TinyShaders

0.3

Generated by Doxygen 1.8.7

Wed Nov 4 2015 17:18:53

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 TinyShaders Class Reference

`#include <TinyShaders.h>`

Data Structures

- struct TShader
- struct TShaderProgram

Public Member Functions

- TinyShaders ()
- ~TinyShaders ()

Static Public Member Functions

- static void Shutdown ()
- static TShaderProgram * GetShaderProgramByName (const char *ProgramName)
- static TShaderProgram * GetShaderProgramByIndex (GLuint ProgramIndex)
- static TShader * GetShaderByName (const GLchar *ShaderName)
- static TShader * GetShaderByIndex (GLuint ShaderIndex)
- static void LoadShader (const GLchar *Name, const GLchar *ShaderFile, GLuint ShaderType)
- static void LoadShaderProgramsFromConfigFile (const GLchar *ConfigFile)
- static void LoadShadersFromConfigFile (const GLchar *ConfigFile)
- static void SaveShaderProgramsToConfigFile (const GLchar *FileName)
- static void BuildProgramFromShaders (const GLchar *ShaderName, std::vector< const GLchar * > Inputs, std::vector< const GLchar * > Outputs, const GLchar *VertexShaderName, const GLchar *Fragment↩ ShaderName, const GLchar *GeometryShaderName, const GLchar *TessContShaderName, const GLchar *TessEvalShaderName)
- static GLboolean ShaderProgramExists (const GLchar *ShaderName)
- static GLboolean ShaderExists (const GLchar *ShaderName)
- static GLvoid LoadShaderFromBuffer (const char *Name, const GLchar *Buffer, GLuint ShaderType)

Private Member Functions

- GLchar * FileToBuffer (const GLchar *Path)
- GLuint StringToShaderType (const char *TypeString)

- const GLchar * ShaderTypeToString (GLuint ShaderType)
- GLvoid AddProgram (TShaderProgram *NewProgram)
- GLvoid AddShader (TShader *NewShader)

Static Private Member Functions

- static TinyShaders * GetInstance ()
- static GLvoid PrintErrorMessage (GLuint ErrorNumber, const GLchar *String=nullptr)

Private Attributes

- std::vector< TShaderProgram * > ShaderPrograms
- std::vector< TShader * > Shaders

Static Private Attributes

- static GLboolean IsInitialized = GL_FALSE
- static TinyShaders * Instance = nullptr

3.1.1 Detailed Description

3.1.2 Constructor & Destructor Documentation

3.1.2.1 TinyShaders::TinyShaders ( ) [inline]

```
00045 {}
```

3.1.2.2 TinyShaders::~TinyShaders ( ) [inline]

```
00046 {}
```

3.1.3 Member Function Documentation

3.1.3.1 GLvoid TinyShaders::AddProgram (TShaderProgram * *NewProgram*) [inline], [private]

```
00902         {
00903             if(NewProgram != nullptr)
00904             {
00905                 if(NewProgram->Compiled)
00906                 {
00907                     GetInstance()->ShaderPrograms.push_back(NewProgram);
00908                 }
00909             }
00910         }
```

3.1.3.2 GLvoid TinyShaders::AddShader (TShader * *NewShader*) [inline], [private]

```
00913         {
00914             if(NewShader != nullptr)
00915             {
00916                 if(NewShader->IsCompiled)
00917                 {
00918                     GetInstance()->Shaders.push_back(NewShader);
00919                 }
00920             }
00921         }
```

3.1.3.3 static void TinyShaders::BuildProgramFromShaders (const GLchar * *ShaderName*, std::vector< const GLchar * > *Inputs*, std::vector< const GLchar * > *Outputs*, const GLchar * *VertexShaderName*, const GLchar * *FragmentShaderName*, const GLchar * *GeometryShaderName*, const GLchar * *TessContShaderName*, const GLchar * *TessEvalShaderName*) [inline], [static]

```
00381         {
00382             if (TinyShaders::IsInitialized)
00383             {
00384                 std::vector<TShader*> Shaders;
00385                 Shaders.push_back(GetShaderByName(VertexShaderName));
00386                 Shaders.push_back(GetShaderByName(FragmentShaderName));
00387                 Shaders.push_back(GetShaderByName(GeometryShaderName));
00388                 Shaders.push_back(GetShaderByName(TessContShaderName));
00389                 Shaders.push_back(GetShaderByName(TessEvalShaderName));
00390 
00391                 TShaderProgram* NewShaderProgram = new TShaderProgram(ShaderName, Inputs, Outputs, Shaders)
      ;
00392                 delete NewShaderProgram;
00393             }
00394             PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED);
00395         }
```

3.1.3.4 GLchar* TinyShaders::FileToBuffer (const GLchar * *Path*) [inline], [private]

```
00799         {
00800             FILE* File = fopen(Path, "rt");
00801 
00802             if (File == nullptr)
00803             {
00804                 PrintErrorMessage(
      TSHADERS_ERROR_INVALIDFILEPATH, Path);
00805                 return nullptr;
00806             }
00807 
00808             //get total byte in given file
00809             fseek(File, 0, SEEK_END);
00810             GLuint FileLength = ftell(File);
00811             fseek(File, 0, SEEK_SET);
00812 
00813             //allocate a file buffer and read the contents of the file
00814             char* Buffer = new char[FileLength + 1];
00815             memset(Buffer, 0, FileLength + 1);
00816             fread(Buffer, sizeof(char), FileLength, File);
00817 
00818             fclose(File);
00819             return Buffer;
00820         }
```

3.1.3.5 static TinyShaders* TinyShaders::GetInstance ( ) [inline], [static], [private]

```
00675         {
00676             if (TinyShaders::IsInitialized)
00677             {
00678                 return TinyShaders::Instance;
00679             }
00680 
00681             TinyShaders::IsInitialized = GL_TRUE;
00682             TinyShaders::Instance = new TinyShaders();
00683             return TinyShaders::Instance;
00684         }
```

3.1.3.6 static TShader* TinyShaders::GetShaderByIndex (GLuint *ShaderIndex*) [inline], [static]

```
00148         {
00149             if (TinyShaders::IsInitialized)
00150             {
00151                 if (ShaderIndex <= GetInstance()->Shaders.size() - 1)
00152                 {
00153                     return GetInstance()->Shaders[ShaderIndex];
00154                 }
00155                 PrintErrorMessage(
      TSHADERS_ERROR_INVALIDSHADERINDEX);
00156                 return nullptr;
00157             }
```

```
00158             PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED);
00159             return nullptr;
00160         }
```

3.1.3.7 static TShader* TinyShaders::GetShaderByName (const GLchar * *ShaderName*) [inline], [static]

```
00122         {
00123             if (TinyShaders::IsInitialized)
00124             {
00125                 if (ShaderName != nullptr)
00126                 {
00127                     for (GLuint Iterator = 0; Iterator < GetInstance()->
      Shaders.size(); Iterator++)
00128                     {
00129                         if (!strcmp(GetInstance()->Shaders[Iterator]->Name, ShaderName))
00130                         {
00131                             return GetInstance()->Shaders[Iterator];
00132                         }
00133                     }
00134                     PrintErrorMessage(
      TSHADERS_ERROR_SHADERNOTFOUND);
00135                     return nullptr;
00136                 }
00137                 PrintErrorMessage(
      TSHADERS_ERROR_INVALIDSHADERNAME);
00138                 return nullptr;
00139             }
00140             PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED);
00141             return nullptr;
00142         }
```

3.1.3.8 static TShaderProgram* TinyShaders::GetShaderProgramByIndex (GLuint *ProgramIndex*) [inline], [static]

```
00104         {
00105             if (TinyShaders::IsInitialized)
00106             {
00107                 if (ProgramIndex >= GetInstance()->ShaderPrograms.size() - 1)
00108                 {
00109                     return GetInstance()->ShaderPrograms[ProgramIndex];
00110                 }
00111                 PrintErrorMessage(
      TSHADERS_ERROR_INVALIDSHADERPROGRAMINDEX);
00112                 return nullptr;
00113             }
00114             PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED);
00115             return nullptr;
00116         }
```

3.1.3.9 static TShaderProgram* TinyShaders::GetShaderProgramByName (const char * *ProgramName*) [inline], [static]

```
00079         {
00080             if (TinyShaders::IsInitialized)
00081             {
00082                 if (ProgramName != nullptr)
00083                 {
00084                     for (GLuint Iterator = 0; Iterator < GetInstance()->
      ShaderPrograms.size(); Iterator++)
00085                     {
00086                         if (!strcmp(GetInstance()->ShaderPrograms[Iterator]->Name,
       ProgramName))
00087                         {
00088                             return GetInstance()->ShaderPrograms[Iterator];
00089                         }
00090                     }
00091                     return nullptr;
00092                 }
00093                 PrintErrorMessage(
      TSHADERS_ERROR_SHADERPROGRAMNOTFOUND);
00094                 return nullptr;
00095             }
00096             PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED);
00097             return nullptr;
00098         }
```

3.1.3.10 static void TinyShaders::LoadShader (const GLchar * *Name*, const GLchar * *ShaderFile*, GLuint *ShaderType*) [inline], [static]

```
00166         {
00167             if (TinyShaders::IsInitialized)
00168 
00169             {
00170                 if (Name != nullptr)
00171 
00172                 {
00173                     if (ShaderType <= 5)
00174                     {
00175                         GetInstance()->Shaders.push_back(new TShader(Name, ShaderType,
      ShaderFile));
00176                     }
00177                     PrintErrorMessage(
      TSHADERS_ERROR_INVALIDSHADERTYPE, GetInstance()->
      ShaderTypeToString(ShaderType));
00178                 }
00179                 PrintErrorMessage(TSHADERS_ERROR_INVALIDSTRING
      );
00180             }
00181             PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED);
00182         }
```

3.1.3.11 static GLvoid TinyShaders::LoadShaderFromBuffer (const char * *Name*, const GLchar * *Buffer*, GLuint *ShaderType*) [inline], [static]

```
00449         {
00450             if(TinyShaders::IsInitialized)
00451             {
00452                 if(Buffer != nullptr)
00453                 {
00454                     if(Name != nullptr)
00455                     {
00456                         if(!ShaderExists(Name))
00457                         {
00458                             TShader* NewShader = new TShader(Name, Buffer, ShaderType);
00459                             delete NewShader;
00460                         }
00461                         PrintErrorMessage(
      TSHADERS_ERROR_SHADERNOTFOUND);
00462                     }
00463                     PrintErrorMessage(
      TSHADERS_ERROR_INVALIDSHADERNAME);
00464                 }
00465                 PrintErrorMessage(TSHADERS_ERROR_INVALIDSTRING
      );
00466             }
00467             PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED);
00468         }
```

