRef(), Transform::GetScaleRef(), Transform::Get↩ StartPositionRef(), 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 377 of file editor.cpp.

```
                                        {
    ImGui::Begin("World Settings");
    std::string presetName = Engine::GetPresetName();
    if (presetName.compare("no preset") != 0)
        presetName = Editor::Make_Display_String(presetName);

      // Allows user to change the preset that is loaded
    ImGui::Text("Presets"); ImGui::SameLine(120);
    if (ImGui::Button(presetName.c_str())) {
        ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDlgKey##5", "Choose File", ".json",
   std::string(getenv("USERPROFILE")) + "/Documents/pEngine/json/preset/");
    }

    if (ImGuiFileDialog::Instance()->Display("ChooseFileDlgKey##5")) {
        if (ImGuiFileDialog::Instance()->IsOk()) {
            std::string filePath = ImGuiFileDialog::Instance()->GetCurrentPath();
            filePath += "/" + ImGuiFileDialog::Instance()->GetCurrentFileName();

            if (Engine::Restart(filePath)) {
                selected_object = -1;
                selected_component = -1;
                object_to_copy = -1;
            }
        }

        ImGuiFileDialog::Instance()->Close();
    }

    ImGui::PushItemWidth(141);

      // Strength of the light being used
    ImGui::Text("Light Power");
    ImGui::SameLine(120); ImGui::InputFloat("##1", &Engine::GetLightPower());

      // Position of the light being used
    ImGui::Text("Light Position");
    ImGui::PushItemWidth(65);
    ImGui::SameLine(120); ImGui::InputFloat("x##4", &Engine::GetLightPos().x);
    ImGui::SameLine(205); ImGui::InputFloat("y##4", &Engine::GetLightPos().y);
    ImGui::SameLine(290); ImGui::InputFloat("z##4", &Engine::GetLightPos().z);
    ImGui::PopItemWidth();

      // Grav const of the engine
    ImGui::Text("Grav Const");
    ImGui::SameLine(120); ImGui::InputDouble("##5", &Engine::GetGravConst());

    ImGui::PopItemWidth();

    ImGui::End();
}
```

References Engine::GetGravConst(), Engine::GetLightPos(), Engine::GetLightPower(), Engine::GetPresetName(), Make_Display_String(), object_to_copy, Engine::Restart(), selected_component, 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 750 of file editor.cpp.

```
{ 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.

```
                         {
      // Initializing the editor
    editor = new Editor;
    if (!editor) {
        Trace::Message("Editor failed to initialize.\n");
        return false;
    }
    editor->selected_object = -1;
    editor->selected_component = -1;
    editor->object_to_copy = -1;

    IMGUI_CHECKVERSION();
    ImGui::CreateContext();

      // Setting up ImGui flags
    ImGui::GetIO().ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard;
    ImGui::GetIO().ConfigFlags |= ImGuiConfigFlags_DockingEnable;
    ImGui::GetIO().ConfigFlags |= ImGuiConfigFlags_ViewportsEnable;

      // Setting style for ImGui
    ImGui::StyleColorsDark();
    if (ImGui::GetIO().ConfigFlags & ImGuiConfigFlags_ViewportsEnable) {
        ImGui::GetStyle().WindowRounding = 0.f;
        ImGui::GetStyle().Colors[ImGuiCol_WindowBg].w = 1.f;
    }

      // Setting up ImGui
    ImGui_ImplGlfw_InitForOpenGL(Graphics::GetWindow(), true);
    ImGui_ImplOpenGL3_Init("#version 330");

    return true;
}
```

References editor, Graphics::GetWindow(), Trace::Message(), object_to_copy, selected_component, and selected_↩ object.

Referenced by Engine::Initialize().

#### 4.4.2.13 Make_Display_String() `std::string Editor::Make_Display_String ( std::string inputString ) [static], [private]`

Removes the filepath from a filename.

**Parameters**

| | |
|---|---|
| *inputString* | Original filename (with filepath) |

**Returns**

std::string

Definition at line 758 of file editor.cpp.

```
                                                             {
    size_t slashLoc = inputString.find_last_of("/");
    size_t dotLoc = inputString.find_last_of(".");

    if (slashLoc == std::string::npos || dotLoc == std::string::npos)
        return inputString;

    std::string newString = inputString.substr(slashLoc + 1, dotLoc);

    return newString;
}
```

Referenced by Display_Components(), Display_Model(), Display_Scripts(), and Display_World_Settings().

### 4.4.2.14 Render() `void Editor::Render ( ) [static]`

Render the editor.

**Returns**

void

Definition at line 114 of file editor.cpp.

```
                        {
    ImGui::Render();
    ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());

    if (ImGui::GetIO().ConfigFlags & ImGuiConfigFlags_ViewportsEnable) {
        GLFWwindow* backup_current_context = glfwGetCurrentContext();
        ImGui::UpdatePlatformWindows();
        ImGui::RenderPlatformWindowsDefault();
        glfwMakeContextCurrent(backup_current_context);
    }
}
```

Referenced by Graphics::Render().

### 4.4.2.15 Reset() `void Editor::Reset ( ) [static]`

Sets selected object to invalid value.

**Returns**

void

Definition at line 147 of file editor.cpp.

```
                       {
    editor->selected_object = -1;
}
```

References editor, and selected_object.

Referenced by Engine::Restart().

#### 4.4.2.16 Shutdown() `void Editor::Shutdown ( ) [static]`

Destroy editor windows and systems.

**Returns**

void

Definition at line 131 of file editor.cpp.

```
                         {
    if (!editor) return;

    ImGui_ImplOpenGL3_Shutdown();
    ImGui_ImplGlfw_Shutdown();
    ImGui::DestroyContext();

    delete editor;
    editor = nullptr;
}
```

References editor.

Referenced by Engine::Shutdown().

#### 4.4.2.17 Update() `void Editor::Update ( ) [static]`

Updates the editor content and calls display functions.

**Returns**

void

Definition at line 73 of file editor.cpp.

```
                       {
      // ImGui update functions
    ImGui_ImplOpenGL3_NewFrame();
    ImGui_ImplGlfw_NewFrame();
    ImGui::NewFrame();

    //ImGui::ShowDemoWindow();

      // Updating whether program should ignore keyboard input
    if (!ImGui::GetIO().WantCaptureKeyboard) {
        editor->takeKeyboardInput = true;
    }
    else {
        editor->takeKeyboardInput = false;
    }

      // Keyboard shortcuts
    if (!editor->takeKeyboardInput) {
          // Save current settings as preset
        if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_LEFT_CONTROL) == GLFW_PRESS) {
            if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_S) == GLFW_PRESS) {
                if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_S) == GLFW_RELEASE) {
                        Engine::Write();
                }
            }
        }
    }

      // Display the different windows
    editor->Display_Dockspace();
    editor->Display_Scene();
    editor->Display_Components();
    editor->Display_World_Settings();
    editor->Display_Camera_Settings();
}
```

References Display_Camera_Settings(), Display_Components(), Display_Dockspace(), Display_Scene(), Display_↩
World_Settings(), editor, Graphics::GetWindow(), takeKeyboardInput, and Engine::Write().

Referenced by Engine::Update().

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

- editor.hpp
- editor.cpp

## 4.5 Engine Class Reference

`#include <engine.hpp>`

### Static Public Member Functions

- static bool Initialize ()

  *Initializes the engine and the systems in the engine.*
- static void Update ()

  *Updates object and camera. Object updates have a fixed time step, camera updates have variable time step.*
- static void Shutdown ()

  *Shutdown systems and then engine.*
- static bool Restart ()

  *Resets the objects in the engine.*
- static bool Restart (std::string presetName)

  *Resets the engine to the given preset.*
- static float GetDeltaTime ()

  *Returns delta time (variable)*
- static float GetDt ()

  *Returns delta time (fixed)*
- static double & GetGravConst ()

  *Returns gravitational constant.*
- static std::string GetPresetName ()

  *Returns the name of the current preset.*
- static float & GetLightPower ()

  *Returns reference to power of the light in the scene.*
- static glm::vec3 & GetLightPos ()

  *Returns reference to the position of the light in the scene.*
- static void Write ()

  *Writes the engine data to a preset file (creates new one if it doesn't already exist)*
- static void SetPresetName (std::string presetName_)

  *Sets the name of the preset file.*

**Private Attributes**

- bool isRunning

  *state of the main loop*
- float deltaTime

  *time between frames*
- float accumulator

  *amount of unused time for physics updates*
- float time

  *total time*
- const float dt = 0.01f

  *fixed delta time for physics updates*
- std::chrono::steady_clock::time_point currentTime

  *current read time*
- std::chrono::steady_clock::time_point newTime

  *newest read time*
- std::chrono::steady_clock::duration timeTaken

  *time between frames*
- double gravConst

  *gravitational constant (used in physics)*
- std::string presetName

  *name of the preset being used*
- float lightPower

  *Power of the light in the scene.*
- glm::vec3 lightPos

  *Position of the light in the scene.*

### 4.5.1 Detailed Description

Engine class

Definition at line 24 of file engine.hpp.

### 4.5.2 Member Function Documentation

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

Returns delta time (variable)

**Returns**

float Variable delta time

Definition at line 214 of file engine.cpp.

```
214 { 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 221 of file engine.cpp.
```
221 { 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 228 of file engine.cpp.
```
228 { 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 249 of file engine.cpp.
```
249 { 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 242 of file engine.cpp.

```
242 { 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 235 of file engine.cpp.

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

References engine, and presetName.

Referenced by Editor::Display_World_Settings().

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

Initializes the engine and the systems in the engine.

**Returns**

true

false

Definition at line 42 of file engine.cpp.

```
42                        {
43       // Initializing engine
44     engine = new Engine;
45     if (!engine) {
46         Trace::Message("Engine was not initialized.\n");
47         return false;
48     }
49 
50       // Initializing random
51     if (!Random::Initialize()) return false;
52 
53       // Reading settings from json
54     File_Reader settings;
55     if (settings.Read_File(std::string(getenv("USERPROFILE")) + "/Documents/pEngine/json/settings.json")) {
56           // Setting up sub systems
57         if (!Camera::Initialize(settings)) return false;
58         if (!Graphics::Initialize(settings)) return false;
59         if (!Model_Data_Manager::Initialize()) return false;
60         if (!Texture_Manager::Initialize()) return false;
61 
62         File_Reader preset;
63 
64         engine->presetName = std::string(getenv("USERPROFILE")) + "/Documents/pEngine/json/preset/" +
       settings.Read_String("preset");
65         if (preset.Read_File(engine->presetName)) {
66             engine->gravConst = preset.Read_Double("gravConst");
67             engine->lightPos = preset.Read_Vec3("lightPos");
68             if (engine->lightPos == glm::vec3(0.f)) {
69                 engine->lightPos = glm::vec3(4, 4, 0);
70             }
71             if (!Object_Manager::Initialize(preset)) return false;
72         }
73         else {
74             engine->presetName = "no preset";
75             if (!Object_Manager::Initialize()) return false;
76         }
77 
78         engine->gravConst = 0.0;
79 
80         engine->lightPower = 1000.f;
81     }
82     else {
83         engine->presetName = "no preset";
84         engine->gravConst = 0.0;
85 
86         engine->lightPower = 1000.f;
87         engine->lightPos = glm::vec3(4, 4, 0);
88 
89           // Setting up sub systems
90         if (!Camera::Initialize()) return false;
91         if (!Graphics::Initialize()) return false;
92         if (!Model_Data_Manager::Initialize()) return false;
93         if (!Texture_Manager::Initialize()) return false;
94         if (!Object_Manager::Initialize()) return false;
95     }
96 
97       // Initializing the editor
98     if (!Editor::Initialize()) return false;
99 
100       // Setting up variables used for dt
101     engine->currentTime = std::chrono::steady_clock::now();
102     engine->accumulator = 0.f;
103     engine->time = 0.f;
104     engine->isRunning = true;
105 
106     return true;
107 }
```

References accumulator, currentTime, engine, gravConst, Random::Initialize(), Editor::Initialize(), Texture_Manager↩ ::Initialize(), Model_Data_Manager::Initialize(), Object_Manager::Initialize(), Camera::Initialize(), Graphics::Initialize(),

isRunning, lightPos, lightPower, Trace::Message(), presetName, File_Reader::Read_Double(), File_Reader::Read_↩
File(), File_Reader::Read_String(), File_Reader::Read_Vec3(), and time.

Referenced by main().

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

Resets the objects in the engine.

**Returns**

true

false

Definition at line 164 of file engine.cpp.

```
                    {
    // Initializing object manager
  File_Reader settings;
  if (! settings.Read_File(std::string(getenv("USERPROFILE")) + "/Documents/pEngine/json/settings.json"))
 return false;

  File_Reader preset;
  if (!preset.Read_File(engine->presetName)) return false;

    // Removing all current objects
  Object_Manager::Shutdown();
  Editor::Reset();

  engine->presetName = settings.Read_String("preset");
  engine->gravConst = preset.Read_Double("gravConst");
  if (!Object_Manager::Initialize(preset)) return false;

  return true;
}
```

References engine, gravConst, Object_Manager::Initialize(), presetName, File_Reader::Read_Double(), File_Reader↩
::Read_File(), File_Reader::Read_String(), Editor::Reset(), and Object_Manager::Shutdown().

Referenced by Editor::Display_World_Settings(), and Graphics::Update().

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

Resets the engine to the given preset.

**Parameters**

| | |
|---|---|
| *presetName* | Given preset |

**Returns**

true

false

Definition at line 190 of file engine.cpp.

```
                                       {
    // Initializing object manager
  File_Reader settings;
  settings.Read_File(std::string(getenv("USERPROFILE")) + "/Documents/pEngine/json/settings.json");
  Trace::Message(presetName + "\n");
  File_Reader preset;
  if (!preset.Read_File(presetName)) return false;

    // Removing all current objects
  Object_Manager::Shutdown();
  Editor::Reset();

  engine->presetName = presetName;
  engine->gravConst = preset.Read_Double("gravConst");
  if (!Object_Manager::Initialize(preset)) return false;

  return true;
}
```

References engine, gravConst, Object_Manager::Initialize(), Trace::Message(), presetName, File_Reader::Read_↩ Double(), File_Reader::Read_File(), Editor::Reset(), and Object_Manager::Shutdown().

