TinyShaders

0.3

Generated by Doxygen 1.8.7

Wed Nov 4 2015 17:54:06

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::shader_t Struct Reference

Public Member Functions

- shader_t (const GLchar *saderName, GLuint shaderType, const GLchar *shaderFilePath)
- shader_t (const GLchar *shaderName, const GLchar *buffer, GLuint shaderType)
- shader_t (void)
- ~shader_t (void)
- void Compile (const GLchar *source)
- void Shutdown (void)

Data Fields

- const GLchar * name
- const GLchar * filePath
- GLuint handle
- GLuint type
- GLuint iD
- GLboolean isCompiled

3.1.1 Detailed Description

3.1.2 Constructor & Destructor Documentation

3.1.2.1 tinyShaders::shader_t::shader_t (const GLchar * *saderName*, GLuint *shaderType*, const GLchar * *shaderFilePath*) [inline]

```
00607                                                                                                                  :
00608                     name( saderName )
00609                 {
00610                     type = shaderType;
00611                     isCompiled = GL_FALSE;
00612                     filePath = shaderFilePath;
00613                     Compile( GetInstance()->FileToBuffer( shaderFilePath ) );
00614                 }
```

3.1.2.2 tinyShaders::shader_t::shader_t (const GLchar * *shaderName*, const GLchar * *buffer*, GLuint *shaderType*) [inline]

```
00617                     : name( shaderName ), type( shaderType )
```

```
00618                 {
00619                     type = shaderType;
00620                     isCompiled = GL_FALSE;
00621                     Compile( buffer );
00622 
00623                 }
```

3.1.2.3 tinyShaders::shader_t::shader_t (void) [inline]

```
00624 {}
```

3.1.2.4 tinyShaders::shader_t::~shader_t (void) [inline]

```
00625 {}
```

3.1.3 Member Function Documentation

3.1.3.1 void tinyShaders::shader_t::Compile (const GLchar * *source*) [inline]

```
00631                 {
00632                     //if the component hasn't been compiled yet
00633                     if ( !isCompiled )
00634                     {
00635                         GLchar errorLog[512];
00636                         GLint successful;
00637 
00638                         if ( source != nullptr )
00639                         {
00640                             handle = glCreateShader( type );
00641                             glShaderSource( handle, 1, ( const GLchar** )&source, 0 );
00642                             glCompileShader( handle );
00643 
00644                             glGetShaderiv( handle, GL_COMPILE_STATUS, &successful );
00645                             glGetShaderInfoLog( handle, sizeof( errorLog ), 0, errorLog );
00646 
00647                             if ( successful != GL_TRUE )
00648                             {
00649                                 TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_FAILED_SHADER_LOAD,
      GetInstance()->ShaderTypeToString( type ) );
00650                                 printf( "%s\n", errorLog );
00651                             }
00652 
00653                             else
00654                             {
00655                                 isCompiled = GL_TRUE;
00656                                 GetInstance()->shaders.push_back( this );
00657                                 iD = GetInstance()->shaders.size() - 1;
00658                             }
00659                         }
00660                         else
00661                         {
00662                             TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_INVALID_SOURCE_FILE );
00663                         }
00664                     }
00665                     else
00666                     {
00667                         //either the file name doesn't exist or the component has already been loaded
00668                         TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_INVALID_FILE_PATH, filePath );
00669                     }
00670                 }
```

3.1.3.2 void tinyShaders::shader_t::Shutdown (void) [inline]

```
00676                 {
00677                     glDeleteShader( handle );
00678                     isCompiled = GL_FALSE;
00679                 }
```

3.1.4 Field Documentation

3.1.4.1 const GLchar* tinyShaders::shader_t::filePath

The FilePath of the component

3.1.4.2 GLuint tinyShaders::shader_t::handle

The handle to the shader in OpenGL

3.1.4.3 GLuint tinyShaders::shader_t::iD

The ID of the shader

3.1.4.4 GLboolean tinyShaders::shader_t::isCompiled

Whether the shader has been compiled

3.1.4.5 const GLchar* tinyShaders::shader_t::name

The name of the shader component

3.1.4.6 GLuint tinyShaders::shader_t::type

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

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

- TinyShaders.h

3.2 tinyShaders::shaderProgram_t Struct Reference

Public Member Functions

- shaderProgram_t (void)
- shaderProgram_t (const GLchar *shaderName, std::vector< const GLchar * > programInputs, std::vector< const GLchar * > programOutputs, std::vector< shader_t * > programShaders)
- shaderProgram_t (const GLchar *shaderName)
- ~shaderProgram_t (void)
- void Shutdown (void)
- GLboolean Compile (void)

Data Fields

- const GLchar * name
- GLuint handle
- GLuint iD
- GLboolean compiled
- std::vector< const GLchar * > inputs
- std::vector< const GLchar * > outputs
- std::vector< shader_t * > shaders

Static Public Attributes

- static const GLuint maxNumShaders = 5

3.2.1 Detailed Description

3.2.2 Constructor & Destructor Documentation

3.2.2.1 tinyShaders::shaderProgram_t::shaderProgram_t (void) [inline]

```
00698                 {
00699                     iD = 0;
00700                 };
```

3.2.2.2 tinyShaders::shaderProgram_t::shaderProgram_t (const GLchar * *shaderName*, std::vector< const GLchar * > *programInputs*, std::vector< const GLchar * > *programOutputs*, std::vector< shader_t * > *programShaders*) [inline]

```
00708                                                           :
00709                     name( shaderName ), inputs( programInputs ),
00710                     outputs( programOutputs ), shaders( programShaders )
00711                 {
00712                     compiled = GL_FALSE;
00713                     Compile();
00714                     //get number of uniform blocks
00715                     if ( GetInstance()->shaderBlocksEvent != nullptr )
00716                     {
00717                         GetInstance()->shaderBlocksEvent(
      handle );
00718                     }
00719                 };
```

3.2.2.3 tinyShaders::shaderProgram_t::shaderProgram_t (const GLchar * *shaderName*) [inline]

```
00724                                                           : name( shaderName )
00725                 {
00726                     compiled = GL_FALSE;
00727                 };
```

3.2.2.4 tinyShaders::shaderProgram_t::~shaderProgram_t (void) [inline]

```
00729 {}
```

3.2.3 Member Function Documentation

3.2.3.1 GLboolean tinyShaders::shaderProgram_t::Compile (void) [inline]

```
00751                 {
00752                     handle = glCreateProgram();
00753                     GLchar errorLog[512];
00754                     GLint successful = GL_FALSE;
00755                     if ( !compiled )
00756                     {
00757                         for ( GLuint iterator = 0; iterator <shaders.size(); iterator++ )
00758                         {
00759                             if ( shaders[iterator] != nullptr )
00760                             {
00761                                 glAttachShader( handle, shaders[iterator]->
      handle );
00762                             }
00763                         }
00764
00765                         // specify vertex input attributes
00766                         for ( GLuint i = 0; i <inputs.size(); ++i )
```

```
00767                           {
00768                               glBindAttribLocation( handle, i, inputs[i] );
00769                           }
00770 
00771                           // specify pixel shader outputs
00772                           for ( GLuint i = 0; i <outputs.size(); ++i )
00773                           {
00774                               glBindFragDataLocation( handle, i, outputs[i] );
00775                           }
00776 
00777                           glLinkProgram( handle );
00778                           glGetProgramiv( handle, GL_LINK_STATUS, &successful );
00779                           glGetProgramInfoLog( handle, sizeof( errorLog ), 0, errorLog );
00780 
00781                           if ( !successful )
00782                           {
00783                               TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_FAILED_SHADER_PROGRAM_LINK,
      name );
00784                               printf( "%s\n", errorLog );
00785                               return GL_FALSE;
00786                           }
00787                           //if a shader successfully compiles then it will add itself to storage
00788                           compiled = GL_TRUE;
00789                           GetInstance()->shaderPrograms.push_back( this );
00790                           iD = GetInstance()->shaderPrograms.size() - 1;
00791                           return GL_TRUE;
00792                       }
00793                       TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_SHADER_PROGRAM_ALREADY_EXISTS,
      name );
00794                       return GL_FALSE;
00795                   }
```

3.2.3.2 void tinyShaders::shaderProgram_t::Shutdown (void) [inline]

```
00735                   {
00736                       glDeleteProgram( handle );
00737 
00738                       for ( GLuint iterator = 0; iterator < GetInstance()->
      shaders.size(); iterator++ )
00739                       {
00740                           GetInstance()->shaders[iterator]->Shutdown();
00741                       }
00742                       shaders.clear();
00743                       inputs.clear();
00744                       outputs.clear();
00745                   }
```

3.2.4 Field Documentation

3.2.4.1 GLboolean tinyShaders::shaderProgram_t::compiled

Whether the shader program has been linked successfully

3.2.4.2 GLuint tinyShaders::shaderProgram_t::handle

The OpenGL handle to the shader program

3.2.4.3 GLuint tinyShaders::shaderProgram_t::iD

The ID of the shader program

3.2.4.4 std::vector< const GLchar* > tinyShaders::shaderProgram_t::inputs

The inputs of the shader program as a vector of strings

3.2.4.5 const GLuint tinyShaders::shaderProgram_t::maxNumShaders = 5 [static]

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

3.2.4.6 const GLchar* tinyShaders::shaderProgram_t::name

The name of the shader program

3.2.4.7 std::vector< const GLchar* > tinyShaders::shaderProgram_t::outputs

The outputs of the shader program as a vector of strings

3.2.4.8 std::vector< shader_t* > tinyShaders::shaderProgram_t::shaders

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

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

- TinyShaders.h

3.3 tinyShaders Class Reference

`#include <TinyShaders.h>`

Data Structures

- struct shader_t
- struct shaderProgram_t

Public Member Functions

- tinyShaders (void)
- ~tinyShaders (void)