3.1.3.12 static void TinyShaders::LoadShaderProgramsFromConfigFile (const GLchar * *ConfigFile*) [inline], [static]

```
00188         {
00189             if (!TinyShaders::IsInitialized)
00190             {
00191                 FILE* pConfigFile = fopen(ConfigFile, "r");
00192                 GLuint NumInputs = 0;
00193                 GLuint NumOutputs = 0;
00194                 GLuint NumPrograms = 0;
00195                 GLuint NumShaders = 0;
00196                 GLuint Iterator = 0;
00197 
00198                 std::vector<const GLchar*> Inputs, Outputs, Paths, Names;
00199                 std::vector<TShader*> Shaders;
00200                 if (pConfigFile)
00201                 {
00202                     //get the total number of shader programs
00203                     fscanf(pConfigFile, "%i\n", &NumPrograms);
00204 
00205                     for (GLuint ProgramIter = 0;
00206                         ProgramIter < NumPrograms;
00207                         ProgramIter++, Paths.clear(), Inputs.clear(), Outputs.clear(), Names.clear(),
      Shaders.clear())
00208                     {
00209                         //get the name of the shader program
```

```
00210                         GLchar* ProgramName = new GLchar[255];
00211                         fscanf(pConfigFile, "%s\n", ProgramName);
00212 
00213                         //this is an anti-trolling measure. If a shader with the same name already exists
      the don't bother making a new one.
00214                         if (!GetInstance()->ShaderProgramExists(ProgramName))
00215                         {
00216                             //get the number of shader inputs
00217                             fscanf(pConfigFile, "%i\n", &NumInputs);
00218 
00219                             //get all inputs
00220                             for (Iterator = 0; Iterator < NumInputs; Iterator++)
00221                             {
00222                                 GLchar* Input = new GLchar[255];
00223                                 fscanf(pConfigFile, "%s\n", Input);
00224                                 Inputs.push_back(Input);
00225                             }
00226 
00227                             //get the number of shader outputs
00228                             fscanf(pConfigFile, "%i\n", &NumOutputs);
00229 
00230                             //get all outputs
00231                             for (Iterator = 0; Iterator < NumOutputs; Iterator++)
00232                             {
00233                                 GLchar* Output = new GLchar[255];
00234                                 fscanf(pConfigFile, "%s\n", Output);
00235                                 Outputs.push_back(Output);
00236                             }
00237 
00238                             //get number of shaders
00239                             fscanf(pConfigFile, "%i\n", &NumShaders);
00240 
00241                             for(GLuint ShaderIter = 0; ShaderIter < NumShaders; ShaderIter++)
00242                             {
00243                                 GLchar* ShaderName = new GLchar[255];
00244                                 GLchar* ShaderPath = new GLchar[255];
00245                                 GLchar* ShaderType = new GLchar[255];
00246 
00247                                 //get shader name
00248                                 fscanf(pConfigFile, "%s\n", ShaderName);
00249 
00250                                 //if the shader hasn't been loaded already then make a new one
00251                                 if(!ShaderExists(ShaderName))
00252                                 {
00253                                     //get type
00254                                     fscanf(pConfigFile, "%s\n", ShaderType);
00255                                     //get file path
00256                                     fscanf(pConfigFile, "%s\n", ShaderPath);
00257 
00258                                     Shaders.push_back(new TShader(ShaderName,
      GetInstance()->StringToShaderType((const char*)ShaderType), ShaderPath));
00259                                 }
00260 
00261                                 else
00262                                 {
00263                                     //if shader already exists then add an existing one from storage, it
      should already be compiled
00264                                     Shaders.push_back(GetShaderByName(ShaderName));
00265                                 }
00266                             }
00267 
00268                             GetInstance()->ShaderPrograms.push_back(new
      TShaderProgram(ProgramName, Inputs, Outputs, Shaders));
00269                         }
00270                         fclose(pConfigFile);
00271                     }
00272                 }
00273                 else
00274                 {
00275                     PrintErrorMessage(
      TSHADERS_ERROR_INVALIDFILEPATH);
00276                 }
00277             }
00278             else
00279             {
00280                 PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED
      );
00281             }
00282         }
```

3.1.3.13 static void TinyShaders::LoadShadersFromConfigFile (const GLchar * *ConfigFile*) [inline], [static]

```
00285         {
00286             if(TinyShaders::IsInitialized)
```

```
00287             {
00288                 FILE* pConfigFile = fopen(ConfigFile, "r+");
00289                 GLuint NumShaders = 0;
00290 
00291                 if(pConfigFile)
00292                 {
00293                     //get the number of shaders to load
00294                     fscanf(pConfigFile, "%i\n\n", &NumShaders);
00295                     GLchar* ShaderName;
00296                     GLchar* ShaderType;
00297                     GLchar* ShaderPath;
00298 
00299                     GLchar empty[255];
00300 
00301                     for(GLuint ShaderIter = 0; ShaderIter < NumShaders;
00302                             ShaderIter++, fscanf(pConfigFile, "\n\n"))
00303                     {
00304                         ShaderName = empty;
00305                         fscanf(pConfigFile, "%s\n", ShaderName);
00306 
00307                         if(!GetInstance()->ShaderExists(ShaderName))
00308                         {
00309                             ShaderType = empty;
00310                             fscanf(pConfigFile, "%s\n", ShaderType);
00311 
00312                             ShaderPath = empty;
00313                             fscanf(pConfigFile, "%s\n", ShaderPath);
00314 
00315                             TShader* NewShader = new TShader(ShaderName,
    GetInstance()->StringToShaderType(ShaderType), ShaderPath);
00316                             delete NewShader;
00317                         }
00318                     }
00319                 }
00320             }
00321         }
```

3.1.3.14 static GLvoid TinyShaders::PrintErrorMessage (GLuint *ErrorNumber*, const GLchar * *String* = nullptr) [inline], [static], [private]

```
00690         {
00691             switch (ErrorNumber)
00692             {
00693             case TSHADERS_ERROR_NOTINITIALIZED:
00694             {
00695                 printf("Error: TinyShaders must first be initialized \n");
00696                 break;
00697             }
00698 
00699             case TSHADERS_ERROR_INVALIDSTRING:
00700             {
00701                 printf("Error: given string is invalid \n");
00702                 break;
00703             }
00704 
00705             case TSHADERS_ERROR_INVALIDSHADERPROGRAMNAME:
00706             {
00707                 printf("Error: given shader name is invalid \n");
00708                 break;
00709             }
00710 
00711             case TSHADERS_ERROR_INVALIDSHADERPROGRAMINDEX:
00712             {
00713                 printf("Error: given shader index is invalid \n");
00714                 break;
00715             }
00716 
00717             case TSHADERS_ERROR_INVALIDSHADERNAME:
00718             {
00719                 printf("Error: given shader component name is invalid \n");
00720                 break;
00721             }
00722 
00723             case TSHADERS_ERROR_INVALIDSHADERINDEX:
00724             {
00725                 printf("Error: given shader component index is invalid \n");
00726                 break;
00727             }
00728 
00729             case TSHADERS_ERROR_INVALIDFILEPATH:
00730             {
00731                 printf("Error: given file path is invalid %s \n", String);
00732                 break;
```

```
00733             }
00734
00735             case TSHADERS_ERROR_SHADERPROGRAMNOTFOUND:
00736             {
00737                 printf("Error: shader with given name %s was not found \n", String);
00738                 break;
00739             }
00740
00741             case TSHADERS_ERROR_SHADERNOTFOUND:
00742             {
00743                 printf("Error: shader component with given name %s was not found \n", String);
00744                 break;
00745             }
00746
00747             case TSHADERS_ERROR_INVALIDSHADERTYPE:
00748             {
00749                 printf("Error: invalid shader type given \n");
00750                 break;
00751             }
00752
00753             case TSHADERS_ERROR_FAILEDSHADERLOAD:
00754             {
00755                 printf("Error: failed to compile %s shader component \n", String);
00756                 break;
00757             }
00758
00759             case TSHADERS_ERROR_FAILEDSHADERPROGRAMLINK:
00760             {
00761                 if (String != nullptr)
00762                 {
00763                     printf("Error: failed to link program %s \n", String);
00764                 }
00765                 break;
00766             }
00767
00768             case TSHADERS_ERROR_SHADEREXISTS:
00769             {
00770                 printf("Error: shader component with this name %s already exists \n", String);
00771                 break;
00772             }
00773
00774             case TSHADERS_ERROR_SHADERPROGRAMEXISTS:
00775             {
00776                 if (String != nullptr)
00777                 {
00778                     printf("Error: shader with this name %s already exists \n", String);
00779                     break;
00780                 }
00781             }
00782
00783             case TSHADERS_ERROR_INVALIDSOURCEFILE:
00784             {
00785                 printf("Given Source file is invalid");
00786                 break;
00787             }
00788             default:
00789             {
00790                 break;
00791             }
00792             }
00793         }
```