#### 4.5.2.10 SetPresetName() `void Engine::SetPresetName ( std::string presetName_ ) [static]`

Sets the name of the preset file.

**Parameters**

| | |
|---|---|
| *preset↩ Name_* | |

**Returns**

void

Definition at line 273 of file engine.cpp.

```
                                             {
  engine->presetName = presetName_;
}
```

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 141 of file engine.cpp.

```
                         {
    if (!engine) return;

      // Shutdown sub systems
    Editor::Shutdown();
    Random::Shutdown();
    Object_Manager::Shutdown();
    Graphics::Shutdown();
    Camera::Shutdown();
    Texture_Manager::Shutdown();
    Model_Data_Manager::Shutdown();

      // Delete engine object
    delete engine;
    engine = nullptr;
}
```

References engine, Random::Shutdown(), Editor::Shutdown(), Model_Data_Manager::Shutdown(), Texture_Manager↩ ::Shutdown(), Camera::Shutdown(), Object_Manager::Shutdown(), and Graphics::Shutdown().

Referenced by main().

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

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

**Returns**

void

Definition at line 115 of file engine.cpp.

```
                       {
      // Calculating dt
    engine->newTime = std::chrono::steady_clock::now();
    engine->timeTaken = engine->newTime - engine->currentTime;
    engine->deltaTime = float(engine->timeTaken.count()) *
        std::chrono::steady_clock::period::num / std::chrono::steady_clock::period::den;
    engine->currentTime = engine->newTime;
    engine->accumulator += engine->deltaTime;

    Editor::Update();
    Camera::Update();
      // Only called when it is time (fixed time step)
    while (engine->accumulator >= engine->dt) {
          // Update objects
        Object_Manager::Update();
          // Update dt related variables
        engine->accumulator -= engine->dt;
        engine->time += engine->dt;
    }
}
```

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 257 of file engine.cpp.

```
                    {
    File_Writer writer;

    writer.Write_Value("gravConst", engine->gravConst);
    writer.Write_Vec3("lightPos", engine->lightPos);
    Object_Manager::Write(writer);

    writer.Write_File(engine->presetName);
}
```

References engine, gravConst, lightPos, presetName, Object_Manager::Write(), File_Writer::Write_File(), File_Writer↩ ::Write_Value(), and File_Writer::Write_Vec3().

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

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

- engine.hpp
- engine.cpp

## 4.6 File_Reader Class Reference

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

### Public Member Functions

- bool Read_File (std::string filename)

  *Reads the json file data into the root variable.*
- int Read_Int (std::string valueName)

  *Reads int from the json file stored in root.*
- std::string Read_String (std::string valueName)

  *Reads std::string from the json file stored in root.*
- glm::vec3 Read_Vec3 (std::string valueName)

  *Reads glm::vec3 from the json file stored in root. glm::vec3 is constructed from an array.*
- bool Read_Bool (std::string valueName)

  *Reads bool from the json file stored in root.*
- float Read_Float (std::string valueName)

  *Reads float from the json stored in root.*
- double Read_Double (std::string valueName)

  *Reads double from the json stored in root.*
- std::string Read_Object_Name (std::string valueName)

  *Reads the name of an object from an object list (preset folder)*
- std::string Read_Object_Template_Name (std::string valueName)

  *Reads the name of the template file for object.*
- glm::vec3 Read_Object_Position (std::string valueName)

  *Reads the position of an object from an object list (preset folder)*
- glm::vec3 Read_Object_Scale (std::string valueName)

  *Reads the scale of an object.*
- std::string Read_Behavior_Name (std::string valueName)

  *Reads the name of the behavior.*

#### Private Attributes

- rapidjson::Document root

  *Holds the data of the json file.*

### 4.6.1 Detailed Description

File_Reader class

Definition at line 24 of file file_reader.hpp.

### 4.6.2 Member Function Documentation

#### 4.6.2.1 Read_Behavior_Name() `std::string File_Reader::Read_Behavior_Name ( std::string valueName )`

Reads the name of the behavior.

**Parameters**

| | |
|---|---|
| *valueName* | Behavior to read |

**Returns**

std::string Name of the behavior

Definition at line 205 of file file_reader.cpp.

```
{
    // Checking if value exists
    if (!root["behaviors"].HasMember(valueName.c_str())) {
        return std::string("");
    }

    return root["behaviors"][valueName.c_str()].GetString();
}
```

Referenced by Behavior::Read().

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

Reads bool from the json file stored in root.

**Parameters**

| | |
|---|---|
| *valueName* | Name of the bool in the json file |

**Returns**

true

false

Definition at line 96 of file file_reader.cpp.

```
                                              {
        // Checking if the value is a bool
      if (!root.HasMember(valueName.c_str())) {
          return false;
      }
      return root[valueName.c_str()].GetBool();
  }
```

### 4.6.2.3 Read_Double() `double File_Reader::Read_Double ( std::string valueName )`

Reads double from the json stored in root.

**Parameters**

| | |
|---|---|
| *valueName* | Name of the double in the json file |

**Returns**

double Value that was read

Definition at line 124 of file file_reader.cpp.

```
                                                    {
        // Checking if the value is a double (has decimal)
      if (!root.HasMember(valueName.c_str())) {
          return false;
      }
      return root[valueName.c_str()].GetDouble();
  }
```

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

### 4.6.2.4 Read_File() `bool File_Reader::Read_File ( std::string filename )`

Reads the json file data into the root variable.

**Parameters**

| | |
|---|---|
| *filename* | Name of the file to be read |

**Returns**

true

false

Definition at line 32 of file file_reader.cpp.

```
                                            {
   // Opening the json file
  std::string fileToOpen = filename;
  FILE* file = fopen(fileToOpen.c_str(), "r");
  if (!file) return false;

  char buffer[65536];
  FileReadStream stream(file, buffer, sizeof(buffer));
  root.ParseStream<0, UTF8<>, FileReadStream>(stream);

  fclose(file);
  return true;
}
```

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

### 4.6.2.5 Read_Float() `float File_Reader::Read_Float ( std::string valueName )`

Reads float from the json stored in root.

**Parameters**

| | |
|---|---|
| *valueName* | Name of the float in the json file |

**Returns**

float Value that was read

Definition at line 110 of file file_reader.cpp.

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

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

#### 4.6.2.6 Read_Int() `int File_Reader::Read_Int ( std::string valueName )`

Reads int from the json file stored in root.

**Parameters**

| | |
|---|---|
| *valueName* | Name of the int in the json file |

**Returns**

int Value that was read

Definition at line 52 of file file_reader.cpp.

```
                                                  {
        // Checking if the value is an int
      if (!root.HasMember(valueName.c_str())) {
          return 0;
      }
      return root[valueName.c_str()].GetInt();
}
```

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

#### 4.6.2.7 Read_Object_Name() `std::string File_Reader::Read_Object_Name ( std::string valueName )`

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

**Parameters**

| | |
|---|---|
| *valueName* | Specifies which object |

**Returns**

std::string Name of the object

Definition at line 138 of file file_reader.cpp.

```
                                                              {
        // Checking if the value exists
      if (!root.HasMember(valueName.c_str())) {
          return std::string("");
      }
      if (!root[valueName.c_str()].HasMember("objectName")) {
          return std::string("");
      }

      return root[valueName.c_str()]["objectName"].GetString();
}
```

Referenced by Object_Manager::ReadList().

#### 4.6.2.8 Read_Object_Position() `glm::vec3 File_Reader::Read_Object_Position ( std::string valueName )`

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

**Parameters**

| | |
|---|---|
| *valueName* | Specifies which object |

**Returns**

glm::vec3 Position of object

Definition at line 174 of file file_reader.cpp.

```
                                                              {
    if (!root[valueName.c_str()].HasMember("position")) {
        return glm::vec3(0.f, 0.f, 0.f);
    }

    Value& array = root[valueName.c_str()]["position"];
    return glm::vec3(array[0].GetFloat(), array[1].GetFloat(), array[2].GetFloat());
}
```

Referenced by Object_Manager::ReadList().

### 4.6.2.9 Read_Object_Scale() `glm::vec3 File_Reader::Read_Object_Scale ( std::string valueName )`

Reads the scale of an object.

**Parameters**

| | |
|---|---|
| *valueName* | |

**Returns**

glm::vec3

Definition at line 189 of file file_reader.cpp.

```
                                                           {
      // Checking if value exists
    if (!root[valueName.c_str()].HasMember("scale")) {
        return glm::vec3(0.f, 0.f, 0.f);
    }

    Value& array = root[valueName.c_str()]["scale"];
    return glm::vec3(array[0].GetFloat(), array[1].GetFloat(), array[2].GetFloat());
}
```

Referenced by Object_Manager::ReadList().

### 4.6.2.10 Read_Object_Template_Name() `std::string File_Reader::Read_Object_Template_Name ( std::string valueName )`

Reads the name of the template file for object.

**Parameters**

| | |
|---|---|
| *valueName* | |

**Returns**

std::string

Definition at line 156 of file file_reader.cpp.

```
                                                                  {
      // Checking if the value exists
    if (!root.HasMember(valueName.c_str())) {
        return std::string("");
    }
    if (!root[valueName.c_str()].HasMember("templateName")) {
        return std::string("");
    }

    return root[valueName.c_str()]["templateName"].GetString();
}
```

Referenced by Object_Manager::ReadList().

#### 4.6.2.11 Read_String() `std::string File_Reader::Read_String ( std::string valueName )`

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

**Parameters**

| | |
|---|---|
| *valueName* | Name of the std::string in the json file |

**Returns**

std::string Value that was read

Definition at line 66 of file file_reader.cpp.

```
                                                   {
      // Checking if the value is a std::string
    if (!root.HasMember(valueName.c_str())) {
        return std::string("");
    }
    return root[valueName.c_str()].GetString();
}
```

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

#### 4.6.2.12 Read_Vec3() `glm::vec3 File_Reader::Read_Vec3 ( std::string valueName )`

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

**Parameters**

| | |
|---|---|
| *valueName* | Name of the glm::vec3 in the json file |

**Returns**

glm::vec3 Value that was read

Definition at line 81 of file file_reader.cpp.

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

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

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

- file_reader.hpp
- file_reader.cpp

## 4.7 File_Writer Class Reference

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

**Public Member Functions**

- File_Writer ()

  *Creates root object to write data into.*
- void Write_File (std::string filename)

  *Writes all the data stored in root to the given filename.*
- void Write_Vec3 (std::string valueName, glm::vec3 value)

  *Write a glm::vec3 into root.*
- void Write_String (std::string valueName, std::string value)

  *Write a std::string into root.*
- template< typename T >
  void Write_Value (std::string valueName, T value)

  *Writes most values to root (can't do strings)*
- void Write_Behavior_Name (std::vector< std::string > &behaviorNames)

  *Writing behaviorNames into nested object and then into root.*
- void Write_Object_Data (Object ∗object)

  *Writing data of an object into root.*

**Private Attributes**

- rapidjson::Document root

  *Holds the data for the json file.*

### 4.7.1 Detailed Description

File_Writer class

Definition at line 30 of file file_writer.hpp.

### 4.7.2 Constructor & Destructor Documentation

#### 4.7.2.1 File_Writer() `File_Writer::File_Writer ( )`

Creates root object to write data into.

Definition at line 27 of file file_writer.cpp.

```
                          {
    root.SetObject();
}
```

### 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**

| | |
|---|---|
| *behaviorNames* | |

Definition at line 88 of file file_writer.cpp.

```
                                                                     {
    Value behaviors(kObjectType);

      // Filling object
    for (unsigned i = 0; i < behaviorNames.size(); ++i) {
        std::string behaviorName = std::string("behavior_" + std::to_string(i));
        Value name(behaviorName.c_str(), SizeType(behaviorName.size()), root.GetAllocator());

        behaviors.AddMember(name, StringRef(behaviorNames[i].c_str()), root.GetAllocator());
    }
```

```
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**

| | |
|---|---|
| *filename* | |

Definition at line 36 of file file_writer.cpp.

```
36                                             {
37      std::string fileToOpen = 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**

| | |
|---|---|
| *object* | |

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()),
       root.GetAllocator());
136     objectData.AddMember(StringRef("templateName"), templateName, root.GetAllocator());
137     objectData.AddMember(StringRef("position"), pos, root.GetAllocator());
138     objectData.AddMember(StringRef("scale"), scale, root.GetAllocator());
139
140       // Nesting object into root
141     std::string objectIdName = "object_" + std::to_string(object->GetId());
142     Value name(objectIdName.c_str(), SizeType(objectIdName.size()), root.GetAllocator());
143     root.AddMember(name, objectData, root.GetAllocator());
144 }
```

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**

| | |
|---|---|
| *valueName* | |
| *value* | |

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**

|  |  |
|---|---|
| *T* |  |

**Parameters**

|  |  |
|---|---|
| *valueName* | Name of value being written to root |
| *value* | Value being written to root |

Definition at line 46 of file file_writer.hpp.

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

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**

|  |  |
|---|---|
| *valueName* | Name of glm::vec3 |
| *value* | glm::vec3 to write |

Definition at line 57 of file file_writer.cpp.

```
                                                {
    // Storing glm::vec3 in array that rapidjson can write
    Value vector3(kArrayType);
    vector3.PushBack(value.x, root.GetAllocator());
    vector3.PushBack(value.y, root.GetAllocator());
    vector3.PushBack(value.z, root.GetAllocator());

    // Writing vector3 into root
    Value name(valueName.c_str(), SizeType(valueName.size()), root.GetAllocator());
    root.AddMember(name, vector3, root.GetAllocator());
}
```

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 Initialize ()

    *Initializes the Graphics system using default values.*
- 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**

| | |
|---|---|
| *width* | |
| *height* | |

Definition at line 51 of file graphics.cpp.