Static Public Member Functions

- static void Shutdown (void)
- static shaderProgram_t * GetShaderProgramByName (const GLchar *programName)
- static shaderProgram_t * GetShaderProgramByIndex (GLuint programIndex)
- static shader_t * GetShaderByName (const GLchar *shaderName)
- static shader_t * 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 *fragmentShader↩ Name, const GLchar *geometryShaderName, const GLchar *tessContShaderName, const GLchar *tess↩ EvalShaderName)
- static GLboolean ShaderProgramExists (const GLchar *shaderName)
- static GLboolean ShaderExists (const GLchar *shaderName)
- static void LoadShaderFromBuffer (const char *name, const GLchar *buffer, GLuint shaderType)
- static GLboolean SetShaderBlockParseEvent (parseBlocks_t shaderBlockParse)

Private Member Functions

- GLchar * FileToBuffer (const GLchar *path) const
- GLuint StringToShaderType (const GLchar *typeString) const
- const GLchar * ShaderTypeToString (GLuint shaderType) const

Static Private Member Functions

- static tinyShaders * GetInstance (void)

Private Attributes

- std::vector< shaderProgram_t * > shaderPrograms
- std::vector< shader_t * > shaders

Static Private Attributes

- static GLboolean isInitialized = GL_FALSE
- static tinyShaders * instance = nullptr
- static parseBlocks_t shaderBlocksEvent = nullptr

3.3.1 Detailed Description

3.3.2 Constructor & Destructor Documentation

3.3.2.1 tinyShaders::tinyShaders (void) [inline]

```
00162 {}
```

3.3.2.2 tinyShaders::~tinyShaders (void) [inline]

```
00163 {}
```

3.3.3 Member Function Documentation

3.3.3.1 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]

```
00504         {
00505             if ( tinyShaders::isInitialized )
00506             {
00507                 std::vector< shader_t* > shaders;
00508                 shaders.push_back( GetShaderByName( vertexShaderName ) );
00509                 shaders.push_back( GetShaderByName( fragmentShaderName ) );
00510                 shaders.push_back( GetShaderByName( geometryShaderName ) );
00511                 shaders.push_back( GetShaderByName( tessContShaderName ) );
00512                 shaders.push_back( GetShaderByName( tessEvalShaderName ) );
00513 
00514                 shaderProgram_t* newShaderProgram = new shaderProgram_t( shaderName, inputs, outputs,
      shaders );
00515                 delete newShaderProgram;
00516             }
00517             TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_NOT_INITIALIZED );
00518         }
```

3.3.3.2 GLchar* tinyShaders::FileToBuffer (const GLchar * *path*) const [inline], [private]

```
00826         {
00827             FILE* file = fopen( path, "rt" );
00828 
00829             if ( file == nullptr )
00830             {
00831                 TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_FILE_PATH, path );
00832                 //printf( "Error: cannot open file %s for reading \n", Path );
00833                 return nullptr;
00834             }
00835 
00836             //get total byte in given file
00837             fseek( file, 0, SEEK_END );
00838             GLuint FileLength = ftell( file );
00839             fseek( file, 0, SEEK_SET );
00840 
00841             //allocate a file buffer and read the contents of the file
00842             GLchar* buffer = new GLchar[FileLength + 1];
00843             memset( buffer, 0, FileLength + 1 );
00844             fread( buffer, sizeof( GLchar ), FileLength, file );
00845 
00846             fclose( file );
00847             return buffer;
00848         }
```

3.3.3.3 static tinyShaders* tinyShaders::GetInstance (void) [inline], [static], [private]

```
00811         {
00812             if ( tinyShaders::isInitialized )
00813             {
00814                 return tinyShaders::instance;
00815             }
00816 
00817             tinyShaders::isInitialized = GL_TRUE;
00818             tinyShaders::instance = new tinyShaders();
00819             return tinyShaders::instance;
00820         }
```

3.3.3.4 static shader_t* tinyShaders::GetShaderByIndex (GLuint *shaderIndex*) [inline], [static]

```
00265         {
00266             if ( tinyShaders::isInitialized )
00267             {
00268                 if ( shaderIndex <= GetInstance()->shaders.size() - 1 )
00269                 {
00270                     return GetInstance()->shaders[shaderIndex];
00271                 }
00272                 TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_SHADER_INDEX );
00273                 return nullptr;
00274             }
00275             TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_NOT_INITIALIZED );
00276             return nullptr;
00277         }
```

3.3.3.5 static shader_t* tinyShaders::GetShaderByName (const GLchar * *shaderName*) [inline], [static]

```
00239         {
00240             if ( tinyShaders::isInitialized )
00241             {
00242                 if ( shaderName != nullptr )
00243                 {
00244                     for ( GLuint iterator = 0; iterator < GetInstance()->
    shaders.size(); iterator++ )
00245                     {
00246                         if ( !strcmp( GetInstance()->shaders[iterator]->name, shaderName
    ) )
00247                         {
00248                             return GetInstance()->shaders[iterator];
00249                         }
00250                     }
00251                     TinyShaders_PrintErrorMessage(
```

```
    TINYSHADERS_ERROR_SHADER_NOT_FOUND );
00252                     return nullptr;
00253                 }
00254                 TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_SHADER_NAME );
00255                 return nullptr;
00256             }
00257             TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_NOT_INITIALIZED );
00258             return nullptr;
00259         }
```

3.3.3.6 static shaderProgram_t* tinyShaders::GetShaderProgramByIndex (GLuint *programIndex*) [inline], [static]

```
00221         {
00222             if ( tinyShaders::isInitialized )
00223             {
00224                 if ( programIndex <= GetInstance()->shaderPrograms.size() - 1 )
00225                 {
00226                     return GetInstance()->shaderPrograms[programIndex];
00227                 }
00228                 TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_SHADER_PROGRAM_INDEX );
00229                 return nullptr;
00230             }
00231             TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_NOT_INITIALIZED );
00232             return nullptr;
00233         }
```

3.3.3.7 static shaderProgram_t* tinyShaders::GetShaderProgramByName (const GLchar * *programName*) [inline], [static]

```
00196         {
00197             if ( tinyShaders::isInitialized )
00198             {
00199                 if ( programName != nullptr )
00200                 {
00201                     for ( GLuint iterator = 0; iterator < GetInstance()->
    shaderPrograms.size(); iterator++ )
00202                     {
00203                         if ( !strcmp( GetInstance()->shaderPrograms[iterator]->
    name, programName ) )
00204                         {
00205                             return GetInstance()->shaderPrograms[iterator];
00206                         }
00207                     }
00208                     return nullptr;
00209                 }
00210                 TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_SHADER_PROGRAM_NOT_FOUND );
00211                 return nullptr;
00212             }
00213             TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_NOT_INITIALIZED );
00214             return nullptr;
00215         }
```

3.3.3.8 static void tinyShaders::LoadShader (const GLchar * *name*, const GLchar * *shaderFile*, GLuint *shaderType*) [inline], [static]

```
00283         {
00284             if ( tinyShaders::isInitialized )
00285             {
00286                 if ( name != nullptr )
00287                 {
00288                     if ( shaderType <= 5 )
00289                     {
00290                         shader_t* newShader = new shader_t( name, shaderType, shaderFile );
00291                     }
00292                     TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_SHADER_TYPE,
    GetInstance()->ShaderTypeToString( shaderType ) );
```

```
00293                 }
00294                 TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_STRING );
00295             }
00296             TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_NOT_INITIALIZED );
00297         }
```

3.3.3.9 static void tinyShaders::LoadShaderFromBuffer (const char * *name*, const GLchar * *buffer*, GLuint *shaderType*) [inline], [static]

```
00569         {
00570             if( tinyShaders::isInitialized )
00571             {
00572                 if( buffer != nullptr )
00573                 {
00574                     if( name != nullptr )
00575                     {
00576                         if( !ShaderExists( name ) )
00577                         {
00578                             shader_t* newShader = new shader_t( name, buffer, shaderType );
00579                             delete newShader;
00580                         }
00581                         TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_SHADER_NOT_FOUND );
00582                     }
00583                     TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_SHADER_NAME );
00584                 }
00585                 TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_STRING );
00586             }
00587             TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_NOT_INITIALIZED );
00588         }
```