3.1.3.15 static void TinyShaders::SaveShaderProgramsToConfigFile (const GLchar * *FileName*) [inline],[static]

```
00324         {
00325             //write total amount of shaders
00326             FILE* pConfigFile = fopen(FileName, "w+");
00327
00328             fprintf(pConfigFile, "%i\n\n", (GLint)GetInstance()->
      ShaderPrograms.size());
00329
00330             for(GLuint ProgramIter = 0; ProgramIter < GetInstance()->
      ShaderPrograms.size(); ProgramIter++)
00331             {
00332                 //write program name
00333                 fprintf(pConfigFile, "%s\n", GetInstance()->
      ShaderPrograms[ProgramIter]->Name);
00334
00335                 //write number of inputs
00336                 fprintf(pConfigFile, "%i\n", (GLint)GetInstance()->
      ShaderPrograms[ProgramIter]->Inputs.size());
00337
00338                 //write inputs
00339                 for(GLuint InputIter = 0; InputIter < GetInstance()->
```

```
      ShaderPrograms[ProgramIter]->Inputs.size(); InputIter++)
00340                 {
00341                     fprintf(pConfigFile, "%s\n", GetInstance()->
      ShaderPrograms[ProgramIter]->Inputs[InputIter]);
00342                 }
00343
00344                 fprintf(pConfigFile, "%i\n", (GLint)GetInstance()->
      ShaderPrograms[ProgramIter]->Outputs.size());
00345
00346                 //write outputs
00347                 for(GLuint OutputIter = 0; OutputIter < GetInstance()->
      ShaderPrograms[ProgramIter]->Outputs.size(); OutputIter++)
00348                 {
00349                     fprintf(pConfigFile, "%s\n", GetInstance()->
      ShaderPrograms[ProgramIter]->Outputs[OutputIter]);
00350                 }
00351
00352                 //write number of shaders
00353                 fprintf(pConfigFile, "%i\n", (GLint)GetInstance()->
      ShaderPrograms[ProgramIter]->Shaders.size());
00354
00355                 for(GLuint ShaderIter = 0; ShaderIter < GetInstance()->
      ShaderPrograms[ProgramIter]->Shaders.size(); ShaderIter++)
00356                 {
00357                     //write shader name
00358                     fprintf(pConfigFile, "%s\n", GetInstance()->
      ShaderPrograms[ProgramIter]->Shaders[ShaderIter]->Name);
00359
00360                     //write shader type
00361                     fprintf(pConfigFile, "%s\n", GetInstance()->
      ShaderTypeToString(GetInstance()->ShaderPrograms[ProgramIter]->
      Shaders[ShaderIter]->Type));
00362
00363                     //write shader file path
00364                     fprintf(pConfigFile, "%s\n", GetInstance()->
      ShaderPrograms[ProgramIter]->Shaders[ShaderIter]->FilePath);
00365                 }
00366             }
00367             fclose(pConfigFile);
00368         }
```

3.1.3.16 static GLboolean TinyShaders::ShaderExists (const GLchar * *ShaderName*) [inline], [static]

```
00425         {
00426             //make sure the name isn't empty
00427             if (ShaderName != nullptr)
00428             {
00429                 //make sure the shader manager has shaders stored already
00430                 if (!GetInstance()->Shaders.empty())
00431                 {
00432                     //for each shader in the shader manager
00433                     for (GLuint Iterator = 0; Iterator < GetInstance()->
      Shaders.size(); Iterator++)
00434                     {
00435                         //if a shader of the same name is the same as an existing shader
00436                         if (!strcmp(ShaderName, GetInstance()->
      Shaders[Iterator]->Name))
00437                         {
00438                             return GL_TRUE;
00439                         }
00440                     }
00441                     return GL_FALSE;
00442                 }
00443                 return GL_FALSE;
00444             }
00445             return GL_FALSE;
00446         }
```

3.1.3.17 static GLboolean TinyShaders::ShaderProgramExists (const GLchar * *ShaderName*) [inline], [static]

```
00401         {
00402             if (ShaderName != nullptr)
00403             {
00404                 if (!GetInstance()->ShaderPrograms.empty())
00405                 {
00406                     for (GLuint Iterator = 0; Iterator < GetInstance()->
      ShaderPrograms.size(); Iterator++)
00407                     {
00408                         if (GetInstance()->ShaderPrograms[Iterator] != nullptr &&
00409                             !strcmp(ShaderName, GetInstance()->
```

```
      ShaderPrograms[Iterator]->Name))
00410                         {
00411                             return GL_TRUE;
00412                         }
00413                     }
00414                     return GL_FALSE;
00415                 }
00416                 return GL_FALSE;
00417             }
00418             return GL_FALSE;
00419         }
```

3.1.3.18 const GLchar* TinyShaders::ShaderTypeToString (GLuint *ShaderType*) [inline], [private]

```
00864         {
00865             switch (ShaderType)
00866             {
00867                 case GL_VERTEX_SHADER:
00868                 {
00869                     return "Vertex";
00870                 }
00871
00872                 case GL_FRAGMENT_SHADER:
00873                 {
00874                     return "Fragment";
00875                 }
00876
00877                 case GL_GEOMETRY_SHADER:
00878                 {
00879                     return "Geometry";
00880                 }
00881
00882                 case GL_TESS_CONTROL_SHADER:
00883                 {
00884                     return "Tessellation Control";
00885                 }
00886
00887                 case GL_TESS_EVALUATION_SHADER:
00888                 {
00889                     return "Tessellation Evaluation";
00890                 }
00891
00892                 default:
00893                 {
00894                     return NULL;
00895                 }
00896             }
00897
00898             return nullptr;
00899         }
```

3.1.3.19 static void TinyShaders::Shutdown ( ) [inline], [static]

```
00053         {
00054             if (TinyShaders::IsInitialized)
00055             {
00056                 for (GLuint Iterator = 0; Iterator < GetInstance()->
      Shaders.size(); Iterator++)
00057                 {
00058                     GetInstance()->Shaders[Iterator]->Shutdown();
00059                      delete GetInstance()->Shaders[Iterator];
00060                 }
00061
00062                 for (GLuint Iterator = 0; Iterator < GetInstance()->
      ShaderPrograms.size(); Iterator++)
00063                 {
00064                     GetInstance()->ShaderPrograms[Iterator]->Shutdown();
00065                     delete GetInstance()->ShaderPrograms[Iterator];
00066                 }
00067
00068                 GetInstance()->ShaderPrograms.clear();
00069                 GetInstance()->Shaders.clear();
00070
00071                 delete Instance;
00072             }
00073         }
```

3.1.3.20 GLuint TinyShaders::StringToShaderType (const char * *TypeString*) [inline], [private]

```
00826         {
00827             if(TypeString != nullptr)
00828             {
00829                 if (!strcmp(TypeString, "Vertex"))
00830                 {
00831                     return GL_VERTEX_SHADER;
00832                 }
00833 
00834                 if (!strcmp(TypeString, "Fragment"))
00835                 {
00836                     return GL_FRAGMENT_SHADER;
00837                 }
00838 
00839                 if (!strcmp(TypeString, "Geometry"))
00840                 {
00841                     return GL_GEOMETRY_SHADER;
00842                 }
00843 
00844                 if (!strcmp(TypeString, "Tessellation Control"))
00845                 {
00846                     return GL_TESS_CONTROL_SHADER;
00847                 }
00848 
00849                 if (!strcmp(TypeString, "Tessellation Evaluation"))
00850                 {
00851                     return GL_TESS_EVALUATION_SHADER;
00852                 }
00853 
00854                 return GL_FALSE;
00855             }
00856             PrintErrorMessage(TSHADERS_ERROR_INVALIDSTRING);
00857             return GL_FALSE;
00858         }
```

3.1.4 Field Documentation

3.1.4.1 TinyShaders * TinyShaders::Instance = nullptr [static], [private]

a static instance of the TinyShaders API

3.1.4.2 GLboolean TinyShaders::IsInitialized = GL_FALSE [static], [private]

Whether TinyShaders has ban initialized

3.1.4.3 std::vector<TShaderProgram*> TinyShaders::ShaderPrograms [private]

all loaded shader programs

3.1.4.4 std::vector<TShader*> TinyShaders::Shaders [private]

all loaded shaders

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

- include/TinyShaders.h

3.2 TinyShaders::TShader Struct Reference

Public Member Functions

- TShader (const GLchar *ShaderName, GLuint ShaderType, const GLchar *ShaderFilePath)
- TShader (const GLchar *ShaderName, const GLchar *Buffer, GLuint ShaderType)

- TShader ()
- ~TShader ()
- GLvoid Compile (const GLchar *Source)
- GLvoid Shutdown ()

Data Fields

- const GLchar * Name
- const GLchar * FilePath
- GLuint Handle
- GLuint Type
- GLuint ID
- GLboolean IsCompiled

3.2.1 Detailed Description

3.2.2 Constructor & Destructor Documentation

3.2.2.1 TinyShaders::TShader::TShader (const GLchar * *ShaderName*, GLuint *ShaderType*, const GLchar * *ShaderFilePath*) [inline]

```
00476                                                                                                   :
00477                     Name(ShaderName)
00478                 {
00479                     Type = ShaderType;
00480                     IsCompiled = GL_FALSE;
00481                     FilePath = ShaderFilePath;
00482                     Compile(GetInstance()->FileToBuffer(ShaderFilePath));
00483                 }
```

3.2.2.2 TinyShaders::TShader::TShader (const GLchar * *ShaderName*, const GLchar * *Buffer*, GLuint *ShaderType*) [inline]

```
00485                     : Name(ShaderName), Type(ShaderType)
00486                 {
00487                     Type = ShaderType;
00488                     IsCompiled = GL_FALSE;
00489                     Compile(Buffer);
00490
00491                 }
```

3.2.2.3 TinyShaders::TShader::TShader ( ) [inline]

```
00492 {}
```

3.2.2.4 TinyShaders::TShader::~TShader ( ) [inline]

```
00493 {}
```

3.2.3 Member Function Documentation

3.2.3.1 GLvoid TinyShaders::TShader::Compile (const GLchar * *Source*) [inline]

```
00499                 {
00500                     //if the component hasn't been compiled yet
00501                     if (!IsCompiled)
00502                     {
```

```
00503                           GLchar ErrorLog[512];
00504                           GLint Successful;
00505
00506                           if (Source != nullptr)
00507                           {
00508                               Handle = glCreateShader(Type);
00509                               glShaderSource(Handle, 1, (const GLchar**)&Source, 0);
00510                               glCompileShader(Handle);
00511
00512                               glGetShaderiv(Handle, GL_COMPILE_STATUS, &Successful);
00513                               glGetShaderInfoLog(Handle, sizeof(ErrorLog), 0, ErrorLog);
00514
00515                               if (Successful != GL_TRUE)
00516                               {
00517                                   PrintErrorMessage(
      TSHADERS_ERROR_FAILEDSHADERLOAD, GetInstance()->
      ShaderTypeToString(Type));
00518                                   printf("%s\n", ErrorLog);
00519                               }
00520
00521                               else
00522                               {
00523                                   IsCompiled = GL_TRUE;
00524                                   ID = GetInstance()->Shaders.size() - 1;
00525                               }
00526                           }
00527                           else
00528                           {
00529                               PrintErrorMessage(
      TSHADERS_ERROR_INVALIDSOURCEFILE);
00530                           }
00531                       }
00532                       else
00533                       {
00534                           //either the file name doesn't exist or the component has already been loaded
00535                           PrintErrorMessage(
      TSHADERS_ERROR_INVALIDFILEPATH, FilePath);
00536                       }
00537                   }
```

3.2.3.2 GLvoid TinyShaders::TShader::Shutdown ( ) [inline]

```
00543                   {
00544                       glDeleteShader(Handle);
00545                       IsCompiled = GL_FALSE;
00546                   }
```

3.2.4 Field Documentation

3.2.4.1 const GLchar* TinyShaders::TShader::FilePath

the FilePath of the component

3.2.4.2 GLuint TinyShaders::TShader::Handle

The handle to the shader in OpenGL

3.2.4.3 GLuint TinyShaders::TShader::ID

the ID of the shader

3.2.4.4 GLboolean TinyShaders::TShader::IsCompiled

Whether the shader has been compiled

3.2.4.5 const GLchar* TinyShaders::TShader::Name

the name of the shader component

3.2.4.6 GLuint TinyShaders::TShader::Type

the type of shader (Vertex, Fragment, etc.)