```
                                        {
      windowSize.first = width;
      windowSize.second = height;
  }
```

### 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**

| | |
|---|---|
| *error* | Error that occurred |
| *description* | Description of error |

**Returns**

void

Definition at line 275 of file graphics.cpp.

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

References Trace::Message().

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

Checking for error in given enum.

**Parameters**

| | |
|---|---|
| *error* | Possible error |

**Returns**

true

false

Definition at line 286 of file graphics.cpp.

```
                                       {
    error = glGetError();
    if (error != GL_NO_ERROR) {
        Trace::Message("Error initializing OpenGl. \n");
        return false;
    }

    return true;
}
```

References Trace::Message().

Referenced by InitializeGL().

#### 4.8.3.3 GetWindow() `GLFWwindow * Graphics::GetWindow ( ) [static]`

Return the graphics window.

**Returns**

GLFWwindow∗

Definition at line 310 of file graphics.cpp.

```
                                 {
    return graphics->window;
}
```

References graphics, and window.

Referenced by Editor::Display_Scene(), Editor::Initialize(), Editor::Update(), Camera::Update(), and Update().

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

Returns window size.

**Returns**

std::pair<int, int>

Definition at line 301 of file graphics.cpp.

```
                                        {
    return graphics->windowSize;
}
```

References graphics, and windowSize.

#### 4.8.3.5 Initialize() [1/2] `bool Graphics::Initialize ( ) [static]`

Initializes the Graphics system using default values.

**Returns**

true

false

Definition at line 115 of file graphics.cpp.

```
                         {
      // Initializing graphics
    graphics = new Graphics(1920, 1080);
    if (!graphics) {
        Trace::Message("Graphics was not initialized.");
        return false;
    }

      // Setting up error recording with graphics
    glfwSetErrorCallback(ErrorCallback);

    if (!glfwInit()) {
        Trace::Message("Could not initialize GLFW.\n");
        return false;
    }

      // Setting up the graphics window
    graphics->window = glfwCreateWindow(graphics->windowSize.first, graphics->windowSize.second,
        "pEngine", nullptr, nullptr);
    if (!graphics->window) {
        Trace::Message("Error creating window.\n");
        return false;
    }

      // Setting up callback functions
    glfwSetCursorPosCallback(graphics->window, Camera::MouseUpdate);

    glfwMakeContextCurrent(graphics->window);
    //glfwSwapInterval(1);
    InitializeGL();

    glewExperimental = GL_TRUE;
    glewInit();

      // Setting up input for keyboard and mouse using glfw library
    glfwSetInputMode(graphics->window, GLFW_STICKY_KEYS, GL_TRUE);
    glfwSetInputMode(graphics->window, GLFW_CURSOR, GLFW_CURSOR_HIDDEN);

    glGenVertexArrays(1, &graphics->vertexArrayId);
    glBindVertexArray(graphics->vertexArrayId);

    if (!Shader::Initialize()) return false;

    return true;
}
```

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

Referenced by Engine::Initialize().

#### 4.8.3.6 Initialize() [2/2]

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

Initializes the Graphics system using the settings in the given data.

**Parameters**

| | |
|---|---|
| *settings* | Settings information |

**Returns**

true

false

Definition at line 63 of file graphics.cpp.

```
                                                  {
      // Initializing graphics
    graphics = new Graphics(settings.Read_Int("windowWidth"), settings.Read_Int("windowHeight"));
    if (!graphics) {
        Trace::Message("Graphics was not initialized.");
        return false;
    }

      // Setting up error recording with graphics
    glfwSetErrorCallback(ErrorCallback);

    if (!glfwInit()) {
        Trace::Message("Could not initialize GLFW.\n");
        return false;
    }

      // Setting up the graphics window
    graphics->window = glfwCreateWindow(graphics->windowSize.first, graphics->windowSize.second,
        "pEngine", nullptr, nullptr);
    if (!graphics->window) {
        Trace::Message("Error creating window.\n");
        return false;
    }

      // Setting up callback functions
    glfwSetCursorPosCallback(graphics->window, Camera::MouseUpdate);

    glfwMakeContextCurrent(graphics->window);
    //glfwSwapInterval(1);
    InitializeGL();

    glewExperimental = GL_TRUE;
    glewInit();

```

```
     // Setting up input for keyboard and mouse using glfw library
    glfwSetInputMode(graphics->window, GLFW_STICKY_KEYS, GL_TRUE);
    glfwSetInputMode(graphics->window, GLFW_CURSOR, GLFW_CURSOR_HIDDEN);

     glGenVertexArrays(1, &graphics->vertexArrayId);
     glBindVertexArray(graphics->vertexArrayId);

     if (!Shader::Initialize(settings)) return false;

     return true;
 }
```

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

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

Initializes the settings of the graphics system.

**Returns**

true

false

Definition at line 167 of file graphics.cpp.

```
                              {
     GLenum error = GL_NO_ERROR;

     glClearColor(0.f, 0.f, 0.f, 1.f);
     if (!Graphics::ErrorCheck(error)) return false;

     glClearDepth(1.f);
     if (!Graphics::ErrorCheck(error)) return false;

     glEnable(GL_DEPTH_TEST);
     if (!Graphics::ErrorCheck(error)) return false;

     glDepthFunc(GL_LEQUAL);
     if (!Graphics::ErrorCheck(error)) return false;

     glShadeModel(GL_SMOOTH);
     if (!Graphics::ErrorCheck(error)) return false;

     glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
     if (!Graphics::ErrorCheck(error)) return false;

     glEnable(GL_CULL_FACE);
     if (!Graphics::ErrorCheck(error)) return false;

     return true;
 }
```

References ErrorCheck().

#### 4.8.3.8 Render() `void Graphics::Render ( ) [static]`

Renders all of the objects in the object_manager.

**Returns**

void

Definition at line 221 of file graphics.cpp.

```
                        {
    // Setting up graphics system for rendering
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  Shader::Update();

  glm::mat4 projection = perspective(radians(Camera::GetFov()), (float)graphics->windowSize.first /
      (float)graphics->windowSize.second, Camera::GetNear(), Camera::GetFar());

    // Getting the view matrix of the camera
  glm::mat4 view = lookAt(
      Camera::GetPosition(),
      Camera::GetPosition() + Camera::GetFront(),
      Camera::GetUp());

    // Rendering all of the objects
  for (unsigned i = 0; i < Object_Manager::GetSize(); ++i) {
      Object* object = Object_Manager::FindObject(i);

      Model* model = object->GetComponent<Model>();
      if (!model) continue;

      model->Draw(projection, view);
  }

  Editor::Render();

  glfwSwapBuffers(graphics->window);
}
```

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.

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

Shutdown the graphics system.

**Returns**

void

Definition at line 255 of file graphics.cpp.

```
                          {
  if (!graphics) return;

  Shader::Shutdown();
  glDeleteVertexArrays(1, &graphics->vertexArrayId);
    // Shutting down opengl
  glfwDestroyWindow(graphics->window);
  glfwTerminate();
    // Deleting graphics object
  delete graphics;
  graphics = nullptr;
}
```

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

Referenced by Engine::Shutdown().

#### 4.8.3.10 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 200 of file graphics.cpp.

```
                  {
    while(!glfwWindowShouldClose(graphics->window)) {
        // Run updates
        Engine::Update();
        Render();
        glfwPollEvents();

        // Check for restart
        if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_R) == GLFW_PRESS && Editor::GetTakeKeyboardInput()) {
            if (glfwGetKey(Graphics::GetWindow(), GLFW_KEY_R) == GLFW_RELEASE) {
                Engine::Restart();
            }
        }
    }
}
```

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

Referenced by main().

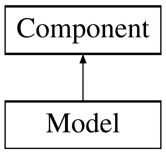
The documentation for this class was generated from the following files:

- 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**

| | |
|---|---|
| *mode_* | Draw mode for opengl |

Definition at line 32 of file model.cpp.

```
: 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**

| | |
|---|---|
| *other* | |

Definition at line 39 of file model.cpp.

```
: 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**

| | |
|---|---|
| *reader* | File with Model data |
| *mode←↩ _* | Draw mode for opengl |

Definition at line 47 of file model.cpp.

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

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.

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

References Model().

#### 4.9.3.2 Draw()

```
void Model::Draw (
        glm::mat4 projection,
        glm::mat4 view )
```

Draw the model.

**Parameters**

| | |
|---|---|
| *projection* | Projection matrix of the scene |
| *view* | View matrix of the scene |

Definition at line 75 of file model.cpp.

```
                                             {
    Transform* transform = GetParent()->GetComponent<Transform>();
    if (!data) return;

    data->Draw(this, transform, projection, view);
}
```

References data, Model_Data::Draw(), Object::GetComponent(), and Component::GetParent().

Referenced by Graphics::Render().

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

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

**Returns**

CType

Definition at line 158 of file model.cpp.

```
                        {
    return CType::CModel;
}
```

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

Returns the filename of the current model.

**Returns**

std::string

Definition at line 131 of file model.cpp.

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

References data, and Model_Data::GetModelName().

Referenced by Editor::Display_Model().

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

Returns pointer to texture object.

**Returns**

Texture*

Definition at line 151 of file model.cpp.

```
{ return texture; }
```

References texture.

Referenced by Model_Data::Draw().

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

Returns the filename of the current texture.

**Returns**

std::string

Definition at line 141 of file model.cpp.

```
                                        {
    if (!texture) return "no texture";
    return texture->GetTextureName();
}
```

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**

| | |
|---|---|
| *reader* | File_reader object that contains Model info |

Definition at line 64 of file model.cpp.

```
                                        {
    data = Model_Data_Manager::Get(reader);
    texture = Texture_Manager::Get(reader);
}
```

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**

| | |
|---|---|
| *reader* | File that contains the name of the model's file |

Definition at line 87 of file model.cpp.

```
{ 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**

| | |
|---|---|
| *modelName* | |

Definition at line 107 of file model.cpp.

```
                                          {
    Model_Data* proxy = Model_Data_Manager::Get(modelName);
    if (!proxy) return;

    data = proxy;
}
```

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**

| *textureName* | |
|---|---|

Definition at line 119 of file model.cpp.

```
                                                 {
    Texture* proxy = Texture_Manager::Get(textureName);
    if (!proxy) return;

    texture = proxy;
}
```

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**

| *writer* | |
|---|---|

Definition at line 94 of file model.cpp.

```
                                        {
    std::string modelName = data->GetModelName();
    std::string textureName = texture->GetTextureName();

    writer.Write_String("modelToLoad", modelName.c_str());
    writer.Write_String("textureToLoad", textureName.c_str());
}
```

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.
```
{}
```

#### 4.10.2.2 Model_Data() [2/2] `Model_Data::Model_Data ( const Model_Data & other )`

Copy constructor.

**Parameters**

| | |
|---|---|
| *other* | |

Definition at line 40 of file model_data.cpp.
```
                                               {
    for (float vert : other.vertices) {
        vertices.emplace_back(vert);
    }
    for (float norm : other.normals) {
        normals.emplace_back(norm);
    }
    for (float uv : other.uvs) {
        uvs.emplace_back(uv);
    }

    vertexbuffer = other.vertexbuffer;
    normalbuffer = other.normalbuffer;
    uvbuffer = other.uvbuffer;
}
```

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.
```
                          {
    glDeleteBuffers(1, &vertexbuffer);
    glDeleteBuffers(1, &uvbuffer);
    glDeleteBuffers(1, &normalbuffer);
}
```

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**

| | |
|---|---|
| *parent* | Model component |
| *transform* | Transform component |
| *projection* | Projection matrix of the scene |
| *view* | View matrix of the scene |

Definition at line 224 of file model_data.cpp.

```
{
    // Creating the MVP (Model * View * Projection) matrix
    glm::mat4 model = glm::mat4(1.f);
    model = glm::translate(model, transform->GetPosition());
    model = glm::rotate(model, (transform->GetRotation().x / 180.f) * glm::pi<float>(), glm::vec3(1, 0, 0));
    model = glm::rotate(model, (transform->GetRotation().y / 180.f) * glm::pi<float>(), glm::vec3(0, 1, 0));
    model = glm::rotate(model, (transform->GetRotation().z / 180.f) * glm::pi<float>(), glm::vec3(0, 0, 1));
    model = glm::scale(model, transform->GetScale());

    // Sending data to the shaders
    glm::mat4 MVP = projection * view * model;
    glUniformMatrix4fv(Shader::GetMatrixId(), 1, GL_FALSE, &MVP[0][0]);
    glUniformMatrix4fv(Shader::GetModelMatrixId(), 1, GL_FALSE, &model[0][0]);
    glUniformMatrix4fv(Shader::GetViewMatrixId(), 1, GL_FALSE, &view[0][0]);

    // Sending light data to the shaders
    glm::vec3 lightPos = Engine::GetLightPos();
    glUniform3f(Shader::GetLightId(), lightPos.x, lightPos.y, lightPos.z);
    glUniform1f(Shader::GetLightPowerId(), Engine::GetLightPower());

    // Setup texture for drawing if it exists
    if (parent->GetTexture())
        parent->GetTexture()->Display();

    // Setup the model vertices
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
    glVertexAttribPointer(
        0,
        3,
        GL_FLOAT,
        GL_FALSE,
        0,
        (void*)0
    );

    // Setup the model uv
    glEnableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER, uvbuffer);
    glVertexAttribPointer(
        1,
        2,
        GL_FLOAT,
        GL_FALSE,
        0,
        (void*)0
    );

    // Setup the model normals
    glEnableVertexAttribArray(2);
    glBindBuffer(GL_ARRAY_BUFFER, normalbuffer);
    glVertexAttribPointer(
        2,
        3,
        GL_FLOAT,
        GL_FALSE,
        0,
        (void*)0
    );

      // Draw the object
    glDrawArrays(GL_TRIANGLES, 0, vertices.size());

      // Disable data sent to shaders
    glDisableVertexAttribArray(0);
    glDisableVertexAttribArray(1);
    glDisableVertexAttribArray(2);

}
```

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

Referenced by Model::Draw().