3.3.3.10 static void tinyShaders::LoadShaderProgramsFromConfigFile (const GLchar * *configFile*) [inline], [static]

```
00303         {
00304             if ( GetInstance()->isInitialized )
00305             {
00306                 FILE* pConfigFile = fopen( configFile, "r" );
00307                 GLuint numInputs = 0;
00308                 GLuint numOutputs = 0;
00309                 GLuint numPrograms = 0;
00310                 GLuint numShaders = 0;
00311                 GLuint iterator = 0;
00312
00313                 std::vector< const GLchar* > inputs, outputs, paths, names;
00314                 std::vector< shader_t* > shaders;
00315                 if ( pConfigFile )
00316                 {
00317                     //get the total number of shader programs
00318                     fscanf( pConfigFile, "%i\n", &numPrograms );
00319
00320                     for ( GLuint programIter = 0;
00321                         programIter <numPrograms;
00322                         programIter++, paths.clear(), inputs.clear(), outputs.clear(), names.clear(),
    shaders.clear() )
00323                     {
00324                         //get the name of the shader program
00325                         GLchar* programName = new GLchar[255];
00326                         fscanf( pConfigFile, "%s\n", programName );
00327                         printf( "%s\n", programName );
00328
00329                         //this is an anti-trolling measure. If a shader with the same name already exists
     the don't bother making a new one.
00330                         if ( !GetInstance()->ShaderProgramExists( programName
    ) )
00331                         {
00332                             //get the number of shader inputs
00333                             fscanf( pConfigFile, "%i\n", &numInputs );
00334
00335                             //get all inputs
00336                             for ( iterator = 0; iterator <numInputs; iterator++ )
00337                             {
00338                                 GLchar* input = new GLchar[255];
00339                                 fscanf( pConfigFile, "%s\n", input );
```

```
00340                                 inputs.push_back( input );
00341                             }
00342 
00343                             //get the number of shader outputs
00344                             fscanf( pConfigFile, "%i\n", &numOutputs );
00345 
00346                             //get all outputs
00347                             for ( iterator = 0; iterator <numOutputs; iterator++ )
00348                             {
00349                                 GLchar* output = new GLchar[255];
00350                                 fscanf( pConfigFile, "%s\n", output );
00351                                 outputs.push_back( output );
00352                             }
00353 
00354                             //get number of shaders
00355                             fscanf( pConfigFile, "%i\n", &numShaders );
00356                             printf( "%i\n", numShaders );
00357 
00358                             for( GLuint iterator = 0; iterator <numShaders; iterator++ )
00359                             {
00360                                 GLchar* shaderName = new GLchar[255];
00361                                 GLchar* shaderPath = new GLchar[255];
00362                                 GLchar* shaderType = new GLchar[255];
00363 
00364                                 //get shader name
00365                                 fscanf( pConfigFile, "%s\n", shaderName );
00366                                 printf( "%s\n", shaderName );
00367 
00368                                 //if the shader hasn't been loaded already then make a new one
00369                                 if( !ShaderExists( shaderName ) )
00370                                 {
00371                                     //get type
00372                                     fscanf( pConfigFile, "%s\n", shaderType );
00373                                     printf( "%s\n", shaderType );
00374                                     //get file path
00375                                     fscanf( pConfigFile, "%s\n", shaderPath );
00376                                     printf( "%s\n", shaderPath );
00377 
00378                                     shaders.push_back( new shader_t( shaderName,
      GetInstance()->StringToShaderType( ( const char* )shaderType ), shaderPath ) )
      ;
00379                                 }
00380 
00381                                 else
00382                                 {
00383                                     //tell scanf to skip a couple lines
00384                                     fscanf( pConfigFile, "%*[^\n]\n %*[^\n]\n", NULL );
00385                                     //if shader already exists then add an existing one from storage, it
       should already be compiled
00386                                     shaders.push_back( GetShaderByName( shaderName ) );
00387                                 }
00388                             }
00389 
00390                             shaderProgram_t* newShaderProgram = new shaderProgram_t( programName, inputs,
      outputs, shaders );
00391                             //get shader block names
00392                         }
00393                     }
00394                     fclose( pConfigFile );
00395                 }
00396                 else
00397                 {
00398                     TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_INVALID_FILE_PATH );
00399                 }
00400             }
00401             else
00402             {
00403                 TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_NOT_INITIALIZED );
00404             }
00405         }
```

3.3.3.11 static void tinyShaders::LoadShadersFromConfigFile (const GLchar * *configFile*) [inline], [static]

```
00408         {
00409             if( tinyShaders::isInitialized )
00410             {
00411                 FILE* pConfigFile = fopen( configFile, "r+" );
00412                 GLuint numShaders = 0;
00413 
00414                 if( pConfigFile )
00415                 {
00416                     //get the number of shaders to load
```

```
00417                   fscanf( pConfigFile, "%i\n", &numShaders );
00418               GLchar* shaderName;
00419                   GLchar* shaderType;
00420                   GLchar* shaderPath;
00421
00422                   GLchar empty[255];
00423
00424                   for( GLuint iterator = 0; iterator <numShaders;
00425                           iterator++, fscanf( pConfigFile, "\n\n" ) )
00426                   {
00427                       shaderName = empty;
00428                       fscanf( pConfigFile, "%s\n", shaderName );
00429
00430                       if( !GetInstance()->ShaderExists( shaderName ) )
00431                       {
00432                           shaderType = empty;
00433                           fscanf( pConfigFile, "%s\n", shaderType );
00434
00435                           shaderPath = empty;
00436                           fscanf( pConfigFile, "%s\n", shaderPath );
00437
00438                           shader_t* newShader = new shader_t( shaderName,
    GetInstance()->StringToShaderType( shaderType ), shaderPath );
00439                           delete newShader;
00440                       }
00441                   }
00442               }
00443           }
00444       }
```

3.3.3.12 static void tinyShaders::SaveShaderProgramsToConfigFile (const GLchar * *fileName*) [inline], [static]

```
00447       {
00448           //write total amount of shaders
00449           FILE* pConfigFile = fopen( fileName, "w+" );
00450
00451           fprintf( pConfigFile, "%i\n\n", ( GLint )GetInstance()->
    shaderPrograms.size() );
00452
00453           for( GLuint programIter = 0; programIter < GetInstance()->
    shaderPrograms.size(); programIter++ )
00454           {
00455               //write program name
00456               fprintf( pConfigFile, "%s\n", GetInstance()->
    shaderPrograms[programIter]->name );
00457
00458               //write number of inputs
00459               fprintf( pConfigFile, "%i\n", ( GLint )GetInstance()->
    shaderPrograms[programIter]->inputs.size() );
00460
00461               //write inputs
00462               for( GLuint inputIter = 0; inputIter < GetInstance()->
    shaderPrograms[programIter]->inputs.size(); inputIter++ )
00463               {
00464                   fprintf( pConfigFile, "%s\n", GetInstance()->
    shaderPrograms[programIter]->inputs[inputIter] );
00465               }
00466
00467               fprintf( pConfigFile, "%i\n", ( GLint )GetInstance()->
    shaderPrograms[programIter]->outputs.size() );
00468
00469               //write outputs
00470               for( GLuint outputIter = 0; outputIter < GetInstance()->
    shaderPrograms[programIter]->outputs.size(); outputIter++ )
00471               {
00472                   fprintf( pConfigFile, "%s\n", GetInstance()->
    shaderPrograms[programIter]->outputs[outputIter] );
00473               }
00474
00475               //write number of shaders
00476               fprintf( pConfigFile, "%i\n", ( GLint )GetInstance()->
    shaderPrograms[programIter]->shaders.size() );
00477
00478               for( GLuint shaderIter = 0; shaderIter < GetInstance()->
    shaderPrograms[programIter]->shaders.size(); shaderIter++ )
00479               {
00480                   //write shader name
00481                   fprintf( pConfigFile, "%s\n", GetInstance()->
    shaderPrograms[programIter]->shaders[shaderIter]->name );
00482
00483                   //write shader type
00484                   fprintf( pConfigFile, "%s\n", GetInstance()->
    ShaderTypeToString( GetInstance()->shaderPrograms[programIter]->
    shaders[shaderIter]->type ) );
```

```
00485
00486                     //write shader file path
00487                     fprintf( pConfigFile, "%s\n", GetInstance()->
      shaderPrograms[programIter]->shaders[shaderIter]->filePath );
00488                 }
00489             }
00490             fclose( pConfigFile );
00491         }
```

3.3.3.13 static GLboolean tinyShaders::SetShaderBlockParseEvent (parseBlocks_t *shaderBlockParse*) [inline], [static]

```
00591         {
00592             if ( GetInstance()->isInitialized )
00593             {
00594                 GetInstance()->shaderBlocksEvent = shaderBlockParse;
00595                 return GL_TRUE;
00596             }
00597             return GL_FALSE;
00598         }
```

3.3.3.14 static GLboolean tinyShaders::ShaderExists (const GLchar * *shaderName*) [inline], [static]

```
00548         {
00549             if ( shaderName != nullptr )
00550             {
00551                 if ( !GetInstance()->shaders.empty() )
00552                 {
00553                     for ( GLuint iterator = 0; iterator < GetInstance()->
      shaders.size(); iterator++ )
00554                     {
00555                         if ( GetInstance()->shaders[iterator] != nullptr &&
00556                             !strcmp( shaderName, GetInstance()->
      shaders[iterator]->name ) )
00557                         {
00558                             return GL_TRUE;
00559                         }
00560                     }
00561                     return GL_FALSE;
00562                 }
00563                 return GL_FALSE;
00564             }
00565             return GL_FALSE;
00566         }
```

3.3.3.15 static GLboolean tinyShaders::ShaderProgramExists (const GLchar * *shaderName*) [inline], [static]

```
00524         {
00525             if ( shaderName != nullptr )
00526             {
00527                 if ( !GetInstance()->shaderPrograms.empty() )
00528                 {
00529                     for ( GLuint iterator = 0; iterator < GetInstance()->
      shaderPrograms.size(); iterator++ )
00530                     {
00531                         if ( GetInstance()->shaderPrograms[iterator] != nullptr &&
00532                             !strcmp( shaderName, GetInstance()->
      shaderPrograms[iterator]->name ) )
00533                         {
00534                             return GL_TRUE;
00535                         }
00536                     }
00537                     return GL_FALSE;
00538                 }
00539                 return GL_FALSE;
00540             }
00541             return GL_FALSE;
00542         }
```

3.3.3.16 const GLchar* tinyShaders::ShaderTypeToString (GLuint *shaderType*) const [inline], [private]

```
00892         {
```

```
00893             switch ( shaderType )
00894             {
00895                 case GL_VERTEX_SHADER:
00896                 {
00897                     return "Vertex";
00898                 }
00899
00900                 case GL_FRAGMENT_SHADER:
00901                 {
00902                     return "Fragment";
00903                 }
00904
00905                 case GL_GEOMETRY_SHADER:
00906                 {
00907                     return "Geometry";
00908                 }
00909
00910                 case GL_TESS_CONTROL_SHADER:
00911                 {
00912                     return "Tessellation Control";
00913                 }
00914
00915                 case GL_TESS_EVALUATION_SHADER:
00916                 {
00917                     return "Tessellation Evaluation";
00918                 }
00919
00920                 default:
00921                 {
00922                     return NULL;
00923                 }
00924             }
00925
00926             return nullptr;
00927         }
```