The documentation for this struct was generated from the following file:

- include/TinyShaders.h

3.3 TinyShaders::TShaderProgram Struct Reference

Public Member Functions

- TShaderProgram ()
- TShaderProgram (const GLchar *ShaderName, std::vector< const GLchar * > ProgramInputs, std::vector< const GLchar * > ProgramOutputs, std::vector< TShader * > ProgramShaders)
- TShaderProgram (const GLchar *ShaderName)
- ~TShaderProgram ()
- GLvoid Shutdown ()
- GLboolean Compile ()

Data Fields

- const GLchar * Name
- GLuint Handle
- GLuint ID
- GLboolean Compiled
- std::vector< const GLchar * > Inputs
- std::vector< const GLchar * > Outputs
- std::vector< TShader * > Shaders

Static Public Attributes

- static GLuint MaxNumShaders = 5

3.3.1 Detailed Description

3.3.2 Constructor & Destructor Documentation

3.3.2.1 TinyShaders::TShaderProgram::TShaderProgram ( ) [inline]

```
00565                 {
00566                     MaxNumShaders = 5;
00567                     ID = 0;
00568                 };
```

3.3.2.2 TinyShaders::TShaderProgram::TShaderProgram (const GLchar * *ShaderName*, std::vector< const GLchar * > *ProgramInputs*, std::vector< const GLchar * > *ProgramOutputs*, std::vector< TShader * > *ProgramShaders*) [inline]

```
00576                                                       :
00577                     Name(ShaderName), Inputs(ProgramInputs),
00578                     Outputs(ProgramOutputs), Shaders(ProgramShaders)
00579                 {
00580                     Compiled = GL_FALSE;
00581                     Compile();
00582                 };
```

3.3.2.3 TinyShaders::TShaderProgram::TShaderProgram (const GLchar * *ShaderName*) [inline]

```
00587                                                       : Name(ShaderName)
00588                 {
00589                     MaxNumShaders = 5;
00590                     Compiled = GL_FALSE;
00591                 };
```

3.3.2.4 TinyShaders::TShaderProgram::~TShaderProgram ( ) [inline]

```
00593 {}
```

3.3.3 Member Function Documentation

3.3.3.1 GLboolean TinyShaders::TShaderProgram::Compile ( ) [inline]

```
00616                 {
00617                     Handle = glCreateProgram();
00618                     GLchar ErrorLog[512];
00619                     GLint Successful = GL_FALSE;
00620                     if (!Compiled)
00621                     {
00622                         for (GLuint Iterator = 0; Iterator < Shaders.size(); Iterator++)
00623                         {
00624                             if (Shaders[Iterator] != nullptr)
00625                             {
00626                                 glAttachShader(Handle, Shaders[Iterator]->
      Handle);
00627                             }
00628                         }
00629
00630                         // specify vertex input attributes
00631                         for (GLuint i = 0; i < Inputs.size(); ++i)
00632                         {
00633                             glBindAttribLocation(Handle, i, Inputs[i]);
00634                         }
00635
00636                         // specify pixel shader outputs
00637                         for (GLuint i = 0; i < Outputs.size(); ++i)
00638                         {
00639                             glBindFragDataLocation(Handle, i, Outputs[i]);
00640                         }
00641
00642                         glLinkProgram(Handle);
00643                         glGetProgramiv(Handle, GL_LINK_STATUS, &Successful);
00644                         glGetProgramInfoLog(Handle, sizeof(ErrorLog), 0, ErrorLog);
00645
00646                         if (!Successful)
00647                         {
00648                             PrintErrorMessage(
      TSHADERS_ERROR_FAILEDSHADERPROGRAMLINK,
      Name);
00649                             printf("%s\n", ErrorLog);
00650                             return GL_FALSE;
00651                         }
00652                         //if a shader successfully compiles then it will add itself to storage. dangerous?
00653                         Compiled = GL_TRUE;
00654                         ID = GetInstance()->ShaderPrograms.size() - 1;
00655                         return GL_TRUE;
00656                     }
00657                     PrintErrorMessage(
      TSHADERS_ERROR_SHADERPROGRAMEXISTS, Name);
```

```
00658                     return GL_FALSE;
00659                 }
```

3.3.3.2 GLvoid TinyShaders::TShaderProgram::Shutdown ( ) [inline]

```
00599                 {
00600                     glDeleteProgram(Handle);
00601 
00602                     for (GLuint Iterator = 0; Iterator < GetInstance()->
      Shaders.size(); Iterator++)
00603                     {
00604                         GetInstance()->Shaders[Iterator]->Shutdown();
00605                         delete GetInstance()->Shaders[Iterator];
00606                     }
00607                     Shaders.clear();
00608                     Inputs.clear();
00609                     Outputs.clear();
00610                 }
```

3.3.4 Field Documentation

3.3.4.1 GLboolean TinyShaders::TShaderProgram::Compiled

Whether the shader program has been linked successfully

3.3.4.2 GLuint TinyShaders::TShaderProgram::Handle

The OpenGL handle to the shader program

3.3.4.3 GLuint TinyShaders::TShaderProgram::ID

the ID of the shader program

3.3.4.4 std::vector<const GLchar*> TinyShaders::TShaderProgram::Inputs

the inputs of the shader program as a vector of strings

3.3.4.5 GLuint TinyShaders::TShaderProgram::MaxNumShaders = 5 [static]

The Maximum number of components a shader program can have. It's always 5

3.3.4.6 const GLchar* TinyShaders::TShaderProgram::Name

The name of the shader program

3.3.4.7 std::vector<const GLchar*> TinyShaders::TShaderProgram::Outputs

the outputs of the shader program as a vector of strings

3.3.4.8 std::vector<TShader*> TinyShaders::TShaderProgram::Shaders

the components that the shader program is comprised of as a vector

The documentation for this struct was generated from the following file:

- include/TinyShaders.h

Chapter 4

File Documentation

4.1 include/TinyShaders.h File Reference

```
#include <list>
#include <vector>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Data Structures

- class TinyShaders
- struct TinyShaders::TShader
- struct TinyShaders::TShaderProgram

Macros

- #define TSHADERS_ERROR_NOTINITIALIZED 1
- #define TSHADERS_ERROR_INVALIDSTRING 2
- #define TSHADERS_ERROR_INVALIDSHADERPROGRAMNAME 3
- #define TSHADERS_ERROR_INVALIDSHADERPROGRAMINDEX 4
- #define TSHADERS_ERROR_INVALIDSHADERNAME 5
- #define TSHADERS_ERROR_INVALIDSHADERINDEX 6
- #define TSHADERS_ERROR_INVALIDFILEPATH 7
- #define TSHADERS_ERROR_SHADERPROGRAMNOTFOUND 8
- #define TSHADERS_ERROR_SHADERNOTFOUND 9
- #define TSHADERS_ERROR_INVALIDSHADERTYPE 10
- #define TSHADERS_ERROR_FAILEDSHADERLOAD 11
- #define TSHADERS_ERROR_FAILEDSHADERPROGRAMLINK 12
- #define TSHADERS_ERROR_SHADEREXISTS 13
- #define TSHADERS_ERROR_SHADERPROGRAMEXISTS 14
- #define TSHADERS_ERROR_INVALIDSOURCEFILE 15

4.1.1 Macro Definition Documentation

4.1.1.1 #define TSHADERS_ERROR_FAILEDSHADERLOAD 11

4.1.1.2 #define TSHADERS_ERROR_FAILEDSHADERPROGRAMLINK 12

4.1.1.3 #define TSHADERS_ERROR_INVALIDFILEPATH 7

4.1.1.4 #define TSHADERS_ERROR_INVALIDSHADERINDEX 6

4.1.1.5 #define TSHADERS_ERROR_INVALIDSHADERNAME 5

4.1.1.6 #define TSHADERS_ERROR_INVALIDSHADERPROGRAMINDEX 4

4.1.1.7 #define TSHADERS_ERROR_INVALIDSHADERPROGRAMNAME 3

4.1.1.8 #define TSHADERS_ERROR_INVALIDSHADERTYPE 10

4.1.1.9 #define TSHADERS_ERROR_INVALIDSOURCEFILE 15

4.1.1.10 #define TSHADERS_ERROR_INVALIDSTRING 2

4.1.1.11 #define TSHADERS_ERROR_NOTINITIALIZED 1

4.1.1.12 #define TSHADERS_ERROR_SHADEREXISTS 13

4.1.1.13 #define TSHADERS_ERROR_SHADERNOTFOUND 9

4.1.1.14 #define TSHADERS_ERROR_SHADERPROGRAMEXISTS 14

4.1.1.15 #define TSHADERS_ERROR_SHADERPROGRAMNOTFOUND 8

[twoside]book

fixltx2e calc doxygen graphicx [utf8]inputenc makeidx multicol multirow warntextcomp textcomp [nointegrals]wasysym [table]xcolor

[T1]fontenc mathptmx [scaled=.90]helvet courier amssymb sectsty

geometry a4paper,top=2.5cm,bottom=2.5cm,left=2.5cm,right=2.5cm

fancyhdr

natbib [titles]tocloft

ifpdf [pdftex,pagebackref=true]hyperref

TinyShaders

0.3

Generated by Doxygen 1.8.7

Wed Nov 4 2015 17:18:53

Contents

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Data Structure Documentation

7.1 TinyShaders Class Reference

`#include <TinyShaders.h>`

Data Structures

- struct TShader
- struct TShaderProgram

Public Member Functions

- TinyShaders ()
- ~TinyShaders ()

Static Public Member Functions

- static void Shutdown ()
- static TShaderProgram * GetShaderProgramByName (const char *ProgramName)
- static TShaderProgram * GetShaderProgramByIndex (GLuint ProgramIndex)
- static TShader * GetShaderByName (const GLchar *ShaderName)
- static TShader * GetShaderByIndex (GLuint ShaderIndex)
- static void LoadShader (const GLchar *Name, const GLchar *ShaderFile, GLuint ShaderType)
- static void LoadShaderProgramsFromConfigFile (const GLchar *ConfigFile)
- static void LoadShadersFromConfigFile (const GLchar *ConfigFile)
- static void SaveShaderProgramsToConfigFile (const GLchar *FileName)
- static void BuildProgramFromShaders (const GLchar *ShaderName, std::vector< const GLchar * > Inputs, std::vector< const GLchar * > Outputs, const GLchar *VertexShaderName, const GLchar *Fragment↩ ShaderName, const GLchar *GeometryShaderName, const GLchar *TessContShaderName, const GLchar *TessEvalShaderName)
- static GLboolean ShaderProgramExists (const GLchar *ShaderName)
- static GLboolean ShaderExists (const GLchar *ShaderName)
- static GLvoid LoadShaderFromBuffer (const char *Name, const GLchar *Buffer, GLuint ShaderType)

Private Member Functions

- GLchar * FileToBuffer (const GLchar *Path)
- GLuint StringToShaderType (const char *TypeString)

- const GLchar * ShaderTypeToString (GLuint ShaderType)
- GLvoid AddProgram (TShaderProgram *NewProgram)
- GLvoid AddShader (TShader *NewShader)

Static Private Member Functions

- static TinyShaders * GetInstance ()
- static GLvoid PrintErrorMessage (GLuint ErrorNumber, const GLchar *String=nullptr)

Private Attributes

- std::vector< TShaderProgram * > ShaderPrograms
- std::vector< TShader * > Shaders

Static Private Attributes

- static GLboolean IsInitialized = GL_FALSE
- static TinyShaders * Instance = nullptr

7.1.1 Detailed Description

7.1.2 Constructor & Destructor Documentation

7.1.2.1 TinyShaders::TinyShaders ( ) [inline]