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

Returns the filename that the models data was gotten from.

**Returns**

string Name of the file that contains model data

Definition at line 299 of file model_data.cpp.

```
{ 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**

| | |
|---|---|
| *reader* | File_Reader object containing the model data |

**Returns**

true

false

Definition at line 73 of file model_data.cpp.

```
                                   {
    std::string modelName_ = reader.Read_String("modelToLoad");

    return Read(modelName_);
}
```

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**

| | |
|---|---|
| *model↵ Name_* | Model's filename |

**Returns**

true

false

Definition at line 86 of file model_data.cpp.

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

References Read().

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

Reads model data from file.

**Parameters**

| | |
|---|---|
| *model↵ Name_* | Model's filename |

**Returns**

true

false

Definition at line 95 of file model_data.cpp.

```
                                      {
  // Opening the file
std::string fileToOpen = std::string(getenv("USERPROFILE")) + "/Documents/pEngine/models/" + modelName_;
  // Setting the name of the file (used in model_data_manager)
modelName = fileToOpen;

 FILE* file = fopen(fileToOpen.c_str(), "r");
 if (!file) {
     file = fopen(modelName_.c_str(), "r");
     if (!file) {
         return false;
     }
     else {
         modelName = modelName_;
     }
 }

   // Creating variables for reading
 std::vector<unsigned> vertex_indices, uv_indices, normal_indices;
 std::vector<glm::vec3> temp_vertices;
 std::vector<glm::vec2> temp_uvs;
 std::vector<glm::vec3> temp_normals;

   // Until the whole file is read
 while (true) {
     char line_header[256];

       // Getting next line of the file
     int res = fscanf(file, "%s", line_header);
     if (res == EOF) break;

       // Checking for which data needs to be read in
     if (strcmp(line_header,"v") == 0) {
         glm::vec3 vertex;
         fscanf(file, "%f %f %f\n", &vertex.x, &vertex.y, &vertex.z);
         temp_vertices.emplace_back(vertex);
         continue;
     }

     if (strcmp(line_header, "vt") == 0) {
         glm::vec2 uv;
         fscanf(file, "%f %f\n", &uv.x, &uv.y);
         temp_uvs.emplace_back(uv);
         continue;
     }

     if (strcmp(line_header, "vn") == 0) {
         glm::vec3 normal;
         fscanf(file, "%f %f %f\n", &normal.x, &normal.y, &normal.z);
         temp_normals.emplace_back(normal);
         continue;
     }

     if (strcmp(line_header, "f") == 0) {
           // Connecting face to previous read vertices, uvs, and normals
         unsigned vertex_index[3], uv_index[3], normal_index[3];
         int matches = fscanf(file, "%d/%d/%d %d/%d/%d %d/%d/%d\n", &vertex_index[0], &uv_index[0], &normal_index[0],
             &vertex_index[1], &uv_index[1], &normal_index[1], &vertex_index[2], &uv_index[2], &normal_index[2]);//,

           // Expects models split into triangles
         if (matches != 9) {
             Trace::Message("File is incompatible with this parser. Export using different settings.");
             return false;
         }

           // Setting vertices for current face
         vertices.emplace_back((temp_vertices[vertex_index[0] - 1]).x);
         vertices.emplace_back((temp_vertices[vertex_index[0] - 1]).y);
         vertices.emplace_back((temp_vertices[vertex_index[0] - 1]).z);

         vertices.emplace_back((temp_vertices[vertex_index[1] - 1]).x);
         vertices.emplace_back((temp_vertices[vertex_index[1] - 1]).y);
         vertices.emplace_back((temp_vertices[vertex_index[1] - 1]).z);

         vertices.emplace_back((temp_vertices[vertex_index[2] - 1]).x);
         vertices.emplace_back((temp_vertices[vertex_index[2] - 1]).y);
         vertices.emplace_back((temp_vertices[vertex_index[2] - 1]).z);

```

```
              // Setting uvs for current face
            uvs.emplace_back((temp_uvs[uv_index[0] - 1]).x);
            uvs.emplace_back((temp_uvs[uv_index[0] - 1]).y);

            uvs.emplace_back((temp_uvs[uv_index[1] - 1]).x);
            uvs.emplace_back((temp_uvs[uv_index[1] - 1]).y);

            uvs.emplace_back((temp_uvs[uv_index[2] - 1]).x);
            uvs.emplace_back((temp_uvs[uv_index[2] - 1]).y);

              // Setting normals for current face
            normals.emplace_back((temp_normals[normal_index[0] - 1]).x);
            normals.emplace_back((temp_normals[normal_index[0] - 1]).y);
            normals.emplace_back((temp_normals[normal_index[0] - 1]).z);

            normals.emplace_back((temp_normals[normal_index[1] - 1]).x);
            normals.emplace_back((temp_normals[normal_index[1] - 1]).y);
            normals.emplace_back((temp_normals[normal_index[1] - 1]).z);

            normals.emplace_back((temp_normals[normal_index[2] - 1]).x);
            normals.emplace_back((temp_normals[normal_index[2] - 1]).y);
            normals.emplace_back((temp_normals[normal_index[2] - 1]).z);
        }
    }

      // Bind vertex data to buffers
    glGenBuffers(1, &vertexbuffer);
    glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(float), &vertices[0], GL_STATIC_DRAW);

      // Bind uv data to buffers
    glGenBuffers(1, &uvbuffer);
    glBindBuffer(GL_ARRAY_BUFFER, uvbuffer);
    glBufferData(GL_ARRAY_BUFFER, uvs.size() * sizeof(float), &uvs[0], GL_STATIC_DRAW);

      // Bind normals data to buffers
    glGenBuffers(1, &normalbuffer);
    glBindBuffer(GL_ARRAY_BUFFER, normalbuffer);
    glBufferData(GL_ARRAY_BUFFER, normals.size() * sizeof(float), &normals[0], GL_STATIC_DRAW);

    return true;
}
```

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**

| | |
|---|---|
| *reader* | 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.

```
                                                         {
    std::string filename = reader.Read_String("modelToLoad");
      // Checks name of file against other model data objects
    for (Model_Data* model_data : model_data_manager->models) {
        if (model_data->GetModelName().compare(filename) == 0) {
            return model_data;
        }
    }

      // Creates new Model_Data object, then adds it to list
    Model_Data* data = new Model_Data;
    data->Load(reader);
    model_data_manager->models.emplace_back(data);

    return data;
}
```

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**

| | |
|---|---|
| *modelName* | 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.

```
                                          {
    // Checks name of file against other model data objects
   for (Model_Data* model_data : model_data_manager->models) {
       if (model_data->GetModelName().compare(modelName) == 0) {
           return model_data;
       }
   }

    // Creates new Model_Data object, then adds it to list
   Model_Data* data = new Model_Data;
   if (!data->Load(modelName)) {
       delete data;
       return nullptr;
   }
   model_data_manager->models.emplace_back(data);

   return data;
}
```

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.

```
                                        {
    // Initializing model_data_manager
   model_data_manager = new Model_Data_Manager;
   if (!model_data_manager) {
       Trace::Message("Model Data Manager was not initialized.\n");
       return false;
   }

   model_data_manager->models.reserve(10);
   return true;
}
```

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

Referenced by Engine::Initialize().

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

Deletes all of the Model_Data objects in the models list then deletes model_data_manager.

**Returns**

void

Definition at line 94 of file model_data_manager.cpp.

```
                                   {
    if (!model_data_manager) return;

      // Deleting all of the Model_Data objects
    for (Model_Data* model_data : model_data_manager->models) {
        if (!model_data) continue;

         delete model_data;
         model_data = nullptr;
     }

     delete model_data_manager;
     model_data_manager = nullptr;
}
```

References model_data_manager, and models.

Referenced by Engine::Shutdown().

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

- model_data_manager.hpp
- model_data_manager.cpp

## 4.12 Object Class Reference

`#include <object.hpp>`

### Public Member Functions

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

- template<typename T >
  void RemoveComponent ()

  *Removes component from object.*
- void SetId (int id_)

  *Sets the id of object.*
- int GetId () const

  *Returns the id of object.*
- void SetName (std::string name_)

  *Sets name of object.*
- std::string GetName () const

  *Returns name of object.*
- std::string & GetNameRef ()

  *Returns reference to the name.*
- void SetTemplateName (std::string templateName_)

  *Sets the name of the template file.*
- std::string GetTemplateName () const

  *Returns the name of the template file.*
- bool Read (std::string objectFilename)

  *Reads object from file. This includes the components of an object.*
- bool ReRead (std::string objectFilename)

  *Reading data into object that already exists.*
- void Write (std::string filePath)

  *Writes the data of the object to a template file.*
- std::unordered_map< CType, Component * > GetComponentList ()

  *Returns the list of components.*
- void Clear ()

  *Clears the components from the object.*

## Private Member Functions

- template<typename T >
  T * GetComponentConst () const

  *Get a component of the object (const)*

## Private Attributes

- std::unordered_map< CType, Component * > components

  *List of components.*
- std::string name

  *Name of the object.*
- std::string templateName

  *Name of the template file used.*
- int id

  *Location of object in object_manager.*

### 4.12.1 Detailed Description

Object class

Definition at line 25 of file object.hpp.

### 4.12.2 Constructor & Destructor Documentation

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

Default constructor.

Definition at line 28 of file object.cpp.

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

Referenced by Clone().

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

Copy constructor.

**Parameters**

| | |
|---|---|
| *other* | Object to be copied |

Definition at line 35 of file object.cpp.

```
                                        {
    SetName(other.GetName());
    SetTemplateName(other.GetTemplateName());

      // Copying Behavior component
    Behavior* behavior = other.GetComponentConst<Behavior>();
    if (behavior) {
        Behavior* newBehavior = new Behavior(*behavior);
        AddComponent(newBehavior);
    }

      // Copying Model component
    Model* model = other.GetComponentConst<Model>();
    if (model) {
        Model* newModel = new Model(*model);
        AddComponent(newModel);
    }

      // Copying Physics component
    Physics* physics = other.GetComponentConst<Physics>();
    if (physics) {
        Physics* newPhysics = new Physics(*physics);
        AddComponent(newPhysics);
    }

    // Copying transform component
    Transform* transform = other.GetComponentConst<Transform>();
    if (transform) {
        Transform* newTransform = new Transform(*transform);
        AddComponent(newTransform);
    }
}
```

References AddComponent(), GetComponentConst(), GetName(), GetTemplateName(), SetName(), and Set←TemplateName().

### 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**

| | |
|---|---|
| *component* | Component to be added |

Definition at line 95 of file object.cpp.

```
                                            {
    component->SetParent(this);
    components.emplace(component->GetCType(), component);
}
```

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 275 of file object.cpp.

```
                      {
    Behavior* behavior = GetComponent<Behavior>();
    Model* model = GetComponent<Model>();
    Physics* physics = GetComponent<Physics>();

    if (behavior) {
        delete behavior;
        behavior = nullptr;
    }
    if (model) {
        delete model;
        model = nullptr;
    }
    if (physics) {
        delete physics;
        physics = nullptr;
    }
}
```

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

Clones this object.

**Returns**

Object∗

Definition at line 73 of file object.cpp.

```
                          {
    return new Object(*this);
}
```

References Object().

#### 4.12.3.4 GetComponent() `template<typename T >`
`T* Object::GetComponent ( ) [inline]`

Get a component of the object.

**Template Parameters**

| | |
|---|---|
| *T* | Component class to return |

**Parameters**

| | |
|---|---|
| *type* | Type of component |

**Returns**

T∗ Pointer to the component

Definition at line 44 of file object.hpp.

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

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**

|  |  |
|---|---|
| *T* | Component class to return |

**Parameters**

|  |  |
|---|---|
| *type* | Type of component |

**Returns**

T∗ Pointer to the component

Definition at line 96 of file object.hpp.
```
                                        {
                 // Searching for component using the type (enum as int)
               auto found = components.find(T::GetCType());
               if (found == components.end()) {
                    return nullptr;
                }
                  // Cast found component into correct type
                return (T*)found->second;
           }
```

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 267 of file object.cpp.
```
                                                             {
    return components;
}
```

References components.

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

Returns the id of object.

**Returns**

unsigned Position in Object_Manager

Definition at line 112 of file object.cpp.
```
{ return id; }
```

References id.

Referenced by Object_Manager::CheckName(), Behavior::ClassSetup(), Editor::Display_Components(), and File_↩
Writer::Write_Object_Data().

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

Returns name of object.

**Returns**

string Name of object

Definition at line 128 of file object.cpp.
```
{ return name; }
```

References name.

Referenced by Object_Manager::CheckName(), Object_Manager::FindObject(), Object(), and File_Writer::Write_↩
Object_Data().

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

Returns reference to the name.

**Returns**

std::string&

Definition at line 135 of file object.cpp.
```
{ return name; }
```

References name.

Referenced by Behavior::ClassSetup().

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

Returns the name of the template file.

**Returns**

std::string

Definition at line 149 of file object.cpp.

```
{ return templateName; }
```

References templateName.

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

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

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

**Parameters**

| | |
|---|---|
| *objectFilename* | |

**Returns**

true

false

Definition at line 158 of file object.cpp.

```
                                        {
      // Getting data from file
    File_Reader object_reader;
    if (!object_reader.Read_File(objectFilename)) return false;

      // Reading Behavior component form file
    Behavior* object_behavior = new Behavior(object_reader);
    AddComponent(object_behavior);

      // Reading Model component from file
    Model* object_model = new Model(object_reader);
    AddComponent(object_model);

      // Reading Physics component from file
    Physics* object_physics = new Physics(object_reader);
    AddComponent(object_physics);

      // Reading Transform component from file
    Transform* object_transform = new Transform(object_reader);
    AddComponent(object_transform);

    return true;
}
```

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

Referenced by Object_Manager::ReadList().

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

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

Removes component from object.

**Template Parameters**

| | |
|---|---|
| *T* | |

Definition at line 60 of file object.hpp.

