

XShaderCompiler

0.08 Alpha

Generated by Doxygen 1.8.11

Contents

1	Main Page	1
2	Namespace Index	3
2.1	Namespace List	3
3	Hierarchical Index	5
3.1	Class Hierarchy	5
4	Class Index	7
4.1	Class List	7
5	Namespace Documentation	9
5.1	Xsc Namespace Reference	9
5.1.1	Detailed Description	11
5.1.2	Enumeration Type Documentation	11
5.1.2.1	InputShaderVersion	11
5.1.2.2	OutputShaderVersion	11
5.1.2.3	ShaderTarget	12
5.1.3	Function Documentation	12
5.1.3.1	CompileShader(const ShaderInput &inputDesc, const ShaderOutput &output↔ Desc, Log *log=nullptr, Reflection::ReflectionData *reflectionData=nullptr)	12
5.2	Xsc::ConsoleManip Namespace Reference	13
5.2.1	Detailed Description	14
5.3	Xsc::Reflection Namespace Reference	14
5.3.1	Detailed Description	14

6	Class Documentation	15
6.1	Xsc::Reflection::BindingSlot Struct Reference	15
6.1.1	Detailed Description	15
6.2	Xsc::ConsoleManip::ColorFlags Struct Reference	15
6.2.1	Detailed Description	16
6.2.2	Member Enumeration Documentation	16
6.2.2.1	anonymous enum	16
6.3	Xsc::Formatting Struct Reference	16
6.3.1	Detailed Description	17
6.4	Xsc::IncludeHandler Class Reference	17
6.4.1	Detailed Description	17
6.4.2	Member Function Documentation	17
6.4.2.1	Include(const std::string &filename, bool useSearchPathsFirst)	17
6.5	Xsc::IndentHandler Class Reference	18
6.5.1	Detailed Description	18
6.6	Xsc::Log Class Reference	18
6.6.1	Detailed Description	19
6.7	Xsc::NameMangling Struct Reference	19
6.7.1	Detailed Description	20
6.7.2	Member Data Documentation	20
6.7.2.1	inputPrefix	20
6.7.2.2	namespacePrefix	20
6.7.2.3	outputPrefix	20
6.7.2.4	reservedWordPrefix	20
6.7.2.5	temporaryPrefix	20
6.7.2.6	useAlwaysSemantics	21
6.8	Xsc::Reflection::NumThreads Struct Reference	21
6.8.1	Detailed Description	21
6.9	Xsc::Options Struct Reference	21
6.9.1	Detailed Description	22

6.10	Xsc::Reflection::ReflectionData Struct Reference	22
6.10.1	Detailed Description	23
6.11	Xsc::Report Class Reference	23
6.11.1	Detailed Description	24
6.11.2	Member Enumeration Documentation	24
6.11.2.1	Types	24
6.11.3	Member Function Documentation	25
6.11.3.1	HasLine() const	25
6.12	Xsc::Reflection::SamplerState Struct Reference	25
6.12.1	Detailed Description	25
6.13	Xsc::ConsoleManip::ScopedColor Class Reference	26
6.13.1	Detailed Description	26
6.13.2	Constructor & Destructor Documentation	26
6.13.2.1	ScopedColor(std::ostream &stream, long front)	26
6.13.2.2	ScopedColor(std::ostream &stream, long front, long back)	26
6.13.2.3	~ScopedColor()	27
6.14	Xsc::ScopedIndent Class Reference	27
6.14.1	Detailed Description	27
6.15	Xsc::ShaderInput Struct Reference	27
6.15.1	Detailed Description	28
6.15.2	Member Data Documentation	28
6.15.2.1	includeHandler	28
6.15.2.2	secondaryEntryPoint	28
6.15.2.3	warnings	29
6.16	Xsc::ShaderOutput Struct Reference	29
6.16.1	Detailed Description	29
6.17	Xsc::StdLog Class Reference	30
6.17.1	Detailed Description	30
6.18	Xsc::VertexSemantic Struct Reference	30
6.18.1	Detailed Description	31

6.19	Xsc::Warnings Struct Reference	31
6.19.1	Detailed Description	31
6.19.2	Member Enumeration Documentation	31
6.19.2.1	anonymous enum	31
6.20	XscBindingSlot Struct Reference	32
6.20.1	Detailed Description	32
6.21	XscFormatting Struct Reference	32
6.21.1	Detailed Description	33
6.22	XscIncludeHandler Struct Reference	33
6.22.1	Detailed Description	33
6.23	XscLog Struct Reference	33
6.23.1	Detailed Description	33
6.24	XscNameMangling Struct Reference	34
6.24.1	Detailed Description	34
6.24.2	Member Data Documentation	34
6.24.2.1	inputPrefix	34
6.24.2.2	namespacePrefix	34
6.24.2.3	outputPrefix	35
6.24.2.4	reservedWordPrefix	35
6.24.2.5	temporaryPrefix	35
6.24.2.6	useAlwaysSemantics	35
6.25	XscNumThreads Struct Reference	35
6.25.1	Detailed Description	36
6.26	XscOptions Struct Reference	36
6.26.1	Detailed Description	36
6.27	XscReflectionData Struct Reference	37
6.27.1	Detailed Description	37
6.28	XscReport Struct Reference	38
6.28.1	Detailed Description	38
6.29	XscSamplerState Struct Reference	38
6.29.1	Detailed Description	39
6.30	XscShaderInput Struct Reference	39
6.30.1	Detailed Description	39
6.30.2	Member Data Documentation	40
6.30.2.1	secondaryEntryPoint	40
6.30.2.2	warnings	40
6.31	XscShaderOutput Struct Reference	40
6.31.1	Detailed Description	41
6.32	XscVertexSemantic Struct Reference	41
6.32.1	Detailed Description	41

Chapter 1

Main Page

Welcome to the XShaderCompiler, Version 0.08 Alpha

Here is a quick start example:

```
#include <Xsc/Xsc.h>
#include <fstream>

int main()
{
    // Open input and output streams
    auto inputStream = std::make_shared<std::ifstream>("Example.hlsl");
    std::ofstream outputStream("Example.VS.vert");

    // Initialize shader input descriptor structure
    Xsc::ShaderInput inputDesc;
    {
        inputDesc.sourceCode      = inputStream;
        inputDesc.shaderVersion   = Xsc::InputShaderVersion::HLSL5
    ;
        inputDesc.entryPoint      = "VS";
        inputDesc.shaderTarget    = Xsc::ShaderTarget::VertexShader
    ;
    }

    // Initialize shader output descriptor structure
    Xsc::ShaderOutput outputDesc;
    {
        outputDesc.sourceCode     = &outputStream;
        outputDesc.shaderVersion  =
        Xsc::OutputShaderVersion::GLSL330;
    }

    // Compile HLSL code into GLSL
    Xsc::StdLog log;
    bool result = Xsc::CompileShader(inputDesc, outputDesc, &log);

    // Show compilation status
    if (result)
        std::cout << "Compilation successful" << std::endl;
    else
        std::cerr << "Compilation failed" << std::endl;

    return 0;
}
```


Chapter 2

Namespace Index

2.1 Namespace List

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

Xsc	Main XShaderCompiler namespace	9
Xsc::ConsoleManip	Namespace for console manipulation	13
Xsc::Reflection	Shader code reflection namespace	14

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Xsc::Reflection::BindingSlot	15
Xsc::ConsoleManip::ColorFlags	15
exception	
Xsc::Report	23
Xsc::Formatting	16
Xsc::IncludeHandler	17
Xsc::IndentHandler	18
Xsc::Log	18
Xsc::StdLog	30
Xsc::NameMangling	19
Xsc::Reflection::NumThreads	21
Xsc::Options	21
Xsc::Reflection::ReflectionData	22
Xsc::Reflection::SamplerState	25
Xsc::ConsoleManip::ScopedColor	26
Xsc::ScopedIndent	27
Xsc::ShaderInput	27
Xsc::ShaderOutput	29
Xsc::VertexSemantic	30
Xsc::Warnings	31
XscBindingSlot	32
XscFormatting	32
XscIncludeHandler	33
XscLog	33
XscNameMangling	34
XscNumThreads	35
XscOptions	36
XscReflectionData	37
XscReport	38
XscSamplerState	38
XscShaderInput	39
XscShaderOutput	40
XscVertexSemantic	41