3.3.3.17 static void tinyShaders::Shutdown (void) [inline], [static]

```
00170         {
00171             if ( tinyShaders::isInitialized )
00172             {
00173                 for ( GLuint iterator = 0; iterator < GetInstance()->
      shaders.size(); iterator++ )
00174                 {
00175                     GetInstance()->shaders[iterator]->Shutdown();
00176                      delete GetInstance()->shaders[iterator];
00177                 }
00178
00179                 for ( GLuint iterator = 0; iterator < GetInstance()->
      shaderPrograms.size(); iterator++ )
00180                 {
00181                     GetInstance()->shaderPrograms[iterator]->Shutdown();
00182                     delete GetInstance()->shaderPrograms[iterator];
00183                 }
00184
00185                 GetInstance()->shaderPrograms.clear();
00186                 GetInstance()->shaders.clear();
00187
00188                 delete instance;
00189             }
00190         }
```

3.3.3.18 GLuint tinyShaders::StringToShaderType (const GLchar * *typeString*) const [inline], [private]

```
00854         {
00855             if( typeString != nullptr )
00856             {
00857                 if ( !strcmp( typeString, "Vertex" ) )
00858                 {
00859                     return GL_VERTEX_SHADER;
00860                 }
00861
00862                 if ( !strcmp( typeString, "Fragment" ) )
00863                 {
00864                     return GL_FRAGMENT_SHADER;
00865                 }
00866
00867                 if ( !strcmp( typeString, "Geometry" ) )
00868                 {
00869                     return GL_GEOMETRY_SHADER;
```

```
00870                 }
00871
00872                 if ( !strcmp( typeString, "Tessellation Control" ) )
00873                 {
00874                     return GL_TESS_CONTROL_SHADER;
00875                 }
00876
00877                 if ( !strcmp( typeString, "Tessellation Evaluation" ) )
00878                 {
00879                     return GL_TESS_EVALUATION_SHADER;
00880                 }
00881
00882                 return GL_FALSE;
00883             }
00884             TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_INVALID_STRING );
00885             return GL_FALSE;
00886         }
```

3.3.4 Field Documentation

3.3.4.1 tinyShaders * tinyShaders::instance = nullptr [static], [private]

A static instance of the TinyShaders API

3.3.4.2 GLboolean tinyShaders::isInitialized = GL_FALSE [static], [private]

Whether TinyShadershas ban initialized

3.3.4.3 parseBlocks_t tinyShaders::shaderBlocksEvent = nullptr [static], [private]

3.3.4.4 std::vector< shaderProgram_t* > tinyShaders::shaderPrograms [private]

All loaded shader programs

3.3.4.5 std::vector< shader_t* > tinyShaders::shaders [private]

All loaded shaders

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

- TinyShaders.h

Chapter 4

File Documentation

4.1 TinyShaders.h File Reference

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

Data Structures

- class tinyShaders
- struct tinyShaders::shader_t
- struct tinyShaders::shaderProgram_t

Macros

- #define TINYSHADERS_ERROR_NOT_INITIALIZED 1
- #define TINYSHADERS_ERROR_INVALID_STRING 2
- #define TINYSHADERS_ERROR_INVALID_SHADER_PROGRAM_NAME 3
- #define TINYSHADERS_ERROR_INVALID_SHADER_PROGRAM_INDEX 4
- #define TINYSHADERS_ERROR_INVALID_SHADER_NAME 5
- #define TINYSHADERS_ERROR_INVALID_SHADER_INDEX 6
- #define TINYSHADERS_ERROR_INVALID_FILE_PATH 7
- #define TINYSHADERS_ERROR_SHADER_PROGRAM_NOT_FOUND 8
- #define TINYSHADERS_ERROR_SHADER_NOT_FOUND 9
- #define TINYSHADERS_ERROR_INVALID_SHADER_TYPE 10
- #define TINYSHADERS_ERROR_FAILED_SHADER_LOAD 11
- #define TINYSHADERS_ERROR_FAILED_SHADER_PROGRAM_LINK 12
- #define TINYSHADERS_ERROR_SHADER_ALREADY_EXISTS 13
- #define TINYSHADERS_ERROR_SHADER_PROGRAM_ALREADY_EXISTS 14
- #define TINYSHADERS_ERROR_INVALID_SOURCE_FILE 15

Typedefs

- typedef void(∗ parseBlocks_t)(GLuint programHandle)

Functions

- static void TinyShaders_PrintErrorMessage (GLuint errorNumber, const GLchar *errorMessage=nullptr)

4.1.1 Macro Definition Documentation

4.1.1.1 #define TINYSHADERS_ERROR_FAILED_SHADER_LOAD 11

4.1.1.2 #define TINYSHADERS_ERROR_FAILED_SHADER_PROGRAM_LINK 12

4.1.1.3 #define TINYSHADERS_ERROR_INVALID_FILE_PATH 7

4.1.1.4 #define TINYSHADERS_ERROR_INVALID_SHADER_INDEX 6

4.1.1.5 #define TINYSHADERS_ERROR_INVALID_SHADER_NAME 5

4.1.1.6 #define TINYSHADERS_ERROR_INVALID_SHADER_PROGRAM_INDEX 4

4.1.1.7 #define TINYSHADERS_ERROR_INVALID_SHADER_PROGRAM_NAME 3

4.1.1.8 #define TINYSHADERS_ERROR_INVALID_SHADER_TYPE 10

4.1.1.9 #define TINYSHADERS_ERROR_INVALID_SOURCE_FILE 15

4.1.1.10 #define TINYSHADERS_ERROR_INVALID_STRING 2

4.1.1.11 #define TINYSHADERS_ERROR_NOT_INITIALIZED 1

4.1.1.12 #define TINYSHADERS_ERROR_SHADER_ALREADY_EXISTS 13

4.1.1.13 #define TINYSHADERS_ERROR_SHADER_NOT_FOUND 9

4.1.1.14 #define TINYSHADERS_ERROR_SHADER_PROGRAM_ALREADY_EXISTS 14

4.1.1.15 #define TINYSHADERS_ERROR_SHADER_PROGRAM_NOT_FOUND 8

4.1.2 Typedef Documentation

4.1.2.1 typedef void(* parseBlocks_t)(GLuint programHandle)

a callback that can gather all the info about the uniform blocks that are in a shader program

4.1.3 Function Documentation

4.1.3.1 static void TinyShaders_PrintErrorMessage (GLuint *errorNumber,* const GLchar * *errorMessage* = `nullptr`) `[inline], [static]`

```
00049 {
00050     switch (errorNumber)
00051     {
00052         case TINYSHADERS_ERROR_NOT_INITIALIZED:
00053         {
00054             printf("Error: TinyShaders must first be initialized \n");
00055             break;
00056         }
00057 
00058         case TINYSHADERS_ERROR_INVALID_STRING:
00059         {
00060             printf("Error: given string is invalid \n");
```

```
00061             break;
00062         }
00063
00064         case TINYSHADERS_ERROR_INVALID_SHADER_PROGRAM_NAME:
00065         {
00066             printf("Error: given shader name is invalid \n");
00067             break;
00068         }
00069
00070         case TINYSHADERS_ERROR_INVALID_SHADER_PROGRAM_INDEX:
00071         {
00072             printf("Error: given shader index is invalid \n");
00073             break;
00074         }
00075
00076         case TINYSHADERS_ERROR_INVALID_SHADER_NAME:
00077         {
00078             printf("Error: given shader component name is invalid \n");
00079             break;
00080         }
00081
00082         case TINYSHADERS_ERROR_INVALID_SHADER_INDEX:
00083         {
00084             printf("Error: given shader component index is invalid \n");
00085             break;
00086         }
00087
00088         case TINYSHADERS_ERROR_INVALID_FILE_PATH:
00089         {
00090             printf("Error: given file path is invalid %s \n", errorMessage);
00091             break;
00092         }
00093
00094         case TINYSHADERS_ERROR_SHADER_PROGRAM_NOT_FOUND:
00095         {
00096             printf("Error: shader with given name %s was not found \n", errorMessage);
00097             break;
00098         }
00099
00100         case TINYSHADERS_ERROR_SHADER_NOT_FOUND:
00101         {
00102             printf("Error: shader component with given name %s was not found \n", errorMessage);
00103             break;
00104         }
00105
00106         case TINYSHADERS_ERROR_INVALID_SHADER_TYPE:
00107         {
00108             printf("Error: invalid shader type given \n");
00109             break;
00110         }
00111
00112         case TINYSHADERS_ERROR_FAILED_SHADER_LOAD:
00113         {
00114             printf("Error: failed to compile %s shader component \n", errorMessage);
00115             break;
00116         }
00117
00118         case TINYSHADERS_ERROR_FAILED_SHADER_PROGRAM_LINK:
00119         {
00120             if (errorMessage != nullptr)
00121             {
00122                 printf("Error: failed to link program %s \n", errorMessage);
00123             }
00124             break;
00125         }
00126
00127         case TINYSHADERS_ERROR_SHADER_ALREADY_EXISTS:
00128         {
00129             printf("Error: shader component with this name %s already exists \n", errorMessage);
00130             break;
00131         }
00132
00133         case TINYSHADERS_ERROR_SHADER_PROGRAM_ALREADY_EXISTS
    :
00134         {
00135             if (errorMessage != nullptr)
00136             {
00137                 printf("Error: shader with this name %s already exists \n", errorMessage);
00138                 break;
00139             }
00140         }
00141
00142         case TINYSHADERS_ERROR_INVALID_SOURCE_FILE:
00143         {
00144             printf("Given Source file is invalid");
00145             break;
00146         }
```

```
00147
00148         default:
00149         {
00150             break;
00151         }
00152     }
00153 }
```