```
                                {
            // Searching for component using the type (enum as int)
          auto found = components.find(T::GetCType());
          if (found == components.end()) return;
            // Delete component
          delete found->second;
          found->second = nullptr;
            // Remove pointer from map
          components.erase(found->first);
      }
```

References components.

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

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

Reading data into object that already exists.

**Parameters**

| | |
|---|---|
| *objectFilename* | Name of template file |

**Returns**

true

false

Definition at line 189 of file object.cpp.

```
                                            {
      // Getting data from file
    File_Reader object_reader;
    if (!object_reader.Read_File(objectFilename)) return false;

    if (name.compare("") == 0)
        SetName(object_reader.Read_String("name"));

    templateName = objectFilename;

      // Reading Model component from file
    Model* object_model = GetComponent<Model>();
    if (!object_model) {
        object_model = new Model;
        AddComponent(object_model);
    }
    object_model->Read(object_reader);

      // Reading Physics component from file
    Physics* object_physics = GetComponent<Physics>();
    if (!object_physics) {
        object_physics = new Physics;
        AddComponent(object_physics);
    }
    object_physics->Read(object_reader);

      // Reading Transform component from file
    Transform* object_transform = GetComponent<Transform>();
    if (!object_transform) {
        object_transform = new Transform;
        AddComponent(object_transform);
    }
    object_transform->Read(object_reader);

      // Reading Behavior component form file
    Behavior* object_behavior = GetComponent<Behavior>();
    if (object_behavior) object_behavior->Clear();
    if (!object_behavior) {
        object_behavior = new Behavior;
        AddComponent(object_behavior);
    }
    object_behavior->Read(object_reader);
    object_behavior->SetupClassesForLua();

    return true;
}
```

References AddComponent(), Behavior::Clear(), name, Behavior::Read(), Model::Read(), Transform::Read(), Physics↩ ::Read(), File_Reader::Read_File(), File_Reader::Read_String(), SetName(), Behavior::SetupClassesForLua(), and templateName.

### 4.12.3.14 SetId() `void Object::SetId ( int id_ )`

Sets the id of object.

**Parameters**

| | |
|---|---|
| *id↩ _↩* | Position in Object_Manager |

Definition at line 105 of file object.cpp.

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

Referenced by Object_Manager::RemoveObject().

### 4.12.3.15 SetName() `void Object::SetName ( std::string name_ )`

Sets name of object.

**Parameters**

| | |
|---|---|
| *name_* | Name of object |

Definition at line 119 of file object.cpp.

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

References Object_Manager::CheckName(), and name.

Referenced by Behavior::ClassSetup(), Editor::Display_Scene(), Object(), Object_Manager::ReadList(), and ReRead().

#### 4.12.3.16 SetTemplateName() `void Object::SetTemplateName ( std::string templateName_ )`

Sets the name of the template file.

**Parameters**

| | |
|---|---|
| *templateName_* | Name of the template file |

Definition at line 142 of file object.cpp.

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

References templateName.

Referenced by Object().

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

Updates object (only physics for now)

Definition at line 81 of file object.cpp.

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

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

Referenced by Object_Manager::Update().

#### 4.12.3.18 Write() `void Object::Write ( std::string filePath )`

Writes the data of the object to a template file.

**Parameters**

| | |
|---|---|
| *filePath* | Filepath for the object being written |

Definition at line 241 of file object.cpp.

```
                                        {
    File_Writer object_writer;
    object_writer.Write_String("name", name);
    templateName = filePath + "/" + name + ".json";
    Trace::Message(templateName + "\n");

    Model* object_model = GetComponent<Model>();
    if (object_model) object_model->Write(object_writer);

    Transform* object_transform = GetComponent<Transform>();
    if (object_transform) object_transform->Write(object_writer);

    Physics* object_physics = GetComponent<Physics>();
    if (object_physics) object_physics->Write(object_writer);

    Behavior* object_behavior = GetComponent<Behavior>();
    if (object_behavior) object_behavior->Write(object_writer);

    object_writer.Write_File(templateName);
}
```

References Trace::Message(), 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 bool Initialize ()

  *Initialize object manager with default values.*
- static void AddObject (Object ∗object)

  *Adds object to object_manager.*
- static Object ∗ FindObject (int id)

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

- static Object ∗ FindObject (std::string objectName)

    *Finds object with the matching name.*
- static unsigned GetSize ()

    *Gets the size of the object_manager object list.*
- static void Update ()

    *Calls the update function for each object in the object list.*
- static void Shutdown ()

    *Deletes all objects in the manager and then the object manager.*
- static std::string CheckName (std::string objectName, int id)

    *Checks if the name of the given object is already being used. If it is being used it applies a number to the back.*
- static void RemoveObject (int id)

    *Removes an object from the object_manager.*
- static void Write (File_Writer &writer)

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

**Private Attributes**

- std::vector< Object ∗ > objects

    *Current objects being tracked by the engine.*

### 4.13.1 Detailed Description

Object_Manager class

Definition at line 25 of file object_manager.hpp.

### 4.13.2 Member Function Documentation

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

Adds object to object_manager.

**Parameters**

| | |
|---|---|
| *object* | Object to be added |

**Returns**

void

Definition at line 72 of file object_manager.cpp.

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

References object_manager, and objects.

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

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

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

**Parameters**

| | |
|---|---|
| *objectName* | |
| *id* | |

**Returns**

std::string

Definition at line 192 of file object_manager.cpp.

```
                                                            {
    // Checking if the name matches any other objects
  int objWithName = 0;
  for (Object* objToCheck : object_manager->objects) {
      if (id != -1 && objToCheck->GetId() == id) continue;
      if (objToCheck->GetName().find(objectName) != std::string::npos)
          ++objWithName;
  }

    // Updating the name
  if (objWithName > 0)
      return objectName + "_" + std::to_string(objWithName);

  return objectName;
}
```

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

Referenced by Object::SetName().

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

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

**Parameters**

| | |
|---|---|
| *id* | Location of object in object_manager object list |

**Returns**

Object∗

Definition at line 84 of file object_manager.cpp.

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

References object_manager, and objects.

Referenced by Behavior::ClassSetup(), Editor::Display_Components(), Editor::Display_Scene(), Graphics::Render(), Shutdown(), Update(), and Physics::UpdateGravity().

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

Finds object with the matching name.

**Parameters**

| | |
|---|---|
| *objectName* | Name to look for |

**Returns**

Object∗

Definition at line 95 of file object_manager.cpp.

```
                                                                  {
    for (Object* object : object_manager->objects) {
        if (objectName.compare(object->GetName()) == 0)
            return object;
    }

     return nullptr;
}
```

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

#### 4.13.2.5 GetSize() `unsigned Object_Manager::GetSize ( ) [static]`

Gets the size of the object_manager object list.

**Returns**

unsigned Size of object list

Definition at line 109 of file object_manager.cpp.

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

References object_manager, and objects.

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

#### 4.13.2.6 Initialize() [1/2] `bool Object_Manager::Initialize ( ) [static]`

Initialize object manager with default values.

**Returns**

true

false

Definition at line 52 of file object_manager.cpp.

```
                                   {
      // Initializing object_manager
    object_manager = new Object_Manager;
    if (!object_manager) {
        Trace::Message("Object Manager was not initialized.");
        return false; // Failed to initialize
    }

      // Adding objects from preset into engine
    object_manager->objects.reserve(10);

    return true; // Successful initialization
}
```

References Trace::Message(), object_manager, and objects.

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

#### 4.13.2.7 Initialize() [2/2] `bool Object_Manager::Initialize ( File_Reader & preset ) [static]`

Initializes the object_manager object. Reads in objects for the given preset.

**Parameters**

| | |
|---|---|
| *preset* | List of objects for this preset |

**Returns**

true

false

Definition at line 31 of file object_manager.cpp.

```
                                                    {
      // Initializing object_manager
    object_manager = new Object_Manager;
    if (!object_manager) {
        Trace::Message("Object Manager was not initialized.");
        return false; // Failed to initialize
    }

      // Adding objects from preset into engine
    object_manager->objects.reserve(10);
    object_manager->ReadList(preset);

    return true; // Successful initialization
}
```

References Trace::Message(), object_manager, objects, and ReadList().

#### 4.13.2.8 ReadList() `void Object_Manager::ReadList ( File_Reader & preset )`

Reads in objects from a preset list that is given.

**Parameters**

| | |
|---|---|
| *preset* | List of objects to be added |

Definition at line 147 of file object_manager.cpp.

```
                                                    {
      // Track which object we are on
    unsigned object_num = 0;

      // Reads objects until there is a failed read
    while (true) {
          // Getting the name of the objects file
        std::string object_name = preset.Read_Object_Name("object_" + std::to_string(object_num));
        std::string template_name = preset.Read_Object_Template_Name("object_" + std::to_string(object_num));
        if (template_name.compare("") == 0) break;

          // Constructing the object
        Object* object = new Object;
        if (!object->Read(std::string(getenv("USERPROFILE")) + "/Documents/pEngine/json/objects/" + template_name)) {
            delete object;
            continue;
        }

        object->SetName(object_name);
        object->SetTemplateName(std::string(getenv("USERPROFILE")) + "/Documents/pEngine/json/objects/" + template_name);
          // Reading in the objects position
        glm::vec3 position = preset.Read_Object_Position("object_" + std::to_string(object_num));
        glm::vec3 scale = preset.Read_Object_Scale("object_" + std::to_string(object_num));
        Transform* transform = object->GetComponent<Transform>();
        transform->SetPosition(position);
        transform->SetStartPosition(position);
        transform->SetScale(scale);
        Behavior* behavior = object->GetComponent<Behavior>();
        behavior->SetupClassesForLua();

          // Adding the object to the manager
        AddObject(object);

        ++object_num;
    }
}
```

References AddObject(), Object::Read(), File_Reader::Read_Object_Name(), File_Reader::Read_Object_Position(), File_Reader::Read_Object_Scale(), File_Reader::Read_Object_Template_Name(), Object::SetName(), Transform::↩ SetPosition(), Transform::SetScale(), Transform::SetStartPosition(), and Behavior::SetupClassesForLua().

Referenced by Initialize().

#### 4.13.2.9 RemoveObject() `void Object_Manager::RemoveObject ( int id ) [static]`

Removes an object from the object_manager.

**Parameters**

| | |
|---|---|
| *id* | id of object to remove |

**Returns**

void

Definition at line 214 of file object_manager.cpp.

```
                                            {
    if (id >= (int)object_manager->objects.size()) return;
    Object* objectToDelete = object_manager->objects[id];

      // Moves all the objects to the right of one being deleted to the left
    unsigned offset = 0;
    for (unsigned objectNum = id + 1; objectNum < object_manager->objects.size(); ++objectNum) {
        Object* objectToSwitch = object_manager->objects[objectNum];
        object_manager->objects[id + offset] = objectToSwitch;
        objectToSwitch->SetId(id + offset++);
    }

      // Deleting the object
    delete objectToDelete;
    objectToDelete = nullptr;
    object_manager->objects.pop_back();
}
```

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

Referenced by Editor::Display_Scene().

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

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

**Returns**

void

Definition at line 127 of file object_manager.cpp.

```
                                    {
    if (!object_manager) return; // If the object_manager doesn't exist

      // Deleting each object in the manager
    for (unsigned i = 0; i < object_manager->objects.size(); ++i) {
        Object* object = object_manager->FindObject(i);
        if (object)
            delete object;
    }

      // Deleting the manager
    delete object_manager;
    object_manager = nullptr;
}
```

References FindObject(), object_manager, and objects.

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

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

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

**Returns**

void

Definition at line 116 of file object_manager.cpp.

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

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

Referenced by Engine::Update().

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

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

**Parameters**

| | |
|---|---|
| *writer* | |

**Returns**

void

Definition at line 238 of file object_manager.cpp.

```
                                  {
    for (Object* object : object_manager->objects) {
        writer.Write_Object_Data(object);
    }
}
```

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

Referenced by Engine::Write().

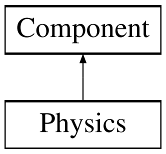
The documentation for this class was generated from the following files:

- object_manager.hpp
- object_manager.cpp

## 4.14 Physics Class Reference

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

Inheritance diagram for Physics:



### Public Member Functions

- Physics ()

  *Creates Physics object with default values.*
- Physics (const Physics &other)

  *Copy constructor.*
- Physics (File_Reader &reader)

  *Creates Physics object using file.*
- Physics ∗ Clone () const

  *Clone Physics object.*
- void SetAcceleration (glm::vec3 accel)

  *Sets acceleration of object.*

- glm::vec3 GetAcceleration () const

  *Returns acceleration of object.*
- glm::vec3 & GetAccelerationRef ()

  *Returns reference to the acceleration of the object.*
- void SetForces (glm::vec3 force)

  *Sets forces acting on object.*
- void AddForce (glm::vec3 force)

  *Adds a force to the current forces acting on the object.*
- glm::vec3 GetForces () const

  *Returns the forces acting on the object.*
- glm::vec3 & GetForcesRef ()

  *Returns reference to the forces acting on the object.*
- void ApplyForce (glm::vec3 direction, float power)

  *Applies force in the given direction using the given power.*
- void SetVelocity (glm::vec3 vel)

  *Sets the velocity of the object.*
- glm::vec3 GetVelocity () const

  *Returns the current velocity of the object.*
- glm::vec3 & GetVelocityRef ()

  *Returns reference to velocity of the object.*
- void SetRotationalVelocity (glm::vec3 rotVel)

  *Sets rotational velocity.*
- glm::vec3 GetRotationalVelocity () const

  *Returns rotational velocity.*
- glm::vec3 & GetRotationalVelocityRef ()

  *Returns reference to rotational velocity.*
- void SetMass (float ma)

  *Sets the mass of the object.*
- float GetMass () const

  *Returns the mass of the object.*
- float & GetMassRef ()

  *Returns reference to mass of the object.*
- void Update ()

  *Updates the physics of the object.*
- void UpdateGravity ()

  *Calculates the gravitational pull each object has on each other.*
- void Read (File_Reader &reader)

  *Reads data for Physics object from file.*
- void Write (File_Writer &writer)

  *Gives physics data to the writer object.*

## Static Public Member Functions

- static CType GetCType ()

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

**Private Attributes**

- glm::vec3 acceleration

  *Acceleration of object.*
- glm::vec3 forces

  *Forces acting on object (reset at end of each update)*
- glm::vec3 velocity

  *Velocity of object.*
- glm::vec3 initialVelocity

  *Starting velocity.*
- glm::vec3 initialAcceleration

  *Starting acceleration.*
- glm::vec3 rotationalVelocity

  *How fast is the object rotating.*
- float mass

  *Mass of object.*

**Additional Inherited Members**

### 4.14.1 Detailed Description

Physics class

Definition at line 25 of file physics.hpp.

### 4.14.2 Constructor & Destructor Documentation

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

Creates Physics object with default values.

Definition at line 32 of file physics.cpp.

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

Referenced by Clone().

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

Copy constructor.

**Parameters**

| | |
|---|---|
| *other* | Physics object to be copied |

Definition at line 41 of file physics.cpp.

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

#### 4.14.2.3 Physics() [3/3] `Physics::Physics ( File_Reader & reader )`

Creates Physics object using file.

**Parameters**

| | |
|---|---|
| *reader* | File to use for making physics object |

Definition at line 50 of file physics.cpp.

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

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**

| | |
|---|---|
| *force* | |

Definition at line 98 of file physics.cpp.