Chapter 4

Class Index

4.1 Class List

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

Xsc::Reflection::BindingSlot	
Binding slot of textures, constant buffers, and fragment targets	15
Xsc::ConsoleManip::ColorFlags	
Output stream color flags enumeration	15
Xsc::Formatting	
Formatting descriptor structure for the output shader	16
Xsc::IncludeHandler	
Interface for handling new include streams	17
Xsc::IndentHandler	
Indentation handler base class	18
Xsc::Log	
Log base class	18
Xsc::NameMangling	
Name mangling descriptor structure for shader input/output variables (also referred to as "varyings"), temporary variables, and reserved keywords	19
Xsc::Reflection::NumThreads	
Number of threads within each work group of a compute shader	21
Xsc::Options	
Structure for additional translation options	21
Xsc::Reflection::ReflectionData	
Structure for shader output statistics (e.g. texture/buffer binding points)	22
Xsc::Report	
Report exception class	23
Xsc::Reflection::SamplerState	
Static sampler state descriptor structure (D3D11_SAMPLER_DESC)	25
Xsc::ConsoleManip::ScopedColor	
Helper class for scoped color stack operations	26
Xsc::ScopedIndent	
Helper class for temporary indentation	27
Xsc::ShaderInput	
Shader input descriptor structure	27
Xsc::ShaderOutput	
Shader output descriptor structure	29
Xsc::StdLog	
Standard output log (uses std::cout to submit a report)	30

Xsc::VertexSemantic	
Vertex shader semantic (or rather attribute) layout structure	30
Xsc::Warnings	
Compiler warning flags	31
XscBindingSlot	
Binding slot of textures, constant buffers, and fragment targets	32
XscFormatting	
Formatting descriptor structure for the output shader	32
XscIncludeHandler	
Include handler structure	33
XscLog	
Output log structure	33
XscNameMangling	
Name mangling descriptor structure for shader input/output variables (also referred to as "vary-ings"), temporary variables, and reserved keywords	34
XscNumThreads	
Number of threads within each work group of a compute shader	35
XscOptions	
Structure for additional translation options	36
XscReflectionData	
Structure for shader output statistics (e.g. texture/buffer binding points)	37
XscReport	
Report structure for warning and error messages	38
XscSamplerState	
Static sampler state descriptor structure (D3D11_SAMPLER_DESC)	38
XscShaderInput	
Shader input descriptor structure	39
XscShaderOutput	
Shader output descriptor structure	40
XscVertexSemantic	
Vertex shader semantic (or rather attribute) layout structure	41

Chapter 5

Namespace Documentation

5.1 Xsc Namespace Reference

Main XShaderCompiler namespace.

Namespaces

- [ConsoleManip](#)
Namespace for console manipulation.
- [Reflection](#)
Shader code reflection namespace.

Classes

- struct [Formatting](#)
Formatting descriptor structure for the output shader.
- class [IncludeHandler](#)
Interface for handling new include streams.
- class [IndentHandler](#)
Indentation handler base class.
- class [Log](#)
Log base class.
- struct [NameMangling](#)
Name mangling descriptor structure for shader input/output variables (also referred to as "varyings"), temporary variables, and reserved keywords.
- struct [Options](#)
Structure for additional translation options.
- class [Report](#)
Report exception class.
- class [ScopedIndent](#)
Helper class for temporary indentation.
- struct [ShaderInput](#)
Shader input descriptor structure.
- struct [ShaderOutput](#)

- *Shader output descriptor structure.*
- class [StdLog](#)
Standard output log (uses std::cout to submit a report).
- struct [VertexSemantic](#)
Vertex shader semantic (or rather attribute) layout structure.
- struct [Warnings](#)
Compiler warning flags.

Enumerations

- enum [ShaderTarget](#) {
[ShaderTarget::Undefined](#), [ShaderTarget::VertexShader](#), [ShaderTarget::TessellationControlShader](#), [ShaderTarget::TessellationEvaluationShader](#),
[ShaderTarget::GeometryShader](#), [ShaderTarget::FragmentShader](#), [ShaderTarget::ComputeShader](#) }
Shader target enumeration.
- enum [InputShaderVersion](#) {
[InputShaderVersion::HLSL3](#) = 3, [InputShaderVersion::HLSL4](#) = 4, [InputShaderVersion::HLSL5](#) = 5, [InputShaderVersion::GLSL](#) = 0x0000ffff,
[InputShaderVersion::ESSL](#) = 0x0001ffff, [InputShaderVersion::VKSL](#) = 0x0002ffff }
Input shader version enumeration.
- enum [OutputShaderVersion](#) {
[OutputShaderVersion::GLSL110](#) = 110, [OutputShaderVersion::GLSL120](#) = 120, [OutputShaderVersion::GLSL130](#) = 130, [OutputShaderVersion::GLSL140](#) = 140,
[OutputShaderVersion::GLSL150](#) = 150, [OutputShaderVersion::GLSL330](#) = 330, [OutputShaderVersion::GLSL400](#) = 400, [OutputShaderVersion::GLSL410](#) = 410,
[OutputShaderVersion::GLSL420](#) = 420, [OutputShaderVersion::GLSL430](#) = 430, [OutputShaderVersion::GLSL440](#) = 440, [OutputShaderVersion::GLSL450](#) = 450,
[OutputShaderVersion::GLSL](#) = 0x0000ffff, [OutputShaderVersion::ESSL100](#) = (0x00010000 + 100), [OutputShaderVersion::ESSL300](#) = (0x00010000 + 300), [OutputShaderVersion::ESSL310](#) = (0x00010000 + 310),
[OutputShaderVersion::ESSL320](#) = (0x00010000 + 320), [OutputShaderVersion::ESSL](#) = 0x0001ffff, [OutputShaderVersion::VKSL450](#) = (0x00020000 + 450), [OutputShaderVersion::VKSL](#) = 0x0002ffff }
Output shader version enumeration.

Functions

- XSC_EXPORT std::string [ToString](#) (const [Reflection::Filter](#) t)
Returns the string representation of the specified 'SamplerState::Filter' type.
- XSC_EXPORT std::string [ToString](#) (const [Reflection::TextureAddressMode](#) t)
Returns the string representation of the specified 'SamplerState::TextureAddressMode' type.
- XSC_EXPORT std::string [ToString](#) (const [Reflection::ComparisonFunc](#) t)
Returns the string representation of the specified 'SamplerState::ComparisonFunc' type.
- XSC_EXPORT void [PrintReflection](#) (std::ostream &stream, const [Reflection::ReflectionData](#) &reflectionData)
Prints the reflection data into the output stream in a human readable format.
- XSC_EXPORT std::string [ToString](#) (const [ShaderTarget](#) target)
Returns the specified shader target as string.
- XSC_EXPORT std::string [ToString](#) (const [InputShaderVersion](#) shaderVersion)
Returns the specified shader input version as string.
- XSC_EXPORT std::string [ToString](#) (const [OutputShaderVersion](#) shaderVersion)
Returns the specified shader output version as string.
- XSC_EXPORT bool [IsLanguageHLSL](#) (const [InputShaderVersion](#) shaderVersion)
Returns true if the shader input version specifies HLSL (for DirectX).

- XSC_EXPORT bool [IsLanguageGLSL](#) (const [InputShaderVersion](#) shaderVersion)
Returns true if the shader input version specifies GLSL (for OpenGL, OpenGL ES, and Vulkan).
- XSC_EXPORT bool [IsLanguageGLSL](#) (const [OutputShaderVersion](#) shaderVersion)
Returns true if the shader output version specifies GLSL (for OpenGL 2+).
- XSC_EXPORT bool [IsLanguageESSL](#) (const [OutputShaderVersion](#) shaderVersion)
Returns true if the shader output version specifies ESSL (for OpenGL ES 2+).
- XSC_EXPORT bool [IsLanguageVKSL](#) (const [OutputShaderVersion](#) shaderVersion)
Returns true if the shader output version specifies VKSL (for Vulkan).
- XSC_EXPORT const std::map< std::string, int > & [GetGLSLExtensionEnumeration](#) ()
Returns the enumeration of all supported GLSL extensions as a map of extension name and version number.
- XSC_EXPORT bool [CompileShader](#) (const [ShaderInput](#) &inputDesc, const [ShaderOutput](#) &outputDesc, [Log](#) *log=nullptr, [Reflection::ReflectionData](#) *reflectionData=nullptr)
Cross compiles the shader code from the specified input stream into the specified output shader code.

5.1.1 Detailed Description

Main XShaderCompiler namespace.

5.1.2 Enumeration Type Documentation

5.1.2.1 enum Xsc::InputShaderVersion [strong]

Input shader version enumeration.

