Standards

Naming

Capitalization

Type	Style	Example
Code File (.h/.cpp/.hpp)	Lowercase Snake	dynamic_array.cpp
Class	Pascal Case	DynamicArray
Functio	Camel Case	AddElement
Enumeration	Pascal Case	BodyColors
Enumeration Value	Uppercase Snake	LIGHT_GRAY
Macro Constant (or reusable constants in general)	Uppercase Snake	DEGREES_IN_RADIANS
Macro Function	Camel Case	macroFunc(X, Y)
Private Data Members	Camel Case	d_requiredStrength
Public Data Members	Camel Case	lifeStyle
Namespace	Pascal Case	StevensDev
Section Header Comment	Uppercase	// FREE OPERATORS
Include Guard	Uppercase Snake	INCLUDED_LIGHT
Other File	Lowercase Snake	standards.odt

Naming Conventions

Type	Style	Example(s)
Class	Noun	GlobalLight
Pure Virtual Class	Noun/Adjective	Light, Cloneable
Data Member	Noun (unless boolean)	d_objectMatrix
Functions	Verb/Action statement (unless boolean)	rotate
Enumeration	Plural Noun	Colors, BufferTypes
Enumeration Value	Singular Noun of Enumeration Type	NAVY_BLUE
Namespace	Generic/Group Identifier	StevensDev
Private Data Member	Prefix with d_	d_weapon
Boolean and Boolean Functions (Non Accessors) Variable	Question form, must be prefixed with is, was, should, can, etc.	IsAvailable, wasCreated, shouldExecute

Constants	Noun/Functional Descriptor	PI, DEGREES_TO_RADIANS
Data Accessor	Name same as the data field but without the d_	time
Data Mutator	Prefix accessor name with set then capitalize accordingly	setTime

Wherever nouns may be used noun clauses may also be used provided they serve as a succinct identifier.

Additional Naming Conventions

- Primitive type specifics should generally never be used in a variable name.
    - i.e. strType, intSize, typeStr, myStringName are unacceptable

Whitespace and Braces

General

- Indentation waterfalls
- Lines must wrap if they exceed the 80 character limit
- Wrapped lines must be indented at least 4 spaces unless it would contradict the line's alignment rules
- Conditional statements, switch statements, and try/catch statements must use curly braces.
- Conditional statement, switch statement, and try/catch keywords must appear on a new line after braces
- The inline keyword must appear on its own line

Specifics

Case	Example
Interior to parenthesis	(1 / rollDice(6)) * ((getBaseDamage())
After comma	d_x(4), d_y(8)
After colon	for (i = 0; i < size; ++i)
Before if, for, while, switch, catch parenthesis	if (!wasFound)

Around operators (not including urnary and pointer-to-member)*	int x = 5 + (y - z);
	int t = item->getSize();

Around Base Class Colon	class Bar : public FooBase
Around Constructor Initialization List Colon	Point2D() : d_x(4), d_y (5)
After * in pointer declarations	char* argv[]
Before & in declarations	inline T const &Min(T const &a)

After right brace in Structures	} foo_t;
After double forward-slash comment opening	// This is a comment.
Case	**Indentation**
Blank line**	None
Namespace Members	None
Class Members	4 spaces
Class/Structure Visibility Keywords	2 spaces
Plain Structure Members	4 spaces
Lambda Body	4 spaces
Preprocessor Directive	None
Member Documentation Comment	2 spaces
Conditional, Switch, Case, Try/Catch Body	4 spaces
Line Continuation	4 spaces
Case	**Wrapping and Alignment**
Curly Braces (Opening and Closing)	Vertically aligned
Base Class List	Place : on new line if long Align subsequent line wraps with first
Constructor Initializer List	Place : on new line if long Align with first item when multi-line
Chained Method Calls	Align when multi-line
Function Parameters	Aligned when multi-line
Function Call Arguments	Aligned when multi-line
Template Parameters	Aligned when multi-line
Template Instantiation Parameters	Aligned when multi-line
For Statement	Aligned when multi-line
Enumeration Constants	Aligned with one item per line if multi-line
General Statements	Aligned when multi-line

* If a ternary operator is just ?: there is no space between the two.

** Blank lines must be completely blank and have no indentation.

Comments

General

There must be in-line comments to provide an explanation for all non-obvious operations. However, the code should not be over-commented either. Not all code requires an explanation so it shouldn't it be provided.

Single Line Comments

- Single line comments must be be used for in-line documentation.
- There must be a space between the // and the comment.
- All non-trivial comments must be a well-formed English and end in a period.

Section Header

- Section headers must entirely uppercase

Multi-Line Comments

- Must use // for multi-line comments
- In general multi-line comments should be avoided except for header documentation

Documentation

All header files must contain post-declaration documentation for all functions. Function documentation must provide use, expected, and unexpected behavior details.

Organization

Class Hierarchy

- Classes may only sub-class exactly one class
- Classes may implement as many “*interfaces*” (pure-virtual classes) as they like
- Class hierarchy may never exceed more than 5 layers

Directory Structure

Directory	**Content**
/build*	Build cache files Output binaries Output libraries
/docs	Portable documentation files
/docs/src	Client dependent documentation files (.odt/.docx)
/ext	External libraries (source)
/include	Additional includes

/libs	Libraries
/src	Main project source files
/test	Unit test source files

* The build directory is not synced on the git repository.

Other

General

- Use well-formed English where appropriate
- No magic numbers
- 0 or nullptr must be used in place of NULL
- No direct heap allocations unless absolutely necessary
- Allocations must be wrapped within an appropriate container class
- Namespaces must have a // End nspsc NAME after the closing curly brace
- Using namespaces cannot be used in the global namespace
- Files must start with a comment that states the name of the file
  - i.e. // build.cpp

Includes

- The first include of a cpp file must be the associated header
- Includes must be listed in alphabetical order
- Includes must be used by the file they are listed in
- No cyclical dependencies

Classes and Structures

- Non-trivial classes must implement orthodox canonical form or appropriately control the copy constructor
- The private section access modifier must appear before the public section access modifier
- The protected access modifier may not be used
- Header declarations cannot contain definitions within the body of a class declaration

- Headers may only contain definitions of inline and free functions (for non-template classes)
- Headers must contain include guards (#pragma once is not a replacement)
- Every class must implement an Ostream insertion operator (<<) that writes a valid json string
    - Must be implemented as a free operator, not a friend

Functions

- Public utility functions must be implemented as static functions interior to a struct
- Helper functions, internal structs, etc. not meant to be visible must use anonymous namespaces unless a private struct makes more sense
- Only pass pointers if the value is mutable
- Primitives should be passed as a copy, const is not necessary
- Non-primitives not meant to be mutated must be passed by a const reference
- Functions that do not modify internal state must be marked with a post-fix const
- Early-out clauses must appear at the beginning of a function definition

End Notes

As this project is still in the early stages of development this document is subject to change.