```
{ 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**

| | |
|---|---|
| *direction* | |
| *power* | |

Definition at line 120 of file physics.cpp.

```
{
    direction = glm::normalize(direction);
    direction *= power;

    AddForce(direction);
}
```

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.

```
{
    return new Physics(*this);
}
```

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.

```
{ return acceleration; }
```

References acceleration.

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

Returns reference to the acceleration of the object.

**Returns**

glm::vec3&

Definition at line 84 of file physics.cpp.

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

References acceleration.

Referenced by Behavior::ClassSetup().

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

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

**Returns**

CType

Definition at line 281 of file physics.cpp.

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

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

Returns the forces acting on the object.

**Returns**

glm::vec3

Definition at line 105 of file physics.cpp.

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

References forces.

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

Returns reference to the forces acting on the object.

**Returns**

glm::vec3&

Definition at line 112 of file physics.cpp.
```
112 { return forces; }
```

References forces.

Referenced by Behavior::ClassSetup().

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

Returns the mass of the object.

**Returns**

float

Definition at line 160 of file physics.cpp.
```
160 { return mass; }
```

References mass.

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

Returns reference to mass of the object.

**Returns**

float&

Definition at line 167 of file physics.cpp.
```
167 { return mass; }
```

References mass.

Referenced by Editor::Display_Physics().

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

Returns rotational velocity.

**Returns**

glm::vec3

Definition at line 181 of file physics.cpp.

```
{ return rotationalVelocity; }
```

References rotationalVelocity.

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

Returns reference to rotational velocity.

**Returns**

glm::vec3&

Definition at line 188 of file physics.cpp.

```
{ return rotationalVelocity; }
```

References rotationalVelocity.

Referenced by Editor::Display_Physics().

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

Returns the current velocity of the object.

**Returns**

glm::vec3

Definition at line 139 of file physics.cpp.

```
{ return velocity; }
```

References velocity.

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

Returns reference to velocity of the object.

**Returns**

glm::vec3&

Definition at line 146 of file physics.cpp.

```
{ return velocity; }
```

References velocity.

Referenced by Behavior::ClassSetup(), and Editor::Display_Physics().

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

Reads data for Physics object from file.

**Parameters**

| | |
|---|---|
| *reader* | File to be read from |

Definition at line 257 of file physics.cpp.

```
                                        {
    initialAcceleration = reader.Read_Vec3("acceleration");
    initialVelocity = reader.Read_Vec3("velocity");
    SetAcceleration(initialAcceleration);
    SetVelocity(initialVelocity);
    SetMass(reader.Read_Float("mass"));
}
```

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

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

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

Sets acceleration of object.

**Parameters**

| | |
|---|---|
| *accel* | |

Definition at line 70 of file physics.cpp.

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

References acceleration.

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

#### 4.14.3.17 SetForces() `void Physics::SetForces (`

```
glm::vec3 force )
```

Sets forces acting on object.

**Parameters**

|  |  |
|---|---|
| *force* |  |

Definition at line 91 of file physics.cpp.

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

References forces.

Referenced by Behavior::ClassSetup().

#### 4.14.3.18 SetMass() `void Physics::SetMass (`

```
float ma )
```

Sets the mass of the object.

**Parameters**

|  |  |
|---|---|
| *ma* |  |

Definition at line 153 of file physics.cpp.

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

References mass.

Referenced by Read().

#### 4.14.3.19 SetRotationalVelocity() `void Physics::SetRotationalVelocity (`

```
glm::vec3 rotVel )
```

Sets rotational velocity.

**Parameters**

| | |
|---|---|
| *rotVel* | New rotational velocity |

Definition at line 174 of file physics.cpp.

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

References rotationalVelocity.

### 4.14.3.20 SetVelocity() `void Physics::SetVelocity ( glm::vec3 vel )`

Sets the velocity of the object.

**Parameters**

| | |
|---|---|
| *vel* | |

Definition at line 132 of file physics.cpp.

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

References velocity.

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

### 4.14.3.21 Update() `void Physics::Update ( )`

Updates the physics of the object.

Definition at line 194 of file physics.cpp.

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

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

Referenced by Object::Update().

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

Calculates the gravitational pull each object has on each other.

Definition at line 221 of file physics.cpp.

```
{
  // Gets the needed components for the current object
Object* object = GetParent();
Transform* transform = object->GetComponent<Transform>();
Physics* physics = object->GetComponent<Physics>();
glm::vec3 position = transform->GetPosition();

  // For each object
for (unsigned i = 0; i < Object_Manager::GetSize(); ++i) {
    if ((int)i == object->GetId()) continue;
      // Gets needed components for the object being checked
    Object* other = Object_Manager::FindObject(i);
    Physics* otherPhysics = other->GetComponent<Physics>();
    Transform* otherTransform = other->GetComponent<Transform>();
    glm::vec3 otherPosition = otherTransform->GetPosition();
      // Finding the distance between the objects
    double distance = sqrt(pow(double(otherPosition.x - position.x), 2.0) +
        pow(double(otherPosition.y - position.y), 2.0) +
        pow(double(otherPosition.z - position.z), 2.0));
      // Calculating the force the objects apply on each other
    double magnitude = Engine::GetGravConst() * ((physics->mass * otherPhysics->mass)) / pow(distance,
  2.0);
      // Getting the direction (normalized)
    glm::vec3 direction = otherPosition - position;
    glm::vec3 normDirection = glm::normalize(direction);
      // Applying gravitational force to normalized direction
    glm::vec3 force = normDirection * float(magnitude);
      // Adding the gravitational force to the forces on object
    physics->AddForce(force);
}
}
```

References Object_Manager::FindObject(), Object::GetComponent(), Engine::GetGravConst(), Component::Get↩Parent(), Transform::GetPosition(), Object_Manager::GetSize(), and mass.

Referenced by Behavior::ClassSetup().

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

Gives physics data to the writer object.

**Parameters**

| | |
|---|---|
| *writer* | |

Definition at line 270 of file physics.cpp.

```
                                       {
    writer.Write_Vec3("acceleration", initialAcceleration);
    writer.Write_Vec3("velocity", initialVelocity);
    writer.Write_Value("mass", mass);
}
```

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.

```
                           {
      // Initializing random
    random = new Random;
    if (!random) {
        Trace::Message("Random failed to initialize.");
        return false;
    }

    return true;
}
```

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**

| | |
|---|---|
| *low* | Lower boundary in random gen |
| *high* | Upper boundary in random gen |

**Returns**

float

Definition at line 70 of file random.cpp.

```
                                                  {
      // Setup random gen
    std::mt19937 gen(random->rd());
    std::uniform_real_distribution<> dist(low, high);
      // Gen random float
    return dist(gen);
}
```

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**

| | |
|---|---|
| *low* | Lower boundary in random gen |
| *high* | Upper boundary in random gen |

**Returns**

vec3

Definition at line 54 of file random.cpp.

```
                                                    {
        // Setup random gen
      std::mt19937 gen(random->rd());
      std::uniform_real_distribution<> dist(low, high);
        // Gen random vec3
      glm::vec3 result_vec3 = { dist(gen), dist(gen), dist(gen) };
      return result_vec3;
}
```

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.

```
                         {
      if (!random) return;

      delete random;
      random = nullptr;
}
```

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 bool Initialize ()

  *Initialize shader with default values.*
- 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 192 of file shader.cpp.

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

```
199 { 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 171 of file shader.cpp.

```
{ 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 185 of file shader.cpp.

```
{ 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 164 of file shader.cpp.

```
{ 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 178 of file shader.cpp.

```
{ return shader->viewMatrixId; }
```

References shader, and viewMatrixId.

Referenced by Model_Data::Draw().

#### 4.16.2.7 Initialize() [1/2] `bool Shader::Initialize ( ) [static]`

Initialize shader with default values.

**Returns**

true

false

Definition at line 50 of file shader.cpp.

```
                          {
    shader = new Shader;
    if (!shader) {
        Trace::Message("Shader failed to initialize.\n");
        return false;
    }

    //LoadShader("src/shaders/vertex.glsl", "src/shaders/fragment.glsl");
    LoadShader(std::string(getenv("USERPROFILE")) + "/Documents/pEngine/shaders/vertex.glsl",
        std::string(getenv("USERPROFILE")) + "/Documents/pEngine/shaders/fragment.glsl");
    return true;
}
```

References LoadShader(), Trace::Message(), and shader.

Referenced by Graphics::Initialize().

#### 4.16.2.8 Initialize() [2/2] `bool Shader::Initialize ( File_Reader & settings ) [static]`

Initializes the shader object.

**Parameters**

| | |
|---|---|
| *settings* | File_Reader object that contains name of shaders to use |

**Returns**

true

false

Definition at line 31 of file shader.cpp.

```
                                            {
    shader = new Shader;
    if (!shader) {
        Trace::Message("Shader failed to initialize.\n");
        return false;
    }

    //LoadShader("src/shaders/vertex.glsl", "src/shaders/fragment.glsl");
    LoadShader(std::string(getenv("USERPROFILE")) + "/Documents/pEngine/shaders/" +
    settings.Read_String("vertexShader") + ".glsl",
        std::string(getenv("USERPROFILE")) + "/Documents/pEngine/shaders/" +
    settings.Read_String("fragShader") + ".glsl");
    return true;
}
```

References LoadShader(), Trace::Message(), File_Reader::Read_String(), and shader.

#### 4.16.2.9 LoadShader()

```
void Shader::LoadShader (
            std::string vertexPath,
            std::string fragmentPath ) [static]
```

Loads the vertex and fragment shader using given filepaths.

**Parameters**

| | |
|---|---|
| *vertexPath* | // Vertex shader filepath |
| *fragmentPath* | // Fragment shader filepath |

**Returns**

void

Definition at line 121 of file shader.cpp.

```
                                                                        {
      // Creating shaders
    GLuint vertShader = glCreateShader(GL_VERTEX_SHADER);
    GLuint fragShader = glCreateShader(GL_FRAGMENT_SHADER);

      // Reading shaders
    std::string vertShaderStr = ReadFile(vertexPath);
    std::string fragShaderStr = ReadFile(fragmentPath);
    const char *vertShaderSrc = vertShaderStr.c_str();
    const char *fragShaderSrc = fragShaderStr.c_str();

      // Compiling shaders
    glShaderSource(vertShader, 1, &vertShaderSrc, nullptr);
    glCompileShader(vertShader);

    glShaderSource(fragShader, 1, &fragShaderSrc, nullptr);
    glCompileShader(fragShader);

      // Attaching shaders to engine
    shader->program = glCreateProgram();
    glAttachShader(shader->program, vertShader);
    glAttachShader(shader->program, fragShader);

      // Cleanup
    glDeleteShader(vertShader);
    glDeleteShader(fragShader);

      // Setting up program
    glLinkProgram(shader->program);
    glUseProgram(shader->program);

    shader->matrixId = glGetUniformLocation(shader->program, "MVP");
    shader->viewMatrixId = glGetUniformLocation(shader->program, "V");
    shader->modelMatrixId = glGetUniformLocation(shader->program, "M");
    shader->lightId = glGetUniformLocation(shader->program, "LightPosition_worldspace");
    shader->lightPowerId = glGetUniformLocation(shader->program, "LightPower");
}
```

References lightId, lightPowerId, matrixId, modelMatrixId, program, ReadFile(), shader, and viewMatrixId.

Referenced by Initialize().

#### 4.16.2.10 ReadFile() `std::string Shader::ReadFile ( std::string *filepath* ) [static]`

Reads shader file into std::string.

**Parameters**

| | |
|---|---|
| *filepath* | Shader file |

**Returns**

std::string

Definition at line 92 of file shader.cpp.

```
                                             {
    std::string content;

      // Opening the shader file
    std::ifstream file(filepath.c_str(), std::ios::in);
    if (!file.is_open()) {
        Trace::Message("Failed to read file: " + filepath + "\n");
        return "";
    }

      // Saving shader file into std::string
    std::string line = "";
    while (!file.eof()) {
        getline(file, line);
        content.append(line + "\n");
    }

      // Closing file and returning std::string
    file.close();
    return content;
}
```

References Trace::Message().

Referenced by LoadShader().

#### 4.16.2.11 Shutdown() `void Shader::Shutdown ( ) [static]`

Shutdown shader.

**Returns**

void

Definition at line 77 of file shader.cpp.

```
                        {
    if (!shader) return;

    glDeleteProgram(shader->program);

    delete shader;
    shader = nullptr;
}
```

References program, and shader.

Referenced by Graphics::Shutdown().

#### 4.16.2.12 Update() `void Shader::Update ( ) [static]`

Tells program to use shader.