Enumerator

HLSL3 HLSL Shader Model 3.0 (DirectX 9).
HLSL4 HLSL Shader Model 4.0 (DirectX 10).
HLSL5 HLSL Shader Model 5.0 (DirectX 11).
GLSL GLSL (OpenGL).
ESSL GLSL (OpenGL ES).
VKSL GLSL (Vulkan).

5.1.2.2 enum Xsc::OutputShaderVersion [strong]

Output shader version enumeration.

Enumerator

GLSL110 GLSL 1.10 (OpenGL 2.0).
GLSL120 GLSL 1.20 (OpenGL 2.1).
GLSL130 GLSL 1.30 (OpenGL 3.0).
GLSL140 GLSL 1.40 (OpenGL 3.1).
GLSL150 GLSL 1.50 (OpenGL 3.2).
GLSL330 GLSL 3.30 (OpenGL 3.3).
GLSL400 GLSL 4.00 (OpenGL 4.0).

GLSL410 GLSL 4.10 (OpenGL 4.1).
GLSL420 GLSL 4.20 (OpenGL 4.2).
GLSL430 GLSL 4.30 (OpenGL 4.3).
GLSL440 GLSL 4.40 (OpenGL 4.4).
GLSL450 GLSL 4.50 (OpenGL 4.5).
GLSL Auto-detect minimal required GLSL version (for OpenGL 2+).
ESSL100 ESSL 1.00 (OpenGL ES 2.0).

Note

Currently not supported!

ESSL300 ESSL 3.00 (OpenGL ES 3.0).

Note

Currently not supported!

ESSL310 ESSL 3.10 (OpenGL ES 3.1).

Note

Currently not supported!

ESSL320 ESSL 3.20 (OpenGL ES 3.2).

Note

Currently not supported!

ESSL Auto-detect minimum required ESSL version (for OpenGL ES 2+).

Note

Currently not supported!

VKSL450 VKSL 4.50 (Vulkan 1.0).

VKSL Auto-detect minimum required VKSL version (for Vulkan/SPIR-V).

5.1.2.3 enum Xsc::ShaderTarget [strong]

Shader target enumeration.

Enumerator

Undefined Undefined shader target.
VertexShader Vertex shader.
TessellationControlShader Tessellation-control (also Hull-) shader.
TessellationEvaluationShader Tessellation-evaluation (also Domain-) shader.
GeometryShader Geometry shader.
FragmentShader Fragment (also Pixel-) shader.
ComputeShader Compute shader.

5.1.3 Function Documentation

5.1.3.1 **XSC_EXPORT** bool Xsc::CompileShader (const ShaderInput & *inputDesc*, const ShaderOutput & *outputDesc*, Log * *log* = nullptr, Reflection::ReflectionData * *reflectionData* = nullptr)

Cross compiles the shader code from the specified input stream into the specified output shader code.

Parameters

in	<i>inputDesc</i>	Input shader code descriptor.
in	<i>outputDesc</i>	Output shader code descriptor.
in	<i>log</i>	Optional pointer to an output log. Inherit from the "Log" class interface. By default null.
out	<i>reflectionData</i>	Optional pointer to a code reflection data structure. By default null.

Returns

True if the code has been translated successfully.

Exceptions

<i>std::invalid_argument</i>	If either the input or output streams are null.
------------------------------	---

See also

[ShaderInput](#)
[ShaderOutput](#)
[Log](#)
[ReflectionData](#)

5.2 Xsc::ConsoleManip Namespace Reference

Namespace for console manipulation.

Classes

- struct [ColorFlags](#)
Output stream color flags enumeration.
- class [ScopedColor](#)
Helper class for scoped color stack operations.

Functions

- void XSC_EXPORT [Enable](#) (bool enable)
Enables or disables console manipulation. By default enabled.
- bool XSC_EXPORT [IsEnabled](#) ()
Returns true if console manipulation is enabled.
- void XSC_EXPORT [PushColor](#) (std::ostream &stream, long front)
Push the specified front color onto the stack.
- void XSC_EXPORT [PushColor](#) (std::ostream &stream, long front, long back)
Push the specified front and back color onto the stack.
- void XSC_EXPORT [PopColor](#) (std::ostream &stream)
Pops the previous front and back colors from the stack.

5.2.1 Detailed Description

Namespace for console manipulation.

5.3 Xsc::Reflection Namespace Reference

Shader code reflection namespace.

Classes

- struct [BindingSlot](#)
Binding slot of textures, constant buffers, and fragment targets.
- struct [NumThreads](#)
Number of threads within each work group of a compute shader.
- struct [ReflectionData](#)
Structure for shader output statistics (e.g. texture/buffer binding points).
- struct [SamplerState](#)
Static sampler state descriptor structure (D3D11_SAMPLER_DESC).

Enumerations

- enum [Filter](#) {
MinMagMipPoint = 0, **MinMagPointMipLinear** = 0x1, **MinPointMagLinearMipPoint** = 0x4, **MinPoint↔
MagMipLinear** = 0x5,
MinLinearMagMipPoint = 0x10, **MinLinearMagPointMipLinear** = 0x11, **MinMagLinearMipPoint** = 0x14,
MinMagMipLinear = 0x15,
Anisotropic = 0x55, **ComparisonMinMagMipPoint** = 0x80, **ComparisonMinMagPointMipLinear** = 0x81,
ComparisonMinPointMagLinearMipPoint = 0x84,
ComparisonMinPointMagMipLinear = 0x85, **ComparisonMinLinearMagMipPoint** = 0x90, **Comparison↔
MinLinearMagPointMipLinear** = 0x91, **ComparisonMinMagLinearMipPoint** = 0x94,
ComparisonMinMagMipLinear = 0x95, **ComparisonAnisotropic** = 0xd5, **MinimumMinMagMipPoint** =
0x100, **MinimumMinMagPointMipLinear** = 0x101,
MinimumMinPointMagLinearMipPoint = 0x104, **MinimumMinPointMagMipLinear** = 0x105, **Minimum↔
MinLinearMagMipPoint** = 0x110, **MinimumMinLinearMagPointMipLinear** = 0x111,
MinimumMinMagLinearMipPoint = 0x114, **MinimumMinMagMipLinear** = 0x115, **MinimumAnisotropic** =
0x155, **MaximumMinMagMipPoint** = 0x180,
MaximumMinMagPointMipLinear = 0x181, **MaximumMinPointMagLinearMipPoint** = 0x184, **Maximum↔
MinPointMagMipLinear** = 0x185, **MaximumMinLinearMagMipPoint** = 0x190,
MaximumMinLinearMagPointMipLinear = 0x191, **MaximumMinMagLinearMipPoint** = 0x194, **Maximum↔
MinMagMipLinear** = 0x195, **MaximumAnisotropic** = 0x1d5 }
Sampler filter enumeration (D3D11_FILTER).
- enum [TextureAddressMode](#) {
Wrap = 1, **Mirror** = 2, **Clamp** = 3, **Border** = 4,
MirrorOnce = 5 }
Texture address mode enumeration (D3D11_TEXTURE_ADDRESS_MODE).
- enum [ComparisonFunc](#) {
Never = 1, **Less** = 2, **Equal** = 3, **LessEqual** = 4,
Greater = 5, **NotEqual** = 6, **GreaterEqual** = 7, **Always** = 8 }
Sample comparison function enumeration (D3D11_COMPARISON_FUNC).

5.3.1 Detailed Description

Shader code reflection namespace.

Chapter 6

Class Documentation

6.1 Xsc::Reflection::BindingSlot Struct Reference

Binding slot of textures, constant buffers, and fragment targets.

```
#include <Reflection.h>
```

Public Attributes

- std::string [ident](#)
Identifier of the binding point.
- int [location](#)
Zero based binding point or location. If this is -1, the location has not been set explicitly.

6.1.1 Detailed Description

Binding slot of textures, constant buffers, and fragment targets.

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

- Reflection.h

6.2 Xsc::ConsoleManip::ColorFlags Struct Reference

Output stream color flags enumeration.

```
#include <ConsoleManip.h>
```

Public Types

- enum {
[Red](#) = (1 << 0), [Green](#) = (1 << 1), [Blue](#) = (1 << 2), [Intens](#) = (1 << 3),
[Black](#) = 0, [Gray](#) = ([Red](#) | [Green](#) | [Blue](#)), [White](#) = ([Gray](#) | [Intens](#)), [Yellow](#) = ([Red](#) | [Green](#) | [Intens](#)),
[Pink](#) = ([Red](#) | [Blue](#) | [Intens](#)), [Cyan](#) = ([Green](#) | [Blue](#) | [Intens](#)) }

6.2.1 Detailed Description

Output stream color flags enumeration.