[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 17:54:06

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::shader_t Struct Reference

Public Member Functions

- shader_t (const GLchar *saderName, GLuint shaderType, const GLchar *shaderFilePath)
- shader_t (const GLchar *shaderName, const GLchar *buffer, GLuint shaderType)
- shader_t (void)
- ~shader_t (void)
- void Compile (const GLchar *source)
- void Shutdown (void)

Data Fields

- const GLchar * name
- const GLchar * filePath
- GLuint handle
- GLuint type
- GLuint iD
- GLboolean isCompiled

7.1.1 Detailed Description

7.1.2 Constructor & Destructor Documentation

7.1.2.1 tinyShaders::shader_t::shader_t (const GLchar * *saderName*, GLuint *shaderType*, const GLchar * *shaderFilePath*) [inline]

```
00607                                                                                                                  :
00608                     name( saderName )
00609                 {
00610                     type = shaderType;
00611                     isCompiled = GL_FALSE;
00612                     filePath = shaderFilePath;
00613                     Compile( GetInstance()->FileToBuffer( shaderFilePath ) );
00614                 }
```

7.1.2.2 tinyShaders::shader_t::shader_t (const GLchar * *shaderName*, const GLchar * *buffer*, GLuint *shaderType*) [inline]

```
00617                     : name( shaderName ), type( shaderType )
```

```
00618                 {
00619                     type = shaderType;
00620                     isCompiled = GL_FALSE;
00621                     Compile( buffer );
00622 
00623                 }
```

7.1.2.3 tinyShaders::shader_t::shader_t (void) [inline]

```
00624 {}
```

7.1.2.4 tinyShaders::shader_t::~shader_t (void) [inline]

```
00625 {}
```

7.1.3 Member Function Documentation

7.1.3.1 void tinyShaders::shader_t::Compile (const GLchar * *source*) [inline]

```
00631                 {
00632                     //if the component hasn't been compiled yet
00633                     if ( !isCompiled )
00634                     {
00635                         GLchar errorLog[512];
00636                         GLint successful;
00637 
00638                         if ( source != nullptr )
00639                         {
00640                             handle = glCreateShader( type );
00641                             glShaderSource( handle, 1, ( const GLchar** )&source, 0 );
00642                             glCompileShader( handle );
00643 
00644                             glGetShaderiv( handle, GL_COMPILE_STATUS, &successful );
00645                             glGetShaderInfoLog( handle, sizeof( errorLog ), 0, errorLog );
00646 
00647                             if ( successful != GL_TRUE )
00648                             {
00649                                 TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_FAILED_SHADER_LOAD,
      GetInstance()->ShaderTypeToString( type ) );
00650                                 printf( "%s\n", errorLog );
00651                             }
00652 
00653                             else
00654                             {
00655                                 isCompiled = GL_TRUE;
00656                                 GetInstance()->shaders.push_back( this );
00657                                 iD = GetInstance()->shaders.size() - 1;
00658                             }
00659                         }
00660                         else
00661                         {
00662                             TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_INVALID_SOURCE_FILE );
00663                         }
00664                     }
00665                     else
00666                     {
00667                         //either the file name doesn't exist or the component has already been loaded
00668                         TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_INVALID_FILE_PATH, filePath );
00669                     }
00670                 }
```

7.1.3.2 void tinyShaders::shader_t::Shutdown (void) [inline]

```
00676                 {
00677                     glDeleteShader( handle );
00678                     isCompiled = GL_FALSE;
00679                 }
```

7.1.4 Field Documentation

7.1.4.1 const GLchar* tinyShaders::shader_t::filePath

The FilePath of the component

7.1.4.2 GLuint tinyShaders::shader_t::handle

The handle to the shader in OpenGL

7.1.4.3 GLuint tinyShaders::shader_t::iD

The ID of the shader

7.1.4.4 GLboolean tinyShaders::shader_t::isCompiled

Whether the shader has been compiled

7.1.4.5 const GLchar* tinyShaders::shader_t::name

The name of the shader component

7.1.4.6 GLuint tinyShaders::shader_t::type

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

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

- TinyShaders.h

7.2 tinyShaders::shaderProgram_t Struct Reference

Public Member Functions

- shaderProgram_t (void)
- shaderProgram_t (const GLchar *shaderName, std::vector< const GLchar * > programInputs, std::vector< const GLchar * > programOutputs, std::vector< shader_t * > programShaders)
- shaderProgram_t (const GLchar *shaderName)
- ~shaderProgram_t (void)
- void Shutdown (void)
- GLboolean Compile (void)

Data Fields

- const GLchar * name
- GLuint handle
- GLuint iD
- GLboolean compiled
- std::vector< const GLchar * > inputs
- std::vector< const GLchar * > outputs
- std::vector< shader_t * > shaders

Static Public Attributes

- static const GLuint maxNumShaders = 5

7.2.1 Detailed Description

7.2.2 Constructor & Destructor Documentation

7.2.2.1 tinyShaders::shaderProgram_t::shaderProgram_t (void) [inline]

```
00698                 {
00699                     iD = 0;
00700                 };
```

7.2.2.2 tinyShaders::shaderProgram_t::shaderProgram_t (const GLchar * *shaderName*, std::vector< const GLchar * > *programInputs*, std::vector< const GLchar * > *programOutputs*, std::vector< shader_t * > *programShaders*) [inline]

```
00708                                                                 :
00709                     name( shaderName ), inputs( programInputs ),
00710                     outputs( programOutputs ), shaders( programShaders )
00711                 {
00712                     compiled = GL_FALSE;
00713                     Compile();
00714                     //get number of uniform blocks
00715                     if ( GetInstance()->shaderBlocksEvent != nullptr )
00716                     {
00717                         GetInstance()->shaderBlocksEvent(
      handle );
00718                     }
00719                 };
```

7.2.2.3 tinyShaders::shaderProgram_t::shaderProgram_t (const GLchar * *shaderName*) [inline]

```
00724                                                             : name( shaderName )
00725                 {
00726                     compiled = GL_FALSE;
00727                 };
```

7.2.2.4 tinyShaders::shaderProgram_t::~shaderProgram_t (void) [inline]

```
00729 {}
```

7.2.3 Member Function Documentation

7.2.3.1 GLboolean tinyShaders::shaderProgram_t::Compile (void) [inline]

```
00751                 {
00752                     handle = glCreateProgram();
00753                     GLchar errorLog[512];
00754                     GLint successful = GL_FALSE;
00755                     if ( !compiled )
00756                     {
00757                         for ( GLuint iterator = 0; iterator <shaders.size(); iterator++ )
00758                         {
00759                             if ( shaders[iterator] != nullptr )
00760                             {
00761                                 glAttachShader( handle, shaders[iterator]->
      handle );
00762                             }
00763                         }
00764
00765                         // specify vertex input attributes
00766                         for ( GLuint i = 0; i <inputs.size(); ++i )
```

```
00767                           {
00768                               glBindAttribLocation( handle, i, inputs[i] );
00769                           }
00770 
00771                           // specify pixel shader outputs
00772                           for ( GLuint i = 0; i <outputs.size(); ++i )
00773                           {
00774                               glBindFragDataLocation( handle, i, outputs[i] );
00775                           }
00776 
00777                           glLinkProgram( handle );
00778                           glGetProgramiv( handle, GL_LINK_STATUS, &successful );
00779                           glGetProgramInfoLog( handle, sizeof( errorLog ), 0, errorLog );
00780 
00781                           if ( !successful )
00782                           {
00783                               TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_FAILED_SHADER_PROGRAM_LINK,
      name );
00784                               printf( "%s\n", errorLog );
00785                               return GL_FALSE;
00786                           }
00787                           //if a shader successfully compiles then it will add itself to storage
00788                           compiled = GL_TRUE;
00789                           GetInstance()->shaderPrograms.push_back( this );
00790                           iD = GetInstance()->shaderPrograms.size() - 1;
00791                           return GL_TRUE;
00792                       }
00793                       TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_SHADER_PROGRAM_ALREADY_EXISTS,
      name );
00794                       return GL_FALSE;
00795                   }
```

7.2.3.2 void tinyShaders::shaderProgram_t::Shutdown (void) [inline]

```
00735                   {
00736                       glDeleteProgram( handle );
00737 
00738                       for ( GLuint iterator = 0; iterator < GetInstance()->
      shaders.size(); iterator++ )
00739                       {
00740                           GetInstance()->shaders[iterator]->Shutdown();
00741                       }
00742                       shaders.clear();
00743                       inputs.clear();
00744                       outputs.clear();
00745                   }
```

7.2.4 Field Documentation

7.2.4.1 GLboolean tinyShaders::shaderProgram_t::compiled

Whether the shader program has been linked successfully

7.2.4.2 GLuint tinyShaders::shaderProgram_t::handle

The OpenGL handle to the shader program

7.2.4.3 GLuint tinyShaders::shaderProgram_t::iD

The ID of the shader program

7.2.4.4 std::vector< const GLchar* > tinyShaders::shaderProgram_t::inputs

The inputs of the shader program as a vector of strings

7.2.4.5 const GLuint tinyShaders::shaderProgram_t::maxNumShaders = 5 [static]

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

7.2.4.6 const GLchar* tinyShaders::shaderProgram_t::name

The name of the shader program

7.2.4.7 std::vector< const GLchar* > tinyShaders::shaderProgram_t::outputs

The outputs of the shader program as a vector of strings

7.2.4.8 std::vector< shader_t* > tinyShaders::shaderProgram_t::shaders

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

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

- TinyShaders.h

7.3 tinyShaders Class Reference

`#include <TinyShaders.h>`

Data Structures

- struct shader_t
- struct shaderProgram_t

Public Member Functions

- tinyShaders (void)
- ~tinyShaders (void)