**Returns**

void

Definition at line 68 of file shader.cpp.

```
                      {
    glUseProgram(shader->program);
}
```

References program, and shader.

Referenced by Graphics::Render().

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

- shader.hpp
- shader.cpp

## 4.17 Texture Class Reference

`#include <texture.hpp>`

### Public Member Functions

- ∼Texture ()
  *Deletes texture data.*
- bool Load (std::string textureName_)
  *Loads in texture with given filename.*
- void Display ()
  *Setup texture for drawing.*
- std::string GetTextureName () const
  *Returns texture name.*
- GLuint GetTextureNum () const
  *Returns texture data id.*

### Static Private Member Functions

- static GLuint LoadDDS (FILE ∗fp)
  *Loads in the given dds file.*

### Private Attributes

- std::string textureName
  *Name of texture.*
- GLuint textureNum
  *Loaded texture data id.*
- GLuint textureId
  *Textures buffer id.*
- bool hasBeenSet
  *Whether there is a texture or not.*

### 4.17.1 Detailed Description

Texture class

Definition at line 23 of file texture.hpp.

### 4.17.2 Constructor & Destructor Documentation

#### 4.17.2.1 ∼Texture() `Texture::∼Texture ( )`

Deletes texture data.

Definition at line 24 of file texture.cpp.

```
24                {
25     glDeleteTextures(1, &textureNum);
26 }
```

References textureNum.

### 4.17.3 Member Function Documentation

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

Setup texture for drawing.

Definition at line 63 of file texture.cpp.

```
{
    if (!hasBeenSet) return;

    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, textureNum);
    glUniform1i(textureId, 0);
}
```

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 76 of file texture.cpp.

```
{ 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 83 of file texture.cpp.

```
{ return textureNum; }
```

References textureNum.

#### 4.17.3.4 Load() `bool Texture::Load ( std::string textureName_ )`

Loads in texture with given filename.

**Parameters**

| | |
|---|---|
| *texture↩ Name_* | Filename of texture |

**Returns**

true

false

Definition at line 35 of file texture.cpp.

```
                                            {
    FILE *fp;
    std::string filename = std::string(getenv("USERPROFILE")) + "/Documents/pEngine/textures/" +
    textureName_;
    textureName = filename ;

      // Opening the file
    fp = fopen(filename.c_str(), "rb");
    if (!fp) {
        fp = fopen(textureName_.c_str(), "rb");
        if (!fp) {
            return false;
        }
        else {
            textureName = textureName_;
        }
    }

    textureNum = Texture::LoadDDS(fp);
    textureId = glGetUniformLocation(Shader::GetProgram(), "myTextureSampler");
    hasBeenSet = true;

    return true;
}
```

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

Referenced by Texture_Manager::Get().

### 4.17.3.5 LoadDDS() `GLuint Texture::LoadDDS ( FILE * fp ) [static], [private]`

Loads in the given dds file.

**Parameters**

| | |
|---|---|
| *fp* | The file stream |

**Returns**

GLuint

Definition at line 94 of file texture.cpp.

```
                              {
    unsigned char header[124];

      // Making sure it is a dds
    char filecode[4];
    fread(filecode, 1, 4, fp);
     if (strncmp(filecode, "DDS ", 4) != 0) {
         fclose(fp);
         return 0;
     }

       // Getting the surface description
     fread(&header, 124, 1, fp);

     unsigned int height      = *(unsigned int*)&(header[8 ]);
     unsigned int width         = *(unsigned int*)&(header[12]);
     unsigned int linearSize     = *(unsigned int*)&(header[16]);
     unsigned int mipMapCount = *(unsigned int*)&(header[24]);
     unsigned int fourCC      = *(unsigned int*)&(header[80]);

     unsigned char * buffer;
     unsigned int bufsize;

     bufsize = mipMapCount > 1 ? linearSize * 2 : linearSize;
     buffer = (unsigned char*)malloc(bufsize * sizeof(unsigned char));
     fread(buffer, 1, bufsize, fp);

       // Close the file
     fclose(fp);

     unsigned int format;
     switch(fourCC) {
         case FOURCC_DXT1:
             format = GL_COMPRESSED_RGBA_S3TC_DXT1_EXT;
             break;
         case FOURCC_DXT3:
             format = GL_COMPRESSED_RGBA_S3TC_DXT3_EXT;
             break;
         case FOURCC_DXT5:
             format = GL_COMPRESSED_RGBA_S3TC_DXT5_EXT;
             break;
         default:
             free(buffer);
             return 0;
     }

     GLuint textureID;
     glGenTextures(1, &textureID);

     glBindTexture(GL_TEXTURE_2D, textureID);
     glPixelStorei(GL_UNPACK_ALIGNMENT,1);

     unsigned int blockSize = (format == GL_COMPRESSED_RGBA_S3TC_DXT1_EXT) ? 8 : 16;
     unsigned int offset = 0;

     for (unsigned int level = 0; level < mipMapCount && (width || height); ++level) {
         unsigned int size = ((width+3)/4)*((height+3)/4)*blockSize;
         glCompressedTexImage2D(GL_TEXTURE_2D, level, format, width, height,
             0, size, buffer + offset);

         offset += size;
         width  /= 2;
         height /= 2;

         if(width < 1) width = 1;
         if(height < 1) height = 1;

     }

     free(buffer);

     return textureID;
}
```

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**

| | |
|---|---|
| *reader* | File_Reader object that contains name of texture |

**Returns**

Texture∗

Definition at line 45 of file texture_manager.cpp.

```
                                                    {
    // Getting texture's filename
    std::string filename = reader.Read_String("textureToLoad");
    // Looking for texture in list of loaded textures
    for (Texture* texture : texture_manager->textures) {
        if (texture->GetTextureName().compare(filename) == 0) {
            return texture;
        }
    }

    // Creating new texture
    Texture* texture = new Texture;
    texture->Load(filename);
    texture_manager->textures.emplace_back(texture);

    return texture;
}
```

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**

| | |
|---|---|
| *textureName* | Name of texture |

**Returns**

Texture∗

Definition at line 71 of file texture_manager.cpp.

```
                                                      {
    // Looking for texture in list of loaded textures
    for (Texture* texture : texture_manager->textures) {
        if (texture->GetTextureName().compare(textureName) == 0) {
            return texture;
        }
    }

    // Creating new texture
    Texture* texture = new Texture;
    if (!texture->Load(textureName)) {
        delete texture;
        return nullptr;
    }
    texture_manager->textures.emplace_back(texture);

    return texture;
}
```

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.

```
                                    {
      // Initializing texture_manager
    texture_manager = new Texture_Manager;
    if (!texture_manager) {
        Trace::Message("Texture Manager was not initialized.\n");
        return false;
    }

      // Reserving space in the texture_manager
    texture_manager->textures.reserve(10);
    return true;
}
```

References Trace::Message(), texture_manager, and textures.

Referenced by Engine::Initialize().

#### 4.18.2.4 Shutdown() `void Texture_Manager::Shutdown ( ) [static]`

Deletes all texture object and then the manager.

**Returns**

void

Definition at line 95 of file texture_manager.cpp.

```
                              {
    if (!texture_manager) return;

    for (Texture* texture : texture_manager->textures) {
        if (!texture) continue;

         delete texture;
         texture = nullptr;
     }

     delete texture_manager;
     texture_manager = nullptr;
 }
```

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.

```
                  {
    trace = new Trace;

      // Opens output file
    trace->trace_stream.open(std::string(getenv("USERPROFILE")) + "/Documents/pEngine/trace.log",
    std::ofstream::out);
    if (!trace->trace_stream) std::cout « "Trace file wasn't opened successfully.\n";
}
```

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**

| | |
|---|---|
| *message* | Message to be printed |

**Returns**

void

Definition at line 40 of file trace.cpp.

```
                                  {
    if (!trace->trace_stream) return;

    trace->trace_stream « message;
    std::cout « message;
}
```

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(), Shader::ReadFile(), Engine::Restart(), and Object::Write().

#### 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.

```
                       {
    // Closing output file
  if (trace->trace_stream) trace->trace_stream.close();

  delete trace;
  trace = nullptr;
}
```

References trace, and trace_stream.

Referenced by main().

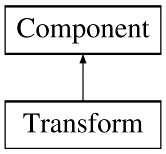
The documentation for this class was generated from the following files:

- 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.

```
                       : Component(CType::CTransform),
    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**

| | |
|---|---|
| *other* | |

Definition at line 27 of file transform.cpp.

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

#### 4.20.2.3 Transform() [3/3] `Transform::Transform ( File_Reader & reader )`

Creates Transform object using file.

**Parameters**

| | |
|---|---|
| *reader* | File to use for making Transform object |

Definition at line 36 of file transform.cpp.

```
                                               : Component(CType::CTransform),
    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)) {
    Read(reader);
}
```

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.

```
                                     {
    return new Transform(*this);
}
```

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.

```
                        {
    return CType::CTransform;
}
```

#### 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.

```
{ 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.

```
{ return position; }
```

References position.

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

#### 4.20.3.5 GetPositionRef() `glm::vec3 & Transform::GetPositionRef ( )`

Returns position reference of object.

**Returns**

glm::vec3&

Definition at line 69 of file transform.cpp.

```
69 { return position; }
```

References position.

Referenced by Behavior::ClassSetup(), and Editor::Display_Transform().

#### 4.20.3.6 GetRotation() `glm::vec3 Transform::GetRotation ( ) const`

Returns rotation of object.

**Returns**

float

Definition at line 118 of file transform.cpp.

```
118 { return rotation; }
```

References rotation.

Referenced by Model_Data::Draw(), and Physics::Update().

#### 4.20.3.7 GetRotationRef() `glm::vec3 & Transform::GetRotationRef ( )`

Returns rotation reference of object.

**Returns**

glm::vec3&

Definition at line 125 of file transform.cpp.

```
125 { return rotation; }
```

References rotation.

Referenced by Behavior::ClassSetup(), and Editor::Display_Transform().

#### 4.20.3.8 GetScale() `glm::vec3 Transform::GetScale ( ) const`

Returns scale of object.

**Returns**

glm::vec3

Definition at line 97 of file transform.cpp.

```
97 { return scale; }
```

References scale.

Referenced by Model_Data::Draw(), and File_Writer::Write_Object_Data().

#### 4.20.3.9 GetScaleRef() `glm::vec3 & Transform::GetScaleRef ( )`

Returns scale reference of object.

**Returns**

glm::vec3&

Definition at line 104 of file transform.cpp.

```
104 { return scale; }
```

References scale.

Referenced by Behavior::ClassSetup(), and Editor::Display_Transform().

#### 4.20.3.10 GetStartPosition() `glm::vec3 Transform::GetStartPosition ( ) const`

Returns the saved start position of the object.

**Returns**

glm::vec3

Definition at line 139 of file transform.cpp.

```
139 { return startPosition; }
```

References startPosition.

Referenced by File_Writer::Write_Object_Data().

#### 4.20.3.11 GetStartPositionRef() `glm::vec3 & Transform::GetStartPositionRef ( )`

Returns a reference to the start position of the object.

**Returns**

glm::vec3&

Definition at line 146 of file transform.cpp.

```
146 { return startPosition; }
```

References startPosition.

Referenced by Behavior::ClassSetup(), and Editor::Display_Transform().

#### 4.20.3.12 Read() `void Transform::Read ( File_Reader & reader )`

Reads data for Transform object from file.

**Parameters**

| | |
|---|---|
| *reader* | 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**

| | |
|---|---|
| *oldPos* | |

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**

| | |
|---|---|
| *pos* | |

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**

| | |
|---|---|
| *rot* | |

Definition at line 111 of file transform.cpp.

```
111 { rotation = rot; }
```

References rotation.

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

#### 4.20.3.16 SetScale() `void Transform::SetScale ( glm::vec3 sca )`

Sets scale of object.

**Parameters**

| | |
|---|---|
| *sca* | |

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**

| | |
|---|---|
| *start↩ Position_* | |

Definition at line 132 of file transform.cpp.