6.2.2 Member Enumeration Documentation

6.2.2.1 anonymous enum

Enumerator

- Red** Red color flag.
- Green** Green color flag.
- Blue** Blue color flag.
- Intens** Intensity color flag.
- Black** Black color flag.
- Gray** Gray color flag (Red | Green | Blue).
- White** White color flag (Gray | Intens).
- Yellow** Yellow color flag (Red | Green | Intens).
- Pink** Pink color flag (Red | Blue | Intens).
- Cyan** Cyan color flag (Green | Blue | Intens).

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

- ConsoleManip.h

6.3 Xsc::Formatting Struct Reference

[Formatting](#) descriptor structure for the output shader.

```
#include <Xsc.h>
```

Public Attributes

- `std::string indent = ""`
Indentation string for code generation. By default `std::string(4, ' ')`.
- `bool blanks = true`
If true, blank lines are allowed. By default true.
- `bool lineMarks = false`
If true, line marks are allowed. By default false.
- `bool compactWrappers = false`
If true, wrapper functions for special intrinsics are written in a compact formatting (i.e. all in one line). By default false.
- `bool alwaysBracedScopes = false`
If true, scopes are always written in braces. By default false.
- `bool newLineOpenScope = true`
If true, the '{'-braces for an open scope gets its own line. If false, braces are written like in Java coding conventions. By default true.
- `bool lineSeparation = true`
If true, auto-formatting of line separation is allowed. By default true.

6.3.1 Detailed Description

[Formatting](#) descriptor structure for the output shader.

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

- Xsc.h

6.4 Xsc::IncludeHandler Class Reference

Interface for handling new include streams.

```
#include <IncludeHandler.h>
```

Public Member Functions

- virtual `std::unique_ptr< std::istream > Include` (const `std::string &filename`, bool `useSearchPathsFirst`)
Returns an input stream for the specified filename.

Public Attributes

- `std::vector< std::string > searchPaths`
List of search paths.

6.4.1 Detailed Description

Interface for handling new include streams.

Remarks

The default implementation will read the files from an `std::ifstream`.

6.4.2 Member Function Documentation

6.4.2.1 `virtual std::unique_ptr<std::istream> Xsc::IncludeHandler::Include (const std::string & filename, bool useSearchPathsFirst) [virtual]`

Returns an input stream for the specified filename.

Parameters

in	<i>includeName</i>	Specifies the include filename.
in	<i>useSearchPathsFirst</i>	Specifies whether to first use the search paths to find the file.

Returns

Unique pointer to the new input stream.

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

- IncludeHandler.h

6.5 Xsc::IndentHandler Class Reference

Indentation handler base class.

```
#include <IndentHandler.h>
```

Public Member Functions

- **IndentHandler** (const std::string &initialIndent=std::string(2, ' '))
- void **SetIndent** (const std::string &indent)
Sets the next indentation string. By default two spaces.
- void **InclIndent** ()
Increments the indentation.
- void **DeclIndent** ()
Decrements the indentation.
- const std::string & **FullIndent** () const
Returns the current full indentation string.

6.5.1 Detailed Description

Indentation handler base class.

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

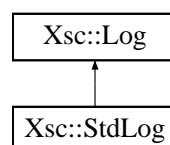
- IndentHandler.h

6.6 Xsc::Log Class Reference

[Log](#) base class.

```
#include <Log.h>
```

Inheritance diagram for Xsc::Log:



Public Member Functions

- virtual void [SubmitReport](#) (const [Report](#) &report)=0
Submits the specified report.
- void [SetIndent](#) (const std::string &indent)
Sets the next indentation string. By default two spaces.
- void [IncIndent](#) ()
Increments the indentation.
- void [DecIndent](#) ()
Decrements the indentation.

Protected Member Functions

- const std::string & [FullIndent](#) () const
Returns the current full indentation string.

6.6.1 Detailed Description

[Log](#) base class.

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

- [Log.h](#)

6.7 Xsc::NameMangling Struct Reference

Name mangling descriptor structure for shader input/output variables (also referred to as "varyings"), temporary variables, and reserved keywords.

```
#include <Xsc.h>
```

Public Attributes

- std::string [inputPrefix](#) = "xsv_"
Name mangling prefix for shader input variables. By default "xsv_".
- std::string [outputPrefix](#) = "xsv_"
Name mangling prefix for shader output variables. By default "xsv_".
- std::string [reservedWordPrefix](#) = "xsr_"
Name mangling prefix for reserved words (such as "texture", "main", "sin" etc.). By default "xsr_".
- std::string [temporaryPrefix](#) = "xst_"
Name mangling prefix for temporary variables. By default "xst_".
- std::string [namespacePrefix](#) = "xsn_"
Name mangling prefix for namespaces like structures or classes. By default "xsn_".
- bool [useAlwaysSemantics](#) = false

6.7.1 Detailed Description

Name mangling descriptor structure for shader input/output variables (also referred to as "varyings"), temporary variables, and reserved keywords.

6.7.2 Member Data Documentation

6.7.2.1 `std::string Xsc::NameMangling::inputPrefix = "xsv_"`

Name mangling prefix for shader input variables. By default "xsv_".

Remarks

This can also be empty or equal to "outputPrefix".

6.7.2.2 `std::string Xsc::NameMangling::namespacePrefix = "xsn_"`

Name mangling prefix for namespaces like structures or classes. By default "xsn_".

Remarks

This can also be empty, but if it's not empty it must not be equal to any of the other prefixes.

6.7.2.3 `std::string Xsc::NameMangling::outputPrefix = "xsv_"`

Name mangling prefix for shader output variables. By default "xsv_".

Remarks

This can also be empty or equal to "inputPrefix".

6.7.2.4 `std::string Xsc::NameMangling::reservedWordPrefix = "xsr_"`

Name mangling prefix for reserved words (such as "texture", "main", "sin" etc.). By default "xsr_".

Remarks

This must not be equal to any of the other prefixes and it must not be empty.

6.7.2.5 `std::string Xsc::NameMangling::temporaryPrefix = "xst_"`

Name mangling prefix for temporary variables. By default "xst_".

Remarks

This must not be equal to any of the other prefixes and it must not be empty.

6.7.2.6 bool Xsc::NameMangling::useAlwaysSemantics = false

If true, shader input/output variables are always renamed to their semantics, even for vertex input and fragment output. Otherwise, their original identifiers are used. By default false.

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

- Xsc.h

6.8 Xsc::Reflection::NumThreads Struct Reference

Number of threads within each work group of a compute shader.

```
#include <Reflection.h>
```

Public Attributes

- int **x** = 0
Number of shader compute threads in X dimension.
- int **y** = 0
Number of shader compute threads in Y dimension.
- int **z** = 0
Number of shader compute threads in Z dimension.

6.8.1 Detailed Description

Number of threads within each work group of a compute shader.

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

- Reflection.h

6.9 Xsc::Options Struct Reference

Structure for additional translation options.

```
#include <Xsc.h>
```

Public Attributes

- bool `optimize` = false
If true, little code optimizations are performed. By default false.
- bool `preprocessOnly` = false
If true, only the preprocessed source code will be written out. By default false.
- bool `validateOnly` = false
If true, the source code is only validated, but no output code will be generated. By default false.
- bool `allowExtensions` = false
If true, the shader output may contain GLSL extensions, if the target shader version is too low. By default false.
- bool `explicitBinding` = false
If true, explicit binding slots are enabled. By default false.
- bool `preserveComments` = false
If true, commentaries are preserved for each statement. By default false.
- bool `preferWrappers` = false
If true, intrinsics are preferred to be implemented as wrappers (instead of inlining). By default false.
- bool `unrollArrayInitializers` = false
If true, array initializations will be unrolled. By default false.
- bool `rowMajorAlignment` = false
If true, matrices have row-major alignment. Otherwise the matrices have column-major alignment. By default false.
- bool `obfuscate` = false
If true, code obfuscation is performed. By default false.
- bool `showAST` = false
If true, the AST (Abstract Syntax Tree) will be written to the log output. By default false.
- bool `showTimes` = false
If true, the timings of the different compilation processes are written to the log output. By default false.

6.9.1 Detailed Description

Structure for additional translation options.