Static Public Member Functions

- static void Shutdown (void)
- static shaderProgram_t * GetShaderProgramByName (const GLchar *programName)
- static shaderProgram_t * GetShaderProgramByIndex (GLuint programIndex)
- static shader_t * GetShaderByName (const GLchar *shaderName)
- static shader_t * 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 *fragmentShader↩ Name, const GLchar *geometryShaderName, const GLchar *tessContShaderName, const GLchar *tess↩ EvalShaderName)
- static GLboolean ShaderProgramExists (const GLchar *shaderName)
- static GLboolean ShaderExists (const GLchar *shaderName)
- static void LoadShaderFromBuffer (const char *name, const GLchar *buffer, GLuint shaderType)
- static GLboolean SetShaderBlockParseEvent (parseBlocks_t shaderBlockParse)

Private Member Functions

- GLchar * FileToBuffer (const GLchar *path) const
- GLuint StringToShaderType (const GLchar *typeString) const
- const GLchar * ShaderTypeToString (GLuint shaderType) const

Static Private Member Functions

- static tinyShaders * GetInstance (void)

Private Attributes

- std::vector< shaderProgram_t * > shaderPrograms
- std::vector< shader_t * > shaders

Static Private Attributes

- static GLboolean isInitialized = GL_FALSE
- static tinyShaders * instance = nullptr
- static parseBlocks_t shaderBlocksEvent = nullptr

7.3.1 Detailed Description

7.3.2 Constructor & Destructor Documentation

7.3.2.1 tinyShaders::tinyShaders (void) [inline]

```
00162 {}
```

7.3.2.2 tinyShaders::~tinyShaders (void) [inline]

```
00163 {}
```

7.3.3 Member Function Documentation

7.3.3.1 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]

```
00504         {
00505             if ( tinyShaders::isInitialized )
00506             {
00507                 std::vector< shader_t* > shaders;
00508                 shaders.push_back( GetShaderByName( vertexShaderName ) );
00509                 shaders.push_back( GetShaderByName( fragmentShaderName ) );
00510                 shaders.push_back( GetShaderByName( geometryShaderName ) );
00511                 shaders.push_back( GetShaderByName( tessContShaderName ) );
00512                 shaders.push_back( GetShaderByName( tessEvalShaderName ) );
00513 
00514                 shaderProgram_t* newShaderProgram = new shaderProgram_t( shaderName, inputs, outputs,
      shaders );
00515                 delete newShaderProgram;
00516             }
00517             TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_NOT_INITIALIZED );
00518         }
```

7.7.3.2 GLchar* tinyShaders::FileToBuffer (const GLchar * *path*) const [inline], [private]

```
00826         {
00827             FILE* file = fopen( path, "rt" );
00828 
00829             if ( file == nullptr )
00830             {
00831                 TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_FILE_PATH, path );
00832                 //printf( "Error: cannot open file %s for reading \n", Path );
00833                 return nullptr;
00834             }
00835 
00836             //get total byte in given file
00837             fseek( file, 0, SEEK_END );
00838             GLuint FileLength = ftell( file );
00839             fseek( file, 0, SEEK_SET );
00840 
00841             //allocate a file buffer and read the contents of the file
00842             GLchar* buffer = new GLchar[FileLength + 1];
00843             memset( buffer, 0, FileLength + 1 );
00844             fread( buffer, sizeof( GLchar ), FileLength, file );
00845 
00846             fclose( file );
00847             return buffer;
00848         }
```

7.7.3.3 static tinyShaders* tinyShaders::GetInstance (void) [inline], [static], [private]

```
00811         {
00812             if ( tinyShaders::isInitialized )
00813             {
00814                 return tinyShaders::instance;
00815             }
00816 
00817             tinyShaders::isInitialized = GL_TRUE;
00818             tinyShaders::instance = new tinyShaders();
00819             return tinyShaders::instance;
00820         }
```

7.7.3.4 static shader_t* tinyShaders::GetShaderByIndex (GLuint *shaderIndex*) [inline], [static]

```
00265         {
00266             if ( tinyShaders::isInitialized )
00267             {
00268                 if ( shaderIndex <= GetInstance()->shaders.size() - 1 )
00269                 {
00270                     return GetInstance()->shaders[shaderIndex];
00271                 }
00272                 TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_SHADER_INDEX );
00273                 return nullptr;
00274             }
00275             TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_NOT_INITIALIZED );
00276             return nullptr;
00277         }
```

7.7.3.5 static shader_t* tinyShaders::GetShaderByName (const GLchar * *shaderName*) [inline], [static]

```
00239         {
00240             if ( tinyShaders::isInitialized )
00241             {
00242                 if ( shaderName != nullptr )
00243                 {
00244                     for ( GLuint iterator = 0; iterator < GetInstance()->
    shaders.size(); iterator++ )
00245                     {
00246                         if ( !strcmp( GetInstance()->shaders[iterator]->name, shaderName
    ) )
00247                         {
00248                             return GetInstance()->shaders[iterator];
00249                         }
00250                     }
00251                     TinyShaders_PrintErrorMessage(
```

```
    TINYSHADERS_ERROR_SHADER_NOT_FOUND );
00252                     return nullptr;
00253                 }
00254                 TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_SHADER_NAME );
00255                 return nullptr;
00256             }
00257             TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_NOT_INITIALIZED );
00258             return nullptr;
00259         }
```

7.3.3.6 static shaderProgram_t* tinyShaders::GetShaderProgramByIndex (GLuint *programIndex*) [inline], [static]

```
00221         {
00222             if ( tinyShaders::isInitialized )
00223             {
00224                 if ( programIndex <= GetInstance()->shaderPrograms.size() - 1 )
00225                 {
00226                     return GetInstance()->shaderPrograms[programIndex];
00227                 }
00228                 TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_SHADER_PROGRAM_INDEX );
00229                 return nullptr;
00230             }
00231             TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_NOT_INITIALIZED );
00232             return nullptr;
00233         }
```

7.3.3.7 static shaderProgram_t* tinyShaders::GetShaderProgramByName (const GLchar * *programName*) [inline], [static]

```
00196         {
00197             if ( tinyShaders::isInitialized )
00198             {
00199                 if ( programName != nullptr )
00200                 {
00201                     for ( GLuint iterator = 0; iterator < GetInstance()->
    shaderPrograms.size(); iterator++ )
00202                     {
00203                         if ( !strcmp( GetInstance()->shaderPrograms[iterator]->
    name, programName ) )
00204                         {
00205                             return GetInstance()->shaderPrograms[iterator];
00206                         }
00207                     }
00208                     return nullptr;
00209                 }
00210                 TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_SHADER_PROGRAM_NOT_FOUND );
00211                 return nullptr;
00212             }
00213             TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_NOT_INITIALIZED );
00214             return nullptr;
00215         }
```

7.3.3.8 static void tinyShaders::LoadShader (const GLchar * *name*, const GLchar * *shaderFile*, GLuint *shaderType*) [inline], [static]

```
00283         {
00284             if ( tinyShaders::isInitialized )
00285             {
00286                 if ( name != nullptr )
00287                 {
00288                     if ( shaderType <= 5 )
00289                     {
00290                         shader_t* newShader = new shader_t( name, shaderType, shaderFile );
00291                     }
00292                     TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_SHADER_TYPE,
    GetInstance()->ShaderTypeToString( shaderType ) );
```

```
00293                 }
00294                 TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_STRING );
00295             }
00296             TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_NOT_INITIALIZED );
00297         }
```

7.3.3.9 static void tinyShaders::LoadShaderFromBuffer (const char * *name*, const GLchar * *buffer*, GLuint *shaderType*) [inline], [static]

```
00569         {
00570             if( tinyShaders::isInitialized )
00571             {
00572                 if( buffer != nullptr )
00573                 {
00574                     if( name != nullptr )
00575                     {
00576                         if( !ShaderExists( name ) )
00577                         {
00578                             shader_t* newShader = new shader_t( name, buffer, shaderType );
00579                             delete newShader;
00580                         }
00581                         TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_SHADER_NOT_FOUND );
00582                     }
00583                     TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_SHADER_NAME );
00584                 }
00585                 TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_INVALID_STRING );
00586             }
00587             TinyShaders_PrintErrorMessage(
    TINYSHADERS_ERROR_NOT_INITIALIZED );
00588         }
```

7.3.3.10 static void tinyShaders::LoadShaderProgramsFromConfigFile (const GLchar * *configFile*) [inline], [static]

```
00303         {
00304             if ( GetInstance()->isInitialized )
00305             {
00306                 FILE* pConfigFile = fopen( configFile, "r" );
00307                 GLuint numInputs = 0;
00308                 GLuint numOutputs = 0;
00309                 GLuint numPrograms = 0;
00310                 GLuint numShaders = 0;
00311                 GLuint iterator = 0;
00312
00313                 std::vector< const GLchar* > inputs, outputs, paths, names;
00314                 std::vector< shader_t* > shaders;
00315                 if ( pConfigFile )
00316                 {
00317                     //get the total number of shader programs
00318                     fscanf( pConfigFile, "%i\n", &numPrograms );
00319
00320                     for ( GLuint programIter = 0;
00321                         programIter <numPrograms;
00322                         programIter++, paths.clear(), inputs.clear(), outputs.clear(), names.clear(),
    shaders.clear() )
00323                     {
00324                         //get the name of the shader program
00325                         GLchar* programName = new GLchar[255];
00326                         fscanf( pConfigFile, "%s\n", programName );
00327                         printf( "%s\n", programName );
00328
00329                         //this is an anti-trolling measure. If a shader with the same name already exists
     the don't bother making a new one.
00330                         if ( !GetInstance()->ShaderProgramExists( programName
    ) )
00331                         {
00332                             //get the number of shader inputs
00333                             fscanf( pConfigFile, "%i\n", &numInputs );
00334
00335                             //get all inputs
00336                             for ( iterator = 0; iterator <numInputs; iterator++ )
00337                             {
00338                                 GLchar* input = new GLchar[255];
00339                                 fscanf( pConfigFile, "%s\n", input );
```

```
00340                                 inputs.push_back( input );
00341                             }
00342 
00343                             //get the number of shader outputs
00344                             fscanf( pConfigFile, "%i\n", &numOutputs );
00345 
00346                             //get all outputs
00347                             for ( iterator = 0; iterator <numOutputs; iterator++ )
00348                             {
00349                                 GLchar* output = new GLchar[255];
00350                                 fscanf( pConfigFile, "%s\n", output );
00351                                 outputs.push_back( output );
00352                             }
00353 
00354                             //get number of shaders
00355                             fscanf( pConfigFile, "%i\n", &numShaders );
00356                             printf( "%i\n", numShaders );
00357 
00358                             for( GLuint iterator = 0; iterator <numShaders; iterator++ )
00359                             {
00360                                 GLchar* shaderName = new GLchar[255];
00361                                 GLchar* shaderPath = new GLchar[255];
00362                                 GLchar* shaderType = new GLchar[255];
00363 
00364                                 //get shader name
00365                                 fscanf( pConfigFile, "%s\n", shaderName );
00366                                 printf( "%s\n", shaderName );
00367 
00368                                 //if the shader hasn't been loaded already then make a new one
00369                                 if( !ShaderExists( shaderName ) )
00370                                 {
00371                                     //get type
00372                                     fscanf( pConfigFile, "%s\n", shaderType );
00373                                     printf( "%s\n", shaderType );
00374                                     //get file path
00375                                     fscanf( pConfigFile, "%s\n", shaderPath );
00376                                     printf( "%s\n", shaderPath );
00377 
00378                                     shaders.push_back( new shader_t( shaderName,
      GetInstance()->StringToShaderType( ( const char* )shaderType ), shaderPath ) )
      ;
00379                                 }
00380 
00381                                 else
00382                                 {
00383                                     //tell scanf to skip a couple lines
00384                                     fscanf( pConfigFile, "%*[^\n]\n %*[^\n]\n", NULL );
00385                                     //if shader already exists then add an existing one from storage, it
       should already be compiled
00386                                     shaders.push_back( GetShaderByName( shaderName ) );
00387                                 }
00388                             }
00389 
00390                             shaderProgram_t* newShaderProgram = new shaderProgram_t( programName, inputs,
      outputs, shaders );
00391                             //get shader block names
00392                         }
00393                     }
00394                     fclose( pConfigFile );
00395                 }
00396                 else
00397                 {
00398                     TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_INVALID_FILE_PATH );
00399                 }
00400             }
00401             else
00402             {
00403                 TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_NOT_INITIALIZED );
00404             }
00405         }
```

7.3.3.11 static void tinyShaders::LoadShadersFromConfigFile (const GLchar * *configFile*) [inline], [static]

```
00408         {
00409             if( tinyShaders::isInitialized )
00410             {
00411                 FILE* pConfigFile = fopen( configFile, "r+" );
00412                 GLuint numShaders = 0;
00413 
00414                 if( pConfigFile )
00415                 {
00416                     //get the number of shaders to load
```

```
00417                   fscanf( pConfigFile, "%i\n", &numShaders );
00418               GLchar* shaderName;
00419                   GLchar* shaderType;
00420                   GLchar* shaderPath;
00421
00422                   GLchar empty[255];
00423
00424                   for( GLuint iterator = 0; iterator <numShaders;
00425                           iterator++, fscanf( pConfigFile, "\n\n" ) )
00426                   {
00427                       shaderName = empty;
00428                       fscanf( pConfigFile, "%s\n", shaderName );
00429
00430                       if( !GetInstance()->ShaderExists( shaderName ) )
00431                       {
00432                           shaderType = empty;
00433                           fscanf( pConfigFile, "%s\n", shaderType );
00434
00435                           shaderPath = empty;
00436                           fscanf( pConfigFile, "%s\n", shaderPath );
00437
00438                           shader_t* newShader = new shader_t( shaderName,
    GetInstance()->StringToShaderType( shaderType ), shaderPath );
00439                           delete newShader;
00440                       }
00441                   }
00442               }
00443           }
00444       }
```

7.3.3.12 static void tinyShaders::SaveShaderProgramsToConfigFile (const GLchar * *fileName*) [inline], [static]

```
00447       {
00448           //write total amount of shaders
00449           FILE* pConfigFile = fopen( fileName, "w+" );
00450
00451           fprintf( pConfigFile, "%i\n\n", ( GLint )GetInstance()->
    shaderPrograms.size() );
00452
00453           for( GLuint programIter = 0; programIter < GetInstance()->
    shaderPrograms.size(); programIter++ )
00454           {
00455               //write program name
00456               fprintf( pConfigFile, "%s\n", GetInstance()->
    shaderPrograms[programIter]->name );
00457
00458               //write number of inputs
00459               fprintf( pConfigFile, "%i\n", ( GLint )GetInstance()->
    shaderPrograms[programIter]->inputs.size() );
00460
00461               //write inputs
00462               for( GLuint inputIter = 0; inputIter < GetInstance()->
    shaderPrograms[programIter]->inputs.size(); inputIter++ )
00463               {
00464                   fprintf( pConfigFile, "%s\n", GetInstance()->
    shaderPrograms[programIter]->inputs[inputIter] );
00465               }
00466
00467               fprintf( pConfigFile, "%i\n", ( GLint )GetInstance()->
    shaderPrograms[programIter]->outputs.size() );
00468
00469               //write outputs
00470               for( GLuint outputIter = 0; outputIter < GetInstance()->
    shaderPrograms[programIter]->outputs.size(); outputIter++ )
00471               {
00472                   fprintf( pConfigFile, "%s\n", GetInstance()->
    shaderPrograms[programIter]->outputs[outputIter] );
00473               }
00474
00475               //write number of shaders
00476               fprintf( pConfigFile, "%i\n", ( GLint )GetInstance()->
    shaderPrograms[programIter]->shaders.size() );
00477
00478               for( GLuint shaderIter = 0; shaderIter < GetInstance()->
    shaderPrograms[programIter]->shaders.size(); shaderIter++ )
00479               {
00480                   //write shader name
00481                   fprintf( pConfigFile, "%s\n", GetInstance()->
    shaderPrograms[programIter]->shaders[shaderIter]->name );
00482
00483                   //write shader type
00484                   fprintf( pConfigFile, "%s\n", GetInstance()->
    ShaderTypeToString( GetInstance()->shaderPrograms[programIter]->
    shaders[shaderIter]->type ) );
```

```
00485
00486                     //write shader file path
00487                     fprintf( pConfigFile, "%s\n", GetInstance()->
      shaderPrograms[programIter]->shaders[shaderIter]->filePath );
00488                 }
00489             }
00490             fclose( pConfigFile );
00491         }
```

7.3.3.13 static GLboolean tinyShaders::SetShaderBlockParseEvent (parseBlocks_t *shaderBlockParse*) [inline], [static]

```
00591         {
00592             if ( GetInstance()->isInitialized )
00593             {
00594                 GetInstance()->shaderBlocksEvent = shaderBlockParse;
00595                 return GL_TRUE;
00596             }
00597             return GL_FALSE;
00598         }
```

7.3.3.14 static GLboolean tinyShaders::ShaderExists (const GLchar * *shaderName*) [inline], [static]

```
00548         {
00549             if ( shaderName != nullptr )
00550             {
00551                 if ( !GetInstance()->shaders.empty() )
00552                 {
00553                     for ( GLuint iterator = 0; iterator < GetInstance()->
      shaders.size(); iterator++ )
00554                     {
00555                         if ( GetInstance()->shaders[iterator] != nullptr &&
00556                             !strcmp( shaderName, GetInstance()->
      shaders[iterator]->name ) )
00557                         {
00558                             return GL_TRUE;
00559                         }
00560                     }
00561                     return GL_FALSE;
00562                 }
00563                 return GL_FALSE;
00564             }
00565             return GL_FALSE;
00566         }
```

7.3.3.15 static GLboolean tinyShaders::ShaderProgramExists (const GLchar * *shaderName*) [inline], [static]

```
00524         {
00525             if ( shaderName != nullptr )
00526             {
00527                 if ( !GetInstance()->shaderPrograms.empty() )
00528                 {
00529                     for ( GLuint iterator = 0; iterator < GetInstance()->
      shaderPrograms.size(); iterator++ )
00530                     {
00531                         if ( GetInstance()->shaderPrograms[iterator] != nullptr &&
00532                             !strcmp( shaderName, GetInstance()->
      shaderPrograms[iterator]->name ) )
00533                         {
00534                             return GL_TRUE;
00535                         }
00536                     }
00537                     return GL_FALSE;
00538                 }
00539                 return GL_FALSE;
00540             }
00541             return GL_FALSE;
00542         }
```

7.3.3.16 const GLchar* tinyShaders::ShaderTypeToString (GLuint *shaderType*) const [inline], [private]

```
00892         {
```

```
00893             switch ( shaderType )
00894             {
00895                 case GL_VERTEX_SHADER:
00896                 {
00897                     return "Vertex";
00898                 }
00899 
00900                 case GL_FRAGMENT_SHADER:
00901                 {
00902                     return "Fragment";
00903                 }
00904 
00905                 case GL_GEOMETRY_SHADER:
00906                 {
00907                     return "Geometry";
00908                 }
00909 
00910                 case GL_TESS_CONTROL_SHADER:
00911                 {
00912                     return "Tessellation Control";
00913                 }
00914 
00915                 case GL_TESS_EVALUATION_SHADER:
00916                 {
00917                     return "Tessellation Evaluation";
00918                 }
00919 
00920                 default:
00921                 {
00922                     return NULL;
00923                 }
00924             }
00925 
00926             return nullptr;
00927         }
```

7.3.3.17 static void tinyShaders::Shutdown (void) [inline], [static]

```
00170         {
00171             if ( tinyShaders::isInitialized )
00172             {
00173                 for ( GLuint iterator = 0; iterator < GetInstance()->
      shaders.size(); iterator++ )
00174                 {
00175                     GetInstance()->shaders[iterator]->Shutdown();
00176                      delete GetInstance()->shaders[iterator];
00177                 }
00178 
00179                 for ( GLuint iterator = 0; iterator < GetInstance()->
      shaderPrograms.size(); iterator++ )
00180                 {
00181                     GetInstance()->shaderPrograms[iterator]->Shutdown();
00182                     delete GetInstance()->shaderPrograms[iterator];
00183                 }
00184 
00185                 GetInstance()->shaderPrograms.clear();
00186                 GetInstance()->shaders.clear();
00187 
00188                 delete instance;
00189             }
00190         }
```

7.3.3.18 GLuint tinyShaders::StringToShaderType (const GLchar * *typeString*) const [inline], [private]

```
00854         {
00855             if( typeString != nullptr )
00856             {
00857                 if ( !strcmp( typeString, "Vertex" ) )
00858                 {
00859                     return GL_VERTEX_SHADER;
00860                 }
00861 
00862                 if ( !strcmp( typeString, "Fragment" ) )
00863                 {
00864                     return GL_FRAGMENT_SHADER;
00865                 }
00866 
00867                 if ( !strcmp( typeString, "Geometry" ) )
00868                 {
00869                     return GL_GEOMETRY_SHADER;
```

```
00870                 }
00871
00872                 if ( !strcmp( typeString, "Tessellation Control" ) )
00873                 {
00874                     return GL_TESS_CONTROL_SHADER;
00875                 }
00876
00877                 if ( !strcmp( typeString, "Tessellation Evaluation" ) )
00878                 {
00879                     return GL_TESS_EVALUATION_SHADER;
00880                 }
00881
00882                 return GL_FALSE;
00883             }
00884             TinyShaders_PrintErrorMessage(
      TINYSHADERS_ERROR_INVALID_STRING );
00885             return GL_FALSE;
00886         }
```

7.3.4 Field Documentation

7.3.4.1 tinyShaders * tinyShaders::instance = nullptr [static], [private]

A static instance of the TinyShaders API

7.3.4.2 GLboolean tinyShaders::isInitialized = GL_FALSE [static], [private]

Whether TinyShadershas ban initialized

7.3.4.3 parseBlocks_t tinyShaders::shaderBlocksEvent = nullptr [static], [private]

7.3.4.4 std::vector< shaderProgram_t* > tinyShaders::shaderPrograms [private]

All loaded shader programs

7.3.4.5 std::vector< shader_t* > tinyShaders::shaders [private]

All loaded shaders

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

- TinyShaders.h

Chapter 8

File Documentation

8.1 TinyShaders.h File Reference

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

Data Structures

- class tinyShaders
- struct tinyShaders::shader_t
- struct tinyShaders::shaderProgram_t

Macros

- #define TINYSHADERS_ERROR_NOT_INITIALIZED 1
- #define TINYSHADERS_ERROR_INVALID_STRING 2
- #define TINYSHADERS_ERROR_INVALID_SHADER_PROGRAM_NAME 3
- #define TINYSHADERS_ERROR_INVALID_SHADER_PROGRAM_INDEX 4
- #define TINYSHADERS_ERROR_INVALID_SHADER_NAME 5
- #define TINYSHADERS_ERROR_INVALID_SHADER_INDEX 6
- #define TINYSHADERS_ERROR_INVALID_FILE_PATH 7
- #define TINYSHADERS_ERROR_SHADER_PROGRAM_NOT_FOUND 8
- #define TINYSHADERS_ERROR_SHADER_NOT_FOUND 9
- #define TINYSHADERS_ERROR_INVALID_SHADER_TYPE 10
- #define TINYSHADERS_ERROR_FAILED_SHADER_LOAD 11
- #define TINYSHADERS_ERROR_FAILED_SHADER_PROGRAM_LINK 12
- #define TINYSHADERS_ERROR_SHADER_ALREADY_EXISTS 13
- #define TINYSHADERS_ERROR_SHADER_PROGRAM_ALREADY_EXISTS 14
- #define TINYSHADERS_ERROR_INVALID_SOURCE_FILE 15

Typedefs

- typedef void(∗ parseBlocks_t)(GLuint programHandle)

Functions

- static void TinyShaders_PrintErrorMessage (GLuint errorNumber, const GLchar *errorMessage=nullptr)

8.1.1 Macro Definition Documentation

8.1.1.1 #define TINYSHADERS_ERROR_FAILED_SHADER_LOAD 11

8.1.1.2 #define TINYSHADERS_ERROR_FAILED_SHADER_PROGRAM_LINK 12

8.1.1.3 #define TINYSHADERS_ERROR_INVALID_FILE_PATH 7

8.1.1.4 #define TINYSHADERS_ERROR_INVALID_SHADER_INDEX 6

8.1.1.5 #define TINYSHADERS_ERROR_INVALID_SHADER_NAME 5

8.1.1.6 #define TINYSHADERS_ERROR_INVALID_SHADER_PROGRAM_INDEX 4

8.1.1.7 #define TINYSHADERS_ERROR_INVALID_SHADER_PROGRAM_NAME 3

8.1.1.8 #define TINYSHADERS_ERROR_INVALID_SHADER_TYPE 10

8.1.1.9 #define TINYSHADERS_ERROR_INVALID_SOURCE_FILE 15

8.1.1.10 #define TINYSHADERS_ERROR_INVALID_STRING 2

8.1.1.11 #define TINYSHADERS_ERROR_NOT_INITIALIZED 1

8.1.1.12 #define TINYSHADERS_ERROR_SHADER_ALREADY_EXISTS 13

8.1.1.13 #define TINYSHADERS_ERROR_SHADER_NOT_FOUND 9

8.1.1.14 #define TINYSHADERS_ERROR_SHADER_PROGRAM_ALREADY_EXISTS 14

8.1.1.15 #define TINYSHADERS_ERROR_SHADER_PROGRAM_NOT_FOUND 8

8.1.2 Typedef Documentation

8.1.2.1 typedef void(* parseBlocks_t)(GLuint programHandle)

a callback that can gather all the info about the uniform blocks that are in a shader program

8.1.3 Function Documentation

8.1.3.1 static void TinyShaders_PrintErrorMessage (GLuint *errorNumber,* const GLchar * *errorMessage* = `nullptr`) [inline], [static]

```
00049 {
00050     switch (errorNumber)
00051     {
00052         case TINYSHADERS_ERROR_NOT_INITIALIZED:
00053         {
00054             printf("Error: TinyShaders must first be initialized \n");
00055             break;
00056         }
00057 
00058         case TINYSHADERS_ERROR_INVALID_STRING:
00059         {
00060             printf("Error: given string is invalid \n");
```

```
00061             break;
00062         }
00063
00064         case TINYSHADERS_ERROR_INVALID_SHADER_PROGRAM_NAME:
00065         {
00066             printf("Error: given shader name is invalid \n");
00067             break;
00068         }
00069
00070         case TINYSHADERS_ERROR_INVALID_SHADER_PROGRAM_INDEX:
00071         {
00072             printf("Error: given shader index is invalid \n");
00073             break;
00074         }
00075
00076         case TINYSHADERS_ERROR_INVALID_SHADER_NAME:
00077         {
00078             printf("Error: given shader component name is invalid \n");
00079             break;
00080         }
00081
00082         case TINYSHADERS_ERROR_INVALID_SHADER_INDEX:
00083         {
00084             printf("Error: given shader component index is invalid \n");
00085             break;
00086         }
00087
00088         case TINYSHADERS_ERROR_INVALID_FILE_PATH:
00089         {
00090             printf("Error: given file path is invalid %s \n", errorMessage);
00091             break;
00092         }
00093
00094         case TINYSHADERS_ERROR_SHADER_PROGRAM_NOT_FOUND:
00095         {
00096             printf("Error: shader with given name %s was not found \n", errorMessage);
00097             break;
00098         }
00099
00100         case TINYSHADERS_ERROR_SHADER_NOT_FOUND:
00101         {
00102             printf("Error: shader component with given name %s was not found \n", errorMessage);
00103             break;
00104         }
00105
00106         case TINYSHADERS_ERROR_INVALID_SHADER_TYPE:
00107         {
00108             printf("Error: invalid shader type given \n");
00109             break;
00110         }
00111
00112         case TINYSHADERS_ERROR_FAILED_SHADER_LOAD:
00113         {
00114             printf("Error: failed to compile %s shader component \n", errorMessage);
00115             break;
00116         }
00117
00118         case TINYSHADERS_ERROR_FAILED_SHADER_PROGRAM_LINK:
00119         {
00120             if (errorMessage != nullptr)
00121             {
00122                 printf("Error: failed to link program %s \n", errorMessage);
00123             }
00124             break;
00125         }
00126
00127         case TINYSHADERS_ERROR_SHADER_ALREADY_EXISTS:
00128         {
00129             printf("Error: shader component with this name %s already exists \n", errorMessage);
00130             break;
00131         }
00132
00133         case TINYSHADERS_ERROR_SHADER_PROGRAM_ALREADY_EXISTS
    :
00134         {
00135             if (errorMessage != nullptr)
00136             {
00137                 printf("Error: shader with this name %s already exists \n", errorMessage);
00138                 break;
00139             }
00140         }
00141
00142         case TINYSHADERS_ERROR_INVALID_SOURCE_FILE:
00143         {
00144             printf("Given Source file is invalid");
00145             break;
00146         }
```

```
00147
00148         default:
00149         {
00150             break;
00151         }
00152     }
00153 }
```