```
00045 {}
```

7.1.2.2 TinyShaders::~TinyShaders ( ) [inline]

```
00046 {}
```

7.1.3 Member Function Documentation

7.1.3.1 GLvoid TinyShaders::AddProgram (TShaderProgram * *NewProgram*) [inline], [private]

```
00902         {
00903             if(NewProgram != nullptr)
00904             {
00905                 if(NewProgram->Compiled)
00906                 {
00907                     GetInstance()->ShaderPrograms.push_back(NewProgram);
00908                 }
00909             }
00910         }
```

7.1.3.2 GLvoid TinyShaders::AddShader (TShader * *NewShader*) [inline], [private]

```
00913         {
00914             if(NewShader != nullptr)
00915             {
00916                 if(NewShader->IsCompiled)
00917                 {
00918                     GetInstance()->Shaders.push_back(NewShader);
00919                 }
00920             }
00921         }
```

7.1.3.3 static void TinyShaders::BuildProgramFromShaders (const GLchar * *ShaderName*, std::vector< const GLchar * > *Inputs*, std::vector< const GLchar * > *Outputs*, const GLchar * *VertexShaderName*, const GLchar * *FragmentShaderName*, const GLchar * *GeometryShaderName*, const GLchar * *TessContShaderName*, const GLchar * *TessEvalShaderName*) [inline], [static]

```
00381         {
00382             if (TinyShaders::IsInitialized)
00383             {
00384                 std::vector<TShader*> Shaders;
00385                 Shaders.push_back(GetShaderByName(VertexShaderName));
00386                 Shaders.push_back(GetShaderByName(FragmentShaderName));
00387                 Shaders.push_back(GetShaderByName(GeometryShaderName));
00388                 Shaders.push_back(GetShaderByName(TessContShaderName));
00389                 Shaders.push_back(GetShaderByName(TessEvalShaderName));
00390 
00391                 TShaderProgram* NewShaderProgram = new TShaderProgram(ShaderName, Inputs, Outputs, Shaders)
      ;
00392                 delete NewShaderProgram;
00393             }
00394             PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED);
00395         }
```

7.1.3.4 GLchar* TinyShaders::FileToBuffer (const GLchar * *Path*) [inline], [private]

```
00799         {
00800             FILE* File = fopen(Path, "rt");
00801 
00802             if (File == nullptr)
00803             {
00804                 PrintErrorMessage(
      TSHADERS_ERROR_INVALIDFILEPATH, Path);
00805                 return nullptr;
00806             }
00807 
00808             //get total byte in given file
00809             fseek(File, 0, SEEK_END);
00810             GLuint FileLength = ftell(File);
00811             fseek(File, 0, SEEK_SET);
00812 
00813             //allocate a file buffer and read the contents of the file
00814             char* Buffer = new char[FileLength + 1];
00815             memset(Buffer, 0, FileLength + 1);
00816             fread(Buffer, sizeof(char), FileLength, File);
00817 
00818             fclose(File);
00819             return Buffer;
00820         }
```

7.1.3.5 static TinyShaders* TinyShaders::GetInstance ( ) [inline], [static], [private]

```
00675         {
00676             if (TinyShaders::IsInitialized)
00677             {
00678                 return TinyShaders::Instance;
00679             }
00680 
00681             TinyShaders::IsInitialized = GL_TRUE;
00682             TinyShaders::Instance = new TinyShaders();
00683             return TinyShaders::Instance;
00684         }
```

7.1.3.6 static TShader* TinyShaders::GetShaderByIndex (GLuint *ShaderIndex*) [inline], [static]

```
00148         {
00149             if (TinyShaders::IsInitialized)
00150             {
00151                 if (ShaderIndex <= GetInstance()->Shaders.size() - 1)
00152                 {
00153                     return GetInstance()->Shaders[ShaderIndex];
00154                 }
00155                 PrintErrorMessage(
      TSHADERS_ERROR_INVALIDSHADERINDEX);
00156                 return nullptr;
00157             }
```

```
00158             PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED);
00159             return nullptr;
00160         }
```

7.1.3.7 static TShader* TinyShaders::GetShaderByName (const GLchar * *ShaderName*) [inline], [static]

```
00122         {
00123             if (TinyShaders::IsInitialized)
00124             {
00125                 if (ShaderName != nullptr)
00126                 {
00127                     for (GLuint Iterator = 0; Iterator < GetInstance()->
      Shaders.size(); Iterator++)
00128                     {
00129                         if (!strcmp(GetInstance()->Shaders[Iterator]->Name, ShaderName))
00130                         {
00131                             return GetInstance()->Shaders[Iterator];
00132                         }
00133                     }
00134                     PrintErrorMessage(
      TSHADERS_ERROR_SHADERNOTFOUND);
00135                     return nullptr;
00136                 }
00137                 PrintErrorMessage(
      TSHADERS_ERROR_INVALIDSHADERNAME);
00138                 return nullptr;
00139             }
00140             PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED);
00141             return nullptr;
00142         }
```

7.1.3.8 static TShaderProgram* TinyShaders::GetShaderProgramByIndex (GLuint *ProgramIndex*) [inline], [static]

```
00104         {
00105             if (TinyShaders::IsInitialized)
00106             {
00107                 if (ProgramIndex >= GetInstance()->ShaderPrograms.size() - 1)
00108                 {
00109                     return GetInstance()->ShaderPrograms[ProgramIndex];
00110                 }
00111                 PrintErrorMessage(
      TSHADERS_ERROR_INVALIDSHADERPROGRAMINDEX);
00112                 return nullptr;
00113             }
00114             PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED);
00115             return nullptr;
00116         }
```

7.1.3.9 static TShaderProgram* TinyShaders::GetShaderProgramByName (const char * *ProgramName*) [inline], [static]

```
00079         {
00080             if (TinyShaders::IsInitialized)
00081             {
00082                 if (ProgramName != nullptr)
00083                 {
00084                     for (GLuint Iterator = 0; Iterator < GetInstance()->
      ShaderPrograms.size(); Iterator++)
00085                     {
00086                         if (!strcmp(GetInstance()->ShaderPrograms[Iterator]->Name,
       ProgramName))
00087                         {
00088                             return GetInstance()->ShaderPrograms[Iterator];
00089                         }
00090                     }
00091                     return nullptr;
00092                 }
00093                 PrintErrorMessage(
      TSHADERS_ERROR_SHADERPROGRAMNOTFOUND);
00094                 return nullptr;
00095             }
00096             PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED);
00097             return nullptr;
00098         }
```

7.1.3.10 static void TinyShaders::LoadShader (const GLchar * *Name*, const GLchar * *ShaderFile*, GLuint *ShaderType*) [inline], [static]

```
00166         {
00167             if (TinyShaders::IsInitialized)
00168 
00169             {
00170                 if (Name != nullptr)
00171 
00172                 {
00173                     if (ShaderType <= 5)
00174                     {
00175                         GetInstance()->Shaders.push_back(new TShader(Name, ShaderType,
      ShaderFile));
00176                     }
00177                     PrintErrorMessage(
      TSHADERS_ERROR_INVALIDSHADERTYPE, GetInstance()->
      ShaderTypeToString(ShaderType));
00178                 }
00179                 PrintErrorMessage(TSHADERS_ERROR_INVALIDSTRING
      );
00180             }
00181             PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED);
00182         }
```

7.1.3.11 static GLvoid TinyShaders::LoadShaderFromBuffer (const char * *Name*, const GLchar * *Buffer*, GLuint *ShaderType*) [inline], [static]