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

- `Xsc.h`

6.10 Xsc::Reflection::ReflectionData Struct Reference

Structure for shader output statistics (e.g. texture/buffer binding points).

```
#include <Reflection.h>
```

Public Attributes

- `std::vector< std::string > macros`
All defined macros after pre-processing.
- `std::vector< BindingSlot > textures`
Texture bindings.
- `std::vector< BindingSlot > storageBuffers`
Storage buffer bindings.
- `std::vector< BindingSlot > constantBuffers`
Constant buffer bindings.
- `std::vector< BindingSlot > inputAttributes`
Shader input attributes.
- `std::vector< BindingSlot > outputAttributes`
Shader output attributes.
- `std::map< std::string, SamplerState > samplerStates`
Static sampler states (identifier, states).
- `NumThreads numThreads`
'numthreads' attribute of a compute shader.

6.10.1 Detailed Description

Structure for shader output statistics (e.g. texture/buffer binding points).

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

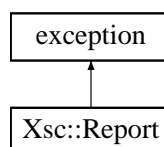
- Reflection.h

6.11 Xsc::Report Class Reference

[Report](#) exception class.

```
#include <Report.h>
```

Inheritance diagram for Xsc::Report:



Public Types

- enum `Types { Types::Info, Types::Warning, Types::Error }`
Report types enumeration.

Public Member Functions

- **Report** (const [Report](#) &)=default
- **Report** & **operator=** (const [Report](#) &)=default
- **Report** (const [Types](#) type, const std::string &message, const std::string &context="")
- **Report** (const [Types](#) type, const std::string &message, const std::string &line, const std::string &marker, const std::string &context="")
- const char * **what** () const override throw ()
Overrides the 'std::exception::what' function.
- void **TakeHints** (std::vector< std::string > &&hints)
Moves the specified hints into this report.
- [Types](#) **Type** () const
Returns the type of this report.
- const std::string & **Context** () const
Returns the context description string (e.g. a function name where the report occurred). This may also be empty.
- const std::string & **Message** () const
Returns the message string.
- const std::string & **Line** () const
Returns the line string where the report occurred. This line never has new-line characters at its end.
- const std::string & **Marker** () const
Returns the line marker string to highlight the area where the report occurred.
- const std::vector< std::string > & **GetHints** () const
Returns the list of optional hints of the report.
- bool **HasLine** () const
Returns true if this report has a line with line marker.

6.11.1 Detailed Description

[Report](#) exception class.

6.11.2 Member Enumeration Documentation

6.11.2.1 enum [Xsc::Report::Types](#) [strong]

[Report](#) types enumeration.

Enumerator

Info Standard information.

Warning Warning message.

Error Error message.

6.11.3 Member Function Documentation

6.11.3.1 bool Xsc::Report::HasLine () const [inline]

Returns true if this report has a line with line marker.

See also

[Line](#)
[Marker](#)

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

- Report.h

6.12 Xsc::Reflection::SamplerState Struct Reference

Static sampler state descriptor structure (D3D11_SAMPLER_DESC).

```
#include <Reflection.h>
```

Public Attributes

- [Filter](#) **filter** = Filter::MinMagMipLinear
- [TextureAddressMode](#) **addressU** = TextureAddressMode::Clamp
- [TextureAddressMode](#) **addressV** = TextureAddressMode::Clamp
- [TextureAddressMode](#) **addressW** = TextureAddressMode::Clamp
- float **mipLODBias** = 0.0f
- unsigned int **maxAnisotropy** = 1u
- [ComparisonFunc](#) **comparisonFunc** = ComparisonFunc::Never
- float **borderColor** [4] = { 0.0f, 0.0f, 0.0f, 0.0f }
- float **minLOD** = -std::numeric_limits<float>::max()
- float **maxLOD** = std::numeric_limits<float>::max()

6.12.1 Detailed Description

Static sampler state descriptor structure (D3D11_SAMPLER_DESC).

Remarks

All members and enumerations have the same values like the one in the "D3D11_SAMPLER_DESC" structure respectively. Thus, they can all be statically casted from and to the original D3D11 values.

See also

[https://msdn.microsoft.com/en-us/library/windows/desktop/ff476207\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ff476207(v=vs.85).aspx)

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

- Reflection.h

6.13 Xsc::ConsoleManip::ScopedColor Class Reference

Helper class for scoped color stack operations.

```
#include <ConsoleManip.h>
```

Public Member Functions

- [ScopedColor](#) (std::ostream &stream, long front)
Constructor with output stream and front color flags.
- [ScopedColor](#) (std::ostream &stream, long front, long back)
Constructor with output stream, and front- and back color flags.
- [~ScopedColor](#) ()
Destructor which will reset the previous color from the output stream.

6.13.1 Detailed Description

Helper class for scoped color stack operations.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 Xsc::ConsoleManip::ScopedColor::ScopedColor (std::ostream & *stream*, long *front*) [inline]

Constructor with output stream and front color flags.

Parameters

in, out	<i>stream</i>	Specifies the output stream for which the scope is to be changed. This is only used for Unix systems.
in	<i>front</i>	Specifies the front color flags. This can be a bitwise OR combination of the entries of the ColorFlags enumeration.

See also

[ColorFlags](#)
[PushColor\(std::ostream&, long\)](#)

6.13.2.2 Xsc::ConsoleManip::ScopedColor::ScopedColor (std::ostream & *stream*, long *front*, long *back*) [inline]

Constructor with output stream, and front- and back color flags.

Parameters

in, out	<i>stream</i>	Specifies the output stream for which the scope is to be changed. This is only used for Unix systems.
in	<i>front</i>	Specifies the front color flags. This can be a bitwise OR combination of the entries of the ColorFlags enumeration.
in	<i>back</i>	Specifies the back color flags. This can be a bitwise OR combination of the entries of the ColorFlags enumeration.

See also

[ColorFlags](#)

[PushColor\(std::ostream&, long, long\)](#)

6.13.2.3 Xsc::ConsoleManip::ScopedColor::~~ScopedColor () `[inline]`

Destructor which will reset the previous color from the output stream.

See also

[PopColor](#)

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

- ConsoleManip.h

6.14 Xsc::ScopedIndent Class Reference

Helper class for temporary indentation.

```
#include <IndentHandler.h>
```

Public Member Functions

- **ScopedIndent** ([IndentHandler](#) &handler)

6.14.1 Detailed Description

Helper class for temporary indentation.

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

- IndentHandler.h

6.15 Xsc::ShaderInput Struct Reference

Shader input descriptor structure.

```
#include <Xsc.h>
```

Public Attributes

- `std::string filename`
Specifies the filename of the input shader code. This is an optional attribute, and only a hint to the compiler.
- `std::shared_ptr< std::istream > sourceCode`
Specifies the input source code stream.
- `InputShaderVersion shaderVersion = InputShaderVersion::HLSL5`
Specifies the input shader version (e.g. `InputShaderVersion::HLSL5` for "HLSL 5"). By default `InputShaderVersion::HLSL5`.
- `ShaderTarget shaderTarget = ShaderTarget::Undefined`
Specifies the target shader (Vertex, Fragment etc.). By default `ShaderTarget::Undefined`.
- `std::string entryPoint = "main"`
Specifies the HLSL shader entry point. By default "main".
- `std::string secondaryEntryPoint`
Specifies the secondary HLSL shader entry point.
- `unsigned int warnings = 0`
Compiler warning flags. This can be a bitwise OR combination of the "Warnings" enumeration entries. By default 0.
- `IncludeHandler * includeHandler = nullptr`
Optional pointer to the implementation of the "IncludeHandler" interface. By default null.

6.15.1 Detailed Description

Shader input descriptor structure.

6.15.2 Member Data Documentation

6.15.2.1 `IncludeHandler* Xsc::ShaderInput::includeHandler = nullptr`

Optional pointer to the implementation of the "IncludeHandler" interface. By default null.

Remarks

If this is null, the default include handler will be used, which will include files with the STL input file streams.

6.15.2.2 `std::string Xsc::ShaderInput::secondaryEntryPoint`

Specifies the secondary HLSL shader entry point.

Remarks

This is only used for a Tessellation-Control Shader (alias Hull Shader) entry point, when a Tessellation-Control Shader (alias Domain Shader) is the output target. This is required to translate all Tessellation-Control attributes (i.e. "partitioning" and "outputtopology") to the Tessellation-Evaluation output shader. If this is empty, the default values for these attributes are used.

6.15.2.3 unsigned int Xsc::ShaderInput::warnings = 0

Compiler warning flags. This can be a bitwise OR combination of the "Warnings" enumeration entries. By default 0.