```
{ 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**

| | |
|---|---|
| *writer* | |

Definition at line 162 of file transform.cpp.

```
                                        {
    writer.Write_Vec3("rotation", rotation);
}
```

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**

| | |
|---|---|
| *vec* | |
| *num* | |

**Returns**

glm::vec3

Definition at line 73 of file vector3_func.cpp.

```
                                                        {
    glm::vec3 returnVec3;
    returnVec3.x = vec.x + num;
    returnVec3.y = vec.y + num;
    returnVec3.z = vec.z + num;

    return vec;
}
```

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**

| | |
|---|---|
| *vec1* | |
| *vec2* | |

**Returns**

glm::vec3

Definition at line 90 of file vector3_func.cpp.

```
                                                                      {
    glm::vec3 returnVec3;

    returnVec3.x = vec1.x + vec2.x;
    returnVec3.y = vec1.y + vec2.y;
    returnVec3.z = vec1.z + vec2.z;

    return vec1;
}
```

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**

| | |
|---|---|
| *vec1* | First input vec3 |
| *vec2* | Second input vec3 |

**Returns**

float

Definition at line 32 of file vector3_func.cpp.

```
{
    return glm::distance(vec1, vec2);
}
```

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**

| | |
|---|---|
| *vec1* | First input vec3 |
| *vec2* | Second input vec3 |

**Returns**

glm::vec3

Definition at line 43 of file vector3_func.cpp.

```
{
    return vec1 - vec2;
}
```

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**

| | |
|---|---|
| *vec* | Input vec3 |

**Returns**

float

Definition at line 62 of file vector3_func.cpp.

```
                                              {
    return glm::length(vec);
}
```

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.

```
                                                   {
    return glm::normalize(vec);
}
```

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.

```
                                   {
    return glm::vec3(0.f, 0.f, 0.f);
}
```

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.

```
                        {
    // Initializing systems
  Trace::Initialize();
  if (!Engine::Initialize()) return -1;
    // Engine update loop
  Graphics::Update();

    // Shutting down systems
  Engine::Shutdown();
  Trace::Shutdown();

  return 0;
}
```

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"
#include "trace.hpp"
```

### Macros

- #define FOURCC_DXT1 0x31545844

  *Equivalent to "DXT1" in ASCII.*
- #define FOURCC_DXT3 0x33545844

  *Equivalent to "DXT3" in ASCII.*
- #define FOURCC_DXT5 0x35545844

  *Equivalent to "DXT5" in ASCII.*

### 5.34.1 Detailed Description

**Author**

Kelson Wysocki ( kelson.wysocki@gmail.com)

**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 <cstdarg>
#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, 79
∼Texture
    Texture, 128

add_float
    Vector3_Func, 147
add_vec3
    Vector3_Func, 148
AddComponent
    Object, 90
AddForce
    Physics, 109
AddObject
    Object_Manager, 99
AddScript
    Behavior, 7
ApplyForce
    Physics, 109

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, 151
behavior.hpp, 151

Camera, 13
    Camera, 15
    GetFar, 16
    GetFov, 16
    GetFront, 16
    GetNear, 17
    GetOriginalMoveSpeed, 17
    GetOriginalSensitivity, 17
    GetOriginalSprintSpeed, 18
    GetPitch, 18
    GetPosition, 18
    GetUp, 19
    GetYaw, 19
    Initialize, 19, 20
    MouseUpdate, 21
    Shutdown, 22
    Update, 22
camera.cpp, 152
camera.hpp, 153
CheckIfCopy
    Behavior, 8
CheckName
    Object_Manager, 100
ClassSetup
    Behavior, 8
Clear
    Behavior, 10
    Object, 90
Clone
    Behavior, 10
    Model, 73
    Object, 90
    Physics, 110
    Transform, 140
Component, 23
    Component, 25
    CType, 24
    GetCType, 25
    GetParent, 25
    SetParent, 25
component.cpp, 153
component.hpp, 154
CType
    Component, 24

Display
    Texture, 129
Display_Camera_Settings
    Editor, 28
Display_Components
    Editor, 28
Display_Dockspace
    Editor, 30
Display_Menu_Bar
    Editor, 30
Display_Model
    Editor, 31
Display_Physics
    Editor, 32
Display_Scene
    Editor, 33
Display_Scripts
    Editor, 34

Display_Transform
 Editor, 36
Display_World_Settings
 Editor, 37
distance
 Vector3_Func, 148
Draw
 Model, 73
 Model_Data, 79

Editor, 26
 Display_Camera_Settings, 28
 Display_Components, 28
 Display_Dockspace, 30
 Display_Menu_Bar, 30
 Display_Model, 31
 Display_Physics, 32
 Display_Scene, 33
 Display_Scripts, 34
 Display_Transform, 36
 Display_World_Settings, 37
 GetTakeKeyboardInput, 37
 Initialize, 38
 Make_Display_String, 38
 Render, 40
 Reset, 40
 Shutdown, 40
 Update, 41
editor.cpp, 154
editor.hpp, 155
Engine, 42
 GetDeltaTime, 43
 GetDt, 43
 GetGravConst, 44
 GetLightPos, 44
 GetLightPower, 44
 GetPresetName, 45
 Initialize, 45
 Restart, 47
 SetPresetName, 48
 Shutdown, 48
 Update, 49
 Write, 49
engine.cpp, 156
engine.hpp, 156
ErrorCallback
 Graphics, 64
ErrorCheck
 Graphics, 65

File_Reader, 50
 Read_Behavior_Name, 51
 Read_Bool, 51
 Read_Double, 52
 Read_File, 52
 Read_Float, 53
 Read_Int, 53
 Read_Object_Name, 55
 Read_Object_Position, 55
 Read_Object_Scale, 56
 Read_Object_Template_Name, 56
 Read_String, 57
 Read_Vec3, 57
file_reader.cpp, 157
file_reader.hpp, 158
File_Writer, 58
 File_Writer, 59
 Write_Behavior_Name, 59
 Write_File, 60
 Write_Object_Data, 60
 Write_String, 61
 Write_Value, 61
 Write_Vec3, 62
file_writer.cpp, 158
file_writer.hpp, 159
FindObject
 Object_Manager, 100, 101

Get
 Model_Data_Manager, 85
 Texture_Manager, 132, 133
get_direction
 Vector3_Func, 149
GetAcceleration
 Physics, 110
GetAccelerationRef
 Physics, 110
GetComponent
 Object, 91
GetComponentConst
 Object, 91
GetComponentList
 Object, 92
GetCType
 Behavior, 10
 Component, 25
 Model, 74
 Physics, 111
 Transform, 140
GetDeltaTime
 Engine, 43
GetDt
 Engine, 43
GetFar
 Camera, 16
GetForces
 Physics, 111
GetForcesRef
 Physics, 111

GetFov
  Camera, 16
GetFront
  Camera, 16
GetGravConst
  Engine, 44
GetId
  Object, 92
GetLightId
  Shader, 122
GetLightPos
  Engine, 44
GetLightPower
  Engine, 44
GetLightPowerId
  Shader, 122
GetMass
  Physics, 112
GetMassRef
  Physics, 112
GetMatrixId
  Shader, 122
GetModelMatrixId
  Shader, 123
GetModelName
  Model, 74
  Model_Data, 81
GetName
  Object, 93
GetNameRef
  Object, 93
GetNear
  Camera, 17
GetOldPosition
  Transform, 141
GetOriginalMoveSpeed
  Camera, 17
GetOriginalSensitivity
  Camera, 17
GetOriginalSprintSpeed
  Camera, 18
GetParent
  Component, 25
GetPitch
  Camera, 18
GetPosition
  Camera, 18
  Transform, 141
GetPositionRef
  Transform, 141
GetPresetName
  Engine, 45
GetProgram
  Shader, 123
GetRotation
  Transform, 142
GetRotationalVelocity
  Physics, 112
GetRotationalVelocityRef
  Physics, 113
GetRotationRef
  Transform, 142
GetScale
  Transform, 142
GetScaleRef
  Transform, 143
GetScripts
  Behavior, 10
GetSize
  Object_Manager, 101
GetStartPosition
  Transform, 143
GetStartPositionRef
  Transform, 143
GetTakeKeyboardInput
  Editor, 37
GetTemplateName
  Object, 93
GetTexture
  Model, 74
GetTextureName
  Model, 75
  Texture, 129
GetTextureNum
  Texture, 129
GetUp
  Camera, 19
GetVelocity
  Physics, 113
GetVelocityRef
  Physics, 113
GetViewMatrixId
  Shader, 123
GetWindow
  Graphics, 65
GetWindowSize
  Graphics, 65
GetYaw
  Camera, 19
Graphics, 63
  ErrorCallback, 64
  ErrorCheck, 65
  GetWindow, 65
  GetWindowSize, 65
  Graphics, 64
  Initialize, 66, 67
  InitializeGL, 68
  Render, 68

Shutdown, 69
Update, 69
graphics.cpp, 159
graphics.hpp, 160

Initialize
Camera, 19, 20
Editor, 38
Engine, 45
Graphics, 66, 67
Model_Data_Manager, 86
Object_Manager, 102
Random, 118
Shader, 124
Texture_Manager, 134
Trace, 136
InitializeGL
Graphics, 68

length
Vector3_Func, 149
Load
Model, 75
Model_Data, 81, 82
Texture, 129
LoadDDS
Texture, 130
LoadShader
Shader, 125

main
main.cpp, 161
main.cpp, 161
main, 161
Make_Display_String
Editor, 38
Message
Trace, 136
Model, 70
Clone, 73
Draw, 73
GetCType, 74
GetModelName, 74
GetTexture, 74
GetTextureName, 75
Load, 75
Model, 72
Read, 76
SwitchModel, 76
SwitchTexture, 76
Write, 77
model.cpp, 162
model.hpp, 162
Model_Data, 77
∼Model_Data, 79
Draw, 79
GetModelName, 81
Load, 81, 82
Model_Data, 78, 79
Read, 82
model_data.cpp, 163
model_data.hpp, 164
Model_Data_Manager, 84
Get, 85
Initialize, 86
Shutdown, 86
model_data_manager.cpp, 164
model_data_manager.hpp, 165
MouseUpdate
Camera, 21

normalize
Vector3_Func, 150

Object, 87
AddComponent, 90
Clear, 90
Clone, 90
GetComponent, 91
GetComponentConst, 91
GetComponentList, 92
GetId, 92
GetName, 93
GetNameRef, 93
GetTemplateName, 93
Object, 89
Read, 94
RemoveComponent, 94
ReRead, 95
SetId, 96
SetName, 96
SetTemplateName, 97
Update, 97
Write, 97
object.cpp, 166
object.hpp, 166
Object_Manager, 98
AddObject, 99
CheckName, 100
FindObject, 100, 101
GetSize, 101
Initialize, 102
ReadList, 103
RemoveObject, 104
Shutdown, 104
Update, 105
Write, 105
object_manager.cpp, 167
object_manager.hpp, 168

Physics, 106
  AddForce, 109
  ApplyForce, 109
  Clone, 110
  GetAcceleration, 110
  GetAccelerationRef, 110
  GetCType, 111
  GetForces, 111
  GetForcesRef, 111
  GetMass, 112
  GetMassRef, 112
  GetRotationalVelocity, 112
  GetRotationalVelocityRef, 113
  GetVelocity, 113
  GetVelocityRef, 113
  Physics, 108, 109
  Read, 114
  SetAcceleration, 114
  SetForces, 115
  SetMass, 115
  SetRotationalVelocity, 115
  SetVelocity, 116
  Update, 116
  UpdateGravity, 117
  Write, 117
physics.cpp, 168
physics.hpp, 169

Random, 118
  Initialize, 118
  random_float, 119
  random_vec3, 119
  Shutdown, 120
random.cpp, 170
random.hpp, 170
random_float
  Random, 119
random_vec3
  Random, 119
Read
  Behavior, 11
  Model, 76
  Model_Data, 82
  Object, 94
  Physics, 114
  Transform, 144
Read_Behavior_Name
  File_Reader, 51
Read_Bool
  File_Reader, 51
Read_Double
  File_Reader, 52
Read_File
  File_Reader, 52
Read_Float
  File_Reader, 53
Read_Int
  File_Reader, 53
Read_Object_Name
  File_Reader, 55
Read_Object_Position
  File_Reader, 55
Read_Object_Scale
  File_Reader, 56
Read_Object_Template_Name
  File_Reader, 56
Read_String
  File_Reader, 57
Read_Vec3
  File_Reader, 57
ReadFile
  Shader, 126
ReadList
  Object_Manager, 103
RemoveComponent
  Object, 94
RemoveObject
  Object_Manager, 104
Render
  Editor, 40
  Graphics, 68
ReRead
  Object, 95
Reset
  Editor, 40
Restart
  Engine, 47

SetAcceleration
  Physics, 114
SetForces
  Physics, 115
SetId
  Object, 96
SetMass
  Physics, 115
SetName
  Object, 96
SetOldPosition
  Transform, 144
SetParent
  Component, 25
SetPosition
  Transform, 144
SetPresetName
  Engine, 48
SetRotation
  Transform, 145

SetRotationalVelocity
    Physics, 115
SetScale
    Transform, 145
SetStartPosition
    Transform, 146
SetTemplateName
    Object, 97
SetupClassesForLua
    Behavior, 11
SetVelocity
    Physics, 116
Shader, 121
    GetLightId, 122
    GetLightPowerId, 122
    GetMatrixId, 122
    GetModelMatrixId, 123
    GetProgram, 123
    GetViewMatrixId, 123
    Initialize, 124
    LoadShader, 125
    ReadFile, 126
    Shutdown, 126
    Update, 127
shader.cpp, 171
shader.hpp, 172
Shutdown
    Camera, 22
    Editor, 40
    Engine, 48
    Graphics, 69
    Model_Data_Manager, 86
    Object_Manager, 104
    Random, 120
    Shader, 126
    Texture_Manager, 134
    Trace, 137
SwitchModel
    Model, 76
SwitchScript
    Behavior, 12
SwitchTexture
    Model, 76

Texture, 127
    ∼Texture, 128
    Display, 129
    GetTextureName, 129
    GetTextureNum, 129
    Load, 129
    LoadDDS, 130
texture.cpp, 172
texture.hpp, 173
Texture_Manager, 132
    Get, 132, 133
    Initialize, 134
    Shutdown, 134
texture_manager.cpp, 174
texture_manager.hpp, 174
Trace, 135
    Initialize, 136
    Message, 136
    Shutdown, 137
trace.cpp, 175
trace.hpp, 176
Transform, 137
    Clone, 140
    GetCType, 140
    GetOldPosition, 141
    GetPosition, 141
    GetPositionRef, 141
    GetRotation, 142
    GetRotationRef, 142
    GetScale, 142
    GetScaleRef, 143
    GetStartPosition, 143
    GetStartPositionRef, 143
    Read, 144
    SetOldPosition, 144
    SetPosition, 144
    SetRotation, 145
    SetScale, 145
    SetStartPosition, 146
    Transform, 139, 140
    Write, 146
transform.cpp, 176
transform.hpp, 177

Update
    Behavior, 12
    Camera, 22
    Editor, 41
    Engine, 49
    Graphics, 69
    Object, 97
    Object_Manager, 105
    Physics, 116
    Shader, 127
UpdateGravity
    Physics, 117

Vector3_Func, 146
    add_float, 147
    add_vec3, 148
    distance, 148
    get_direction, 149
    length, 149
    normalize, 150
    zero_vec3, 150

vector3_func.cpp, 177
vector3_func.hpp, 178

Write
  Behavior, 13
  Engine, 49
  Model, 77
  Object, 97
  Object_Manager, 105
  Physics, 117
  Transform, 146
Write_Behavior_Name
  File_Writer, 59
Write_File
  File_Writer, 60
Write_Object_Data
  File_Writer, 60
Write_String
  File_Writer, 61
Write_Value
  File_Writer, 61
Write_Vec3
  File_Writer, 62

zero_vec3
  Vector3_Func, 150