```
00449         {
00450             if(TinyShaders::IsInitialized)
00451             {
00452                 if(Buffer != nullptr)
00453                 {
00454                     if(Name != nullptr)
00455                     {
00456                         if(!ShaderExists(Name))
00457                         {
00458                             TShader* NewShader = new TShader(Name, Buffer, ShaderType);
00459                             delete NewShader;
00460                         }
00461                         PrintErrorMessage(
      TSHADERS_ERROR_SHADERNOTFOUND);
00462                     }
00463                     PrintErrorMessage(
      TSHADERS_ERROR_INVALIDSHADERNAME);
00464                 }
00465                 PrintErrorMessage(TSHADERS_ERROR_INVALIDSTRING
      );
00466             }
00467             PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED);
00468         }
```

7.1.3.12 static void TinyShaders::LoadShaderProgramsFromConfigFile (const GLchar * *ConfigFile*) [inline], [static]

```
00188         {
00189             if (!TinyShaders::IsInitialized)
00190             {
00191                 FILE* pConfigFile = fopen(ConfigFile, "r");
00192                 GLuint NumInputs = 0;
00193                 GLuint NumOutputs = 0;
00194                 GLuint NumPrograms = 0;
00195                 GLuint NumShaders = 0;
00196                 GLuint Iterator = 0;
00197 
00198                 std::vector<const GLchar*> Inputs, Outputs, Paths, Names;
00199                 std::vector<TShader*> Shaders;
00200                 if (pConfigFile)
00201                 {
00202                     //get the total number of shader programs
00203                     fscanf(pConfigFile, "%i\n", &NumPrograms);
00204 
00205                     for (GLuint ProgramIter = 0;
00206                         ProgramIter < NumPrograms;
00207                         ProgramIter++, Paths.clear(), Inputs.clear(), Outputs.clear(), Names.clear(),
      Shaders.clear())
00208                     {
00209                         //get the name of the shader program
```

```
00210                         GLchar* ProgramName = new GLchar[255];
00211                         fscanf(pConfigFile, "%s\n", ProgramName);
00212 
00213                         //this is an anti-trolling measure. If a shader with the same name already exists
      the don't bother making a new one.
00214                         if (!GetInstance()->ShaderProgramExists(ProgramName))
00215                         {
00216                             //get the number of shader inputs
00217                             fscanf(pConfigFile, "%i\n", &NumInputs);
00218 
00219                             //get all inputs
00220                             for (Iterator = 0; Iterator < NumInputs; Iterator++)
00221                             {
00222                                 GLchar* Input = new GLchar[255];
00223                                 fscanf(pConfigFile, "%s\n", Input);
00224                                 Inputs.push_back(Input);
00225                             }
00226 
00227                             //get the number of shader outputs
00228                             fscanf(pConfigFile, "%i\n", &NumOutputs);
00229 
00230                             //get all outputs
00231                             for (Iterator = 0; Iterator < NumOutputs; Iterator++)
00232                             {
00233                                 GLchar* Output = new GLchar[255];
00234                                 fscanf(pConfigFile, "%s\n", Output);
00235                                 Outputs.push_back(Output);
00236                             }
00237 
00238                             //get number of shaders
00239                             fscanf(pConfigFile, "%i\n", &NumShaders);
00240 
00241                             for(GLuint ShaderIter = 0; ShaderIter < NumShaders; ShaderIter++)
00242                             {
00243                                 GLchar* ShaderName = new GLchar[255];
00244                                 GLchar* ShaderPath = new GLchar[255];
00245                                 GLchar* ShaderType = new GLchar[255];
00246 
00247                                 //get shader name
00248                                 fscanf(pConfigFile, "%s\n", ShaderName);
00249 
00250                                 //if the shader hasn't been loaded already then make a new one
00251                                 if(!ShaderExists(ShaderName))
00252                                 {
00253                                     //get type
00254                                     fscanf(pConfigFile, "%s\n", ShaderType);
00255                                     //get file path
00256                                     fscanf(pConfigFile, "%s\n", ShaderPath);
00257 
00258                                     Shaders.push_back(new TShader(ShaderName,
      GetInstance()->StringToShaderType((const char*)ShaderType), ShaderPath));
00259                                 }
00260 
00261                                 else
00262                                 {
00263                                     //if shader already exists then add an existing one from storage, it
      should already be compiled
00264                                     Shaders.push_back(GetShaderByName(ShaderName));
00265                                 }
00266                             }
00267 
00268                             GetInstance()->ShaderPrograms.push_back(new
      TShaderProgram(ProgramName, Inputs, Outputs, Shaders));
00269                         }
00270                         fclose(pConfigFile);
00271                     }
00272                 }
00273                 else
00274                 {
00275                     PrintErrorMessage(
      TSHADERS_ERROR_INVALIDFILEPATH);
00276                 }
00277             }
00278             else
00279             {
00280                 PrintErrorMessage(TSHADERS_ERROR_NOTINITIALIZED
      );
00281             }
00282         }
```

7.1.3.13 static void TinyShaders::LoadShadersFromConfigFile (const GLchar * *ConfigFile*) [inline], [static]

```
00285         {
00286             if(TinyShaders::IsInitialized)
```

```
00287             {
00288                 FILE* pConfigFile = fopen(ConfigFile, "r+");
00289                 GLuint NumShaders = 0;
00290 
00291                 if(pConfigFile)
00292                 {
00293                     //get the number of shaders to load
00294                     fscanf(pConfigFile, "%i\n\n", &NumShaders);
00295                     GLchar* ShaderName;
00296                     GLchar* ShaderType;
00297                     GLchar* ShaderPath;
00298 
00299                     GLchar empty[255];
00300 
00301                     for(GLuint ShaderIter = 0; ShaderIter < NumShaders;
00302                             ShaderIter++, fscanf(pConfigFile, "\n\n"))
00303                     {
00304                         ShaderName = empty;
00305                         fscanf(pConfigFile, "%s\n", ShaderName);
00306 
00307                         if(!GetInstance()->ShaderExists(ShaderName))
00308                         {
00309                             ShaderType = empty;
00310                             fscanf(pConfigFile, "%s\n", ShaderType);
00311 
00312                             ShaderPath = empty;
00313                             fscanf(pConfigFile, "%s\n", ShaderPath);
00314 
00315                             TShader* NewShader = new TShader(ShaderName,
    GetInstance()->StringToShaderType(ShaderType), ShaderPath);
00316                             delete NewShader;
00317                         }
00318                     }
00319                 }
00320             }
00321         }
```

7.1.3.14 static GLvoid TinyShaders::PrintErrorMessage (GLuint *ErrorNumber*, const GLchar * *String* = `nullptr`) [inline], [static], [private]

```
00690         {
00691             switch (ErrorNumber)
00692             {
00693             case TSHADERS_ERROR_NOTINITIALIZED:
00694             {
00695                 printf("Error: TinyShaders must first be initialized \n");
00696                 break;
00697             }
00698 
00699             case TSHADERS_ERROR_INVALIDSTRING:
00700             {
00701                 printf("Error: given string is invalid \n");
00702                 break;
00703             }
00704 
00705             case TSHADERS_ERROR_INVALIDSHADERPROGRAMNAME:
00706             {
00707                 printf("Error: given shader name is invalid \n");
00708                 break;
00709             }
00710 
00711             case TSHADERS_ERROR_INVALIDSHADERPROGRAMINDEX:
00712             {
00713                 printf("Error: given shader index is invalid \n");
00714                 break;
00715             }
00716 
00717             case TSHADERS_ERROR_INVALIDSHADERNAME:
00718             {
00719                 printf("Error: given shader component name is invalid \n");
00720                 break;
00721             }
00722 
00723             case TSHADERS_ERROR_INVALIDSHADERINDEX:
00724             {
00725                 printf("Error: given shader component index is invalid \n");
00726                 break;
00727             }
00728 
00729             case TSHADERS_ERROR_INVALIDFILEPATH:
00730             {
00731                 printf("Error: given file path is invalid %s \n", String);
00732                 break;
```

```
00733             }
00734
00735             case TSHADERS_ERROR_SHADERPROGRAMNOTFOUND:
00736             {
00737                 printf("Error: shader with given name %s was not found \n", String);
00738                 break;
00739             }
00740
00741             case TSHADERS_ERROR_SHADERNOTFOUND:
00742             {
00743                 printf("Error: shader component with given name %s was not found \n", String);
00744                 break;
00745             }
00746
00747             case TSHADERS_ERROR_INVALIDSHADERTYPE:
00748             {
00749                 printf("Error: invalid shader type given \n");
00750                 break;
00751             }
00752
00753             case TSHADERS_ERROR_FAILEDSHADERLOAD:
00754             {
00755                 printf("Error: failed to compile %s shader component \n", String);
00756                 break;
00757             }
00758
00759             case TSHADERS_ERROR_FAILEDSHADERPROGRAMLINK:
00760             {
00761                 if (String != nullptr)
00762                 {
00763                     printf("Error: failed to link program %s \n", String);
00764                 }
00765                 break;
00766             }
00767
00768             case TSHADERS_ERROR_SHADEREXISTS:
00769             {
00770                 printf("Error: shader component with this name %s already exists \n", String);
00771                 break;
00772             }
00773
00774             case TSHADERS_ERROR_SHADERPROGRAMEXISTS:
00775             {
00776                 if (String != nullptr)
00777                 {
00778                     printf("Error: shader with this name %s already exists \n", String);
00779                     break;
00780                 }
00781             }
00782
00783             case TSHADERS_ERROR_INVALIDSOURCEFILE:
00784             {
00785                 printf("Given Source file is invalid");
00786                 break;
00787             }
00788             default:
00789             {
00790                 break;
00791             }
00792             }
00793         }
```

7.1.3.15 static void TinyShaders::SaveShaderProgramsToConfigFile (const GLchar * *FileName*) [inline], [static]

```
00324         {
00325             //write total amount of shaders
00326             FILE* pConfigFile = fopen(FileName, "w+");
00327
00328             fprintf(pConfigFile, "%i\n\n", (GLint)GetInstance()->
      ShaderPrograms.size());
00329
00330             for(GLuint ProgramIter = 0; ProgramIter < GetInstance()->
      ShaderPrograms.size(); ProgramIter++)
00331             {
00332                 //write program name
00333                 fprintf(pConfigFile, "%s\n", GetInstance()->
      ShaderPrograms[ProgramIter]->Name);
00334
00335                 //write number of inputs
00336                 fprintf(pConfigFile, "%i\n", (GLint)GetInstance()->
      ShaderPrograms[ProgramIter]->Inputs.size());
00337
00338                 //write inputs
00339                 for(GLuint InputIter = 0; InputIter < GetInstance()->
```