See also

[Warnings](#)

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

- Xsc.h

6.16 Xsc::ShaderOutput Struct Reference

Shader output descriptor structure.

```
#include <Xsc.h>
```

Public Attributes

- `std::string filename`
Specifies the filename of the output shader code. This is an optional attribute, and only a hint to the compiler.
- `std::ostream * sourceCode = nullptr`
Specifies the output source code stream. This will contain the output code. This must not be null when passed to the "CompileShader" function!
- `OutputShaderVersion shaderVersion = OutputShaderVersion::GLSL`
Specifies the output shader version. By default [OutputShaderVersion::GLSL](#) (to auto-detect minimum required version).
- `std::vector< VertexSemantic > vertexSemantics`
Optional list of vertex semantic layouts, to bind a vertex attribute (semantic name) to a location index (only used when 'explicitBinding' is true).
- `Options options`
Additional options to configure the code generation.
- `Formatting formatting`
Output code formatting descriptor.
- `NameMangling nameMangling`
Specifies the options for name mangling.

6.16.1 Detailed Description

Shader output descriptor structure.

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

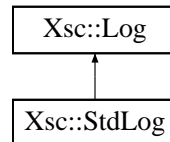
- Xsc.h

6.17 Xsc::StdLog Class Reference

Standard output log (uses std::cout to submit a report).

```
#include <Log.h>
```

Inheritance diagram for Xsc::StdLog:



Public Member Functions

- void [SumitReport](#) (const [Report](#) &report) override
Implements the base class interface.
- void [PrintAll](#) (bool verbose=true)
Prints all submitted reports to the standard output.

Additional Inherited Members

6.17.1 Detailed Description

Standard output log (uses std::cout to submit a report).

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

- [Log.h](#)

6.18 Xsc::VertexSemantic Struct Reference

Vertex shader semantic (or rather attribute) layout structure.

```
#include <Xsc.h>
```

Public Attributes

- std::string [semantic](#)
Specifies the shader semantic (or rather attribute).
- int [location](#)
Specifies the binding location.

6.18.1 Detailed Description

Vertex shader semantic (or rather attribute) layout structure.

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

- Xsc.h

6.19 Xsc::Warnings Struct Reference

Compiler warning flags.

```
#include <Xsc.h>
```

Public Types

- enum : unsigned int {
[Basic](#) = (1 << 0), [Syntax](#) = (1 << 1), [PreProcessor](#) = (1 << 2), [UnusedVariables](#) = (1 << 3),
[EmptyStatementBody](#) = (1 << 4), [ImplicitTypeConversions](#) = (1 << 5), [DeclarationShadowing](#) = (1 << 6),
[UnlocatedObjects](#) = (1 << 7),
[RequiredExtensions](#) = (1 << 8), [CodeReflection](#) = (1 << 9), [All](#) = (~0u) }

6.19.1 Detailed Description

Compiler warning flags.

6.19.2 Member Enumeration Documentation

6.19.2.1 anonymous enum : unsigned int

Enumerator

- Basic** Warning for basic issues (control path, disabled code etc.).
- Syntax** Warning for syntactic issues.
- PreProcessor** Warning for pre-processor issues.
- UnusedVariables** Warning for unused variables.
- EmptyStatementBody** Warning for statements with empty body.
- ImplicitTypeConversions** Warning for specific implicit type conversions.
- DeclarationShadowing** Warning for declarations that shadow a previous local (e.g. for-loops or variables in class hierarchy).
- UnlocatedObjects** Warning for optional objects that where not found.
- RequiredExtensions** Warning for required extensions in the output code.
- CodeReflection** Warning for issues during code reflection.
- All** All warnings.

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

- Xsc.h

6.20 XscBindingSlot Struct Reference

Binding slot of textures, constant buffers, and fragment targets.

```
#include <ReflectionC.h>
```

Public Attributes

- const char * [ident](#)
Identifier of the binding point.
- int [location](#)
Zero based binding point or location. If this is -1, the location has not been set explicitly.

6.20.1 Detailed Description

Binding slot of textures, constant buffers, and fragment targets.

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

- ReflectionC.h

6.21 XscFormatting Struct Reference

Formatting descriptor structure for the output shader.

```
#include <XscC.h>
```

Public Attributes

- const char * [indent](#)
Indentation string for code generation. By default 4 spaces.
- bool [blanks](#)
If true, blank lines are allowed. By default true.
- bool [lineMarks](#)
If true, line marks are allowed. By default false.
- bool [compactWrappers](#)
If true, wrapper functions for special intrinsics are written in a compact formatting (i.e. all in one line). By default false.
- bool [alwaysBracedScopes](#)
If true, scopes are always written in braces. By default false.
- bool [newLineOpenScope](#)
If true, the '{'-braces for an open scope gets its own line. If false, braces are written like in Java coding conventions. By default true.
- bool [lineSeparation](#)
If true, auto-formatting of line separation is allowed. By default true.

6.21.1 Detailed Description

Formatting descriptor structure for the output shader.

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

- XscC.h

6.22 XscIncludeHandler Struct Reference

Include handler structure.

```
#include <IncludeHandlerC.h>
```

Public Attributes

- XSC_PFN_HANDLE_INCLUDE [handleIncludePfn](#)
Function pointer to handle the '#include'-directives.
- const char ** [searchPaths](#)
Pointer to an array of search paths. This must be either NULL, point to an array where the last entry is always NULL.

6.22.1 Detailed Description

Include handler structure.

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

- IncludeHandlerC.h

6.23 XscLog Struct Reference

Output log structure.

```
#include <LogC.h>
```

Public Attributes

- XSC_PFN_HANDLE_REPORT [handleReportPfn](#)
Function pointer to handle compiler reports.

6.23.1 Detailed Description

Output log structure.

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

- LogC.h

6.24 XscNameMangling Struct Reference

Name mangling descriptor structure for shader input/output variables (also referred to as "varyings"), temporary variables, and reserved keywords.

```
#include <XscC.h>
```

Public Attributes

- const char * [inputPrefix](#)
Name mangling prefix for shader input variables. By default "xsv_".
- const char * [outputPrefix](#)
Name mangling prefix for shader output variables. By default "xsv_".
- const char * [reservedWordPrefix](#)
Name mangling prefix for reserved words (such as "texture", "main", "sin" etc.). By default "xsr_".
- const char * [temporaryPrefix](#)
Name mangling prefix for temporary variables. By default "xst_".
- const char * [namespacePrefix](#)
Name mangling prefix for namespaces like structures or classes. By default "xsn_".
- bool [useAlwaysSemantics](#)

6.24.1 Detailed Description

Name mangling descriptor structure for shader input/output variables (also referred to as "varyings"), temporary variables, and reserved keywords.

6.24.2 Member Data Documentation

6.24.2.1 const char* XscNameMangling::inputPrefix

Name mangling prefix for shader input variables. By default "xsv_".

Remarks

This can also be empty or equal to "outputPrefix".

6.24.2.2 const char* XscNameMangling::namespacePrefix

Name mangling prefix for namespaces like structures or classes. By default "xsn_".

Remarks

This can also be empty, but if it's not empty it must not be equal to any of the other prefixes.

6.24.2.3 `const char* XscNameMangling::outputPrefix`

Name mangling prefix for shader output variables. By default "xsv_".

Remarks

This can also be empty or equal to "inputPrefix".

6.24.2.4 `const char* XscNameMangling::reservedWordPrefix`

Name mangling prefix for reserved words (such as "texture", "main", "sin" etc.). By default "xsr_".

Remarks

This must not be equal to any of the other prefixes and it must not be empty.

6.24.2.5 `const char* XscNameMangling::temporaryPrefix`

Name mangling prefix for temporary variables. By default "xst_".

Remarks

This must not be equal to any of the other prefixes and it must not be empty.

6.24.2.6 `bool XscNameMangling::useAlwaysSemantics`

If true, shader input/output variables are always renamed to their semantics, even for vertex input and fragment output. Otherwise, their original identifiers are used. By default false.

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

- XscC.h

6.25 XscNumThreads Struct Reference

Number of threads within each work group of a compute shader.

```
#include <ReflectionC.h>
```

Public Attributes

- `int x`
- `int y`
- `int z`

6.25.1 Detailed Description

Number of threads within each work group of a compute shader.

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

- ReflectionC.h

6.26 XscOptions Struct Reference

Structure for additional translation options.

```
#include <XscC.h>
```

Public Attributes

- bool [optimize](#)
If true, little code optimizations are performed. By default false.
- bool [preprocessOnly](#)
If true, only the preprocessed source code will be written out. By default false.
- bool [validateOnly](#)
If true, the source code is only validated, but no output code will be generated. By default false.
- bool [allowExtensions](#)
If true, the shader output may contain GLSL extensions, if the target shader version is too low. By default false.
- bool [explicitBinding](#)
If true, explicit binding slots are enabled. By default false.
- bool [preserveComments](#)
If true, commentaries are preserved for each statement. By default false.
- bool [preferWrappers](#)
If true, intrinsics are preferred to be implemented as wrappers (instead of inlining). By default false.
- bool [unrollArrayInitializers](#)
If true, array initializations will be unrolled. By default false.
- bool [rowMajorAlignment](#)
If true, matrices have row-major alignment. Otherwise the matrices have column-major alignment. By default false.
- bool [obfuscate](#)
If true, code obfuscation is performed. By default false.
- bool [showAST](#)
If true, the AST (Abstract Syntax Tree) will be written to the log output. By default false.
- bool [showTimes](#)
If true, the timings of the different compilation processes are written to the log output. By default false.

6.26.1 Detailed Description

Structure for additional translation options.

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

- XscC.h

6.27 XscReflectionData Struct Reference

Structure for shader output statistics (e.g. texture/buffer binding points).

```
#include <ReflectionC.h>
```

Public Attributes

- const char ** [macros](#)
All defined macros after pre-processing.
- size_t [macrosCount](#)
Number of elements in 'macros'.
- const struct [XscBindingSlot](#) * [textures](#)
Texture bindings.
- size_t [texturesCount](#)
Number of elements in 'textures'.
- const struct [XscBindingSlot](#) * [storageBuffers](#)
Storage buffer bindings.
- size_t [storageBuffersCount](#)
Number of elements in 'storageBuffers'.
- const struct [XscBindingSlot](#) * [constantBuffers](#)
Constant buffer bindings.
- size_t [constantBufferCounts](#)
Number of elements in 'constantBuffers'.
- const struct [XscBindingSlot](#) * [inputAttributes](#)
Shader input attributes.
- size_t [inputAttributesCount](#)
Number of elements in 'inputAttributes'.
- const struct [XscBindingSlot](#) * [outputAttributes](#)
Shader output attributes.
- size_t [outputAttributesCount](#)
Number of elements in 'outputAttributes'.
- const struct [XscSamplerState](#) * [samplerStates](#)
Static sampler states (identifier, states).
- size_t [samplerStatesCount](#)
Number of elements in 'samplerStates'.
- struct [XscNumThreads](#) [numThreads](#)
'numthreads' attribute of a compute shader.

6.27.1 Detailed Description

Structure for shader output statistics (e.g. texture/buffer binding points).

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

- [ReflectionC.h](#)

6.28 XscReport Struct Reference

Report structure for warning and error messages.

```
#include <ReportC.h>
```

Public Attributes

- enum XscReportType **type**
Specifies the report type.
- const char * **context**
Specifies the context description string (e.g. a function name where the report occurred). This may also be NULL.
- const char * **message**
Specifies the message string.
- const char * **line**
Specifies the line string where the report occurred. This line never has new-line characters at its end. This may also be NULL.
- const char * **marker**
Specifies the line marker string to highlight the area where the report occurred. This may also be NULL.
- const char ** **hints**
Specifies the list of optional hints of the report. This may also be NULL.
- size_t **hintsCount**
Specifies the number of hints. If 'hints' is NULL, this is 0.

6.28.1 Detailed Description

Report structure for warning and error messages.

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

- ReportC.h

6.29 XscSamplerState Struct Reference

Static sampler state descriptor structure (D3D11_SAMPLER_DESC).

```
#include <ReflectionC.h>
```

Public Attributes

- const char * **ident**
- enum XscFilter **filter**
- enum XscTextureAddressMode **addressU**
- enum XscTextureAddressMode **addressV**
- enum XscTextureAddressMode **addressW**
- float **mipLODBias**
- unsigned int **maxAnisotropy**
- enum XscComparisonFunc **comparisonFunc**
- float **borderColor** [4]
- float **minLOD**
- float **maxLOD**

6.29.1 Detailed Description

Static sampler state descriptor structure (D3D11_SAMPLER_DESC).

Remarks

All members and enumerations have the same values like the one in the "D3D11_SAMPLER_DESC" structure respectively. Thus, they can all be statically casted from and to the original D3D11 values.

See also

[https://msdn.microsoft.com/en-us/library/windows/desktop/ff476207\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ff476207(v=vs.85).aspx)

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

- ReflectionC.h

6.30 XscShaderInput Struct Reference

Shader input descriptor structure.

```
#include <XscC.h>
```

Public Attributes

- const char * [filename](#)
Specifies the filename of the input shader code. This is an optional attribute, and only a hint to the compiler. By default NULL.
- const char * [sourceCode](#)
Specifies the input source code. This must not be null when passed to the "XscCompileShader" function!
- enum XscInputShaderVersion [shaderVersion](#)
Specifies the input shader version (e.g. XscEInputHLSL5 for "HLSL 5"). By default XscEInputHLSL5.
- enum XscShaderTarget [shaderTarget](#)
Specifies the target shader (Vertex, Fragment etc.). By default XscUndefinedShader.
- const char * [entryPoint](#)
Specifies the HLSL shader entry point. By default "main".
- const char * [secondaryEntryPoint](#)
Specifies the secondary HLSL shader entry point. By default NULL.
- unsigned int [warnings](#)
Compiler warning flags. This can be a bitwise OR combination of the "XscWarnings" enumeration entries. By default 0.
- struct [XscIncludeHandler](#) [includeHandler](#)
Include handler member which contains a function pointer to handle '#include'-directives.

6.30.1 Detailed Description

Shader input descriptor structure.

6.30.2 Member Data Documentation

6.30.2.1 `const char* XscShaderInput::secondaryEntryPoint`

Specifies the secondary HLSL shader entry point. By default NULL.

Remarks

This is only used for a Tessellation-Control Shader (alias Hull Shader) entry point, when a Tessellation-
Control Shader (alias Domain Shader) is the output target. This is required to translate all Tessellation-Control attributes (i.e. "partitioning" and "outputtopology") to the Tessellation-Evaluation output shader. If this is empty, the default values for these attributes are used.

6.30.2.2 `unsigned int XscShaderInput::warnings`

Compiler warning flags. This can be a bitwise OR combination of the "XscWarnings" enumeration entries. By default 0.

See also

Warnings

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

- XscC.h

6.31 XscShaderOutput Struct Reference

Shader output descriptor structure.

```
#include <XscC.h>
```

Public Attributes

- `const char * filename`
Specifies the filename of the output shader code. This is an optional attribute, and only a hint to the compiler.
- `const char ** sourceCode`
Specifies the output source code. This will contain the output code. This must not be null when passed to the "XscCompileShader" function!
- `enum XscOutputShaderVersion shaderVersion`
Specifies the output shader version. By default XscEOutputGLSL (to auto-detect minimum required version).
- `const struct XscVertexSemantic * vertexSemantics`
Optional list of vertex semantic layouts, to bind a vertex attribute (semantic name) to a location index (only used when 'explicitBinding' is true). By default NULL.
- `size_t vertexSemanticsCount`
Number of elements the 'vertexSemantics' member points to. By default 0.
- `struct XscOptions options`
Additional options to configure the code generation.
- `struct XscFormatting formatting`
Output code formatting descriptor.
- `struct XscNameMangling nameMangling`
Specifies the options for name mangling.

6.31.1 Detailed Description

Shader output descriptor structure.

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

- XscC.h

6.32 XscVertexSemantic Struct Reference

Vertex shader semantic (or rather attribute) layout structure.

```
#include <XscC.h>
```

Public Attributes

- const char * [semantic](#)
Specifies the shader semantic (or rather attribute).
- int [location](#)
Specifies the binding location.

6.32.1 Detailed Description

Vertex shader semantic (or rather attribute) layout structure.

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

- XscC.h

Index

- ~ScopedColor
 - Xsc::ConsoleManip::ScopedColor, [27](#)
- All
 - Xsc::Warnings, [31](#)
- Basic
 - Xsc::Warnings, [31](#)
- Black
 - Xsc::ConsoleManip::ColorFlags, [16](#)
- Blue
 - Xsc::ConsoleManip::ColorFlags, [16](#)
- CodeReflection
 - Xsc::Warnings, [31](#)
- CompileShader
 - Xsc, [12](#)
- ComputeShader
 - Xsc, [12](#)
- Cyan
 - Xsc::ConsoleManip::ColorFlags, [16](#)
- DeclarationShadowing
 - Xsc::Warnings, [31](#)
- ESSL100
 - Xsc, [12](#)
- ESSL300
 - Xsc, [12](#)
- ESSL310
 - Xsc, [12](#)
- ESSL320
 - Xsc, [12](#)
- ESSL
 - Xsc, [11](#), [12](#)
- EmptyStatementBody
 - Xsc::Warnings, [31](#)
- Error
 - Xsc::Report, [24](#)
- FragmentShader
 - Xsc, [12](#)
- GLSL110
 - Xsc, [11](#)
- GLSL120
 - Xsc, [11](#)
- GLSL130
 - Xsc, [11](#)
- GLSL140
 - Xsc, [11](#)
- GLSL150
 - Xsc, [11](#)
- GLSL330
 - Xsc, [11](#)
- GLSL400
 - Xsc, [11](#)
- GLSL410
 - Xsc, [11](#)
- GLSL420
 - Xsc, [12](#)
- GLSL430
 - Xsc, [12](#)
- GLSL440
 - Xsc, [12](#)
- GLSL450
 - Xsc, [12](#)
- GLSL
 - Xsc, [11](#), [12](#)
- GeometryShader
 - Xsc, [12](#)
- Gray
 - Xsc::ConsoleManip::ColorFlags, [16](#)
- Green
 - Xsc::ConsoleManip::ColorFlags, [16](#)
- HLSL3
 - Xsc, [11](#)
- HLSL4
 - Xsc, [11](#)
- HLSL5
 - Xsc, [11](#)
- HasLine
 - Xsc::Report, [25](#)
- ImplicitTypeConversions
 - Xsc::Warnings, [31](#)
- Include
 - Xsc::IncludeHandler, [17](#)
- includeHandler
 - Xsc::ShaderInput, [28](#)
- Info
 - Xsc::Report, [24](#)
- inputPrefix
 - Xsc::NameMangling, [20](#)
 - XscNameMangling, [34](#)
- InputShaderVersion
 - Xsc, [11](#)
- Intens
 - Xsc::ConsoleManip::ColorFlags, [16](#)

- namespacePrefix
 - Xsc::NameMangling, [20](#)
 - XscNameMangling, [34](#)
- outputPrefix
 - Xsc::NameMangling, [20](#)
 - XscNameMangling, [34](#)
- OutputShaderVersion
 - Xsc, [11](#)
- Pink
 - Xsc::ConsoleManip::ColorFlags, [16](#)
- PreProcessor
 - Xsc::Warnings, [31](#)
- Red
 - Xsc::ConsoleManip::ColorFlags, [16](#)
- RequiredExtensions
 - Xsc::Warnings, [31](#)
- reservedWordPrefix
 - Xsc::NameMangling, [20](#)
 - XscNameMangling, [35](#)
- ScopedColor
 - Xsc::ConsoleManip::ScopedColor, [26](#)
- secondaryEntryPoint
 - Xsc::ShaderInput, [28](#)
 - XscShaderInput, [40](#)
- ShaderTarget
 - Xsc, [12](#)
- Syntax
 - Xsc::Warnings, [31](#)
- temporaryPrefix
 - Xsc::NameMangling, [20](#)
 - XscNameMangling, [35](#)
- TessellationControlShader
 - Xsc, [12](#)
- TessellationEvaluationShader
 - Xsc, [12](#)
- Types
 - Xsc::Report, [24](#)
- Undefined
 - Xsc, [12](#)
- UnlocatedObjects
 - Xsc::Warnings, [31](#)
- UnusedVariables
 - Xsc::Warnings, [31](#)
- useAlwaysSemantics
 - Xsc::NameMangling, [20](#)
 - XscNameMangling, [35](#)
- VKSL450
 - Xsc, [12](#)
- VKSL
 - Xsc, [11](#), [12](#)
- VertexShader
 - Xsc, [12](#)
- Warning
 - Xsc::Report, [24](#)
- warnings
 - Xsc::ShaderInput, [28](#)
 - XscShaderInput, [40](#)
- White
 - Xsc::ConsoleManip::ColorFlags, [16](#)
- Xsc, [9](#)
 - CompileShader, [12](#)
 - ComputeShader, [12](#)
 - ESSL100, [12](#)
 - ESSL300, [12](#)
 - ESSL310, [12](#)
 - ESSL320, [12](#)
 - ESSL, [11](#), [12](#)
 - FragmentShader, [12](#)
 - GLSL110, [11](#)
 - GLSL120, [11](#)
 - GLSL130, [11](#)
 - GLSL140, [11](#)
 - GLSL150, [11](#)
 - GLSL330, [11](#)
 - GLSL400, [11](#)
 - GLSL410, [11](#)
 - GLSL420, [12](#)
 - GLSL430, [12](#)
 - GLSL440, [12](#)
 - GLSL450, [12](#)
 - GLSL, [11](#), [12](#)
 - GeometryShader, [12](#)
 - HLSL3, [11](#)
 - HLSL4, [11](#)
 - HLSL5, [11](#)
 - InputShaderVersion, [11](#)
 - OutputShaderVersion, [11](#)
 - ShaderTarget, [12](#)
 - TessellationControlShader, [12](#)
 - TessellationEvaluationShader, [12](#)
 - Undefined, [12](#)
 - VKSL450, [12](#)
 - VKSL, [11](#), [12](#)
 - VertexShader, [12](#)
- Xsc::ConsoleManip, [13](#)
- Xsc::ConsoleManip::ColorFlags, [15](#)
 - Black, [16](#)
 - Blue, [16](#)
 - Cyan, [16](#)
 - Gray, [16](#)
 - Green, [16](#)
 - Intens, [16](#)
 - Pink, [16](#)
 - Red, [16](#)
 - White, [16](#)
 - Yellow, [16](#)
- Xsc::ConsoleManip::ScopedColor, [26](#)
 - ~ScopedColor, [27](#)
 - ScopedColor, [26](#)
- Xsc::Formatting, [16](#)

- Xsc::IncludeHandler, [17](#)
 - Include, [17](#)
- Xsc::IndentHandler, [18](#)
- Xsc::Log, [18](#)
- Xsc::NameMangling, [19](#)
 - inputPrefix, [20](#)
 - namespacePrefix, [20](#)
 - outputPrefix, [20](#)
 - reservedWordPrefix, [20](#)
 - temporaryPrefix, [20](#)
 - useAlwaysSemantics, [20](#)
- Xsc::Options, [21](#)
- Xsc::Reflection, [14](#)
- Xsc::Reflection::BindingSlot, [15](#)
- Xsc::Reflection::NumThreads, [21](#)
- Xsc::Reflection::ReflectionData, [22](#)
- Xsc::Reflection::SamplerState, [25](#)
- Xsc::Report, [23](#)
 - Error, [24](#)
 - HasLine, [25](#)
 - Info, [24](#)
 - Types, [24](#)
 - Warning, [24](#)
- Xsc::ScopedIndent, [27](#)
- Xsc::ShaderInput, [27](#)
 - includeHandler, [28](#)
 - secondaryEntryPoint, [28](#)
 - warnings, [28](#)
- Xsc::ShaderOutput, [29](#)
- Xsc::StdLog, [30](#)
- Xsc::VertexSemantic, [30](#)
- Xsc::Warnings, [31](#)
 - All, [31](#)
 - Basic, [31](#)
 - CodeReflection, [31](#)
 - DeclarationShadowing, [31](#)
 - EmptyStatementBody, [31](#)
 - ImplicitTypeConversions, [31](#)
 - PreProcessor, [31](#)
 - RequiredExtensions, [31](#)
 - Syntax, [31](#)
 - UnlocatedObjects, [31](#)
 - UnusedVariables, [31](#)
- XscBindingSlot, [32](#)
- XscFormatting, [32](#)
- XscIncludeHandler, [33](#)
- XscLog, [33](#)
- XscNameMangling, [34](#)
 - inputPrefix, [34](#)
 - namespacePrefix, [34](#)
 - outputPrefix, [34](#)
 - reservedWordPrefix, [35](#)
 - temporaryPrefix, [35](#)
 - useAlwaysSemantics, [35](#)
- XscNumThreads, [35](#)
- XscOptions, [36](#)
- XscReflectionData, [37](#)
- XscReport, [38](#)
- XscSamplerState, [38](#)
- XscShaderInput, [39](#)
 - secondaryEntryPoint, [40](#)
 - warnings, [40](#)
- XscShaderOutput, [40](#)
- XscVertexSemantic, [41](#)
- Yellow
 - Xsc::ConsoleManip::ColorFlags, [16](#)