```
      ShaderPrograms[ProgramIter]->Inputs.size(); InputIter++)
00340                 {
00341                     fprintf(pConfigFile, "%s\n", GetInstance()->
      ShaderPrograms[ProgramIter]->Inputs[InputIter]);
00342                 }
00343
00344                 fprintf(pConfigFile, "%i\n", (GLint)GetInstance()->
      ShaderPrograms[ProgramIter]->Outputs.size());
00345
00346                 //write outputs
00347                 for(GLuint OutputIter = 0; OutputIter < GetInstance()->
      ShaderPrograms[ProgramIter]->Outputs.size(); OutputIter++)
00348                 {
00349                     fprintf(pConfigFile, "%s\n", GetInstance()->
      ShaderPrograms[ProgramIter]->Outputs[OutputIter]);
00350                 }
00351
00352                 //write number of shaders
00353                 fprintf(pConfigFile, "%i\n", (GLint)GetInstance()->
      ShaderPrograms[ProgramIter]->Shaders.size());
00354
00355                 for(GLuint ShaderIter = 0; ShaderIter < GetInstance()->
      ShaderPrograms[ProgramIter]->Shaders.size(); ShaderIter++)
00356                 {
00357                     //write shader name
00358                     fprintf(pConfigFile, "%s\n", GetInstance()->
      ShaderPrograms[ProgramIter]->Shaders[ShaderIter]->Name);
00359
00360                     //write shader type
00361                     fprintf(pConfigFile, "%s\n", GetInstance()->
      ShaderTypeToString(GetInstance()->ShaderPrograms[ProgramIter]->
      Shaders[ShaderIter]->Type));
00362
00363                     //write shader file path
00364                     fprintf(pConfigFile, "%s\n", GetInstance()->
      ShaderPrograms[ProgramIter]->Shaders[ShaderIter]->FilePath);
00365                 }
00366             }
00367             fclose(pConfigFile);
00368         }
```

7.1.3.16 static GLboolean TinyShaders::ShaderExists (const GLchar * *ShaderName*) [inline], [static]

```
00425         {
00426             //make sure the name isn't empty
00427             if (ShaderName != nullptr)
00428             {
00429                 //make sure the shader manager has shaders stored already
00430                 if (!GetInstance()->Shaders.empty())
00431                 {
00432                     //for each shader in the shader manager
00433                     for (GLuint Iterator = 0; Iterator < GetInstance()->
      Shaders.size(); Iterator++)
00434                     {
00435                         //if a shader of the same name is the same as an existing shader
00436                         if (!strcmp(ShaderName, GetInstance()->
      Shaders[Iterator]->Name))
00437                         {
00438                             return GL_TRUE;
00439                         }
00440                     }
00441                     return GL_FALSE;
00442                 }
00443                 return GL_FALSE;
00444             }
00445             return GL_FALSE;
00446         }
```

7.1.3.17 static GLboolean TinyShaders::ShaderProgramExists (const GLchar * *ShaderName*) [inline], [static]

```
00401         {
00402             if (ShaderName != nullptr)
00403             {
00404                 if (!GetInstance()->ShaderPrograms.empty())
00405                 {
00406                     for (GLuint Iterator = 0; Iterator < GetInstance()->
      ShaderPrograms.size(); Iterator++)
00407                     {
00408                         if (GetInstance()->ShaderPrograms[Iterator] != nullptr &&
00409                             !strcmp(ShaderName, GetInstance()->
```

```
      ShaderPrograms[Iterator]->Name))
00410                         {
00411                             return GL_TRUE;
00412                         }
00413                     }
00414                     return GL_FALSE;
00415                 }
00416                 return GL_FALSE;
00417             }
00418             return GL_FALSE;
00419         }
```

7.1.3.18 const GLchar* TinyShaders::ShaderTypeToString (GLuint *ShaderType*) [inline], [private]

```
00864         {
00865             switch (ShaderType)
00866             {
00867                 case GL_VERTEX_SHADER:
00868                 {
00869                     return "Vertex";
00870                 }
00871
00872                 case GL_FRAGMENT_SHADER:
00873                 {
00874                     return "Fragment";
00875                 }
00876
00877                 case GL_GEOMETRY_SHADER:
00878                 {
00879                     return "Geometry";
00880                 }
00881
00882                 case GL_TESS_CONTROL_SHADER:
00883                 {
00884                     return "Tessellation Control";
00885                 }
00886
00887                 case GL_TESS_EVALUATION_SHADER:
00888                 {
00889                     return "Tessellation Evaluation";
00890                 }
00891
00892                 default:
00893                 {
00894                     return NULL;
00895                 }
00896             }
00897
00898             return nullptr;
00899         }
```

7.1.3.19 static void TinyShaders::Shutdown ( ) [inline], [static]

```
00053         {
00054             if (TinyShaders::IsInitialized)
00055             {
00056                 for (GLuint Iterator = 0; Iterator < GetInstance()->
      Shaders.size(); Iterator++)
00057                 {
00058                     GetInstance()->Shaders[Iterator]->Shutdown();
00059                      delete GetInstance()->Shaders[Iterator];
00060                 }
00061
00062                 for (GLuint Iterator = 0; Iterator < GetInstance()->
      ShaderPrograms.size(); Iterator++)
00063                 {
00064                     GetInstance()->ShaderPrograms[Iterator]->Shutdown();
00065                     delete GetInstance()->ShaderPrograms[Iterator];
00066                 }
00067
00068                 GetInstance()->ShaderPrograms.clear();
00069                 GetInstance()->Shaders.clear();
00070
00071                 delete Instance;
00072             }
00073         }
```

7.1.3.20 GLuint TinyShaders::StringToShaderType (const char * *TypeString*) [inline], [private]

```
00826         {
00827             if(TypeString != nullptr)
00828             {
00829                 if (!strcmp(TypeString, "Vertex"))
00830                 {
00831                     return GL_VERTEX_SHADER;
00832                 }
00833 
00834                 if (!strcmp(TypeString, "Fragment"))
00835                 {
00836                     return GL_FRAGMENT_SHADER;
00837                 }
00838 
00839                 if (!strcmp(TypeString, "Geometry"))
00840                 {
00841                     return GL_GEOMETRY_SHADER;
00842                 }
00843 
00844                 if (!strcmp(TypeString, "Tessellation Control"))
00845                 {
00846                     return GL_TESS_CONTROL_SHADER;
00847                 }
00848 
00849                 if (!strcmp(TypeString, "Tessellation Evaluation"))
00850                 {
00851                     return GL_TESS_EVALUATION_SHADER;
00852                 }
00853 
00854                 return GL_FALSE;
00855             }
00856             PrintErrorMessage(TSHADERS_ERROR_INVALIDSTRING);
00857             return GL_FALSE;
00858         }
```

7.1.4 Field Documentation

7.1.4.1 TinyShaders * TinyShaders::Instance = nullptr [static], [private]

a static instance of the TinyShaders API

7.1.4.2 GLboolean TinyShaders::IsInitialized = GL_FALSE [static], [private]

Whether TinyShaders has ban initialized

7.1.4.3 std::vector<TShaderProgram*> TinyShaders::ShaderPrograms [private]

all loaded shader programs

7.1.4.4 std::vector<TShader*> TinyShaders::Shaders [private]

all loaded shaders

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

- include/TinyShaders.h

7.2 TinyShaders::TShader Struct Reference

Public Member Functions

- TShader (const GLchar *ShaderName, GLuint ShaderType, const GLchar *ShaderFilePath)
- TShader (const GLchar *ShaderName, const GLchar *Buffer, GLuint ShaderType)

- TShader ()
- ~TShader ()
- GLvoid Compile (const GLchar *Source)
- GLvoid Shutdown ()

Data Fields

- const GLchar * Name
- const GLchar * FilePath
- GLuint Handle
- GLuint Type
- GLuint ID
- GLboolean IsCompiled

7.2.1 Detailed Description

7.2.2 Constructor & Destructor Documentation

7.2.2.1 TinyShaders::TShader::TShader (const GLchar * *ShaderName*, GLuint *ShaderType*, const GLchar * *ShaderFilePath*) [inline]

```
00476                                                                                                   :
00477                     Name(ShaderName)
00478                 {
00479                     Type = ShaderType;
00480                     IsCompiled = GL_FALSE;
00481                     FilePath = ShaderFilePath;
00482                     Compile(GetInstance()->FileToBuffer(ShaderFilePath));
00483                 }
```

7.2.2.2 TinyShaders::TShader::TShader (const GLchar * *ShaderName*, const GLchar * *Buffer*, GLuint *ShaderType*) [inline]

```
00485                     : Name(ShaderName), Type(ShaderType)
00486                 {
00487                     Type = ShaderType;
00488                     IsCompiled = GL_FALSE;
00489                     Compile(Buffer);
00490
00491                 }
```

7.2.2.3 TinyShaders::TShader::TShader ( ) [inline]

```
00492 {}
```

7.2.2.4 TinyShaders::TShader::~TShader ( ) [inline]

```
00493 {}
```

7.2.3 Member Function Documentation

7.2.3.1 GLvoid TinyShaders::TShader::Compile (const GLchar * *Source*) [inline]

```
00499                 {
00500                     //if the component hasn't been compiled yet
00501                     if (!IsCompiled)
00502                     {
```

```
00503                         GLchar ErrorLog[512];
00504                         GLint Successful;
00505 
00506                         if (Source != nullptr)
00507                         {
00508                             Handle = glCreateShader(Type);
00509                             glShaderSource(Handle, 1, (const GLchar**)&Source, 0);
00510                             glCompileShader(Handle);
00511 
00512                             glGetShaderiv(Handle, GL_COMPILE_STATUS, &Successful);
00513                             glGetShaderInfoLog(Handle, sizeof(ErrorLog), 0, ErrorLog);
00514 
00515                             if (Successful != GL_TRUE)
00516                             {
00517                                 PrintErrorMessage(
      TSHADERS_ERROR_FAILEDSHADERLOAD, GetInstance()->
      ShaderTypeToString(Type));
00518                                 printf("%s\n", ErrorLog);
00519                             }
00520 
00521                             else
00522                             {
00523                                 IsCompiled = GL_TRUE;
00524                                 ID = GetInstance()->Shaders.size() - 1;
00525                             }
00526                         }
00527                         else
00528                         {
00529                             PrintErrorMessage(
      TSHADERS_ERROR_INVALIDSOURCEFILE);
00530                         }
00531                     }
00532                     else
00533                     {
00534                         //either the file name doesn't exist or the component has already been loaded
00535                         PrintErrorMessage(
      TSHADERS_ERROR_INVALIDFILEPATH, FilePath);
00536                     }
00537                 }
```

7.2.3.2 GLvoid TinyShaders::TShader::Shutdown ( ) [inline]

```
00543                 {
00544                     glDeleteShader(Handle);
00545                     IsCompiled = GL_FALSE;
00546                 }
```

7.2.4 Field Documentation

7.2.4.1 const GLchar* TinyShaders::TShader::FilePath

the FilePath of the component

7.2.4.2 GLuint TinyShaders::TShader::Handle

The handle to the shader in OpenGL

7.2.4.3 GLuint TinyShaders::TShader::ID

the ID of the shader

7.2.4.4 GLboolean TinyShaders::TShader::IsCompiled

Whether the shader has been compiled

7.2.4.5 const GLchar* TinyShaders::TShader::Name

the name of the shader component

7.2.4.6 GLuint TinyShaders::TShader::Type

the type of shader (Vertex, Fragment, etc.)

The documentation for this struct was generated from the following file:

- include/TinyShaders.h

7.3 TinyShaders::TShaderProgram Struct Reference

Public Member Functions

- TShaderProgram ()
- TShaderProgram (const GLchar *ShaderName, std::vector< const GLchar * > ProgramInputs, std::vector< const GLchar * > ProgramOutputs, std::vector< TShader * > ProgramShaders)
- TShaderProgram (const GLchar *ShaderName)
- ~TShaderProgram ()
- GLvoid Shutdown ()
- GLboolean Compile ()

Data Fields

- const GLchar * Name
- GLuint Handle
- GLuint ID
- GLboolean Compiled
- std::vector< const GLchar * > Inputs
- std::vector< const GLchar * > Outputs
- std::vector< TShader * > Shaders

Static Public Attributes

- static GLuint MaxNumShaders = 5

7.3.1 Detailed Description

7.3.2 Constructor & Destructor Documentation

7.3.2.1 TinyShaders::TShaderProgram::TShaderProgram ( ) [inline]

```
00565                 {
00566                     MaxNumShaders = 5;
00567                     ID = 0;
00568                 };
```

7.3.2.2 TinyShaders::TShaderProgram::TShaderProgram (const GLchar * *ShaderName*, std::vector< const GLchar * > *ProgramInputs*, std::vector< const GLchar * > *ProgramOutputs*, std::vector< TShader * > *ProgramShaders*) [inline]

```
00576                                                       :
00577                     Name(ShaderName), Inputs(ProgramInputs),
00578                     Outputs(ProgramOutputs), Shaders(ProgramShaders)
00579                 {
00580                     Compiled = GL_FALSE;
00581                     Compile();
00582                 };
```

7.3.2.3 TinyShaders::TShaderProgram::TShaderProgram (const GLchar * *ShaderName*) [inline]

```
00587                                                       : Name(ShaderName)
00588                 {
00589                     MaxNumShaders = 5;
00590                     Compiled = GL_FALSE;
00591                 };
```

7.3.2.4 TinyShaders::TShaderProgram::~TShaderProgram ( ) [inline]

```
00593 {}
```

7.3.3 Member Function Documentation

7.3.3.1 GLboolean TinyShaders::TShaderProgram::Compile ( ) [inline]

```
00616                 {
00617                     Handle = glCreateProgram();
00618                     GLchar ErrorLog[512];
00619                     GLint Successful = GL_FALSE;
00620                     if (!Compiled)
00621                     {
00622                         for (GLuint Iterator = 0; Iterator < Shaders.size(); Iterator++)
00623                         {
00624                             if (Shaders[Iterator] != nullptr)
00625                             {
00626                                 glAttachShader(Handle, Shaders[Iterator]->
      Handle);
00627                             }
00628                         }
00629
00630                         // specify vertex input attributes
00631                         for (GLuint i = 0; i < Inputs.size(); ++i)
00632                         {
00633                             glBindAttribLocation(Handle, i, Inputs[i]);
00634                         }
00635
00636                         // specify pixel shader outputs
00637                         for (GLuint i = 0; i < Outputs.size(); ++i)
00638                         {
00639                             glBindFragDataLocation(Handle, i, Outputs[i]);
00640                         }
00641
00642                         glLinkProgram(Handle);
00643                         glGetProgramiv(Handle, GL_LINK_STATUS, &Successful);
00644                         glGetProgramInfoLog(Handle, sizeof(ErrorLog), 0, ErrorLog);
00645
00646                         if (!Successful)
00647                         {
00648                             PrintErrorMessage(
      TSHADERS_ERROR_FAILEDSHADERPROGRAMLINK,
      Name);
00649                             printf("%s\n", ErrorLog);
00650                             return GL_FALSE;
00651                         }
00652                         //if a shader successfully compiles then it will add itself to storage. dangerous?
00653                         Compiled = GL_TRUE;
00654                         ID = GetInstance()->ShaderPrograms.size() - 1;
00655                         return GL_TRUE;
00656                     }
00657                     PrintErrorMessage(
      TSHADERS_ERROR_SHADERPROGRAMEXISTS, Name);
```

```
00658                     return GL_FALSE;
00659                 }
```

7.3.3.2 GLvoid TinyShaders::TShaderProgram::Shutdown ( ) [inline]

```
00599                 {
00600                     glDeleteProgram(Handle);
00601 
00602                     for (GLuint Iterator = 0; Iterator < GetInstance()->
      Shaders.size(); Iterator++)
00603                     {
00604                         GetInstance()->Shaders[Iterator]->Shutdown();
00605                         delete GetInstance()->Shaders[Iterator];
00606                     }
00607                     Shaders.clear();
00608                     Inputs.clear();
00609                     Outputs.clear();
00610                 }
```

7.3.4 Field Documentation

7.3.4.1 GLboolean TinyShaders::TShaderProgram::Compiled

Whether the shader program has been linked successfully

7.3.4.2 GLuint TinyShaders::TShaderProgram::Handle

The OpenGL handle to the shader program

7.3.4.3 GLuint TinyShaders::TShaderProgram::ID

the ID of the shader program

7.3.4.4 std::vector<const GLchar*> TinyShaders::TShaderProgram::Inputs

the inputs of the shader program as a vector of strings

7.3.4.5 GLuint TinyShaders::TShaderProgram::MaxNumShaders = 5 [static]

The Maximum number of components a shader program can have. It's always 5

7.3.4.6 const GLchar* TinyShaders::TShaderProgram::Name

The name of the shader program

7.3.4.7 std::vector<const GLchar*> TinyShaders::TShaderProgram::Outputs

the outputs of the shader program as a vector of strings

7.3.4.8 std::vector<TShader*> TinyShaders::TShaderProgram::Shaders

the components that the shader program is comprised of as a vector

The documentation for this struct was generated from the following file:

- include/TinyShaders.h

Chapter 8

File Documentation

8.1 include/TinyShaders.h File Reference

```
#include <list>
#include <vector>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Data Structures

- class TinyShaders
- struct TinyShaders::TShader
- struct TinyShaders::TShaderProgram

Macros

- #define TSHADERS_ERROR_NOTINITIALIZED 1
- #define TSHADERS_ERROR_INVALIDSTRING 2
- #define TSHADERS_ERROR_INVALIDSHADERPROGRAMNAME 3
- #define TSHADERS_ERROR_INVALIDSHADERPROGRAMINDEX 4
- #define TSHADERS_ERROR_INVALIDSHADERNAME 5
- #define TSHADERS_ERROR_INVALIDSHADERINDEX 6
- #define TSHADERS_ERROR_INVALIDFILEPATH 7
- #define TSHADERS_ERROR_SHADERPROGRAMNOTFOUND 8
- #define TSHADERS_ERROR_SHADERNOTFOUND 9
- #define TSHADERS_ERROR_INVALIDSHADERTYPE 10
- #define TSHADERS_ERROR_FAILEDSHADERLOAD 11
- #define TSHADERS_ERROR_FAILEDSHADERPROGRAMLINK 12
- #define TSHADERS_ERROR_SHADEREXISTS 13
- #define TSHADERS_ERROR_SHADERPROGRAMEXISTS 14
- #define TSHADERS_ERROR_INVALIDSOURCEFILE 15

8.1.1 Macro Definition Documentation

8.1.1.1 #define TSHADERS_ERROR_FAILEDSHADERLOAD 11

8.1.1.2 #define TSHADERS_ERROR_FAILEDSHADERPROGRAMLINK 12

8.1.1.3 #define TSHADERS_ERROR_INVALIDFILEPATH 7

8.1.1.4 #define TSHADERS_ERROR_INVALIDSHADERINDEX 6

8.1.1.5 #define TSHADERS_ERROR_INVALIDSHADERNAME 5

8.1.1.6 #define TSHADERS_ERROR_INVALIDSHADERPROGRAMINDEX 4

8.1.1.7 #define TSHADERS_ERROR_INVALIDSHADERPROGRAMNAME 3

8.1.1.8 #define TSHADERS_ERROR_INVALIDSHADERTYPE 10

8.1.1.9 #define TSHADERS_ERROR_INVALIDSOURCEFILE 15

8.1.1.10 #define TSHADERS_ERROR_INVALIDSTRING 2

8.1.1.11 #define TSHADERS_ERROR_NOTINITIALIZED 1

8.1.1.12 #define TSHADERS_ERROR_SHADEREXISTS 13

8.1.1.13 #define TSHADERS_ERROR_SHADERNOTFOUND 9

8.1.1.14 #define TSHADERS_ERROR_SHADERPROGRAMEXISTS 14

8.1.1.15 #define TSHADERS_ERROR_SHADERPROGRAMNOTFOUND 8

8.2 File List

Here is a list of all files with brief descriptions: