Photon Common Library

2.0.0-beta

Generated by Doxygen 1.11.0

1 Photon Common Library 1

2 Namespace Index 3

2.1 Namespace List . . . 3

3 Concept Index 5

3.1 Concepts . . . 5

4 Hierarchical Index 7

4.1 Class Hierarchy . . . 7

5 Class Index 9

5.1 Class List . . . 9

6 File Index 11

6.1 File List . . . 11

7 Namespace Documentation 13

7.1 ph Namespace Reference . . . 13

7.1.1 Detailed Description . . . 16

7.1.2 Typedef Documentation . . . 16

7.1.2.1 Exception . . . 16

7.1.2.2 float32 . . . 16

7.1.2.3 float64 . . . 16

7.1.2.4 hiInteger . . . 16

7.1.2.5 hiReal . . . 16

7.1.2.6 int16 . . . 16

7.1.2.7 int16f . . . 16

7.1.2.8 int32 . . . 16

7.1.2.9 int32f . . . 17

7.1.2.10 int64 . . . 17

7.1.2.11 int64f . . . 17

7.1.2.12 int8 . . . 17

7.1.2.13 int8f . . . 17

7.1.2.14 integer . . . 17

7.1.2.15 LogHandler . . . 17

7.1.2.16 real . . . 17

7.1.2.17 StdUnorderedStringSet . . . 17

7.1.2.18 TAlignedMemoryUniquePtr . . . 18

7.1.2.19 TStdUnorderedStringMap . . . 18

7.1.2.20 uint16 . . . 18

7.1.2.21 uint16f . . . 18

7.1.2.22 uint32 . . . 18

7.1.2.23 uint32f . . . 18

7.1.2.24 uint64 . . . 18
7.1.2.25 uint64f . . . 18
7.1.2.26 uint8 . . . 18
7.1.2.27 uint8f . . . 19
7.1.3 Enumeration Type Documentation . . . 19
7.1.3.1 ELogLevel . . . 19
7.1.4 Function Documentation . . . 20
7.1.4.1 debug_break() . . . 20
7.1.4.2 from_bytes() . . . 20
7.1.4.3 get_core_log_groups() . . . 20
7.1.4.4 is_once() . . . 20
7.1.4.5 lossless_cast() [1/2] . . . 20
7.1.4.6 lossless_cast() [2/2] . . . 21
7.1.4.7 lossless_float_cast() . . . 21
7.1.4.8 lossless_integer_cast() . . . 21
7.1.4.9 make_aligned_memory() . . . 21
7.1.4.10 make_array() . . . 22
7.1.4.11 obtain_stack_trace() . . . 22
7.1.4.12 operator"""_r() . . . 22
7.1.4.13 PH_DECLARE_LOG_GROUP() . . . 22
7.1.4.14 PH_DEFINE_LOG_GROUP() . . . 22
7.1.4.15 reverse_bytes() . . . 22
7.1.4.16 sizeof_in_bits() . . . 22
7.1.4.17 start_implicit_lifetime_as() . . . 23
7.1.4.18 start_implicit_lifetime_as_array() . . . 23
7.1.4.19 throw_formatted() . . . 23
7.1.4.20 to_bytes() . . . 23
7.2 ph::detail Namespace Reference . . . 23
7.2.1 Detailed Description . . . 24
7.2.2 Function Documentation . . . 24
7.2.2.1 allocate_aligned_memory() . . . 24
7.2.2.2 free_aligned_memory() . . . 25
7.2.2.3 make_array() . . . 25
7.2.2.4 on_assertion_failed() . . . 25
7.2.2.5 output_assertion_message() . . . 25
7.2.2.6 output_not_implemented_warning() . . . 25
7.2.3 Variable Documentation . . . 26
7.2.3.1 DEPENDENT_FALSE . . . 26
7.3 ph::detail::core_logging Namespace Reference . . . 26
7.3.1 Detailed Description . . . 26
7.3.2 Function Documentation . . . 26
7.3.2.1 add_log_group() . . . 26

7.3.2.2 exit() . . . 27
7.3.2.3 get_logger() . . . 27
7.3.2.4 init() . . . 27
7.3.2.5 log_to_logger() . . . 27
7.4 ph::detail::stats Namespace Reference . . . 27
7.5 ph::math Namespace Reference . . . 28
7.5.1 Function Documentation . . . 28
7.5.1.1 ceil_div() . . . 28
7.5.1.2 is_power_of_2() . . . 28
7.5.1.3 next_multiple() . . . 28
7.5.1.4 next_power_of_2_multiple() . . . 29
7.6 ph::os Namespace Reference . . . 29
7.6.1 Enumeration Type Documentation . . . 29
7.6.1.1 EWindowsVersion . . . 29
7.6.2 Function Documentation . . . 30
7.6.2.1 get_executable_path() . . . 30
7.6.2.2 get_L1_cache_line_size_in_bytes() . . . 30
7.6.2.3 get_windows_version() . . . 30
7.7 ph::string_utils Namespace Reference . . . 30
7.7.1 Detailed Description . . . 32
7.7.2 Enumeration Type Documentation . . . 32
7.7.2.1 EWhitespace . . . 32
7.7.3 Function Documentation . . . 32
7.7.3.1 AZ_to_az() [1/2] . . . 32
7.7.3.2 AZ_to_az() [2/2] . . . 32
7.7.3.3 az_to_AZ() [1/2] . . . 33
7.7.3.4 az_to_AZ() [2/2] . . . 33
7.7.3.5 cut_ends() . . . 33
7.7.3.6 cut_head() . . . 33
7.7.3.7 cut_tail() . . . 34
7.7.3.8 erase_all() . . . 34
7.7.3.9 get_whitespaces() . . . 34
7.7.3.10 has_any_of() . . . 34
7.7.3.11 has_none_of() . . . 35
7.7.3.12 is_whitespace() . . . 35
7.7.3.13 next_token() . . . 35
7.7.3.14 parse_float() . . . 35
7.7.3.15 parse_int() . . . 35
7.7.3.16 parse_number() . . . 36
7.7.3.17 repeat() . . . 36
7.7.3.18 stringify_float() . . . 36
7.7.3.19 stringify_int() . . . 36

7.7.3.20 stringify_int_alphabetic() . . . 37
7.7.3.21 stringify_number() [1/3] . . . 37
7.7.3.22 stringify_number() [2/3] . . . 38
7.7.3.23 stringify_number() [3/3] . . . 38
7.7.3.24 trim() . . . 38
7.7.3.25 trim_head() . . . 39
7.7.3.26 trim_tail() . . . 39
7.8 ph::string_utils::detail_from_to_char Namespace Reference . . . 39
7.8.1 Function Documentation . . . 39
7.8.1.1 throw_from_std_errc_if_has_error() . . . 39
7.9 ph::string_utils::table Namespace Reference . . . 40
7.9.1 Variable Documentation . . . 40
7.9.1.1 ASCII_TO_LOWER . . . 40
7.9.1.2 ASCII_TO_UPPER . . . 40
7.9.1.3 BASE62_DIGITS . . . 40
7.9.1.4 common_whitespaces . . . 40
7.9.1.5 standard_whitespaces . . . 40

8 Concept Documentation **41**
8.1 ph::CPhotonException Concept Reference . . . 41
8.1.1 Concept definition . . . 41
8.2 ph::detail::CPermissiveImplicitLifetime Concept Reference . . . 41
8.2.1 Concept definition . . . 41
8.2.2 Detailed Description . . . 41
8.3 ph::string_utils::CHasToString Concept Reference . . . 41
8.3.1 Concept definition . . . 41

9 Class Documentation **43**
9.1 ph::detail::AlignedMemoryDeleter Struct Reference . . . 43
9.1.1 Member Function Documentation . . . 43
9.1.1.1 operator()() [1/2] . . . 43
9.1.1.2 operator()() [2/2] . . . 43
9.2 ph::CommandLineArguments Class Reference . . . 43
9.2.1 Detailed Description . . . 44
9.2.2 Constructor & Destructor Documentation . . . 44
9.2.2.1 CommandLineArguments() . . . 44
9.2.3 Member Function Documentation . . . 44
9.2.3.1 getProgramName() . . . 44
9.2.3.2 isEmpty() . . . 45
9.2.3.3 retrieve() . . . 45
9.2.3.4 retrieveFloat() . . . 45
9.2.3.5 retrieveInt() . . . 45
9.2.3.6 retrieveOptionArguments() . . . 45

9.2.3.7 retrieveString() . . . 45
9.2.3.8 retrieveStrings() [1/2] . . . 46
9.2.3.9 retrieveStrings() [2/2] . . . 46
9.3 ph::Config Class Reference . . . 46
9.3.1 Member Function Documentation . . . 47
9.3.1.1 RENDERER_RESOURCE_DIRECTORY() . . . 47
9.4 ph::FileIOError Class Reference . . . 47
9.4.1 Constructor & Destructor Documentation . . . 48
9.4.1.1 FileIOError() [1/3] . . . 48
9.4.1.2 FileIOError() [2/3] . . . 48
9.4.1.3 FileIOError() [3/3] . . . 48
9.4.2 Member Function Documentation . . . 48
9.4.2.1 whatStr() . . . 48
9.5 ph::FilesystemError Class Reference . . . 48
9.5.1 Constructor & Destructor Documentation . . . 49
9.5.1.1 FilesystemError() [1/2] . . . 49
9.5.1.2 FilesystemError() [2/2] . . . 49
9.5.2 Member Function Documentation . . . 49
9.5.2.1 IOException() [1/2] . . . 49
9.5.2.2 IOException() [2/2] . . . 49
9.5.2.3 whatStr() . . . 50
9.6 ph::TimerStatsReport::GroupedTimeRecord Struct Reference . . . 50
9.6.1 Constructor & Destructor Documentation . . . 50
9.6.1.1 GroupedTimeRecord() . . . 50
9.6.2 Member Data Documentation . . . 50
9.6.2.1 count . . . 50
9.6.2.2 groupName . . . 50
9.6.2.3 subgroups . . . 50
9.6.2.4 totalMicroseconds . . . 51
9.7 ph::detail::HeterogeneousStringHash Struct Reference . . . 51
9.7.1 Member Typedef Documentation . . . 51
9.7.1.1 is_transparent . . . 51
9.7.2 Member Function Documentation . . . 51
9.7.2.1 operator()() [1/3] . . . 51
9.7.2.2 operator()() [2/3] . . . 51
9.7.2.3 operator()() [3/3] . . . 52
9.8 ph::IllegalOperationException Class Reference . . . 52
9.8.1 Member Function Documentation . . . 52
9.8.1.1 LogicalException() [1/2] . . . 52
9.8.1.2 LogicalException() [2/2] . . . 52
9.9 ph::IniFile Class Reference . . . 53
9.9.1 Detailed Description . . . 53

9.9.2 Constructor & Destructor Documentation . . . 53
9.9.2.1 IniFile() [1/2] . . . 53
9.9.2.2 IniFile() [2/2] . . . 54
9.9.3 Member Function Documentation . . . 54
9.9.3.1 append() . . . 54
9.9.3.2 clear() . . . 54
9.9.3.3 findPropertyIndex() . . . 54
9.9.3.4 findSectionIndex() . . . 54
9.9.3.5 getCurrentSectionName() . . . 54
9.9.3.6 getPropertyName() . . . 54
9.9.3.7 getPropertyValue() . . . 54
9.9.3.8 getSectionName() . . . 55
9.9.3.9 numProperties() . . . 55
9.9.3.10 numSections() . . . 55
9.9.3.11 read() . . . 55
9.9.3.12 save() . . . 55
9.9.3.13 setCurrentSection() [1/2] . . . 55
9.9.3.14 setCurrentSection() [2/2] . . . 55
9.9.3.15 setProperty() [1/2] . . . 55
9.9.3.16 setProperty() [2/2] . . . 56
9.10 ph::InvalidArgumentException Class Reference . . . 56
9.10.1 Member Function Documentation . . . 56
9.10.1.1 LogicalException() [1/2] . . . 56
9.10.1.2 LogicalException() [2/2] . . . 57
9.11 ph::IOException Class Reference . . . 57
9.11.1 Constructor & Destructor Documentation . . . 57
9.11.1.1 IOException() [1/2] . . . 57
9.11.1.2 IOException() [2/2] . . . 58
9.12 ph::Logger Class Reference . . . 58
9.12.1 Constructor & Destructor Documentation . . . 58
9.12.1.1 Logger() . . . 58
9.12.2 Member Function Documentation . . . 58
9.12.2.1 addLogHandler() . . . 58
9.12.2.2 log() [1/3] . . . 58
9.12.2.3 log() [2/3] . . . 59
9.12.2.4 log() [3/3] . . . 59
9.12.2.5 makeColoredStdOutLogPrinter() . . . 59
9.12.2.6 makeStdOutLogPrinter() . . . 59
9.13 ph::LogGroup Struct Reference . . . 59
9.13.1 Member Data Documentation . . . 59
9.13.1.1 category . . . 59
9.13.1.2 groupName . . . 60

9.14 ph::LogGroups Class Reference . . . 60
9.14.1 Constructor & Destructor Documentation . . . 60
9.14.1.1 LogGroups() [1/2] . . . 60
9.14.1.2 LogGroups() [2/2] . . . 60
9.14.2 Member Function Documentation . . . 60
9.14.2.1 addGroup() . . . 60
9.14.2.2 getGroup() . . . 60
9.14.2.3 numGroups() . . . 61
9.14.2.4 operator=() . . . 61
9.15 ph::LogicalException Class Reference . . . 61
9.15.1 Detailed Description . . . 61
9.15.2 Constructor & Destructor Documentation . . . 61
9.15.2.1 LogicalException() [1/2] . . . 61
9.15.2.2 LogicalException() [2/2] . . . 62
9.15.2.3 ~LogicalException() . . . 62
9.15.3 Member Function Documentation . . . 62
9.15.3.1 whatStr() . . . 62
9.16 ph::NumericException Class Reference . . . 62
9.16.1 Member Function Documentation . . . 63
9.16.1.1 RuntimeException() [1/2] . . . 63
9.16.1.2 RuntimeException() [2/2] . . . 63
9.17 ph::OutOfRangeException Class Reference . . . 63
9.17.1 Member Function Documentation . . . 63
9.17.1.1 LogicalException() [1/2] . . . 63
9.17.1.2 LogicalException() [2/2] . . . 64
9.18 ph::OverflowException Class Reference . . . 64
9.19 ph::RuntimeException Class Reference . . . 65
9.19.1 Detailed Description . . . 65
9.19.2 Constructor & Destructor Documentation . . . 65
9.19.2.1 RuntimeException() [1/2] . . . 65
9.19.2.2 RuntimeException() [2/2] . . . 65
9.19.2.3 ~RuntimeException() . . . 65
9.19.3 Member Function Documentation . . . 66
9.19.3.1 whatStr() . . . 66
9.20 ph::detail::stats::ScopedTimer Struct Reference . . . 66
9.20.1 Member Typedef Documentation . . . 66
9.20.1.1 Clock . . . 66
9.20.2 Constructor & Destructor Documentation . . . 66
9.20.2.1 ScopedTimer() . . . 66
9.20.2.2 ~ScopedTimer() . . . 67
9.20.3 Member Data Documentation . . . 67
9.20.3.1 counter . . . 67

9.20.3.2 startTime . . . 67
9.21 ph::detail::stats::TimeCounter Struct Reference . . . 67
9.21.1 Constructor & Destructor Documentation . . . 67
9.21.1.1 TimeCounter() . . . 67
9.21.2 Member Function Documentation . . . 68
9.21.2.1 addMicroseconds() . . . 68
9.21.3 Member Data Documentation . . . 68
9.21.3.1 category . . . 68
9.21.3.2 count . . . 68
9.21.3.3 name . . . 68
9.21.3.4 totalMicroseconds . . . 68
9.22 ph::TimerStatsReport::TimeRecord Struct Reference . . . 68
9.22.1 Constructor & Destructor Documentation . . . 69
9.22.1.1 TimeRecord() . . . 69
9.22.2 Member Data Documentation . . . 69
9.22.2.1 category . . . 69
9.22.2.2 count . . . 69
9.22.2.3 name . . . 69
9.22.2.4 totalMicroseconds . . . 69
9.23 ph::TimerStatsReport Class Reference . . . 69
9.23.1 Constructor & Destructor Documentation . . . 70
9.23.1.1 TimerStatsReport() . . . 70
9.23.2 Member Function Documentation . . . 70
9.23.2.1 averagedReport() . . . 70
9.23.2.2 detailedReport() . . . 70
9.23.2.3 getGroupedTimeRecord() . . . 70
9.23.2.4 proportionalReport() . . . 70
9.23.2.5 rawReport() . . . 70
9.24 ph::Timestamp Class Reference . . . 70
9.24.1 Detailed Description . . . 71
9.24.2 Constructor & Destructor Documentation . . . 71
9.24.2.1 Timestamp() . . . 71
9.24.3 Member Function Documentation . . . 71
9.24.3.1 toHMS() . . . 71
9.24.3.2 toHMSMicroseconds() . . . 71
9.24.3.3 toHMSMilliseconds() . . . 71
9.24.3.4 toString() . . . 71
9.24.3.5 toYMD() . . . 71
9.24.3.6 toYMDHMS() . . . 72
9.24.3.7 toYMDHMSMicroseconds() . . . 72
9.24.3.8 toYMDHMSMilliseconds() . . . 72
9.25 ph::UninitializedObjectException Class Reference . . . 72

9.25.1 Member Function Documentation . . . 73
9.25.1.1 LogicalException() [1/2] . . . 73
9.25.1.2 LogicalException() [2/2] . . . 73

10 File Documentation **75**

10.1 Documentation/namespace_ph.dox File Reference . . . 75
10.2 Documentation/namespace_ph_detail.dox File Reference . . . 75
10.3 Documentation/namespace_ph_detail_core_logging.dox File Reference . . . 75
10.4 Documentation/namespace_ph_string_utils.dox File Reference . . . 75
10.5 Include/Common/assertion.h File Reference . . . 76
10.5.1 Macro Definition Documentation . . . 77
10.5.1.1 PH_ASSERT . . . 77
10.5.1.2 PH_ASSERT_EQ . . . 77
10.5.1.3 PH_ASSERT_GE . . . 77
10.5.1.4 PH_ASSERT_GT . . . 77
10.5.1.5 PH_ASSERT_IN_RANGE . . . 77
10.5.1.6 PH_ASSERT_IN_RANGE_EXCLUSIVE . . . 78
10.5.1.7 PH_ASSERT_IN_RANGE_INCLUSIVE . . . 78
10.5.1.8 PH_ASSERT_LE . . . 78
10.5.1.9 PH_ASSERT_LT . . . 78
10.5.1.10 PH_ASSERT_MSG . . . 78
10.5.1.11 PH_ASSERT_NE . . . 78
10.5.1.12 PH_ASSERT_UNREACHABLE_SECTION . . . 79
10.5.1.13 PH_INTERNAL_RANGE_MSG . . . 79
10.5.1.14 PH_STATIC_ASSERT_DEPENDENT_FALSE . . . 79
10.6 assertion.h . . . 79
10.7 Include/Common/compiler.h File Reference . . . 80
10.7.1 Detailed Description . . . 80
10.7.1.1 Definitions for Detecting Compilers . . . 81
10.7.1.2 Additional Attributes . . . 81
10.7.2 Macro Definition Documentation . . . 81
10.7.2.1 PH_COMPILER_IS_CLANG . . . 81
10.7.2.2 PH_COMPILER_IS_GCC . . . 81
10.7.2.3 PH_COMPILER_IS_MSVC . . . 81
10.8 compiler.h . . . 81
10.9 Include/Common/config.h File Reference . . . 82
10.9.1 Macro Definition Documentation . . . 83
10.9.1.1 PH_ABORT_ON_ASSERTION_FAILED . . . 83
10.9.1.2 PH_CONFIG_DIRECTORY . . . 83
10.9.1.3 PH_DEBUG . . . 83
10.9.1.4 PH_ENABLE_DEBUG_LOG . . . 83
10.9.1.5 PH_ENABLE_HIT_EVENT_STATS . . . 84

10.9.1.6 PH_ENGINE_VERSION . . . 84
10.9.1.7 PH_ENSURE_LOCKFREE_ALGORITHMS_ARE_LOCKLESS . . . 84
10.9.1.8 PH_HIT_PROBE_CACHE_BYTES . . . 84
10.9.1.9 PH_HIT_PROBE_DEPTH . . . 84
10.9.1.10 PH_INTERNAL_RESOURCE_DIRECTORY . . . 84
10.9.1.11 PH_LOG_FILE_DIRECTRY . . . 84
10.9.1.12 PH_MEMORY_ARENA_DEFAULT_BLOCK_SIZE_IN_BYTES . . . 84
10.9.1.13 PH_NUMERIC_IMAGE_MAX_ELEMENTS . . . 85
10.9.1.14 PH_PRINT_STACK_TRACE_ON_ASSERTION_FAILED . . . 85
10.9.1.15 PH_PROFILING . . . 85
10.9.1.16 PH_PSDL_VERSION . . . 85
10.9.1.17 PH_RENDER_MODE . . . 85
10.9.1.18 PH_RENDER_MODE_ACES . . . 85
10.9.1.19 PH_RENDER_MODE_FULL_SPECTRAL . . . 85
10.9.1.20 PH_RENDER_MODE_LINEAR_SRGB . . . 85
10.9.1.21 PH_RENDER_MODE_SPECTRAL . . . 85
10.9.1.22 PH_RENDERER_RESOURCE_DIRECTORY . . . 86
10.9.1.23 PH_SCRIPT_DIRECTORY . . . 86
10.9.1.24 PH_SDL_MAX_FIELDS . . . 86
10.9.1.25 PH_SDL_MAX_FUNCTIONS . . . 86
10.9.1.26 PH_SPECTRUM_SAMPLED_MAX_WAVELENGTH_NM . . . 86
10.9.1.27 PH_SPECTRUM_SAMPLED_MIN_WAVELENGTH_NM . . . 86
10.9.1.28 PH_SPECTRUM_SAMPLED_NUM_SAMPLES . . . 86
10.9.1.29 PH_STRICT_ASYMMETRIC_IMPORTANCE_TRANSPORT . . . 86
10.9.1.30 PH_STRICT_FLOATING_POINT_SIZES . . . 87
10.9.1.31 PH_STRICT_OBJECT_LIFETIME . . . 87
10.9.1.32 PH_TFUNCTION_DEFAULT_MIN_SIZE_IN_BYTES . . . 87
10.9.1.33 PH_USE_DOUBLE_REAL . . . 87
10.10 config.h . . . 87
10.11 Include/Common/Config/IniFile.h File Reference . . . 88
10.12 IniFile.h . . . 89
10.13 Include/Common/Container/detail.h File Reference . . . 90
10.14 detail.h . . . 91
10.15 Include/Common/Container/StdUnorderedStringSet.h File Reference . . . 91
10.16 StdUnorderedStringSet.h . . . 92
10.17 Include/Common/Container/TStdUnorderedStringMap.h File Reference . . . 92
10.18 TStdUnorderedStringMap.h . . . 93
10.19 Include/Common/debug.h File Reference . . . 93
10.19.1 Macro Definition Documentation . . . 93
10.19.1.1 PH_DEBUG_BREAK . . . 93
10.20 debug.h . . . 93
10.21 Include/Common/exceptions.h File Reference . . . 94

10.22 exceptions.h . . . 95
10.23 Include/Common/io_exceptions.h File Reference . . . 96
10.24 io_exceptions.h . . . 96
10.25 Include/Common/Log/ELogLevel.h File Reference . . . 97
10.26 ELogLevel.h . . . 98
10.27 Include/Common/Log/Logger.h File Reference . . . 98
10.28 Logger.h . . . 99
10.29 Include/Common/Log/logger_fwd.h File Reference . . . 99
10.30 logger_fwd.h . . . 100
10.31 Include/Common/logging.h File Reference . . . 100
10.31.1 Detailed Description . . . 102
10.31.2 Macro Definition Documentation . . . 102
10.31.2.1 PH_DEBUG_LOG . . . 102
10.31.2.2 PH_DEBUG_LOG_ONCE . . . 102
10.31.2.3 PH_DEBUG_LOG_STRING . . . 102
10.31.2.4 PH_DEBUG_LOG_STRING_ONCE . . . 102
10.31.2.5 PH_DECLARE_LOG_GROUP . . . 102
10.31.2.6 PH_DEFAULT_DEBUG_LOG . . . 103
10.31.2.7 PH_DEFAULT_DEBUG_LOG_ONCE . . . 103
10.31.2.8 PH_DEFAULT_DEBUG_LOG_STRING . . . 103
10.31.2.9 PH_DEFAULT_DEBUG_LOG_STRING_ONCE . . . 103
10.31.2.10 PH_DEFAULT_LOG . . . 103
10.31.2.11 PH_DEFAULT_LOG_STRING . . . 103
10.31.2.12 PH_DEFINE_EXTERNAL_LOG_GROUP . . . 104
10.31.2.13 PH_DEFINE_INLINE_LOG_GROUP . . . 104
10.31.2.14 PH_DEFINE_INTERNAL_LOG_GROUP . . . 104
10.31.2.15 PH_DEFINE_LOG_GROUP . . . 104
10.31.2.16 PH_LOG . . . 105
10.31.2.17 PH_LOG_FORMAT_STRING_TO_CORE_LOGGER . . . 105
10.31.2.18 PH_LOG_RAW_STRING_TO_CORE_LOGGER . . . 105
10.31.2.19 PH_LOG_STRING . . . 105
10.32 logging.h . . . 106
10.33 Include/Common/macro.h File Reference . . . 108
10.33.1 Detailed Description . . . 108
10.33.2 Macro Definition Documentation . . . 108
10.33.2.1 PH_CONCAT_2 . . . 108
10.33.2.2 PH_CONCAT_3 . . . 108
10.33.2.3 PH_CONCAT_4 . . . 108
10.33.2.4 PH_CONCAT_5 . . . 109
10.33.2.5 PH_CONCAT_6 . . . 109
10.33.2.6 PH_CONCAT_7 . . . 109
10.33.2.7 PH_CONCAT_8 . . . 109

10.33.2.8 PH_NO_OP . . . 109
10.34 macro.h . . . 110
10.35 Include/Common/math_basics.h File Reference . . . 110
10.35.1 Detailed Description . . . 110
10.36 math_basics.h . . . 111
10.37 Include/Common/memory.h File Reference . . . 111
10.37.1 Detailed Description . . . 112
10.38 memory.h . . . 113
10.39 Include/Common/memory.ipp File Reference . . . 113
10.40 memory.ipp . . . 114
10.41 Include/Common/os.h File Reference . . . 116
10.41.1 Detailed Description . . . 117
10.41.2 Macro Definition Documentation . . . 117
10.41.2.1 PH_OPERATING_SYSTEM_IS_LINUX . . . 117
10.41.2.2 PH_OPERATING_SYSTEM_IS_OSX . . . 117
10.41.2.3 PH_OPERATING_SYSTEM_IS_WINDOWS . . . 117
10.42 os.h . . . 118
10.43 Include/Common/primitive_type.h File Reference . . . 118
10.44 primitive_type.h . . . 120
10.45 Include/Common/profiling.h File Reference . . . 120
10.45.1 Detailed Description . . . 121
10.45.2 Macro Definition Documentation . . . 121
10.45.2.1 PH_DEFINE_PROFILE_UNIT_NAME . . . 121
10.45.2.2 PH_PROFILE_LOOP_BEGIN . . . 121
10.45.2.3 PH_PROFILE_LOOP_END . . . 121
10.45.2.4 PH_PROFILE_LOOP_MARK . . . 121
10.45.2.5 PH_PROFILE_NAME_THIS_THREAD . . . 121
10.45.2.6 PH_PROFILE_NAMED_SCOPE . . . 121
10.45.2.7 PH_PROFILE_SCOPE . . . 121
10.46 profiling.h . . . 122
10.47 Include/Common/stats.h File Reference . . . 122
10.47.1 Macro Definition Documentation . . . 123
10.47.1.1 PH_DEFINE_EXTERNAL_TIMER_STAT . . . 123
10.47.1.2 PH_DEFINE_INLINE_TIMER_STAT . . . 123
10.47.1.3 PH_DEFINE_INTERNAL_TIMER_STAT . . . 123
10.47.1.4 PH_SCOPED_TIMER . . . 124
10.48 stats.h . . . 124
10.49 Include/Common/ThirdParty/lib_tracy.h File Reference . . . 125
10.50 lib_tracy.h . . . 125
10.51 Include/Common/utility.h File Reference . . . 125
10.51.1 Macro Definition Documentation . . . 126
10.51.1.1 PH_NOT_IMPLEMENTED_WARNING . . . 126

10.52 utility.h . . . 127
10.53 Include/Common/utility.ipp File Reference . . . 127
10.54 utility.ipp . . . 128
10.55 Include/Common/Utility/CommandLineArguments.h File Reference . . . 130
10.56 CommandLineArguments.h . . . 130
10.57 Include/Common/Utility/string_utils.h File Reference . . . 132
10.57.1 Detailed Description . . . 134
10.57.2 Macro Definition Documentation . . . 134
10.57.2.1 PH_DEFINE_INLINE_TO_STRING_FORMATTER . . . 134
10.57.2.2 PH_DEFINE_INLINE_TO_STRING_FORMATTER_SPECIALIZATION . . . 135
10.57.2.3 PH_DEFINE_INLINE_TO_STRING_FORMATTER_TEMPLATE . . . 135
10.58 string_utils.h . . . 135
10.59 Include/Common/Utility/string_utils_table.h File Reference . . . 141
10.60 string_utils_table.h . . . 142
10.61 Include/Common/Utility/Timestamp.h File Reference . . . 143
10.62 Timestamp.h . . . 143
10.63 README.md File Reference . . . 144
10.64 Source/Common/assertion.cpp File Reference . . . 144
10.65 Source/Common/config.cpp File Reference . . . 144
10.66 Source/Common/Config/IniFile.cpp File Reference . . . 145
10.67 Source/Common/debug.cpp File Reference . . . 145
10.67.1 Macro Definition Documentation . . . 145
10.67.1.1 psnip_trap . . . 145
10.68 Source/Common/exception.cpp File Reference . . . 145
10.69 Source/Common/Log/Logger.cpp File Reference . . . 146
10.70 Source/Common/logging.cpp File Reference . . . 146
10.71 Source/Common/memory.cpp File Reference . . . 147
10.72 Source/Common/os.cpp File Reference . . . 147
10.73 Source/Common/profiling.cpp File Reference . . . 148
10.74 Source/Common/stats.cpp File Reference . . . 148
10.75 Source/Common/utility.cpp File Reference . . . 148
10.76 Source/Common/Utility/CommandLineArguments.cpp File Reference . . . 149
10.77 Source/Common/Utility/Timestamp.cpp File Reference . . . 149

Index **151**

Chapter 1

Photon Common Library

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

ph
The root for all renderer implementations . . . 13
ph::detail
Implementation detail mainly for internal usages . . . 23
ph::detail::core_logging
Core logging functionalities. Most logs will output information (logs) via a main logger, which we refer to as "core logger". This namespace contains implementation details for core logging functionalities . . . 26
ph::detail::stats . . . 27
ph::math . . . 28
ph::os . . . 29
ph::string_utils
Contains various string manipulation helpers . . . 30
ph::string_utils::detail_from_to_char . . . 39
ph::string_utils::table . . . 40

Chapter 3

Concept Index

3.1 Concepts

Here is a list of all concepts with brief descriptions:

ph::CPhotonException 41
ph::detail::CPermissiveImplicitLifetime 41
ph::string_utils::CHasToString 41

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

ph::detail::AlignedMemoryDeleter . . . 43
ph::CommandLineArguments . . . 43
ph::Config . . . 46
ph::TimerStatsReport::GroupedTimeRecord . . . 50
ph::detail::HeterogeneousStringHash . . . 51
ph::IniFile . . . 53
ph::Logger . . . 58
ph::LogGroup . . . 59
ph::LogGroups . . . 60
std::logic_error
    ph::LogicalException . . . 61
        ph::IllegalOperationException . . . 52
        ph::InvalidArgumentException . . . 56
        ph::OutOfRangeException . . . 63
        ph::UninitializedObjectException . . . 72
std::runtime_error
    ph::RuntimeException . . . 65
        ph::IOException . . . 57
            ph::FileIOError . . . 47
            ph::FilesystemError . . . 48
        ph::NumericException . . . 62
            ph::OverflowException . . . 64
ph::detail::stats::ScopedTimer . . . 66
ph::detail::stats::TimeCounter . . . 67
ph::TimerStatsReport::TimeRecord . . . 68
ph::TimerStatsReport . . . 69
ph::Timestamp . . . 70

Chapter 5

Class Index

5.1 Class List

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

ph::detail::AlignedMemoryDeleter . . . 43
ph::CommandLineArguments
Helper for parsing command line arguments . . . 43
ph::Config . . . 46
ph::FileIOError . . . 47
ph::FilesystemError . . . 48
ph::TimerStatsReport::GroupedTimeRecord . . . 50
ph::detail::HeterogeneousStringHash . . . 51
ph::IllegalOperationException . . . 52
ph::IniFile
INI file I/O. This class is useful for recording various settings across the entire engine project. As a low-level I/O class in `Common` library, it can be used regardless the engine initialization state (see `init_render_engine()` and `exit_render_engine()` in `Engine` library for more details) . . . 53
ph::InvalidArgumentException . . . 56
ph::IOException . . . 57
ph::Logger . . . 58
ph::LogGroup . . . 59
ph::LogGroups . . . 60
ph::LogicalException
General exception thrown on logical error . . . 61
ph::NumericException . . . 62
ph::OutOfRangeException . . . 63
ph::OverflowException . . . 64
ph::RuntimeException
General exception thrown on runtime error . . . 65
ph::detail::stats::ScopedTimer . . . 66
ph::detail::stats::TimeCounter . . . 67
ph::TimerStatsReport::TimeRecord . . . 68
ph::TimerStatsReport . . . 69
ph::Timestamp
Represents a point in time . . . 70
ph::UninitializedObjectException . . . 72

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Include/Common/assertion.h . . . 76
Include/Common/compiler.h
    Compiler-related information and utilities . . . 80
Include/Common/config.h . . . 82
Include/Common/debug.h . . . 93
Include/Common/exceptions.h . . . 94
Include/Common/io_exceptions.h . . . 96
Include/Common/logging.h
    Logging functions . . . 100
Include/Common/macro.h
    Useful macro definitions for general operations . . . 108
Include/Common/math_basics.h
    Basic math utilities. For more math functions, see `Engine` project's `math.h` . . . 110
Include/Common/memory.h
    Low-level memory allocation routines . . . 111
Include/Common/memory.ipp . . . 113
Include/Common/os.h
    Operating system detection macros and utilities . . . 116
Include/Common/primitive_type.h . . . 118
Include/Common/profiling.h
    Profiling functions . . . 120
Include/Common/stats.h . . . 122
Include/Common/utility.h . . . 125
Include/Common/utility.ipp . . . 127
Include/Common/Config/IniFile.h . . . 88
Include/Common/Container/detail.h . . . 90
Include/Common/Container/StdUnorderedStringSet.h . . . 91
Include/Common/Container/TStdUnorderedStringMap.h . . . 92
Include/Common/Log/ELogLevel.h . . . 97
Include/Common/Log/Logger.h . . . 98
Include/Common/Log/logger_fwd.h . . . 99
Include/Common/ThirdParty/lib_tracy.h . . . 125
Include/Common/Utility/CommandLineArguments.h . . . 130
Include/Common/Utility/string_utils.h
    String manipulation helpers . . . 132

Include/Common/Utility/string_utils_table.h . . . 141
Include/Common/Utility/Timestamp.h . . . 143
Source/Common/assertion.cpp . . . 144
Source/Common/config.cpp . . . 144
Source/Common/debug.cpp . . . 145
Source/Common/exception.cpp . . . 145
Source/Common/logging.cpp . . . 146
Source/Common/memory.cpp . . . 147
Source/Common/os.cpp . . . 147
Source/Common/profiling.cpp . . . 148
Source/Common/stats.cpp . . . 148
Source/Common/utility.cpp . . . 148
Source/Common/Config/IniFile.cpp . . . 145
Source/Common/Log/Logger.cpp . . . 146
Source/Common/Utility/CommandLineArguments.cpp . . . 149
Source/Common/Utility/Timestamp.cpp . . . 149

Chapter 7

Namespace Documentation

7.1 ph Namespace Reference

The root for all renderer implementations.

Namespaces

- namespace detail

  Implementation detail mainly for internal usages.
- namespace math
- namespace os
- namespace string_utils

  Contains various string manipulation helpers.

Classes

- class CommandLineArguments

  Helper for parsing command line arguments.
- class Config
- class FileIOError
- class FilesystemError
- class IllegalOperationException
- class IniFile

  INI file I/O. This class is useful for recording various settings across the entire engine project. As a low-level I/O class in `Common` library, it can be used regardless the engine initialization state (see `init_render_engine()` and `exit_render_engine()` in `Engine` library for more details).
- class InvalidArgumentException
- class IOException
- class Logger
- struct LogGroup
- class LogGroups
- class LogicalException

  General exception thrown on logical error.
- class NumericException
- class OutOfRangeException
- class OverflowException
- class RuntimeException

  General exception thrown on runtime error.
- class TimerStatsReport
- class Timestamp

  Represents a point in time.
- class UninitializedObjectException

Concepts

- concept CPhotonException

Typedefs

- using StdUnorderedStringSet

  Unordered `std::string` set with support for heterogeneous string key lookup. Supports `std::string_view` and literal (C-style) string lookup in addition to the original `std::string` lookup. The heterogeneous access can save redundant dynamic allocations when querying the map.
- template<typename Value >
  using TStdUnorderedStringMap

  Unordered `std::string` map with support for heterogeneous string key lookup. Supports `std::string_view` and literal (C-style) string lookup in addition to the original `std::string` lookup. The heterogeneous access can save redundant dynamic allocations when querying the map.
- using Exception = std::exception
- using LogHandler = std::function<void(ELogLevel logLevel, std::string_view logString)>
- template<typename T >
  using TAlignedMemoryUniquePtr = std::unique_ptr<T, detail::AlignedMemoryDeleter>
- using real = float
- using integer = int
- using hiReal = float64
- using hiInteger = int64

- using int8 = std::int8_t

  Fixed-size integer types.
- using uint8 = std::uint8_t
- using int16 = std::int16_t
- using uint16 = std::uint16_t
- using int32 = std::int32_t
- using uint32 = std::uint32_t
- using int64 = std::int64_t
- using uint64 = std::uint64_t

- using int8f = std::int_fast8_t

  Fastest integer types with size guarantee. For example, `uint32f` is an unsigned integer with at least 32 bits.
- using uint8f = std::uint_fast8_t
- using int16f = std::int_fast16_t
- using uint16f = std::uint_fast16_t
- using int32f = std::int_fast32_t
- using uint32f = std::uint_fast32_t
- using int64f = std::int_fast64_t
- using uint64f = std::uint_fast64_t

- using float32 = float

  Fixed-size floating-point types.
- using float64 = double

Enumerations

- enum class ELogLevel {
  Debug , Note , Warning , Error ,
  DebugOnce , NoteOnce , WarningOnce , ErrorOnce }

Functions

- void debug_break ()
- std::string obtain_stack_trace ()
- template<CPhotonException ExceptionType, typename... Args>
  void throw_formatted (const std::format_string< Args... > msgFormat, Args &&... args)
- constexpr bool is_once (const ELogLevel logLevel)
- LogGroups get_core_log_groups ()
- PH_DECLARE_LOG_GROUP (PhotonRenderer)
- template<typename T = void>
  auto make_aligned_memory (std::size_t numBytes, std::size_t alignmentInBytes) -> TAlignedMemoryUniquePtr< T >

  Create an aligned memory resource.
- template<typename T >
  void from_bytes (const std::byte *srcBytes, T *out_dstValue)
- template<typename T >
  void to_bytes (const T &srcValue, std::byte *out_dstBytes)
- template<std::size_t N>
  void reverse_bytes (std::byte *bytes)
- template<typename T >
  T * start_implicit_lifetime_as (void *ptr) noexcept

  Wrapper for `std::start_lifetime_as()`. Primarily a fallback when C++23 is not available. This function may touch the storage. For cv overloads or one that does not touch the storage, use `std::start_lifetime_as()` (requires C++23).
- template<typename T >
  T * start_implicit_lifetime_as_array (void *ptr, std::size_t numArrayElements) noexcept

  Wrapper for `std::start_lifetime_as_array()`. Primarily a fallback when C++23 is not available. This function may touch the storage. For cv overloads or one that does not touch the storage, use `std::start_lifetime_as_array()` (requires C++23).
- constexpr real operator""_r (const long double cookedValue)
- template<typename T >
  consteval std::size_t sizeof_in_bits ()

  Calculates number of bits an instance of type `T` occupies.
- template<typename T , std::size_t N>
  constexpr std::array< T, N > make_array (const T &element)

  Creates an `std::array` filled with the same element.
- template<std::integral DstType, std::integral SrcType>
  DstType lossless_integer_cast (const SrcType &src)
- template<std::floating_point DstType, std::floating_point SrcType>
  DstType lossless_float_cast (const SrcType &src)
- template<typename DstType , typename SrcType >
  DstType lossless_cast (const SrcType &src)

  Cast numeric value to another type without any loss of information. If there is any possible overflow or numeric precision loss, exception is thrown.
- template<typename DstType , typename SrcType >
  DstType lossless_cast (const SrcType &src, DstType *const out_dst)
- PH_DEFINE_LOG_GROUP (PhotonRenderer, Core)

7.1.1 Detailed Description

The root for all renderer implementations.

7.1.2 Typedef Documentation

7.1.2.1 Exception

```
using ph::Exception = std::exception
```

7.1.2.2 float32

```
using ph::float32 = float
```

Fixed-size floating-point types.

7.1.2.3 float64

```
using ph::float64 = double
```

7.1.2.4 hiInteger

```
using ph::hiInteger = int64
```

7.1.2.5 hiReal

```
using ph::hiReal = float64
```

7.1.2.6 int16

```
using ph::int16 = std::int16_t
```

7.1.2.7 int16f

```
using ph::int16f = std::int_fast16_t
```

7.1.2.8 int32

```
using ph::int32 = std::int32_t
```

7.1.2.9 int32f

```
using ph::int32f = std::int_fast32_t
```

7.1.2.10 int64

```
using ph::int64 = std::int64_t
```

7.1.2.11 int64f

```
using ph::int64f = std::int_fast64_t
```

7.1.2.12 int8

```
using ph::int8 = std::int8_t
```

Fixed-size integer types.

7.1.2.13 int8f

```
using ph::int8f = std::int_fast8_t
```

Fastest integer types with size guarantee. For example, `uint32f` is an unsigned integer with at least 32 bits.

7.1.2.14 integer

```
using ph::integer = int
```

7.1.2.15 LogHandler

```
using ph::LogHandler = std::function<void(ELogLevel logLevel, std::string_view logString)>
```

7.1.2.16 real

```
using ph::real = float
```

7.1.2.17 StdUnorderedStringSet

```
using ph::StdUnorderedStringSet
```

Initial value:

```
std::unordered_set<
    std::string, detail::HeterogeneousStringHash, std::equal_to<»
```

Unordered `std::string` set with support for heterogeneous string key lookup. Supports `std::string_view` and literal (C-style) string lookup in addition to the original `std::string` lookup. The heterogeneous access can save redundant dynamic allocations when querying the map.

7.1.2.18 TAlignedMemoryUniquePtr

```
template<typename T >
using ph::TAlignedMemoryUniquePtr = std::unique_ptr<T, detail::AlignedMemoryDeleter>
```

7.1.2.19 TStdUnorderedStringMap

```
template<typename Value >
using ph::TStdUnorderedStringMap
```

Initial value:

```
 std::unordered_map<
    std::string, Value, detail::HeterogeneousStringHash, std::equal_to<»
```

Unordered `std::string` map with support for heterogeneous string key lookup. Supports `std::string_view` and literal (C-style) string lookup in addition to the original `std::string` lookup. The heterogeneous access can save redundant dynamic allocations when querying the map.

7.1.2.20 uint16

```
using ph::uint16 = std::uint16_t
```

7.1.2.21 uint16f

```
using ph::uint16f = std::uint_fast16_t
```

7.1.2.22 uint32

```
using ph::uint32 = std::uint32_t
```

7.1.2.23 uint32f

```
using ph::uint32f = std::uint_fast32_t
```

7.1.2.24 uint64

```
using ph::uint64 = std::uint64_t
```

7.1.2.25 uint64f

```
using ph::uint64f = std::uint_fast64_t
```

7.1.2.26 uint8

```
using ph::uint8 = std::uint8_t
```

7.1.2.27 uint8f

```
using ph::uint8f = std::uint_fast8_t
```

7.1.3 Enumeration Type Documentation

7.1.3.1 ELogLevel

```
enum class ph::ELogLevel  [strong]
```

Enumerator

Debug	
Note	
Warning	
Error	
DebugOnce	
NoteOnce	
WarningOnce	
ErrorOnce	

7.1.4 Function Documentation

7.1.4.1 debug_break()

```
void ph::debug_break ()
```

7.1.4.2 from_bytes()

```
template<typename T >
void ph::from_bytes (
            const std::byte * srcBytes,
            T * out_dstValue)  [inline]
```

7.1.4.3 get_core_log_groups()

```
LogGroups ph::get_core_log_groups ()
```

7.1.4.4 is_once()

```
bool ph::is_once (
            const ELogLevel logLevel)  [inline], [constexpr]
```

7.1.4.5 lossless_cast() [1/2]

```
template<typename DstType , typename SrcType >
DstType ph::lossless_cast (
            const SrcType & src)  [inline]
```

Cast numeric value to another type without any loss of information. If there is any possible overflow or numeric precision loss, exception is thrown.

Exceptions

OverflowException	If overflow happens.
Numericxception	If any numeric precision loss happens.

7.1.4.6 lossless_cast() [2/2]

```
template<typename DstType , typename SrcType >
DstType ph::lossless_cast (
            const SrcType & src,
            DstType *const out_dst)  [inline]
```

7.1.4.7 lossless_float_cast()

```
template<std::floating_point DstType, std::floating_point SrcType>
DstType ph::lossless_float_cast (
            const SrcType & src)  [inline]
```

7.1.4.8 lossless_integer_cast()

```
template<std::integral DstType, std::integral SrcType>
DstType ph::lossless_integer_cast (
            const SrcType & src)  [inline]
```

7.1.4.9 make_aligned_memory()

```
template<typename T = void>
auto ph::make_aligned_memory (
            std::size_t numBytes,
            std::size_t alignmentInBytes) -> TAlignedMemoryUniquePtr<T>  [inline]
```

Create an aligned memory resource.

The returned memory resource will follow the life time of `std::unique_ptr`. Note that the memory allocated by this function is raw memory–placement new is required before any use of the memory content, otherwise it is UB by C++ standard (for non-implicit-lifetime types). For implicit-lifetime types, accessing the raw memory without placement new has defined behavior after C++20.

Template Parameters

T	The type to create memory for. `alignmentInBytes` should be compatible with the type.

Parameters

numBytes	Number of bytes to allocate. Must be an integer multiple of `alignmentInBytes`.
alignmentInBytes	How many bytes to align (so the returned pointer is an integer multiple of `alignmentInBytes`). Must be an integer power of 2 and a multiple of `sizeof(void*)`.

Returns

Pointer to the beginning of newly allocated memory. `nullptr` on failure.

Note

This function is thread safe.

7.1.4.10 make_array()

```
template<typename T , std::size_t N>
std::array< T, N > ph::make_array (
            const T & element)  [inline], [constexpr]
```

Creates an `std::array` filled with the same element.

Note

The element does not need to be default-constructible.

7.1.4.11 obtain_stack_trace()

```
std::string ph::obtain_stack_trace ()
```

7.1.4.12 operator"""_r()

```
real ph::operator""_r (
            const long double cookedValue)  [inline], [constexpr]
```

7.1.4.13 PH_DECLARE_LOG_GROUP()

```
ph::PH_DECLARE_LOG_GROUP (
            PhotonRenderer )
```

7.1.4.14 PH_DEFINE_LOG_GROUP()

```
ph::PH_DEFINE_LOG_GROUP (
            PhotonRenderer ,
            Core )
```

7.1.4.15 reverse_bytes()

```
template<std::size_t N>
void ph::reverse_bytes (
            std::byte * bytes)  [inline]
```

7.1.4.16 sizeof_in_bits()

```
template<typename T >
std::size_t ph::sizeof_in_bits ()  [inline], [consteval]
```

Calculates number of bits an instance of type `T` occupies.

7.1.4.17 start_implicit_lifetime_as()

```
template<typename T >
T * ph::start_implicit_lifetime_as (
            void * ptr)  [inline], [noexcept]
```

Wrapper for `std::start_lifetime_as()`. Primarily a fallback when C++23 is not available. This function may touch the storage. For cv overloads or one that does not touch the storage, use `std::start_lifetime_as()` (requires C++23).

Note

Starting lifetime of an array of `unsigned char` or `std::byte` implicitly creates objects within the region of storage. See [intro.object] section 13 (https://timsong-cpp.github.io/cppwp/intro.object#13).

7.1.4.18 start_implicit_lifetime_as_array()

```
template<typename T >
T * ph::start_implicit_lifetime_as_array (
            void * ptr,
            std::size_t numArrayElements)  [inline], [noexcept]
```

Wrapper for `std::start_lifetime_as_array()`. Primarily a fallback when C++23 is not available. This function may touch the storage. For cv overloads or one that does not touch the storage, use `std::start_lifetime_as_array()` (requires C++23).

Note

Starting lifetime of an array of `unsigned char` or `std::byte` implicitly creates objects within the region of storage. See [intro.object] section 13 (https://timsong-cpp.github.io/cppwp/intro.object#13).

7.1.4.19 throw_formatted()

```
template<CPhotonException ExceptionType, typename...  Args>
void ph::throw_formatted (
            const std::format_string< Args...  > msgFormat,
            Args &&...  args)  [inline]
```

7.1.4.20 to_bytes()

```
template<typename T >
void ph::to_bytes (
            const T & srcValue,
            std::byte * out_dstBytes)  [inline]
```

7.2 ph::detail Namespace Reference

Implementation detail mainly for internal usages.

Namespaces

- namespace core_logging

  Core logging functionalities. Most logs will output information (logs) via a main logger, which we refer to as "core logger". This namespace contains implementation details for core logging functionalities.
- namespace stats

Classes

- struct AlignedMemoryDeleter
- struct HeterogeneousStringHash

Concepts

- concept CPermissiveImplicitLifetime

Functions

- void output_assertion_message (const std::string &filename, const std::string &lineNumber, const std::string &condition, const std::string &message)
- void on_assertion_failed ()
- void * allocate_aligned_memory (std::size_t numBytes, std::size_t alignmentInBytes)
- void free_aligned_memory (void *ptr)
- void output_not_implemented_warning (const std::string &filename, const std::string &lineNumber)
- template<typename T , std::size_t... Is>
  constexpr std::array< T, sizeof...(Is)> make_array (T element, std::index_sequence< Is... >)

Variables

- template<typename T >
  constexpr bool DEPENDENT_FALSE = false

7.2.1 Detailed Description

Implementation detail mainly for internal usages.

7.2.2 Function Documentation

7.2.2.1 allocate_aligned_memory()

```
void * ph::detail::allocate_aligned_memory (
            std::size_t numBytes,
            std::size_t alignmentInBytes)  [nodiscard]
```

Parameters

numBytes	Number of bytes to allocate. Must be an integer multiple of `alignmentInBytes`.
alignmentInBytes	How many bytes to align (so the returned pointer is an integer multiple of `alignmentInBytes`). Must be an integer power of 2 and a multiple of `sizeof(void*)`.

Returns

Pointer to the beginning of newly allocated memory. `nullptr` on failure.

Note

Call free_aligned_memory(void*) to deallocate the memory. The implementation is based on `malloc()`, and object lifetime can be reasoned w.r.t. `malloc()`. This function is thread safe.

7.2.2.2 free_aligned_memory()

```
void ph::detail::free_aligned_memory (
            void * ptr)
```

Parameters

ptr	The memory to be deallocated. `ptr` must be allocated by allocate_aligned_memory(std::size_t, std::size_t). If `ptr` is `nullptr`, no action is performed.

Note

This function is thread safe.

7.2.2.3 make_array()

```
template<typename T , std::size_t...  Is>
std::array< T, sizeof...(Is)> ph::detail::make_array (
            T element,
            std::index_sequence< Is...  > )  [inline], [constexpr]
```

7.2.2.4 on_assertion_failed()

```
void ph::detail::on_assertion_failed ()
```

7.2.2.5 output_assertion_message()

```
void ph::detail::output_assertion_message (
            const std::string & filename,
            const std::string & lineNumber,
            const std::string & condition,
            const std::string & message)
```

7.2.2.6 output_not_implemented_warning()

```
void ph::detail::output_not_implemented_warning (
            const std::string & filename,
            const std::string & lineNumber)
```

7.2.3 Variable Documentation

7.2.3.1 DEPENDENT_FALSE

```
template<typename T >
bool ph::detail::DEPENDENT_FALSE = false  [inline], [constexpr]
```

7.3 ph::detail::core_logging Namespace Reference

Core logging functionalities. Most logs will output information (logs) via a main logger, which we refer to as "core logger". This namespace contains implementation details for core logging functionalities.

Functions

- void init ()

  Initializes core logging functionalities. Any logging is only valid after calling `init()`.
- void exit ()

  Terminates core logging functionalities. Cleanup after logging is finished.
- Logger & get_logger ()

  Get the core logger.
- std::size_t add_log_group (std::string_view groupName, std::string_view category="")

  Add a log group to the core logger.
- void log_to_logger (const Logger &logger, std::string_view groupName, ELogLevel logLevel, std::string_view logMessage)

  Log information to the specified logger.

7.3.1 Detailed Description

Core logging functionalities. Most logs will output information (logs) via a main logger, which we refer to as "core logger". This namespace contains implementation details for core logging functionalities.

7.3.2 Function Documentation

7.3.2.1 add_log_group()

```
std::size_t ph::detail::core_logging::add_log_group (
            std::string_view groupName,
            std::string_view category = "")
```

Add a log group to the core logger.

Note

Thread-safe.

7.3.2.2 exit()

```
void ph::detail::core_logging::exit ()
```

Terminates core logging functionalities. Cleanup after logging is finished.

7.3.2.3 get_logger()

```
Logger & ph::detail::core_logging::get_logger ()
```

Get the core logger.

Note

Const methods of core logger are thread-safe.

7.3.2.4 init()

```
void ph::detail::core_logging::init ()
```

Initializes core logging functionalities. Any logging is only valid after calling `init()`.

7.3.2.5 log_to_logger()

```
void ph::detail::core_logging::log_to_logger (
            const Logger & logger,
            std::string_view groupName,
            ELogLevel logLevel,
            std::string_view logMessage)
```

Log information to the specified logger.

Note

Thread-safe if the logger is thread-safe.

7.4 ph::detail::stats Namespace Reference

Classes

- struct ScopedTimer
- struct TimeCounter

7.5 ph::math Namespace Reference

Functions

- template<typename T >
  constexpr bool is_power_of_2 (const T value)

  Determines whether `value` is a power of 2 number.
- template<std::integral T>
  T ceil_div (const T numerator, const T denominator)

  Divide `numerator` by `denominator` and round up to integer. Both inputs must be positive integer. Specifically, `numerator` >= 0 and `denominator` > 0.
- template<std::integral T>
  T next_multiple (const T value, const T multiple)

  *Get the next number that is an integer multiple of `multiple`. Specifically, get the minimum number x = C * `multiple` >= `value` where C is an integer >= 0. Currently supports positive integers only.*
- template<std::integral T>
  T next_power_of_2_multiple (const T value, const T multiple)

  Same as next_multiple(T, T) except that `multiple` must be a power of 2 number.

7.5.1 Function Documentation

7.5.1.1 ceil_div()

```
template<std::integral T>
T ph::math::ceil_div (
            const T numerator,
            const T denominator)  [inline]
```

Divide `numerator` by `denominator` and round up to integer. Both inputs must be positive integer. Specifically, `numerator` >= 0 and `denominator` > 0.

7.5.1.2 is_power_of_2()

```
template<typename T >
bool ph::math::is_power_of_2 (
            const T value)  [inline], [constexpr]
```

Determines whether `value` is a power of 2 number.

7.5.1.3 next_multiple()

```
template<std::integral T>
T ph::math::next_multiple (
            const T value,
            const T multiple)  [inline]
```

Get the next number that is an integer multiple of `multiple`. Specifically, get the minimum number x = C * `multiple` >= `value` where C is an integer >= 0. Currently supports positive integers only.

7.5.1.4 next_power_of_2_multiple()

```
template<std::integral T>
T ph::math::next_power_of_2_multiple (
            const T value,
            const T multiple)  [inline]
```

Same as next_multiple(T, T) except that `multiple` must be a power of 2 number.

7.6 ph::os Namespace Reference

Enumerations

- enum class EWindowsVersion {
  Unknown = 0 , Windows_2000 , Windows_XP , Windows_Vista ,
  Windows_7 , Windows_8 , Windows_8_1 , Windows_10 }

Functions

- EWindowsVersion get_windows_version ()

  Get current Windows version at runtime.
- std::size_t get_L1_cache_line_size_in_bytes ()

  Get size of L1 cache at runtime.
- std::filesystem::path get_executable_path ()

  Get the path to the currently running executable. Answering the question, "Where am I?".

7.6.1 Enumeration Type Documentation

7.6.1.1 EWindowsVersion

```
enum class ph::os::EWindowsVersion  [strong]
```

Enumerator

Unknown	
Windows_2000	
Windows_XP	
Windows_Vista	
Windows_7	
Windows_8	
Windows_8_1	
Windows_10	

7.6.2 Function Documentation

7.6.2.1 get_executable_path()

```
std::filesystem::path ph::os::get_executable_path ()
```

Get the path to the currently running executable. Answering the question, "Where am I?".

Returns

Path to the executable. Empty if cannot obtain the path.

7.6.2.2 get_L1_cache_line_size_in_bytes()

```
std::size_t ph::os::get_L1_cache_line_size_in_bytes ()
```

Get size of L1 cache at runtime.

Returns

Size in bytes.

7.6.2.3 get_windows_version()

```
EWindowsVersion ph::os::get_windows_version ()
```

Get current Windows version at runtime.

7.7 ph::string_utils Namespace Reference

Contains various string manipulation helpers.

Namespaces

- namespace detail_from_to_char
- namespace table

Concepts

- concept CHasToString

Enumerations

- enum class EWhitespace { Common , Standard }

Functions

- template<EWhitespace TYPE = EWhitespace::Common>
  std::string_view get_whitespaces ()
- template<EWhitespace TYPE = EWhitespace::Common>
  constexpr bool is_whitespace (const char ch)
- bool has_any_of (const std::string_view srcStr, const std::string_view candidates)
- bool has_none_of (const std::string_view srcStr, const std::string_view candidates)
- std::string_view cut_head (const std::string_view srcStr, const std::string_view candidates)

  Remove characters from the beginning.
- std::string_view cut_tail (const std::string_view srcStr, const std::string_view candidates)

  Remove characters from the end.
- std::string_view cut_ends (const std::string_view srcStr, const std::string_view candidates)

  Remove characters from both ends.
- template<EWhitespace TYPE = EWhitespace::Common>
  std::string_view trim_head (const std::string_view srcStr)

  Remove white spaces from the beginning.
- template<EWhitespace TYPE = EWhitespace::Common>
  std::string_view trim_tail (const std::string_view srcStr)

  Remove white spaces from the end.
- template<EWhitespace TYPE = EWhitespace::Common>
  std::string_view trim (const std::string_view srcStr)

  Remove white spaces from both ends.
- std::string_view next_token (std::string_view srcStr, std::string_view *const out_remainingStr=nullptr, const std::string_view tokenSeparators=get_whitespaces<>())

  Retrieve a token from a string.
- char az_to_AZ (const char ch)

  Convert lower-case characters to upper-case.
- char AZ_to_az (const char ch)

  Convert upper-case characters to lower-case.
- void az_to_AZ (std::string &str)

  Convert lower-case characters to upper-case.
- void AZ_to_az (std::string &str)

  Convert upper-case characters to lower-case.
- std::string repeat (const std::string_view str, const std::size_t n)

  Repeat the input string for N times.
- void erase_all (std::string &str, const char ch)

  Remove all occurrence of a character in the string.
- template<typename T >
  T parse_float (const std::string_view floatStr)

  Returns a float by processing its string representation. Supports float, double, and long double.
- template<typename T >
  T parse_int (std::string_view intStr)

  Returns an integer by processing its string representation. Supports the following:
- template<typename NumberType >
  NumberType parse_number (const std::string_view numberStr)

  Returns a number by processing its string representation. Accepts all types supported by parse_float(std::string_view) and parse_int(std::string_view).
- template<typename T >
  std::size_t stringify_float (const T value, char *const out_buffer, const std::size_t bufferSize)

  Converts a float to string.

- template<std::integral T>
  std::size_t stringify_int_alphabetic (const T value, char *const out_buffer, const std::size_t bufferSize, const int base)

  Converts an integer to base [2, 62] string.
- template<std::integral T>
  std::size_t stringify_int (const T value, char *const out_buffer, const std::size_t bufferSize, const int base=10)

  Converts an integer to string.
- template<typename NumberType >
  std::size_t stringify_number (const NumberType value, char *const out_buffer, const std::size_t bufferSize)

  Converts a number to string. Accepts all types supported by stringify_float(T, char, std::size_t) and stringify_int(T, char*, std::size_t). The written string is not null terminated.*
- template<typename NumberType >
  std::string & stringify_number (const NumberType value, std::string &out_str, const std::size_t maxChars=64)

  Converts a number to string. Similar to `stringify_number(NumberType, char*, std::size_t)`, except that this variant writes to `std::string` and the resulting string is guaranteed to be null terminated (by calling `std::string::c_str()`).
- template<typename NumberType >
  std::string stringify_number (const NumberType value, const std::size_t maxChars=64)

  Converts a number to string. Similar to `stringify_number(NumberType, std::string&, std::size_t)`, except that this variant creates a new string.

7.7.1 Detailed Description

Contains various string manipulation helpers.

7.7.2 Enumeration Type Documentation

7.7.2.1 EWhitespace

```
enum class ph::string_utils::EWhitespace  [strong]
```

Enumerator

Common	Smaller set of whitespace characters that are often seen (see `table::common_whitespaces`).
Standard	Complete set of whitespace characters (see `table::standard_whitespaces`).

7.7.3 Function Documentation

7.7.3.1 AZ_to_az() [1/2]

```
char ph::string_utils::AZ_to_az (
            const char ch)  [inline]
```

Convert upper-case characters to lower-case.

Characters that are not English alphabets, or being lower-case already, will be preserved.

7.7.3.2 AZ_to_az() [2/2]

```
void ph::string_utils::AZ_to_az (
            std::string & str)  [inline]
```

Convert upper-case characters to lower-case.

Characters that are not English alphabets, or being lower-case already, will be preserved.

Parameters

in,out	*str*	String that is going to be converted in-place.

7.7.3.3 az_to_AZ() [1/2]

```
char ph::string_utils::az_to_AZ (
            const char ch)  [inline]
```

Convert lower-case characters to upper-case.

Characters that are not English alphabets, or being upper-case already, will be preserved.

7.7.3.4 az_to_AZ() [2/2]

```
void ph::string_utils::az_to_AZ (
            std::string & str)  [inline]
```

Convert lower-case characters to upper-case.

Characters that are not English alphabets, or being upper-case already, will be preserved.

Parameters

in,out	*str*	String that is going to be converted in-place.

7.7.3.5 cut_ends()

```
std::string_view ph::string_utils::cut_ends (
            const std::string_view srcStr,
            const std::string_view candidates)  [inline]
```

Remove characters from both ends.

Characters in `srcStr` will be removed from both ends if they match any of the character in `candidates`. The process stops once a mismatch is encountered.

Parameters

srcStr	String that is going to be cut.
candidates	The character set used to remove characters from `srcStr`.

Returns

The cut string.

7.7.3.6 cut_head()

```
std::string_view ph::string_utils::cut_head (
            const std::string_view srcStr,
            const std::string_view candidates)  [inline]
```

Remove characters from the beginning.

Characters in `srcStr` will be removed from the beginning if they match any of the character in `candidates`. The process stops once a mismatch is encountered.

Parameters

srcStr	String that is going to be cut.
candidates	The character set used to remove characters from `srcStr`.

Returns

The cut string.

7.7.3.7 cut_tail()

```
std::string_view ph::string_utils::cut_tail (
            const std::string_view srcStr,
            const std::string_view candidates)  [inline]
```

Remove characters from the end.

Characters in `srcStr` will be removed from the end if they match any of the character in `candidates`. The process stops once a mismatch is encountered.

Parameters

srcStr	String that is going to be cut.
candidates	The character set used to remove characters from `srcStr`.

Returns

The cut string.

7.7.3.8 erase_all()

```
void ph::string_utils::erase_all (
            std::string & str,
            const char ch)  [inline]
```

Remove all occurrence of a character in the string.

7.7.3.9 get_whitespaces()

```
template<EWhitespace TYPE = EWhitespace::Common>
std::string_view ph::string_utils::get_whitespaces ()  [inline]
```

7.7.3.10 has_any_of()

```
bool ph::string_utils::has_any_of (
            const std::string_view srcStr,
            const std::string_view candidates)  [inline]
```

7.7.3.11 has_none_of()

```
bool ph::string_utils::has_none_of (
            const std::string_view srcStr,
            const std::string_view candidates)  [inline]
```

7.7.3.12 is_whitespace()

```
template<EWhitespace TYPE = EWhitespace::Common>
bool ph::string_utils::is_whitespace (
            const char ch)  [inline], [constexpr]
```

7.7.3.13 next_token()

```
std::string_view ph::string_utils::next_token (
            std::string_view srcStr,
            std::string_view *const out_remainingStr = nullptr,
            const std::string_view tokenSeparators = get_whitespaces<>())  [inline]
```

Retrieve a token from a string.

Parameters

	srcStr	The string that token is going to be retrieved from.
out	*out_remainingStr*	If not null, stores the string with the retrieved token and its separator removed. Pointing to `srcStr` is valid, e.g., `next_token(str, &str)`.
	tokenSeparators	Charactors that separate the tokens. Defaults to whitespace characters.

7.7.3.14 parse_float()

```
template<typename T >
T ph::string_utils::parse_float (
            const std::string_view floatStr)  [inline]
```

Returns a float by processing its string representation. Supports float, double, and long double.

7.7.3.15 parse_int()

```
template<typename T >
T ph::string_utils::parse_int (
            std::string_view intStr)  [inline]
```

Returns an integer by processing its string representation. Supports the following:

1. Supports all signed and unsigned standard integer types (including `bool`).
2. Supports both base 10 (no prefix) and base 16 (0x prefix) inputs.

7.7.3.16 parse_number()

```
template<typename NumberType >
NumberType ph::string_utils::parse_number (
            const std::string_view numberStr)  [inline]
```

Returns a number by processing its string representation. Accepts all types supported by parse_float(std::string_view) and parse_int(std::string_view).

7.7.3.17 repeat()

```
std::string ph::string_utils::repeat (
            const std::string_view str,
            const std::size_t n)  [inline]
```

Repeat the input string for N times.

7.7.3.18 stringify_float()

```
template<typename T >
std::size_t ph::string_utils::stringify_float (
            const T value,
            char *const out_buffer,
            const std::size_t bufferSize)  [inline]
```

Converts a float to string.

Supports all built-in floating point types (e.g., float, double, and long double). The function expects a large enough `bufferSize` determined by the caller. The written string is not null terminated. By default, the stringified float guarantees round-trip conversion–feeding the converted string `s` from `value` to `parse_float()` will result in the same value.

Parameters

out_buffer	The buffer for storing the string.
bufferSize	Size of `out_buffer`.

Returns

Number of characters written to `out_buffer`.

Note

No dynamic memory allocation is performed.

7.7.3.19 stringify_int()

```
template<std::integral T>
std::size_t ph::string_utils::stringify_int (
            const T value,
            char *const out_buffer,
            const std::size_t bufferSize,
            const int base = 10)  [inline]
```

Converts an integer to string.

Supports all signed and unsigned standard integer types (including `bool`). The function expects a large enough `bufferSize` determined by the caller. The written string is not null terminated.

Parameters

out_buffer	The buffer for storing the string.
bufferSize	Size of `out_buffer`.

Returns

Number of characters written to `out_buffer`.

Note

No dynamic memory allocation is performed.

7.7.3.20 stringify_int_alphabetic()

```
template<std::integral T>
std::size_t ph::string_utils::stringify_int_alphabetic (
            const T value,
            char *const out_buffer,
            const std::size_t bufferSize,
            const int base)  [inline]
```

Converts an integer to base [2, 62] string.

Supports all signed and unsigned standard integer types (including `bool`). The function expects a large enough `bufferSize` determined by the caller. The written string is not null terminated.

Parameters

out_buffer	The buffer for storing the string.
bufferSize	Size of `out_buffer`.

Returns

Number of characters written to `out_buffer`.

Note

No dynamic memory allocation is performed.

7.7.3.21 stringify_number() [1/3]

```
template<typename NumberType >
std::size_t ph::string_utils::stringify_number (
            const NumberType value,
            char *const out_buffer,
            const std::size_t bufferSize)  [inline]
```

Converts a number to string. Accepts all types supported by stringify_float(T, char*, std::size_t) and stringify_int(T, char*, std::size_t). The written string is not null terminated.

Returns

Number of characters written to `out_buffer`.

7.7.3.22 stringify_number() [2/3]

```
template<typename NumberType >
std::string ph::string_utils::stringify_number (
            const NumberType value,
            const std::size_t maxChars = 64)  [inline]
```

Converts a number to string. Similar to `stringify_number(NumberType, std::string&, std::size_t)`, except that this variant creates a new string.

Parameters

out_str	The string to append the result to.

Returns

A new string that stores the number.

7.7.3.23 stringify_number() [3/3]

```
template<typename NumberType >
std::string & ph::string_utils::stringify_number (
            const NumberType value,
            std::string & out_str,
            const std::size_t maxChars = 64)  [inline]
```

Converts a number to string. Similar to `stringify_number(NumberType, char*, std::size_t)`, except that this variant writes to `std::string` and the resulting string is guaranteed to be null terminated (by calling `std::string::c_str()`).

Parameters

out_str	The string to append the result to.

Returns

`out_str` for convenience.

7.7.3.24 trim()

```
template<EWhitespace TYPE = EWhitespace::Common>
std::string_view ph::string_utils::trim (
            const std::string_view srcStr)  [inline]
```

Remove white spaces from both ends.

Parameters

srcStr	String that is going to be trimmed.

Returns

The trimmed string.

7.7.3.25 trim_head()

```
template<EWhitespace TYPE = EWhitespace::Common>
std::string_view ph::string_utils::trim_head (
            const std::string_view srcStr)  [inline]
```

Remove white spaces from the beginning.

Parameters

srcStr	String that is going to be trimmed.

Returns

The trimmed string.

7.7.3.26 trim_tail()

```
template<EWhitespace TYPE = EWhitespace::Common>
std::string_view ph::string_utils::trim_tail (
            const std::string_view srcStr)  [inline]
```

Remove white spaces from the end.

Parameters

srcStr	String that is going to be trimmed.

Returns

The trimmed string.

7.8 ph::string_utils::detail_from_to_char Namespace Reference

Functions

- void throw_from_std_errc_if_has_error (const std::errc errorCode)

7.8.1 Function Documentation

7.8.1.1 throw_from_std_errc_if_has_error()

```
void ph::string_utils::detail_from_to_char::throw_from_std_errc_if_has_error (
            const std::errc errorCode)  [inline]
```

7.9 ph::string_utils::table Namespace Reference

Variables

- constexpr std::string_view common_whitespaces = " \n\r\t"

  Commonly used whitespace characters.
- constexpr std::string_view standard_whitespaces = " \n\r\t\v\f"

  Standard whitespace characters.
- constexpr std::array< unsigned char, 256 > ASCII_TO_UPPER
- constexpr std::array< unsigned char, 256 > ASCII_TO_LOWER
- constexpr std::array< unsigned char, 62 > BASE62_DIGITS

7.9.1 Variable Documentation

7.9.1.1 ASCII_TO_LOWER

```
std::array<unsigned char, 256> ph::string_utils::table::ASCII_TO_LOWER  [inline], [constexpr]
```

Table for mapping standard ASCII character codes to lower case, i.e., A∼Z are mapped to a∼z (a∼z are also mapped to a∼z itself, so a case check can be eliminated). Any other codes will be left unchanged.

7.9.1.2 ASCII_TO_UPPER

```
std::array<unsigned char, 256> ph::string_utils::table::ASCII_TO_UPPER  [inline], [constexpr]
```

Table for mapping standard ASCII character codes to upper case, i.e., a∼z are mapped to A∼Z (A∼Z are also mapped to A∼Z itself, so a case check can be eliminated). Any other codes will be left unchanged.

7.9.1.3 BASE62_DIGITS

```
std::array<unsigned char, 62> ph::string_utils::table::BASE62_DIGITS  [inline], [constexpr]
```

Initial value:

```
=
{{
    '0', '1', '2', '3', '4', '5', '6', '7',
    '8', '9', 'a', 'b', 'c', 'd', 'e', 'f',
    'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
    'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
    'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D',
    'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
    'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
    'U', 'V', 'W', 'X', 'Y', 'Z'
}}
```

7.9.1.4 common_whitespaces

```
std::string_view ph::string_utils::table::common_whitespaces = " \n\r\t"  [inline], [constexpr]
```

Commonly used whitespace characters.

Ordered from high frequency to low frequency (approximated).

7.9.1.5 standard_whitespaces

```
std::string_view ph::string_utils::table::standard_whitespaces = " \n\r\t\v\f"  [inline],
[constexpr]
```

Standard whitespace characters.

Should idealy contain characters `ch` that satisfy `std::isspace(ch) == true`.

Chapter 8

Concept Documentation

8.1 ph::CPhotonException Concept Reference

`#include <exceptions.h>`

8.1.1 Concept definition

```
template<typename T>
concept ph::CPhotonException =
    std::is_base_of_v<RuntimeException, T> ||
    std::is_base_of_v<LogicalException, T>
```

8.2 ph::detail::CPermissiveImplicitLifetime Concept Reference

8.2.1 Concept definition

```
template<typename T>
concept ph::detail::CPermissiveImplicitLifetime =  std::disjunction_v<
    std::is_scalar<T>,
    std::is_array<T>,
    std::is_aggregate<T>,
    std::conjunction<
        std::is_trivially_destructible<T>,
        std::disjunction<
            std::is_trivially_default_constructible<T>,
            std::is_trivially_copy_constructible<T>,
            std::is_trivially_move_constructible<T>»
```

8.2.2 Detailed Description

Test whether `T` is an implicit-lifetime type. This concept is not exactly `std::is_implicit_lifetime_v`, it is slightly permissive in the sense that true implicit-lifetime type cannot have a user-provided destructor, and we are only testing if it is trivially destructible. See P2674R0 "A trait for implicit lifetime types" for more details.

8.3 ph::string_utils::CHasToString Concept Reference

`#include <string_utils.h>`

8.3.1 Concept definition

```
template<typename ObjType>
concept ph::string_utils::CHasToString =  requires (const ObjType& obj)
{
    { obj.toString() } -> std::convertible_to<std::string_view>;
}
```

Chapter 9

Class Documentation

9.1 ph::detail::AlignedMemoryDeleter Struct Reference

`#include <memory.h>`

Public Member Functions

- void operator() (void *const ptr) const
- void operator() (const void *const ptr) const

9.1.1 Member Function Documentation

9.1.1.1 operator()() [1/2]

```
void ph::detail::AlignedMemoryDeleter::operator() (
            const void *const ptr) const  [inline]
```

9.1.1.2 operator()() [2/2]

```
void ph::detail::AlignedMemoryDeleter::operator() (
            void *const ptr) const  [inline]
```

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

- Include/Common/memory.h

9.2 ph::CommandLineArguments Class Reference

Helper for parsing command line arguments.

`#include <CommandLineArguments.h>`

Public Member Functions

- CommandLineArguments (int argc, char ∗argv[ ])
- std::string getProgramName () const

  Get the program name.
- bool isEmpty () const

  Check if there are arguments yet to be parsed.
- std::string retrieveString (const std::string &defaultString="")

  Get the first argument passed in and remove it from the internal buffer.
- std::vector< std::string > retrieveStrings (std::size_t numValues)

  Get the first N arguments passed in and remove them from the internal buffer.
- std::vector< std::string > retrieveOptionArguments (const std::string &optionPrefix)

  Get the arguments for an option. This method assumes that the options specified are of the form "{- | --}<optionName> <arg0> <arg1> ...", i.e., options have a single or double dash prefix followed by its name, then the actual arguments. Careful that some input forms may still require manual treatment (using retrieve()) such as a filename starting with a dash or a negative number, since they can be misinterpreted as the next option and cause the argument list for the current option being ended prematurely.
- std::vector< std::string > retrieveStrings (const std::string &startingPrefix, const std::string &endingPrefix, bool shouldIncludeStart=true, bool shouldIncludeEnd=true)

  Get the arguments between a specified range.
- template<typename T >
  T retrieveInt (T defaultInt=0)

  Get an integer from the arguments. Similar to retrieveString(const std::string&), while the result is converted to an integer.
- template<typename T >
  T retrieveFloat (T defaultFloat=0.0f)

  Get a float from the arguments. Similar to retrieveString(const std::string&), while the result is converted to a float.
- template<typename T >
  std::optional< T > retrieve ()

9.2.1 Detailed Description

Helper for parsing command line arguments.

9.2.2 Constructor & Destructor Documentation

9.2.2.1 CommandLineArguments()

```
ph::CommandLineArguments::CommandLineArguments (
            int argc,
            char * argv[])
```

9.2.3 Member Function Documentation

9.2.3.1 getProgramName()

```
std::string ph::CommandLineArguments::getProgramName () const  [inline]
```

Get the program name.

Returns

An empty string if program name is not available.

9.2.3.2 isEmpty()

```
bool ph::CommandLineArguments::isEmpty () const  [inline]
```

Check if there are arguments yet to be parsed.

9.2.3.3 retrieve()

```
template<typename T >
std::optional< T > ph::CommandLineArguments::retrieve ()  [inline]
```

9.2.3.4 retrieveFloat()

```
template<typename T >
T ph::CommandLineArguments::retrieveFloat (
            T defaultFloat = 0.0f)  [inline]
```

Get a float from the arguments. Similar to retrieveString(const std::string&), while the result is converted to a float.

9.2.3.5 retrieveInt()

```
template<typename T >
T ph::CommandLineArguments::retrieveInt (
            T defaultInt = 0)  [inline]
```

Get an integer from the arguments. Similar to retrieveString(const std::string&), while the result is converted to an integer.

9.2.3.6 retrieveOptionArguments()

```
std::vector< std::string > ph::CommandLineArguments::retrieveOptionArguments (
            const std::string & optionPrefix)
```

Get the arguments for an option. This method assumes that the options specified are of the form "{- | --}<optionName> <arg0> <arg1> ...", i.e., options have a single or double dash prefix followed by its name, then the actual arguments. Careful that some input forms may still require manual treatment (using retrieve()) such as a filename starting with a dash or a negative number, since they can be misinterpreted as the next option and cause the argument list for the current option being ended prematurely.

Parameters

optionPrefix	The option's prefix.

9.2.3.7 retrieveString()

```
std::string ph::CommandLineArguments::retrieveString (
            const std::string & defaultString = "")
```

Get the first argument passed in and remove it from the internal buffer.

Parameters

defaultString	A default value if the operation cannot be done (such as isEmpty() is true).

9.2.3.8 retrieveStrings() [1/2]

```
std::vector< std::string > ph::CommandLineArguments::retrieveStrings (
            const std::string & startingPrefix,
            const std::string & endingPrefix,
            bool shouldIncludeStart = true,
            bool shouldIncludeEnd = true)
```

Get the arguments between a specified range.

Parameters

startingPrefix	The first argument's prefix.
endingPrefix	The last argument's prefix.
shouldIncludeStart	Whether to include the first matching argument.
shouldIncludeEnd	Whether to include the last matching argument.

9.2.3.9 retrieveStrings() [2/2]

```
std::vector< std::string > ph::CommandLineArguments::retrieveStrings (
            std::size_t numValues)
```

Get the first N arguments passed in and remove them from the internal buffer.

Parameters

numValues	Number of values to be retrieved at once.

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

- Include/Common/Utility/CommandLineArguments.h
- Source/Common/Utility/CommandLineArguments.cpp

9.3 ph::Config Class Reference

```
#include <config.h>
```

Static Public Member Functions

- static std::string & RENDERER_RESOURCE_DIRECTORY ()

9.3.1 Member Function Documentation

9.3.1.1 RENDERER_RESOURCE_DIRECTORY()

```
std::string & ph::Config::RENDERER_RESOURCE_DIRECTORY ()  [static]
```

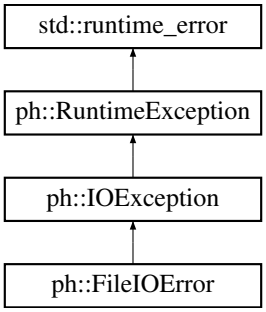
The documentation for this class was generated from the following files:

- Include/Common/config.h
- Source/Common/config.cpp

9.4 ph::FileIOError Class Reference

```
#include <io_exceptions.h>
```

Inheritance diagram for ph::FileIOError:



Public Member Functions

- FileIOError (const std::string &message)
- FileIOError (const char *message)
- FileIOError (const std::string &message, std::string filename)
- std::string whatStr () const override

Public Member Functions inherited from ph::IOException

- IOException (const std::string &message)
- IOException (const char *message)

Public Member Functions inherited from ph::RuntimeException

- RuntimeException (const std::string &message)
- RuntimeException (const char *message)
- ~RuntimeException () override=default

9.4.1 Constructor & Destructor Documentation

9.4.1.1 FileIOError() [1/3]

```
ph::FileIOError::FileIOError (
            const std::string & message)  [inline], [explicit]
```

9.4.1.2 FileIOError() [2/3]

```
ph::FileIOError::FileIOError (
            const char * message)  [inline], [explicit]
```

9.4.1.3 FileIOError() [3/3]

```
ph::FileIOError::FileIOError (
            const std::string & message,
            std::string filename)  [inline]
```

9.4.2 Member Function Documentation

9.4.2.1 whatStr()

```
std::string ph::FileIOError::whatStr () const  [inline], [override], [virtual]
```

Reimplemented from ph::RuntimeException.

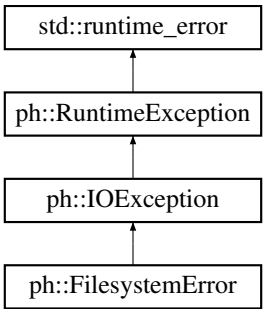
The documentation for this class was generated from the following file:

- Include/Common/io_exceptions.h

9.5 ph::FilesystemError Class Reference

```
#include <io_exceptions.h>
```

Inheritance diagram for ph::FilesystemError:



Public Member Functions

- FilesystemError (std::error_code errorCode)
- FilesystemError (const std::string &message, std::error_code errorCode)
- std::string whatStr () const override
- IOException (const std::string &message)
- IOException (const char *message)

Public Member Functions inherited from ph::IOException

- IOException (const std::string &message)
- IOException (const char *message)

Public Member Functions inherited from ph::RuntimeException

- RuntimeException (const std::string &message)
- RuntimeException (const char *message)
- ~RuntimeException () override=default

9.5.1 Constructor & Destructor Documentation

9.5.1.1 FilesystemError() [1/2]

```
ph::FilesystemError::FilesystemError (
            std::error_code errorCode)  [inline], [explicit]
```

9.5.1.2 FilesystemError() [2/2]

```
ph::FilesystemError::FilesystemError (
            const std::string & message,
            std::error_code errorCode)  [inline]
```

9.5.2 Member Function Documentation

9.5.2.1 IOException() [1/2]

```
ph::IOException::IOException (
            const char * message)  [inline], [explicit]
```

9.5.2.2 IOException() [2/2]

```
ph::IOException::IOException (
            const std::string & message)  [inline], [explicit]
```

9.5.2.3 whatStr()

```
std::string ph::FilesystemError::whatStr () const  [inline], [override], [virtual]
```

Reimplemented from ph::RuntimeException.

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

- Include/Common/io_exceptions.h

9.6 ph::TimerStatsReport::GroupedTimeRecord Struct Reference

```
#include <stats.h>
```

Public Member Functions

- GroupedTimeRecord ()

Public Attributes

- std::string groupName
- std::uint64_t totalMicroseconds
- std::uint64_t count
- std::vector< GroupedTimeRecord > subgroups

9.6.1 Constructor & Destructor Documentation

9.6.1.1 GroupedTimeRecord()

```
ph::TimerStatsReport::GroupedTimeRecord::GroupedTimeRecord ()
```

9.6.2 Member Data Documentation

9.6.2.1 count

```
std::uint64_t ph::TimerStatsReport::GroupedTimeRecord::count
```

9.6.2.2 groupName

```
std::string ph::TimerStatsReport::GroupedTimeRecord::groupName
```

9.6.2.3 subgroups

```
std::vector<GroupedTimeRecord> ph::TimerStatsReport::GroupedTimeRecord::subgroups
```

9.6.2.4 totalMicroseconds

```
std::uint64_t ph::TimerStatsReport::GroupedTimeRecord::totalMicroseconds
```

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

- Include/Common/stats.h
- Source/Common/stats.cpp

9.7 ph::detail::HeterogeneousStringHash Struct Reference

```
#include <detail.h>
```

Public Types

- using is_transparent = void

Public Member Functions

- std::size_t operator() (const char *txt) const
- std::size_t operator() (std::string_view txt) const
- std::size_t operator() (const std::string &txt) const

9.7.1 Member Typedef Documentation

9.7.1.1 is_transparent

```
using ph::detail::HeterogeneousStringHash::is_transparent = void
```

9.7.2 Member Function Documentation

9.7.2.1 operator()() [1/3]

```
std::size_t ph::detail::HeterogeneousStringHash::operator() (
            const char * txt) const  [inline], [nodiscard]
```

9.7.2.2 operator()() [2/3]

```
std::size_t ph::detail::HeterogeneousStringHash::operator() (
            const std::string & txt) const  [inline], [nodiscard]
```

9.7.2.3 operator()() [3/3]

```
std::size_t ph::detail::HeterogeneousStringHash::operator() (
            std::string_view txt) const  [inline], [nodiscard]
```

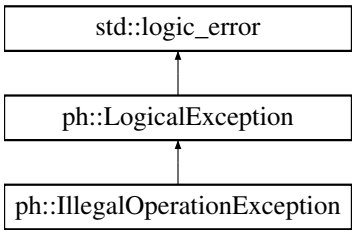
The documentation for this struct was generated from the following file:

- Include/Common/Container/detail.h

9.8 ph::IllegalOperationException Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for ph::IllegalOperationException:



Public Member Functions

- LogicalException (const std::string &message)
- LogicalException (const char *message)

Public Member Functions inherited from ph::LogicalException

- LogicalException (const std::string &message)
- LogicalException (const char *message)
- ~LogicalException () override=default
- virtual std::string whatStr () const

9.8.1 Member Function Documentation

9.8.1.1 LogicalException() [1/2]

```
ph::LogicalException::LogicalException (
            const char * message)  [explicit]
```

9.8.1.2 LogicalException() [2/2]

```
ph::LogicalException::LogicalException (
            const std::string & message)  [explicit]
```

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

- Include/Common/exceptions.h

9.9 ph::IniFile Class Reference

INI file I/O. This class is useful for recording various settings across the entire engine project. As a low-level I/O class in `Common` library, it can be used regardless the engine initialization state (see `init_render_engine()` and `exit_render_engine()` in `Engine` library for more details).

`#include <IniFile.h>`

Public Member Functions

- IniFile ()

  Creates a file with no content. An empty section is made current.
- IniFile (const std::string &iniFilePath)
- void save (const std::string &iniFilePath)
- void clear ()
- std::size_t numSections () const
- std::string_view getSectionName (std::size_t sectionIdx) const
- std::string_view getCurrentSectionName () const
- std::optional< std::size_t > findSectionIndex (std::string_view sectionName) const
- void setCurrentSection (std::size_t sectionIdx)
- void setCurrentSection (std::string_view sectionName, bool createIfNotExist=true)
- std::size_t numProperties () const
- std::string_view getPropertyName (std::size_t propertyIdx) const
- std::string_view getPropertyValue (std::size_t propertyIdx) const
- std::optional< std::size_t > findPropertyIndex (std::string_view propertyName) const
- void setProperty (std::size_t propertyIdx, std::string_view propertyValue)

  Set a property under current section by index.
- void setProperty (std::string_view propertyName, std::string_view propertyValue, bool createIfNotExist=true)

  Set a property under current section by name.
- void append (const IniFile &other)

  Add another INI file to this one. All properties from the other file will be added to this one. New sections will be created if they were not in this file. Properties will be overwritten if they were already defined in this file.

Static Public Member Functions

- static IniFile read (const std::string &iniFilePath)

9.9.1 Detailed Description

INI file I/O. This class is useful for recording various settings across the entire engine project. As a low-level I/O class in `Common` library, it can be used regardless the engine initialization state (see `init_render_engine()` and `exit_render_engine()` in `Engine` library for more details).

9.9.2 Constructor & Destructor Documentation

9.9.2.1 IniFile() [1/2]

`ph::IniFile::IniFile ()`

Creates a file with no content. An empty section is made current.

9.9.2.2 IniFile() [2/2]

```
ph::IniFile::IniFile (
            const std::string & iniFilePath)  [explicit]
```

9.9.3 Member Function Documentation

9.9.3.1 append()

```
void ph::IniFile::append (
            const IniFile & other)
```

Add another INI file to this one. All properties from the other file will be added to this one. New sections will be created if they were not in this file. Properties will be overwritten if they were already defined in this file.

9.9.3.2 clear()

```
void ph::IniFile::clear ()  [inline]
```

9.9.3.3 findPropertyIndex()

```
std::optional< std::size_t > ph::IniFile::findPropertyIndex (
            std::string_view propertyName) const
```

9.9.3.4 findSectionIndex()

```
std::optional< std::size_t > ph::IniFile::findSectionIndex (
            std::string_view sectionName) const
```

9.9.3.5 getCurrentSectionName()

```
std::string_view ph::IniFile::getCurrentSectionName () const  [inline]
```

9.9.3.6 getPropertyName()

```
std::string_view ph::IniFile::getPropertyName (
            std::size_t propertyIdx) const  [inline]
```

9.9.3.7 getPropertyValue()

```
std::string_view ph::IniFile::getPropertyValue (
            std::size_t propertyIdx) const  [inline]
```

9.9.3.8 getSectionName()

```
std::string_view ph::IniFile::getSectionName (
            std::size_t sectionIdx) const  [inline]
```

9.9.3.9 numProperties()

```
std::size_t ph::IniFile::numProperties () const  [inline]
```

9.9.3.10 numSections()

```
std::size_t ph::IniFile::numSections () const  [inline]
```

9.9.3.11 read()

```
IniFile ph::IniFile::read (
            const std::string & iniFilePath)  [static]
```

9.9.3.12 save()

```
void ph::IniFile::save (
            const std::string & iniFilePath)
```

9.9.3.13 setCurrentSection() [1/2]

```
void ph::IniFile::setCurrentSection (
            std::size_t sectionIdx)  [inline]
```

9.9.3.14 setCurrentSection() [2/2]

```
void ph::IniFile::setCurrentSection (
            std::string_view sectionName,
            bool createIfNotExist = true)
```

9.9.3.15 setProperty() [1/2]

```
void ph::IniFile::setProperty (
            std::size_t propertyIdx,
            std::string_view propertyValue)  [inline]
```

Set a property under current section by index.

9.9.3.16 setProperty() [2/2]

```
void ph::IniFile::setProperty (
            std::string_view propertyName,
            std::string_view propertyValue,
            bool createIfNotExist = true)
```

Set a property under current section by name.

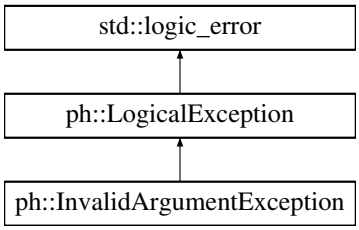
The documentation for this class was generated from the following files:

- Include/Common/Config/IniFile.h
- Source/Common/Config/IniFile.cpp

9.10 ph::InvalidArgumentException Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for ph::InvalidArgumentException:



Public Member Functions

- LogicalException (const std::string &message)
- LogicalException (const char *message)

Public Member Functions inherited from ph::LogicalException

- LogicalException (const std::string &message)
- LogicalException (const char *message)
- ~LogicalException () override=default
- virtual std::string whatStr () const

9.10.1 Member Function Documentation

9.10.1.1 LogicalException() [1/2]

```
ph::LogicalException::LogicalException (
            const char * message)  [explicit]
```

9.10.1.2 LogicalException() [2/2]

```
ph::LogicalException::LogicalException (
            const std::string & message)  [explicit]
```

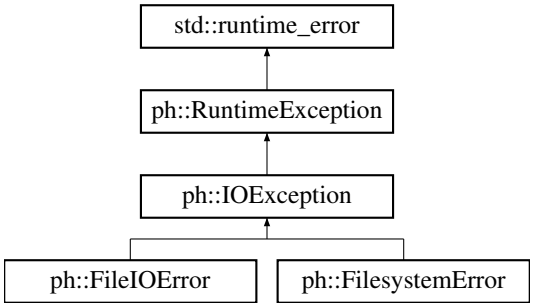
The documentation for this class was generated from the following file:

- Include/Common/exceptions.h

9.11 ph::IOException Class Reference

```
#include <io_exceptions.h>
```

Inheritance diagram for ph::IOException:



Public Member Functions

- IOException (const std::string &message)
- IOException (const char *message)

Public Member Functions inherited from ph::RuntimeException

- RuntimeException (const std::string &message)
- RuntimeException (const char *message)
- ~RuntimeException () override=default
- virtual std::string whatStr () const

9.11.1 Constructor & Destructor Documentation

9.11.1.1 IOException() [1/2]

```
ph::IOException::IOException (
            const std::string & message)  [inline], [explicit]
```

9.11.1.2 IOException() [2/2]

```
ph::IOException::IOException (
            const char * message)  [inline], [explicit]
```

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

- Include/Common/io_exceptions.h

9.12 ph::Logger Class Reference

```
#include <Logger.h>
```

Public Member Functions

- Logger ()
- void log (std::string_view message) const
- void log (ELogLevel logLevel, std::string_view message) const
- void log (std::string_view name, ELogLevel logLevel, std::string_view message) const
- void addLogHandler (LogHandler logHandler)

  Add a log handler that can deal with log messages. Log handler must be copyable.

Static Public Member Functions

- static LogHandler makeStdOutLogPrinter ()
- static LogHandler makeColoredStdOutLogPrinter ()

9.12.1 Constructor & Destructor Documentation

9.12.1.1 Logger()

```
ph::Logger::Logger ()
```

9.12.2 Member Function Documentation

9.12.2.1 addLogHandler()

```
void ph::Logger::addLogHandler (
            LogHandler logHandler)
```

Add a log handler that can deal with log messages. Log handler must be copyable.

9.12.2.2 log() [1/3]

```
void ph::Logger::log (
            ELogLevel logLevel,
            std::string_view message) const
```

9.12.2.3 log() [2/3]

```
void ph::Logger::log (
            std::string_view message) const
```

9.12.2.4 log() [3/3]

```
void ph::Logger::log (
            std::string_view name,
            ELogLevel logLevel,
            std::string_view message) const
```

9.12.2.5 makeColoredStdOutLogPrinter()

```
auto ph::Logger::makeColoredStdOutLogPrinter ()  [static]
```

9.12.2.6 makeStdOutLogPrinter()

```
auto ph::Logger::makeStdOutLogPrinter ()  [static]
```

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

- Include/Common/Log/Logger.h
- Source/Common/Log/Logger.cpp

9.13 ph::LogGroup Struct Reference

```
#include <logging.h>
```

Public Attributes

- std::string groupName
- std::string category

9.13.1 Member Data Documentation

9.13.1.1 category

```
std::string ph::LogGroup::category
```

9.13.1.2 groupName

```
std::string ph::LogGroup::groupName
```

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

- Include/Common/logging.h

9.14 ph::LogGroups Class Reference

```
#include <logging.h>
```

Public Member Functions

- LogGroups ()=default
- LogGroups (const LogGroups &other)=default
- LogGroups & operator= (const LogGroups &rhs)=default
- std::size_t addGroup (std::string_view groupName, std::string_view category="")
- std::size_t numGroups () const
- const LogGroup & getGroup (std::size_t index) const

9.14.1 Constructor & Destructor Documentation

9.14.1.1 LogGroups() [1/2]

```
ph::LogGroups::LogGroups ()  [inline], [default]
```

9.14.1.2 LogGroups() [2/2]

```
ph::LogGroups::LogGroups (
            const LogGroups & other)  [inline], [default]
```

9.14.2 Member Function Documentation

9.14.2.1 addGroup()

```
std::size_t ph::LogGroups::addGroup (
            std::string_view groupName,
            std::string_view category = "")
```

9.14.2.2 getGroup()

```
const LogGroup & ph::LogGroups::getGroup (
            std::size_t index) const
```

9.14.2.3 numGroups()

```
std::size_t ph::LogGroups::numGroups () const
```

9.14.2.4 operator=()

```
LogGroups & ph::LogGroups::operator= (
            const LogGroups & rhs)  [inline], [default]
```

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

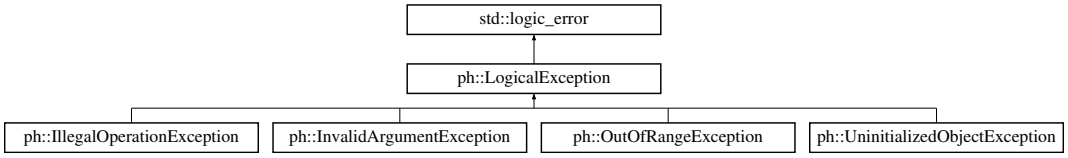
- Include/Common/logging.h
- Source/Common/logging.cpp

9.15 ph::LogicalException Class Reference

General exception thrown on logical error.

```
#include <exceptions.h>
```

Inheritance diagram for ph::LogicalException:



Public Member Functions

- LogicalException (const std::string &message)
- LogicalException (const char *message)
- ~LogicalException () override=default
- virtual std::string whatStr () const

9.15.1 Detailed Description

General exception thrown on logical error.

9.15.2 Constructor & Destructor Documentation

9.15.2.1 LogicalException() [1/2]

```
ph::LogicalException::LogicalException (
            const std::string & message)  [explicit]
```

9.15.2.2 LogicalException() [2/2]

```
ph::LogicalException::LogicalException (
            const char * message)  [explicit]
```

9.15.2.3 ~LogicalException()

```
ph::LogicalException::~LogicalException ()  [inline], [override], [default]
```

9.15.3 Member Function Documentation

9.15.3.1 whatStr()

```
std::string ph::LogicalException::whatStr () const  [virtual]
```

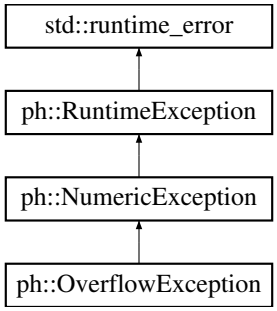
The documentation for this class was generated from the following files:

- Include/Common/exceptions.h
- Source/Common/exception.cpp

9.16 ph::NumericException Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for ph::NumericException:



Public Member Functions

- RuntimeException (const std::string &message)
- RuntimeException (const char *message)

Public Member Functions inherited from ph::RuntimeException

- RuntimeException (const std::string &message)
- RuntimeException (const char *message)
- ~RuntimeException () override=default
- virtual std::string whatStr () const

9.16.1 Member Function Documentation

9.16.1.1 RuntimeException() [1/2]

```
ph::RuntimeException::RuntimeException (
            const char * message)  [explicit]
```

9.16.1.2 RuntimeException() [2/2]

```
ph::RuntimeException::RuntimeException (
            const std::string & message)  [explicit]
```

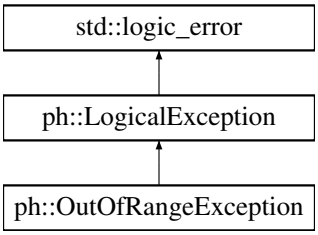
The documentation for this class was generated from the following file:

- Include/Common/exceptions.h

9.17 ph::OutOfRangeException Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for ph::OutOfRangeException:



Public Member Functions

- LogicalException (const std::string &message)
- LogicalException (const char *message)

Public Member Functions inherited from ph::LogicalException

- LogicalException (const std::string &message)
- LogicalException (const char *message)
- ~LogicalException () override=default
- virtual std::string whatStr () const

9.17.1 Member Function Documentation

9.17.1.1 LogicalException() [1/2]

```
ph::LogicalException::LogicalException (
            const char * message)  [explicit]
```

9.17.1.2 LogicalException() [2/2]

```
ph::LogicalException::LogicalException (
            const std::string & message)  [explicit]
```

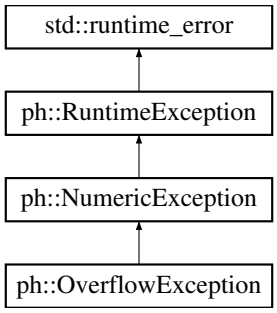
The documentation for this class was generated from the following file:

- Include/Common/exceptions.h

9.18 ph::OverflowException Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for ph::OverflowException:



Additional Inherited Members

Public Member Functions inherited from ph::NumericException

- RuntimeException (const std::string &message)
- RuntimeException (const char *message)

Public Member Functions inherited from ph::RuntimeException

- RuntimeException (const std::string &message)
- RuntimeException (const char *message)
- ~RuntimeException () override=default
- virtual std::string whatStr () const

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

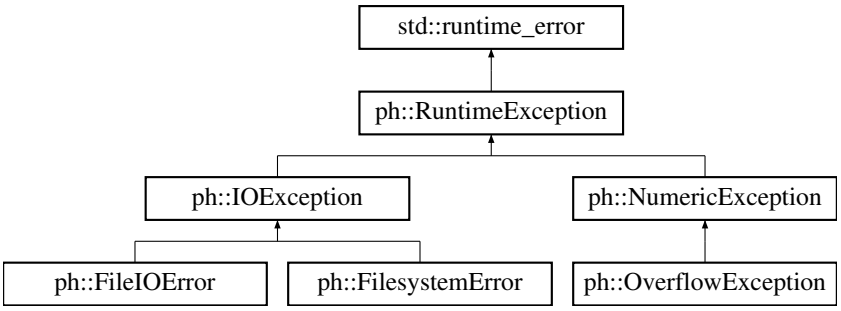
- Include/Common/exceptions.h

9.19 ph::RuntimeException Class Reference

General exception thrown on runtime error.

`#include <exceptions.h>`

Inheritance diagram for ph::RuntimeException:



Public Member Functions

- RuntimeException (const std::string &message)
- RuntimeException (const char *message)
- ~RuntimeException () override=default
- virtual std::string whatStr () const

9.19.1 Detailed Description

General exception thrown on runtime error.

9.19.2 Constructor & Destructor Documentation

9.19.2.1 RuntimeException() [1/2]

```
ph::RuntimeException::RuntimeException (
            const std::string & message)  [explicit]
```

9.19.2.2 RuntimeException() [2/2]

```
ph::RuntimeException::RuntimeException (
            const char * message)  [explicit]
```

9.19.2.3 ~RuntimeException()

```
ph::RuntimeException::~RuntimeException ()  [inline], [override], [default]
```

9.19.3 Member Function Documentation

9.19.3.1 whatStr()

```
std::string ph::RuntimeException::whatStr () const  [virtual]
```

Reimplemented in ph::FileIOError, and ph::FilesystemError.

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

- Include/Common/exceptions.h
- Source/Common/exception.cpp

9.20 ph::detail::stats::ScopedTimer Struct Reference

```
#include <stats.h>
```

Public Types

- using Clock = std::chrono::steady_clock

Public Member Functions

- ScopedTimer (TimeCounter &counter)
- ~ScopedTimer ()

Public Attributes

- TimeCounter & counter
- Clock::time_point startTime

9.20.1 Member Typedef Documentation

9.20.1.1 Clock

```
using ph::detail::stats::ScopedTimer::Clock = std::chrono::steady_clock
```

9.20.2 Constructor & Destructor Documentation

9.20.2.1 ScopedTimer()

```
ph::detail::stats::ScopedTimer::ScopedTimer (
            TimeCounter & counter)  [explicit]
```

9.20.2.2 ~ScopedTimer()

```
ph::detail::stats::ScopedTimer::~ScopedTimer ()
```

9.20.3 Member Data Documentation

9.20.3.1 counter

```
TimeCounter& ph::detail::stats::ScopedTimer::counter
```

9.20.3.2 startTime

```
Clock::time_point ph::detail::stats::ScopedTimer::startTime
```

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

- Include/Common/stats.h
- Source/Common/stats.cpp

9.21 ph::detail::stats::TimeCounter Struct Reference

```
#include <stats.h>
```

Public Member Functions

- TimeCounter (std::string name, std::string category)
- void addMicroseconds (std::uint64_t microseconds)

Public Attributes

- const std::string name
- const std::string category
- std::atomic_uint64_t totalMicroseconds
- std::atomic_uint64_t count

9.21.1 Constructor & Destructor Documentation

9.21.1.1 TimeCounter()

```
ph::detail::stats::TimeCounter::TimeCounter (
            std::string name,
            std::string category)
```

9.21.2 Member Function Documentation

9.21.2.1 addMicroseconds()

```
void ph::detail::stats::TimeCounter::addMicroseconds (
            std::uint64_t microseconds)
```

9.21.3 Member Data Documentation

9.21.3.1 category

```
const std::string ph::detail::stats::TimeCounter::category
```

9.21.3.2 count

```
std::atomic_uint64_t ph::detail::stats::TimeCounter::count
```

9.21.3.3 name

```
const std::string ph::detail::stats::TimeCounter::name
```

9.21.3.4 totalMicroseconds

```
std::atomic_uint64_t ph::detail::stats::TimeCounter::totalMicroseconds
```

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

- Include/Common/stats.h
- Source/Common/stats.cpp

9.22 ph::TimerStatsReport::TimeRecord Struct Reference

```
#include <stats.h>
```

Public Member Functions

- TimeRecord ()

Public Attributes

- std::string name
- std::string category
- std::uint64_t totalMicroseconds
- std::uint64_t count

9.22.1 Constructor & Destructor Documentation

9.22.1.1 TimeRecord()

```
ph::TimerStatsReport::TimeRecord::TimeRecord ()
```

9.22.2 Member Data Documentation

9.22.2.1 category

```
std::string ph::TimerStatsReport::TimeRecord::category
```

9.22.2.2 count

```
std::uint64_t ph::TimerStatsReport::TimeRecord::count
```

9.22.2.3 name

```
std::string ph::TimerStatsReport::TimeRecord::name
```

9.22.2.4 totalMicroseconds

```
std::uint64_t ph::TimerStatsReport::TimeRecord::totalMicroseconds
```

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

- Include/Common/stats.h
- Source/Common/stats.cpp

9.23 ph::TimerStatsReport Class Reference

```
#include <stats.h>
```

Classes

- struct GroupedTimeRecord
- struct TimeRecord

Public Member Functions

- TimerStatsReport ()
- GroupedTimeRecord getGroupedTimeRecord () const
- std::string proportionalReport () const
- std::string averagedReport () const
- std::string detailedReport () const
- std::string rawReport () const

9.23.1 Constructor & Destructor Documentation

9.23.1.1 TimerStatsReport()

```
ph::TimerStatsReport::TimerStatsReport ()
```

9.23.2 Member Function Documentation

9.23.2.1 averagedReport()

```
std::string ph::TimerStatsReport::averagedReport () const
```

9.23.2.2 detailedReport()

```
std::string ph::TimerStatsReport::detailedReport () const
```

9.23.2.3 getGroupedTimeRecord()

```
TimerStatsReport::GroupedTimeRecord ph::TimerStatsReport::getGroupedTimeRecord () const
```

9.23.2.4 proportionalReport()

```
std::string ph::TimerStatsReport::proportionalReport () const
```

9.23.2.5 rawReport()

```
std::string ph::TimerStatsReport::rawReport () const
```

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

- Include/Common/stats.h
- Source/Common/stats.cpp

9.24 ph::Timestamp Class Reference

Represents a point in time.

```
#include <Timestamp.h>
```

Public Member Functions

- Timestamp ()
- std::string toYMD () const
- std::string toHMS () const
- std::string toHMSMilliseconds () const
- std::string toHMSMicroseconds () const
- std::string toYMDHMS () const
- std::string toYMDHMSMilliseconds () const
- std::string toYMDHMSMicroseconds () const
- std::string toString () const

9.24.1 Detailed Description

Represents a point in time.

This class represents the local time on which an instance of it has been created. It is guaranteed to be thread-safe.

9.24.2 Constructor & Destructor Documentation

9.24.2.1 Timestamp()

```
ph::Timestamp::Timestamp ()  [inline]
```

9.24.3 Member Function Documentation

9.24.3.1 toHMS()

```
std::string ph::Timestamp::toHMS () const
```

9.24.3.2 toHMSMicroseconds()

```
std::string ph::Timestamp::toHMSMicroseconds () const
```

9.24.3.3 toHMSMilliseconds()

```
std::string ph::Timestamp::toHMSMilliseconds () const
```

9.24.3.4 toString()

```
std::string ph::Timestamp::toString () const
```

9.24.3.5 toYMD()

```
std::string ph::Timestamp::toYMD () const
```

9.24.3.6 toYMDHMS()

```
std::string ph::Timestamp::toYMDHMS () const
```

9.24.3.7 toYMDHMSMicroseconds()

```
std::string ph::Timestamp::toYMDHMSMicroseconds () const
```

9.24.3.8 toYMDHMSMilliseconds()

```
std::string ph::Timestamp::toYMDHMSMilliseconds () const
```

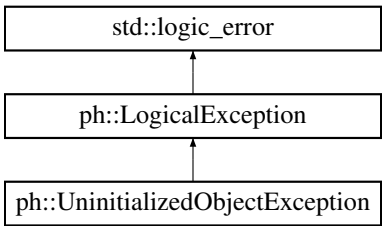
The documentation for this class was generated from the following files:

- Include/Common/Utility/Timestamp.h
- Source/Common/Utility/Timestamp.cpp

9.25 ph::UninitializedObjectException Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for ph::UninitializedObjectException:



Public Member Functions

- LogicalException (const std::string &message)
- LogicalException (const char *message)

Public Member Functions inherited from ph::LogicalException

- LogicalException (const std::string &message)
- LogicalException (const char *message)
- ~LogicalException () override=default
- virtual std::string whatStr () const

9.25.1 Member Function Documentation

9.25.1.1 LogicalException() [1/2]

```
ph::LogicalException::LogicalException (
            const char * message)  [explicit]
```

9.25.1.2 LogicalException() [2/2]

```
ph::LogicalException::LogicalException (
            const std::string & message)  [explicit]
```

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

- Include/Common/exceptions.h

Chapter 10

File Documentation

10.1 Documentation/namespace_ph.dox File Reference

Namespaces

- namespace ph

  The root for all renderer implementations.

10.2 Documentation/namespace_ph_detail.dox File Reference

Namespaces

- namespace ph::detail

  Implementation detail mainly for internal usages.

10.3 Documentation/namespace_ph_detail_core_logging.dox File Reference

Namespaces

- namespace ph::detail::core_logging

  Core logging functionalities. Most logs will output information (logs) via a main logger, which we refer to as "core logger". This namespace contains implementation details for core logging functionalities.

10.4 Documentation/namespace_ph_string_utils.dox File Reference

Namespaces

- namespace ph::string_utils

  Contains various string manipulation helpers.

10.5 Include/Common/assertion.h File Reference

```
#include "Common/config.h"
#include "Common/macro.h"
#include <string>
```

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::detail

  Implementation detail mainly for internal usages.

Macros

- #define PH_ASSERT_MSG(condition, message) PH_NO_OP()
- #define PH_INTERNAL_RANGE_MSG(value, lowerBound, upperBound, lowerBoundSymbol, upperBound↩ Symbol) PH_NO_OP()
- #define PH_ASSERT(condition) PH_ASSERT_MSG(condition, "")
- #define PH_ASSERT_UNREACHABLE_SECTION() PH_ASSERT_MSG(false, "executing supposedly unreachable code")
- #define PH_ASSERT_EQ(a, b) PH_ASSERT_MSG(a == b, std::string(#a) + " = " + std::to_string(a) + ", " + #b + " = " + std::to_string(b))
- #define PH_ASSERT_NE(a, b) PH_ASSERT_MSG(a != b, std::string(#a) + " = " + std::to_string(a) + ", " + #b + " = " + std::to_string(b))
- #define PH_ASSERT_GT(a, b) PH_ASSERT_MSG(a > b, std::string(#a) + " = " + std::to_string(a) + ", " + #b + " = " + std::to_string(b))
- #define PH_ASSERT_LT(a, b) PH_ASSERT_MSG(a < b, std::string(#a) + " = " + std::to_string(a) + ", " + #b + " = " + std::to_string(b))
- #define PH_ASSERT_GE(a, b) PH_ASSERT_MSG(a >= b, std::string(#a) + " = " + std::to_string(a) + ", " + #b + " = " + std::to_string(b))
- #define PH_ASSERT_LE(a, b) PH_ASSERT_MSG(a <= b, std::string(#a) + " = " + std::to_string(a) + ", " + #b + " = " + std::to_string(b))
- #define PH_ASSERT_IN_RANGE(value, begin, end) PH_ASSERT_MSG(begin <= value && value < end, PH_INTERNAL_RANGE_MSG(value, begin, end, "[", ")"))

  Assert that `value` is within [`begin`, `end`).
- #define PH_ASSERT_IN_RANGE_INCLUSIVE(value, lowerBound, upperBound) PH_ASSERT_MSG(lower↩ Bound <= value && value <= upperBound, PH_INTERNAL_RANGE_MSG(value, lowerBound, upperBound, "[", "]"))

  Similar to PH_ASSERT_IN_RANGE(3), except the bounds are inclusive.
- #define PH_ASSERT_IN_RANGE_EXCLUSIVE(value, lowerBound, upperBound) PH_ASSERT_MSG(lower↩ Bound < value && value < upperBound, PH_INTERNAL_RANGE_MSG(value, lowerBound, upperBound, "(", ")"))

  Similar to PH_ASSERT_IN_RANGE(3), except the bounds are exclusive.
- #define PH_STATIC_ASSERT_DEPENDENT_FALSE(DependentType, message) static_assert(::ph::detail::DEPENDENT_FAL Type>, #message)

Functions

- void ph::detail::output_assertion_message (const std::string &filename, const std::string &lineNumber, const std::string &condition, const std::string &message)
- void ph::detail::on_assertion_failed ()

Variables

- template<typename T >
  constexpr bool ph::detail::DEPENDENT_FALSE = false

10.5.1 Macro Definition Documentation

10.5.1.1 PH_ASSERT

```
#define PH_ASSERT(
              condition)  PH_ASSERT_MSG(condition, "")
```

10.5.1.2 PH_ASSERT_EQ

```
#define PH_ASSERT_EQ(
              a,
              b)  PH_ASSERT_MSG(a == b, std::string(#a) + " = " + std::to_string(a) + ", " +
#b + " = " + std::to_string(b))
```

10.5.1.3 PH_ASSERT_GE

```
#define PH_ASSERT_GE(
              a,
              b)  PH_ASSERT_MSG(a >= b, std::string(#a) + " = " + std::to_string(a) + ", " +
#b + " = " + std::to_string(b))
```

10.5.1.4 PH_ASSERT_GT

```
#define PH_ASSERT_GT(
              a,
              b)  PH_ASSERT_MSG(a > b, std::string(#a) + " = " + std::to_string(a) + ", " + #b
+ " = " + std::to_string(b))
```

10.5.1.5 PH_ASSERT_IN_RANGE

```
#define PH_ASSERT_IN_RANGE(
              value,
              begin,
              end)  PH_ASSERT_MSG(begin <= value && value < end, PH_INTERNAL_RANGE_MSG(value,
begin, end, "[", ")"))
```

Assert that `value` is within [`begin`, `end`).

10.5.1.6 PH_ASSERT_IN_RANGE_EXCLUSIVE

```
#define PH_ASSERT_IN_RANGE_EXCLUSIVE(
              value,
              lowerBound,
              upperBound)  PH_ASSERT_MSG(lowerBound < value && value < upperBound, PH_INTERNAL_RANGE_MSG(valu
lowerBound, upperBound, "(", ")"))
```

Similar to PH_ASSERT_IN_RANGE(3), except the bounds are exclusive.

10.5.1.7 PH_ASSERT_IN_RANGE_INCLUSIVE

```
#define PH_ASSERT_IN_RANGE_INCLUSIVE(
              value,
              lowerBound,
              upperBound)  PH_ASSERT_MSG(lowerBound <= value && value <= upperBound, PH_INTERNAL_RANGE_MSG(va
lowerBound, upperBound, "[", "]"))
```

Similar to PH_ASSERT_IN_RANGE(3), except the bounds are inclusive.

10.5.1.8 PH_ASSERT_LE

```
#define PH_ASSERT_LE(
              a,
              b)  PH_ASSERT_MSG(a <= b, std::string(#a) + " = " + std::to_string(a) + ", " +
#b + " = " + std::to_string(b))
```

10.5.1.9 PH_ASSERT_LT

```
#define PH_ASSERT_LT(
              a,
              b)  PH_ASSERT_MSG(a < b, std::string(#a) + " = " + std::to_string(a) + ", " + #b
+ " = " + std::to_string(b))
```

10.5.1.10 PH_ASSERT_MSG

```
#define PH_ASSERT_MSG(
              condition,
              message) PH_NO_OP()
```

10.5.1.11 PH_ASSERT_NE

```
#define PH_ASSERT_NE(
              a,
              b)  PH_ASSERT_MSG(a != b, std::string(#a) + " = " + std::to_string(a) + ", " +
#b + " = " + std::to_string(b))
```

10.5.1.12 PH_ASSERT_UNREACHABLE_SECTION

```
#define PH_ASSERT_UNREACHABLE_SECTION()  PH_ASSERT_MSG(false, "executing supposedly unreachable
code")
```

10.5.1.13 PH_INTERNAL_RANGE_MSG

```
#define PH_INTERNAL_RANGE_MSG(
              value,
              lowerBound,
              upperBound,
              lowerBoundSymbol,
              upperBoundSymbol) PH_NO_OP()
```

10.5.1.14 PH_STATIC_ASSERT_DEPENDENT_FALSE

```
#define PH_STATIC_ASSERT_DEPENDENT_FALSE(
              DependentType,
              message)  static_assert(::ph::detail::DEPENDENT_FALSE<DependentType>, #message)
```

10.6 assertion.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Common/config.h"
00004 #include "Common/macro.h"
00005
00006 #include <string>
00007
00008 namespace ph::detail
00009 {
00010
00011 void output_assertion_message(
00012     const std::string& filename,
00013     const std::string& lineNumber,
00014     const std::string& condition,
00015     const std::string& message);
00016
00017 void on_assertion_failed();
00018
00019 }// end namespace ph::detail
00020
00021 #if PH_DEBUG
00022
00023 #define PH_ASSERT_MSG(condition, failMessage)\
00024     do\
00025     {\
00026         if(!(condition))\
00027         {\
00028             ::ph::detail::output_assertion_message(\
00029                 std::string(__FILE__),\
00030                 std::to_string(__LINE__),\
00031                 std::string(#condition),\
00032                 std::string((failMessage)));\
00033             \
00034             ::ph::detail::on_assertion_failed();\
00035         }\
00036     } while(0)
00037
00038 #define PH_INTERNAL_RANGE_MSG(value, lowerBound, upperBound, lowerBoundSymbol, upperBoundSymbol)\
00039     (std::string(#value) + " = " + std::to_string(value) + ", asserted to be in range = " + \
00040     lowerBoundSymbol + std::to_string(lowerBound) + ", " + std::to_string(upperBound) +
      upperBoundSymbol)
00041
00042 #else
```

```
00043
00044 #define PH_ASSERT_MSG(condition, message) PH_NO_OP()
00045 #define PH_INTERNAL_RANGE_MSG(value, lowerBound, upperBound, lowerBoundSymbol, upperBoundSymbol)
      PH_NO_OP()
00046
00047 #endif
00048
00049 #define PH_ASSERT(condition)\
00050     PH_ASSERT_MSG(condition, "")
00051
00052 #define PH_ASSERT_UNREACHABLE_SECTION()\
00053     PH_ASSERT_MSG(false, "executing supposedly unreachable code")
00054
00055 #define PH_ASSERT_EQ(a, b)\
00056     PH_ASSERT_MSG(a == b, std::string(#a) + " = " + std::to_string(a) + ", " + #b + " = " +
      std::to_string(b))
00057
00058 #define PH_ASSERT_NE(a, b)\
00059     PH_ASSERT_MSG(a != b, std::string(#a) + " = " + std::to_string(a) + ", " + #b + " = " +
      std::to_string(b))
00060
00061 #define PH_ASSERT_GT(a, b)\
00062     PH_ASSERT_MSG(a > b, std::string(#a) + " = " + std::to_string(a) + ", " + #b + " = " +
      std::to_string(b))
00063
00064 #define PH_ASSERT_LT(a, b)\
00065     PH_ASSERT_MSG(a < b, std::string(#a) + " = " + std::to_string(a) + ", " + #b + " = " +
      std::to_string(b))
00066
00067 #define PH_ASSERT_GE(a, b)\
00068     PH_ASSERT_MSG(a >= b, std::string(#a) + " = " + std::to_string(a) + ", " + #b + " = " +
      std::to_string(b))
00069
00070 #define PH_ASSERT_LE(a, b)\
00071     PH_ASSERT_MSG(a <= b, std::string(#a) + " = " + std::to_string(a) + ", " + #b + " = " +
      std::to_string(b))
00072
00075 #define PH_ASSERT_IN_RANGE(value, begin, end)\
00076     PH_ASSERT_MSG(begin <= value && value < end, PH_INTERNAL_RANGE_MSG(value, begin, end, "[", ")"))
00077
00080 #define PH_ASSERT_IN_RANGE_INCLUSIVE(value, lowerBound, upperBound)\
00081     PH_ASSERT_MSG(lowerBound <= value && value <= upperBound, PH_INTERNAL_RANGE_MSG(value, lowerBound,
      upperBound, "[", "]"))
00082
00085 #define PH_ASSERT_IN_RANGE_EXCLUSIVE(value, lowerBound, upperBound)\
00086     PH_ASSERT_MSG(lowerBound < value && value < upperBound, PH_INTERNAL_RANGE_MSG(value, lowerBound,
      upperBound, "(", ")"))
00087
00088
00089 namespace ph::detail
00090 {
00091
00092 template<typename T>
00093 inline constexpr bool DEPENDENT_FALSE = false;
00094
00095 }// end namespace ph::detail
00096
00097 #define PH_STATIC_ASSERT_DEPENDENT_FALSE(DependentType, message)\
00098     static_assert(::ph::detail::DEPENDENT_FALSE<DependentType>, #message)
```

10.7 Include/Common/compiler.h File Reference

Compiler-related information and utilities.

Macros

- #define PH_COMPILER_IS_CLANG 0
- #define PH_COMPILER_IS_GCC 0
- #define PH_COMPILER_IS_MSVC 0

10.7.1 Detailed Description

Compiler-related information and utilities.

10.7.1.1 Definitions for Detecting Compilers

- `PH_COMPILER_IS_MSVC`: 1 if the compiler is the Microsoft Visual C++ compiler (MSVC), 0 otherwise
- `PH_COMPILER_IS_GCC`: 1 if the compiler is the GNU C++ compiler, 0 otherwise
- `PH_COMPILER_IS_CLANG`: 1 if the compiler is Clang C++ compiler, 0 otherwise

10.7.1.2 Additional Attributes

Introduces additional attributes for use with the standard C++ attribute syntax.

- `[[PH_ALWAYS_INLINE]]`: Always attempt to inline the target

10.7.2 Macro Definition Documentation

10.7.2.1 PH_COMPILER_IS_CLANG

```
#define PH_COMPILER_IS_CLANG 0
```

10.7.2.2 PH_COMPILER_IS_GCC

```
#define PH_COMPILER_IS_GCC 0
```

10.7.2.3 PH_COMPILER_IS_MSVC

```
#define PH_COMPILER_IS_MSVC 0
```

10.8 compiler.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00020 // TODO: distinguish between versions
00021
00022 #if defined(__clang__)
00023     #define PH_COMPILER_IS_CLANG 1
00024 #elif defined(__GNUG__)
00025     #define PH_COMPILER_IS_GCC 1
00026 #elif defined(_MSC_VER)
00027     #define PH_COMPILER_IS_MSVC 1
00028 #endif
00029
00030 #ifndef PH_COMPILER_IS_CLANG
00031     #define PH_COMPILER_IS_CLANG 0
00032 #endif
00033
00034 #ifndef PH_COMPILER_IS_GCC
00035     #define PH_COMPILER_IS_GCC 0
00036 #endif
00037
00038 #ifndef PH_COMPILER_IS_MSVC
00039     #define PH_COMPILER_IS_MSVC 0
00040 #endif
00041
00042 #if PH_COMPILER_IS_MSVC
00043     #define PH_ALWAYS_INLINE msvc::forceinline
00044 #elif PH_COMPILER_IS_GCC
00045     #define PH_ALWAYS_INLINE gnu::always_inline
00046 #elif PH_COMPILER_IS_CLANG
00047     #define PH_ALWAYS_INLINE clang::always_inline
00048 #else
00049     #error "Unrecognized compiler: cannot define `PH_ALWAYS_INLINE`"
00050 #endif
```

10.9 Include/Common/config.h File Reference

```
#include <cstddef>
#include <string>
```

Classes

- class ph::Config

Namespaces

- namespace ph

  The root for all renderer implementations.

Macros

- #define PH_ENGINE_VERSION "2.0.0"

  Version of Photon.
- #define PH_PSDL_VERSION "1.1.0"

  Version of Photon Scene Description Language.
- #define PH_DEBUG 0

  Enable debug functionalities. Assertions will be enabled on debug mode.
- #define PH_PROFILING 0
- #define PH_ABORT_ON_ASSERTION_FAILED 1

  Abort the engine on assertion fail.
- #define PH_PRINT_STACK_TRACE_ON_ASSERTION_FAILED 1

  Perform and print a stack trace when assertion failed.
- #define PH_STRICT_FLOATING_POINT_SIZES 1

  Assuring floating point types has specified sizes.
- #define PH_USE_DOUBLE_REAL 0

  Use double precision real numbers.
- #define PH_ENABLE_DEBUG_LOG PH_DEBUG

  Enable debug log level.
- #define PH_ENSURE_LOCKFREE_ALGORITHMS_ARE_LOCKLESS 1
- #define PH_MEMORY_ARENA_DEFAULT_BLOCK_SIZE_IN_BYTES (static_cast<std::size_t>(512) * 1024)

  Default block size for memory arena. Default value is 512 KiB.
- #define PH_TFUNCTION_DEFAULT_MIN_SIZE_IN_BYTES (static_cast<std::size_t>(64))
- #define PH_ENABLE_HIT_EVENT_STATS 0

  Enable statistics recording for hit events.
- #define PH_STRICT_OBJECT_LIFETIME 0

  Being strict about object lifetime. Some compiler versions and different standards may be using an object lifetime model that can be optimized more aggressively (e.g., constant folding on const instances), which may cause UB or malfunctions on some low-level code. It is then advisible to turn on this option if such behavior is observed on the host platform. Note that turning this option off does not mean the source code will be non-conforming to the standard, it means the opposite–follow the latest standard strictly. Turning this option on is a fallback when things do not go as planned.
- #define PH_LOG_FILE_DIRECTRY "./Logs/"

  Directory that stores engine log file.

- #define PH_CONFIG_DIRECTORY "./Config/"
- #define PH_SCRIPT_DIRECTORY "./Script/"
- #define PH_INTERNAL_RESOURCE_DIRECTORY "./InternalResource/"

  Resources that are integrated as part of the renderer.
- #define PH_RENDERER_RESOURCE_DIRECTORY "./Photon-v2-Resource/Resource/"

  Resources that are optional for the renderer.
- #define PH_RENDER_MODE_LINEAR_SRGB 0
- #define PH_RENDER_MODE_ACES 1
- #define PH_RENDER_MODE_SPECTRAL 2
- #define PH_RENDER_MODE_FULL_SPECTRAL 3
- #define PH_RENDER_MODE PH_RENDER_MODE_LINEAR_SRGB
- #define PH_STRICT_ASYMMETRIC_IMPORTANCE_TRANSPORT 0

  Being strict about the symmetricity of importance transport. There are many sources of asymmetry between light and importance transport, but not all of them can be handled gracefully. One example is when shading normals are used for light transport, it is equivalent to using an asymmetric, modified BSDF. However, the correction factor to restore consistency between light and importance transport can have high variance and makes the result unusable for some scenes. This option (when disabled) attempt to fix that by introducing a small bias.
- #define PH_SPECTRUM_SAMPLED_MIN_WAVELENGTH_NM 350
- #define PH_SPECTRUM_SAMPLED_MAX_WAVELENGTH_NM 850
- #define PH_SPECTRUM_SAMPLED_NUM_SAMPLES 100
- #define PH_HIT_PROBE_DEPTH 8
- #define PH_SDL_MAX_FIELDS 64
- #define PH_SDL_MAX_FUNCTIONS 64
- #define PH_HIT_PROBE_CACHE_BYTES 32

  Number of available bytes for a probe's cache. Note that a byte is not necessarily 8-bit.
- #define PH_NUMERIC_IMAGE_MAX_ELEMENTS 4

10.9.1 Macro Definition Documentation

10.9.1.1 PH_ABORT_ON_ASSERTION_FAILED

```
#define PH_ABORT_ON_ASSERTION_FAILED 1
```

Abort the engine on assertion fail.

10.9.1.2 PH_CONFIG_DIRECTORY

```
#define PH_CONFIG_DIRECTORY "./Config/"
```

10.9.1.3 PH_DEBUG

```
#define PH_DEBUG 0
```

Enable debug functionalities. Assertions will be enabled on debug mode.

10.9.1.4 PH_ENABLE_DEBUG_LOG

```
#define PH_ENABLE_DEBUG_LOG PH_DEBUG
```

Enable debug log level.

10.9.1.5 PH_ENABLE_HIT_EVENT_STATS

```
#define PH_ENABLE_HIT_EVENT_STATS 0
```

Enable statistics recording for hit events.

10.9.1.6 PH_ENGINE_VERSION

```
#define PH_ENGINE_VERSION "2.0.0"
```

Version of Photon.

10.9.1.7 PH_ENSURE_LOCKFREE_ALGORITHMS_ARE_LOCKLESS

```
#define PH_ENSURE_LOCKFREE_ALGORITHMS_ARE_LOCKLESS 1
```

10.9.1.8 PH_HIT_PROBE_CACHE_BYTES

```
#define PH_HIT_PROBE_CACHE_BYTES 32
```

Number of available bytes for a probe's cache. Note that a byte is not necessarily 8-bit.

10.9.1.9 PH_HIT_PROBE_DEPTH

```
#define PH_HIT_PROBE_DEPTH 8
```

10.9.1.10 PH_INTERNAL_RESOURCE_DIRECTORY

```
#define PH_INTERNAL_RESOURCE_DIRECTORY "./InternalResource/"
```

Resources that are integrated as part of the renderer.

10.9.1.11 PH_LOG_FILE_DIRECTRY

```
#define PH_LOG_FILE_DIRECTRY "./Logs/"
```

Directory that stores engine log file.

10.9.1.12 PH_MEMORY_ARENA_DEFAULT_BLOCK_SIZE_IN_BYTES

```
#define PH_MEMORY_ARENA_DEFAULT_BLOCK_SIZE_IN_BYTES (static_cast<std::size_t>(512) * 1024)
```

Default block size for memory arena. Default value is 512 KiB.

10.9.1.13 PH_NUMERIC_IMAGE_MAX_ELEMENTS

```
#define PH_NUMERIC_IMAGE_MAX_ELEMENTS 4
```

10.9.1.14 PH_PRINT_STACK_TRACE_ON_ASSERTION_FAILED

```
#define PH_PRINT_STACK_TRACE_ON_ASSERTION_FAILED 1
```

Perform and print a stack trace when assertion failed.

10.9.1.15 PH_PROFILING

```
#define PH_PROFILING 0
```

10.9.1.16 PH_PSDL_VERSION

```
#define PH_PSDL_VERSION "1.1.0"
```

Version of Photon Scene Description Language.

10.9.1.17 PH_RENDER_MODE

```
#define PH_RENDER_MODE PH_RENDER_MODE_LINEAR_SRGB
```

10.9.1.18 PH_RENDER_MODE_ACES

```
#define PH_RENDER_MODE_ACES 1
```

10.9.1.19 PH_RENDER_MODE_FULL_SPECTRAL

```
#define PH_RENDER_MODE_FULL_SPECTRAL 3
```

10.9.1.20 PH_RENDER_MODE_LINEAR_SRGB

```
#define PH_RENDER_MODE_LINEAR_SRGB 0
```

10.9.1.21 PH_RENDER_MODE_SPECTRAL

```
#define PH_RENDER_MODE_SPECTRAL 2
```

10.9.1.22 PH_RENDERER_RESOURCE_DIRECTORY

```
#define PH_RENDERER_RESOURCE_DIRECTORY "./Photon-v2-Resource/Resource/"
```

Resources that are optional for the renderer.

10.9.1.23 PH_SCRIPT_DIRECTORY

```
#define PH_SCRIPT_DIRECTORY "./Script/"
```

10.9.1.24 PH_SDL_MAX_FIELDS

```
#define PH_SDL_MAX_FIELDS 64
```

10.9.1.25 PH_SDL_MAX_FUNCTIONS

```
#define PH_SDL_MAX_FUNCTIONS 64
```

10.9.1.26 PH_SPECTRUM_SAMPLED_MAX_WAVELENGTH_NM

```
#define PH_SPECTRUM_SAMPLED_MAX_WAVELENGTH_NM 850
```

10.9.1.27 PH_SPECTRUM_SAMPLED_MIN_WAVELENGTH_NM

```
#define PH_SPECTRUM_SAMPLED_MIN_WAVELENGTH_NM 350
```

10.9.1.28 PH_SPECTRUM_SAMPLED_NUM_SAMPLES

```
#define PH_SPECTRUM_SAMPLED_NUM_SAMPLES 100
```

10.9.1.29 PH_STRICT_ASYMMETRIC_IMPORTANCE_TRANSPORT

```
#define PH_STRICT_ASYMMETRIC_IMPORTANCE_TRANSPORT 0
```

Being strict about the symmetricity of importance transport. There are many sources of asymmetry between light and importance transport, but not all of them can be handled gracefully. One example is when shading normals are used for light transport, it is equivalent to using an asymmetric, modified BSDF. However, the correction factor to restore consistency between light and importance transport can have high variance and makes the result unusable for some scenes. This option (when disabled) attempt to fix that by introducing a small bias.

See also: "SPPM implementation is not symmetric" https://github.com/mmp/pbrt-v3/issues/209

10.9.1.30 PH_STRICT_FLOATING_POINT_SIZES

```
#define PH_STRICT_FLOATING_POINT_SIZES 1
```

Assuring floating point types has specified sizes.

10.9.1.31 PH_STRICT_OBJECT_LIFETIME

```
#define PH_STRICT_OBJECT_LIFETIME 0
```

Being strict about object lifetime. Some compiler versions and different standards may be using an object lifetime model that can be optimized more aggressively (e.g., constant folding on const instances), which may cause UB or malfunctions on some low-level code. It is then advisible to turn on this option if such behavior is observed on the host platform. Note that turning this option off does not mean the source code will be non-conforming to the standard, it means the opposite–follow the latest standard strictly. Turning this option on is a fallback when things do not go as planned.

10.9.1.32 PH_TFUNCTION_DEFAULT_MIN_SIZE_IN_BYTES

```
#define PH_TFUNCTION_DEFAULT_MIN_SIZE_IN_BYTES (static_cast<std::size_t>(64))
```

10.9.1.33 PH_USE_DOUBLE_REAL

```
#define PH_USE_DOUBLE_REAL 0
```

Use double precision real numbers.

10.10 config.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <cstddef>
00004 #include <string>
00005
00007 // Core Settings                                                    //
00009
00012 #define PH_ENGINE_VERSION "2.0.0"
00013
00016 #define PH_PSDL_VERSION "1.1.0"
00017
00021 #ifdef PH_CONFIG_ENABLE_DEBUG
00022     #define PH_DEBUG 1
00023 #else
00024     #define PH_DEBUG 0
00025 #endif
00026
00027 #ifdef PH_CONFIG_ENABLE_PROFILING
00028 #define PH_PROFILING 1
00029 #else
00030 #define PH_PROFILING 0
00031 #endif
00032
00035 #define PH_ABORT_ON_ASSERTION_FAILED 1
00036
00039 #define PH_PRINT_STACK_TRACE_ON_ASSERTION_FAILED 1
00040
00043 #define PH_STRICT_FLOATING_POINT_SIZES 1
00044
00045 // Log as soon as possible (primarily for debugging).
```

```
00046 //#define PH_UNBUFFERED_LOG
00047
00050 #ifdef PH_CONFIG_DOUBLE_PRECISION_REAL
00051 #define PH_USE_DOUBLE_REAL 1
00052 #else
00053 #define PH_USE_DOUBLE_REAL 0
00054 #endif
00055
00058 #define PH_ENABLE_DEBUG_LOG PH_DEBUG
00059
00060 #define PH_ENSURE_LOCKFREE_ALGORITHMS_ARE_LOCKLESS 1
00061
00065 #define PH_MEMORY_ARENA_DEFAULT_BLOCK_SIZE_IN_BYTES (static_cast<std::size_t>(512) * 1024)
00066
00067 #define PH_TFUNCTION_DEFAULT_MIN_SIZE_IN_BYTES (static_cast<std::size_t>(64))
00068
00071 #define PH_ENABLE_HIT_EVENT_STATS 0
00072
00081 #define PH_STRICT_OBJECT_LIFETIME 0
00082
00085 #define PH_LOG_FILE_DIRECTRY "./Logs/"
00086
00087 #define PH_CONFIG_DIRECTORY "./Config/"
00088 #define PH_SCRIPT_DIRECTORY "./Script/"
00089
00092 #define PH_INTERNAL_RESOURCE_DIRECTORY "./InternalResource/"
00093
00096 #define PH_RENDERER_RESOURCE_DIRECTORY "./Photon-v2-Resource/Resource/"
00097
00099 // Render Modes                                                    //
00101
00102 #define PH_RENDER_MODE_LINEAR_SRGB 0
00103 #define PH_RENDER_MODE_ACES 1
00104 #define PH_RENDER_MODE_SPECTRAL 2
00105 #define PH_RENDER_MODE_FULL_SPECTRAL 3
00106
00107 #define PH_RENDER_MODE PH_RENDER_MODE_LINEAR_SRGB
00108
00118 #define PH_STRICT_ASYMMETRIC_IMPORTANCE_TRANSPORT 0
00119
00121 // Data Structures                                                 //
00123
00124 #define PH_SPECTRUM_SAMPLED_MIN_WAVELENGTH_NM 350
00125 #define PH_SPECTRUM_SAMPLED_MAX_WAVELENGTH_NM 850
00126 #define PH_SPECTRUM_SAMPLED_NUM_SAMPLES       100
00127 #define PH_HIT_PROBE_DEPTH                    8
00128
00129 #define PH_SDL_MAX_FIELDS    64
00130 #define PH_SDL_MAX_FUNCTIONS 64
00131
00135 #define PH_HIT_PROBE_CACHE_BYTES 32
00136
00137 #define PH_NUMERIC_IMAGE_MAX_ELEMENTS 4
00138
00139 namespace ph
00140 {
00141
00142 class Config final
00143 {
00144 public:
00145     static std::string& RENDERER_RESOURCE_DIRECTORY();
00146 };
00147
00148 }// end namespace ph
```

10.11 Include/Common/Config/IniFile.h File Reference

```
#include "Common/assertion.h"
#include <string>
#include <string_view>
#include <vector>
#include <cstddef>
#include <optional>
#include <utility>
```

Classes

- class ph::IniFile

  INI file I/O. This class is useful for recording various settings across the entire engine project. As a low-level I/O class in `Common` library, it can be used regardless the engine initialization state (see `init_render_engine()` and `exit_render_engine()` in `Engine` library for more details).

Namespaces

- namespace ph

  The root for all renderer implementations.

10.12 IniFile.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Common/assertion.h"
00004
00005 #include <string>
00006 #include <string_view>
00007 #include <vector>
00008 #include <cstddef>
00009 #include <optional>
00010 #include <utility>
00011
00012 namespace ph
00013 {
00014
00020 class IniFile final
00021 {
00022 public:
00023     static IniFile read(const std::string& iniFilePath);
00024
00025 public:
00028     IniFile();
00029
00030     explicit IniFile(const std::string& iniFilePath);
00031
00032     void save(const std::string& iniFilePath);
00033     void clear();
00034
00035     std::size_t numSections() const;
00036     std::string_view getSectionName(std::size_t sectionIdx) const;
00037     std::string_view getCurrentSectionName() const;
00038     std::optional<std::size_t> findSectionIndex(std::string_view sectionName) const;
00039     void setCurrentSection(std::size_t sectionIdx);
00040     void setCurrentSection(std::string_view sectionName, bool createIfNotExist = true);
00041
00042     std::size_t numProperties() const;
00043     std::string_view getPropertyName(std::size_t propertyIdx) const;
00044     std::string_view getPropertyValue(std::size_t propertyIdx) const;
00045     std::optional<std::size_t> findPropertyIndex(std::string_view propertyName) const;
00046
00049     void setProperty(std::size_t propertyIdx, std::string_view propertyValue);
00050
00053     void setProperty(
00054         std::string_view propertyName,
00055         std::string_view propertyValue,
00056         bool createIfNotExist = true);
00057
00063     void append(const IniFile& other);
00064
00065 private:
00066     struct IniSection final
00067     {
00068         std::string name;
00069         std::vector<std::pair<std::string, std::string>> keyValPairs;
00070     };
00071
00072     IniSection& getIniSection(std::size_t sectionIdx);
00073     const IniSection& getIniSection(std::size_t sectionIdx) const;
00074
00075     std::vector<IniSection> m_sections;
```

```
00076     std::size_t             m_currentSectionIdx;
00077 };
00078
00079 // In-header Implementations:
00080
00081 inline std::size_t IniFile::numSections() const
00082 {
00083     return m_sections.size();
00084 }
00085
00086 inline std::size_t IniFile::numProperties() const
00087 {
00088     return getIniSection(m_currentSectionIdx).keyValPairs.size();
00089 }
00090
00091 inline std::string_view IniFile::getSectionName(const std::size_t sectionIdx) const
00092 {
00093     return getIniSection(sectionIdx).name;
00094 }
00095
00096 inline std::string_view IniFile::getCurrentSectionName() const
00097 {
00098     return getIniSection(m_currentSectionIdx).name;
00099 }
00100
00101 inline void IniFile::setCurrentSection(const std::size_t sectionIdx)
00102 {
00103     PH_ASSERT_LT(sectionIdx, m_sections.size());
00104
00105     m_currentSectionIdx = sectionIdx;
00106 }
00107
00108 inline std::string_view IniFile::getPropertyName(const std::size_t propertyIdx) const
00109 {
00110     const IniSection& section = getIniSection(m_currentSectionIdx);
00111
00112     PH_ASSERT_LT(propertyIdx, section.keyValPairs.size());
00113     return section.keyValPairs[propertyIdx].first;
00114 }
00115
00116 inline std::string_view IniFile::getPropertyValue(const std::size_t propertyIdx) const
00117 {
00118     const IniSection& section = getIniSection(m_currentSectionIdx);
00119
00120     PH_ASSERT_LT(propertyIdx, section.keyValPairs.size());
00121     return section.keyValPairs[propertyIdx].second;
00122 }
00123
00124 inline void IniFile::setProperty(const std::size_t propertyIdx, const std::string_view propertyValue)
00125 {
00126     IniSection& section = getIniSection(m_currentSectionIdx);
00127
00128     PH_ASSERT_LT(propertyIdx, section.keyValPairs.size());
00129     section.keyValPairs[propertyIdx].second = propertyValue;
00130 }
00131
00132 inline IniFile::IniSection& IniFile::getIniSection(const std::size_t sectionIdx)
00133 {
00134     PH_ASSERT_LT(sectionIdx, m_sections.size());
00135
00136     return m_sections[sectionIdx];
00137 }
00138
00139 inline const IniFile::IniSection& IniFile::getIniSection(const std::size_t sectionIdx) const
00140 {
00141     PH_ASSERT_LT(sectionIdx, m_sections.size());
00142
00143     return m_sections[sectionIdx];
00144 }
00145
00146 inline void IniFile::clear()
00147 {
00148     m_sections.clear();
00149 }
00150
00151 }// end namespace ph
```

10.13 Include/Common/Container/detail.h File Reference

```
#include <cstddef>
#include <string>
```

```
#include <string_view>
#include <functional>
```

Classes

- struct ph::detail::HeterogeneousStringHash

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::detail

  Implementation detail mainly for internal usages.

10.14 detail.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <cstddef>
00004 #include <string>
00005 #include <string_view>
00006 #include <functional>
00007
00008 namespace ph::detail
00009 {
00010
00011 // References:
00012 // [1] https://www.cppstories.com/2021/heterogeneous-access-cpp20/
00013 // [2] https://en.cppreference.com/w/cpp/container/unordered_map/find
00014 struct HeterogeneousStringHash
00015 {
00016     using is_transparent = void;
00017
00018     [[nodiscard]]
00019     std::size_t operator () (const char* txt) const
00020     {
00021         return std::hash<std::string_view>{}(txt);
00022     }
00023
00024     [[nodiscard]]
00025     std::size_t operator () (std::string_view txt) const
00026     {
00027         return std::hash<std::string_view>{}(txt);
00028     }
00029
00030     [[nodiscard]]
00031     std::size_t operator () (const std::string& txt) const
00032     {
00033         return std::hash<std::string>{}(txt);
00034     }
00035 };
00036
00037 }// end namespace ph::detail
```

10.15 Include/Common/Container/StdUnorderedStringSet.h File Reference

```
#include "Common/Container/detail.h"
#include <unordered_set>
```

Namespaces

- namespace ph

  The root for all renderer implementations.

Typedefs

- using ph::StdUnorderedStringSet

  Unordered `std::string` set with support for heterogeneous string key lookup. Supports `std::string_view` and literal (C-style) string lookup in addition to the original `std::string` lookup. The heterogeneous access can save redundant dynamic allocations when querying the map.

10.16 StdUnorderedStringSet.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Common/Container/detail.h"
00004
00005 #include <unordered_set>
00006
00007 namespace ph
00008 {
00009
00015 using StdUnorderedStringSet = std::unordered_set<
00016     std::string, detail::HeterogeneousStringHash, std::equal_to<»;
00017
00018 }// end namespace ph
```

10.17 Include/Common/Container/TStdUnorderedStringMap.h File Reference

```
#include "Common/Container/detail.h"
#include <unordered_map>
```

Namespaces

- namespace ph

  The root for all renderer implementations.

Typedefs

- template<typename Value >
  using ph::TStdUnorderedStringMap

  Unordered `std::string` map with support for heterogeneous string key lookup. Supports `std::string_view` and literal (C-style) string lookup in addition to the original `std::string` lookup. The heterogeneous access can save redundant dynamic allocations when querying the map.

10.18 TStdUnorderedStringMap.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Common/Container/detail.h"
00004
00005 #include <unordered_map>
00006
00007 namespace ph
00008 {
00009
00015 template<typename Value>
00016 using TStdUnorderedStringMap = std::unordered_map<
00017     std::string, Value, detail::HeterogeneousStringHash, std::equal_to<»;
00018
00019 }// end namespace ph
```

10.19 Include/Common/debug.h File Reference

```
#include <string>
```

Namespaces

- namespace ph

  The root for all renderer implementations.

Macros

- #define PH_DEBUG_BREAK() ::ph::debug_break()

Functions

- void ph::debug_break ()
- std::string ph::obtain_stack_trace ()

10.19.1 Macro Definition Documentation

10.19.1.1 PH_DEBUG_BREAK

```
#define PH_DEBUG_BREAK() ::ph::debug_break()
```

10.20 debug.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <string>
00004
00005 namespace ph
00006 {
00007
00008 void debug_break();
00009 std::string obtain_stack_trace();
00010
00011 }// end namespace ph
00012
00013 #define PH_DEBUG_BREAK() ::ph::debug_break()
```

10.21 Include/Common/exceptions.h File Reference

```
#include <stdexcept>
#include <string>
#include <string_view>
#include <format>
#include <type_traits>
#include <utility>
```

Classes

- class ph::RuntimeException

  General exception thrown on runtime error.
- class ph::LogicalException

  General exception thrown on logical error.
- class ph::NumericException
- class ph::OverflowException
- class ph::UninitializedObjectException
- class ph::IllegalOperationException
- class ph::InvalidArgumentException
- class ph::OutOfRangeException

Namespaces

- namespace ph

  The root for all renderer implementations.

Concepts

- concept ph::CPhotonException

Typedefs

- using ph::Exception = std::exception

Functions

- template<CPhotonException ExceptionType, typename... Args>
  void ph::throw_formatted (const std::format_string< Args... > msgFormat, Args &&... args)

10.22 exceptions.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <stdexcept>
00004 #include <string>
00005 #include <string_view>
00006 #include <format>
00007 #include <type_traits>
00008 #include <utility>
00009
00010 namespace ph
00011 {
00012
00013 // Note: When adding new base types, the implementation of `CPhotonException` needs to be updated.
00014
00015 // A convenient "catch all" (including std exceptions) type when handling exceptions
00016 using Exception = std::exception;
00017
00020 class RuntimeException : public std::runtime_error
00021 {
00022 public:
00023     explicit RuntimeException(const std::string& message);
00024     explicit RuntimeException(const char* message);
00025     inline ~RuntimeException() override = default;
00026
00027     virtual std::string whatStr() const;
00028 };
00029
00032 class LogicalException : public std::logic_error
00033 {
00034 public:
00035     explicit LogicalException(const std::string& message);
00036     explicit LogicalException(const char* message);
00037     inline ~LogicalException() override = default;
00038
00039     virtual std::string whatStr() const;
00040 };
00041
00042 class NumericException : public RuntimeException
00043 {
00044 public:
00045     using RuntimeException::RuntimeException;
00046 };
00047
00048 class OverflowException : public NumericException
00049 {
00050 public:
00051     using NumericException::NumericException;
00052 };
00053
00054 class UninitializedObjectException : public LogicalException
00055 {
00056 public:
00057     using LogicalException::LogicalException;
00058 };
00059
00060 class IllegalOperationException : public LogicalException
00061 {
00062 public:
00063     using LogicalException::LogicalException;
00064 };
00065
00066 class InvalidArgumentException : public LogicalException
00067 {
00068 public:
00069     using LogicalException::LogicalException;
00070 };
00071
00072 class OutOfRangeException : public LogicalException
00073 {
00074 public:
00075     using LogicalException::LogicalException;
00076 };
00077
00078 template<typename T>
00079 concept CPhotonException =
00080     std::is_base_of_v<RuntimeException, T> ||
00081     std::is_base_of_v<LogicalException, T>;
00082
00083 template<CPhotonException ExceptionType, typename... Args>
00084 [[noreturn]]
00085 inline void throw_formatted(const std::format_string<Args...> msgFormat, Args&&... args)
00086 {
```

```
00087     // Intentionally not forwarding to `std::make_format_args` due to P2905R2
00088     throw ExceptionType(
00089         std::vformat(msgFormat.get(), std::make_format_args(args...)));
00090 }
00091
00092 }// end namespace ph
```

10.23 Include/Common/io_exceptions.h File Reference

```
#include "Common/exceptions.h"
#include <string>
#include <utility>
#include <format>
#include <system_error>
```

Classes

- class ph::IOException
- class ph::FileIOError
- class ph::FilesystemError

Namespaces

- namespace ph

  The root for all renderer implementations.

10.24 io_exceptions.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Common/exceptions.h"
00004
00005 #include <string>
00006 #include <utility>
00007 #include <format>
00008 #include <system_error>
00009
00010 namespace ph
00011 {
00012
00013 class IOException : public RuntimeException
00014 {
00015 public:
00016     explicit IOException(const std::string& message);
00017     explicit IOException(const char* message);
00018 };
00019
00020 class FileIOError : public IOException
00021 {
00022 public:
00023     explicit FileIOError(const std::string& message);
00024     explicit FileIOError(const char* message);
00025     FileIOError(const std::string& message, std::string filename);
00026
00027     std::string whatStr() const override;
00028
00029 private:
00030     std::string m_filename;
00031 };
00032
00033 class FilesystemError : public IOException
```

```
00034 {
00035 public:
00036     using IOException::IOException;
00037
00038     explicit FilesystemError(std::error_code errorCode);
00039     FilesystemError(const std::string& message, std::error_code errorCode);
00040
00041     std::string whatStr() const override;
00042
00043 private:
00044     std::error_code m_errorCode;
00045 };
00046
00047 // In-header Implementations:
00048
00049 inline IOException::IOException(const std::string& message) :
00050     RuntimeException(message)
00051 {}
00052
00053 inline IOException::IOException(const char* const message) :
00054     RuntimeException(message)
00055 {}
00056
00057 inline FileIOError::FileIOError(const std::string& message) :
00058     IOException(message),
00059     m_filename()
00060 {}
00061
00062 inline FileIOError::FileIOError(const char* const message) :
00063     IOException(message),
00064     m_filename()
00065 {}
00066
00067 inline FileIOError::FileIOError(const std::string& message, std::string filename) :
00068     IOException(message),
00069     m_filename(std::move(filename))
00070 {}
00071
00072 inline std::string FileIOError::whatStr() const
00073 {
00074     std::string filenameInfo = m_filename.empty() ? "(unavailable)" : m_filename;
00075
00076     return std::format("{} | filename <{}>", IOException::whatStr(), filenameInfo);
00077 }
00078
00079 inline FilesystemError::FilesystemError(std::error_code errorCode)
00080     : FilesystemError("", errorCode)
00081 {}
00082
00083 inline FilesystemError::FilesystemError(const std::string& message, std::error_code errorCode)
00084     : IOException(message)
00085     , m_errorCode(errorCode)
00086 {}
00087
00088 inline std::string FilesystemError::whatStr() const
00089 {
00090     auto baseMsg = IOException::whatStr();
00091     auto errorCodeMsg = m_errorCode.message();
00092     if(errorCodeMsg.empty())
00093     {
00094         return baseMsg;
00095     }
00096     else
00097     {
00098         return std::format("{} ({})", baseMsg, errorCodeMsg);
00099     }
00100 }
00101
00102 }// end namespace ph
```

10.25 Include/Common/Log/ELogLevel.h File Reference

```
#include "Common/Log/logger_fwd.h"
```

Namespaces

- namespace ph

  The root for all renderer implementations.

Enumerations

- enum class ph::ELogLevel {
  ph::Debug , ph::Note , ph::Warning , ph::Error ,
  ph::DebugOnce , ph::NoteOnce , ph::WarningOnce , ph::ErrorOnce }

Functions

- constexpr bool ph::is_once (const ELogLevel logLevel)

10.26 ELogLevel.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Common/Log/logger_fwd.h"
00004
00005 namespace ph
00006 {
00007
00008 enum class ELogLevel
00009 {
00010     Debug,
00011     Note,
00012     Warning,
00013     Error,
00014     DebugOnce,
00015     NoteOnce,
00016     WarningOnce,
00017     ErrorOnce
00018 };
00019
00020 inline constexpr bool is_once(const ELogLevel logLevel)
00021 {
00022     switch(logLevel)
00023     {
00024     case ELogLevel::DebugOnce:
00025     case ELogLevel::NoteOnce:
00026     case ELogLevel::WarningOnce:
00027     case ELogLevel::ErrorOnce:
00028         return true;
00029
00030     default:
00031         return false;
00032     }
00033 }
00034
00035 }// end namespace ph
```

10.27 Include/Common/Log/Logger.h File Reference

```
#include "Common/Log/logger_fwd.h"
#include <string>
#include <string_view>
#include <vector>
#include <functional>
#include <memory>
```

Classes

- class ph::Logger

Namespaces

- namespace ph

  The root for all renderer implementations.

10.28 Logger.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Common/Log/logger_fwd.h"
00004
00005 #include <string>
00006 #include <string_view>
00007 #include <vector>
00008 #include <functional>
00009 #include <memory>
00010
00011 namespace ph
00012 {
00013
00014 class Logger final
00015 {
00016 public:
00017     Logger();
00018
00019     void log(std::string_view message) const;
00020
00021     void log(ELogLevel logLevel, std::string_view message) const;
00022
00023     void log(
00024         std::string_view name,
00025         ELogLevel        logLevel,
00026         std::string_view message) const;
00027
00031     void addLogHandler(LogHandler logHandler);
00032
00033 public:
00034     static LogHandler makeStdOutLogPrinter();
00035     static LogHandler makeColoredStdOutLogPrinter();
00036
00037 private:
00038     std::vector<LogHandler> m_logHandlers;
00039
00040     static std::string makeLogString(
00041         std::string_view name,
00042         ELogLevel        logLevel,
00043         std::string_view message);
00044
00045     static bool shouldStdOutPrintWithoutBuffering(ELogLevel logLevel);
00046 };
00047
00048 }// end namespace ph
```

10.29 Include/Common/Log/logger_fwd.h File Reference

```
#include <functional>
#include <string_view>
```

Namespaces

- namespace ph

  The root for all renderer implementations.

Typedefs

- using ph::LogHandler = std::function<void(ELogLevel logLevel, std::string_view logString)>

10.30 logger_fwd.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <functional>
00004 #include <string_view>
00005
00006 namespace ph
00007 {
00008
00009 enum class ELogLevel;
00010 class Logger;
00011
00012 using LogHandler = std::function<void(ELogLevel logLevel, std::string_view logString)>;
00013
00014 }// end namespace ph
```

10.31 Include/Common/logging.h File Reference

Logging functions.

```
#include "Common/Log/logger_fwd.h"
#include "Common/Log/ELogLevel.h"
#include "Common/config.h"
#include "Common/macro.h"
#include <string>
#include <vector>
#include <cstddef>
#include <format>
```

Classes

- struct ph::LogGroup
- class ph::LogGroups

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::detail

  Implementation detail mainly for internal usages.
- namespace ph::detail::core_logging

  Core logging functionalities. Most logs will output information (logs) via a main logger, which we refer to as "core logger". This namespace contains implementation details for core logging functionalities.

Macros

- #define PH_DECLARE_LOG_GROUP(groupName) const ::ph::Logger& internal_impl_logger_access_##groupName()

  Declares a logger. The logger should be defined using `PH_DEFINE_LOG_GROUP()` somewhere in the source (preferably in a .cpp file).
- #define PH_DEFINE_LOG_GROUP(groupName, category)

  Defines a logger.
- #define PH_DEFINE_INLINE_LOG_GROUP(groupName, category)
- #define PH_DEFINE_INTERNAL_LOG_GROUP(groupName, category)

  Defines a logger that is private to a .cpp file. Can only appear one time in a translation unit, preferably in a .cpp file.
- #define PH_DEFINE_EXTERNAL_LOG_GROUP(groupName, category) PH_DEFINE_INLINE_LOG_GROUP(groupName, category)

  Defines a public logger in a header file. The logger will be usable anywhere that includes the header file containing this call.
- #define PH_LOG_RAW_STRING_TO_CORE_LOGGER(groupName, level, rawStringExpr)
- #define PH_LOG_FORMAT_STRING_TO_CORE_LOGGER(groupName, level, formatString, ...)
- #define PH_DEBUG_LOG_STRING(groupName, rawString) PH_NO_OP()
- #define PH_DEBUG_LOG_STRING_ONCE(groupName, rawString) PH_NO_OP()
- #define PH_DEBUG_LOG(groupName, formatString, ...) PH_NO_OP()
- #define PH_DEBUG_LOG_ONCE(groupName, formatString, ...) PH_NO_OP()
- #define PH_LOG_STRING(groupName, level, rawString) PH_LOG_RAW_STRING_TO_CORE_LOGGER(groupName, level, rawString)
- #define PH_LOG(groupName, level, formatString, ...) PH_LOG_FORMAT_STRING_TO_CORE_LOGGER(groupName, level, formatString, __VA_ARGS__)

- #define PH_DEFAULT_DEBUG_LOG_STRING(rawString, ...) PH_NO_OP()

  A set of helper utility macros to log using Photon renderer's default log group.
- #define PH_DEFAULT_DEBUG_LOG_STRING_ONCE(rawString, ...) PH_NO_OP()
- #define PH_DEFAULT_DEBUG_LOG(formatString, ...) PH_NO_OP()
- #define PH_DEFAULT_DEBUG_LOG_ONCE(formatString, ...) PH_NO_OP()
- #define PH_DEFAULT_LOG_STRING(level, rawString) PH_LOG_STRING(PhotonRenderer, level, rawString)
- #define PH_DEFAULT_LOG(level, formatString, ...) PH_LOG(PhotonRenderer, level, formatString, __VA_ARGS__)

Functions

- LogGroups ph::get_core_log_groups ()
- void ph::detail::core_logging::init ()

  Initializes core logging functionalities. Any logging is only valid after calling `init()`.
- void ph::detail::core_logging::exit ()

  Terminates core logging functionalities. Cleanup after logging is finished.
- Logger & ph::detail::core_logging::get_logger ()

  Get the core logger.
- std::size_t ph::detail::core_logging::add_log_group (std::string_view groupName, std::string_view category="")

  Add a log group to the core logger.
- void ph::detail::core_logging::log_to_logger (const Logger &logger, std::string_view groupName, ELogLevel logLevel, std::string_view logMessage)

  Log information to the specified logger.
- ph::PH_DECLARE_LOG_GROUP (PhotonRenderer)

10.31.1 Detailed Description

Logging functions.

Note on loggers: All logging functionalities are thread-safe when using pre-defined macros. It is not advisible to log in class dtor, especially for static instances as they may live outside of engine's lifetime. Also make sure not to call any logging functions when the logger is not initialized.

10.31.2 Macro Definition Documentation

10.31.2.1 PH_DEBUG_LOG

```
#define PH_DEBUG_LOG(
              groupName,
              formatString,
              ...) PH_NO_OP()
```

10.31.2.2 PH_DEBUG_LOG_ONCE

```
#define PH_DEBUG_LOG_ONCE(
              groupName,
              formatString,
              ...) PH_NO_OP()
```

10.31.2.3 PH_DEBUG_LOG_STRING

```
#define PH_DEBUG_LOG_STRING(
              groupName,
              rawString) PH_NO_OP()
```

10.31.2.4 PH_DEBUG_LOG_STRING_ONCE

```
#define PH_DEBUG_LOG_STRING_ONCE(
              groupName,
              rawString) PH_NO_OP()
```

10.31.2.5 PH_DECLARE_LOG_GROUP

```
#define PH_DECLARE_LOG_GROUP(
              groupName)  const ::ph::Logger& internal_impl_logger_access_##groupName()
```

Declares a logger. The logger should be defined using PH_DEFINE_LOG_GROUP() somewhere in the source (preferably in a .cpp file).

10.31.2.6 PH_DEFAULT_DEBUG_LOG

```
#define PH_DEFAULT_DEBUG_LOG(
              formatString,
              ...) PH_NO_OP()
```

10.31.2.7 PH_DEFAULT_DEBUG_LOG_ONCE

```
#define PH_DEFAULT_DEBUG_LOG_ONCE(
              formatString,
              ...) PH_NO_OP()
```

10.31.2.8 PH_DEFAULT_DEBUG_LOG_STRING

```
#define PH_DEFAULT_DEBUG_LOG_STRING(
              rawString,
              ...) PH_NO_OP()
```

A set of helper utility macros to log using Photon renderer's default log group.

Note

Default log group macros are for uses in the `ph` namespace. For general usages, use the custom log group macros.

10.31.2.9 PH_DEFAULT_DEBUG_LOG_STRING_ONCE

```
#define PH_DEFAULT_DEBUG_LOG_STRING_ONCE(
              rawString,
              ...) PH_NO_OP()
```

10.31.2.10 PH_DEFAULT_LOG

```
#define PH_DEFAULT_LOG(
              level,
              formatString,
              ...) PH_LOG(PhotonRenderer, level, formatString, __VA_ARGS__)
```

10.31.2.11 PH_DEFAULT_LOG_STRING

```
#define PH_DEFAULT_LOG_STRING(
              level,
              rawString) PH_LOG_STRING(PhotonRenderer, level, rawString)
```

10.31.2.12 PH_DEFINE_EXTERNAL_LOG_GROUP

```
#define PH_DEFINE_EXTERNAL_LOG_GROUP(
                groupName,
                category) PH_DEFINE_INLINE_LOG_GROUP(groupName, category)
```

Defines a public logger in a header file. The logger will be usable anywhere that includes the header file containing this call.

10.31.2.13 PH_DEFINE_INLINE_LOG_GROUP

```
#define PH_DEFINE_INLINE_LOG_GROUP(
                groupName,
                category)
```

Value:

```
    inline const ::ph::Logger& internal_impl_logger_access_##groupName()\
    {\
        static const std::size_t logGroupIndex = ::ph::detail::core_logging::add_log_group(#groupName,
      #category);\
    \
        return ::ph::detail::core_logging::get_logger();\
    }
```

10.31.2.14 PH_DEFINE_INTERNAL_LOG_GROUP

```
#define PH_DEFINE_INTERNAL_LOG_GROUP(
                groupName,
                category)
```

Value:

```
    namespace\
    {\
        PH_DEFINE_INLINE_LOG_GROUP(groupName, category);\
    }
```

Defines a logger that is private to a .cpp file. Can only appear one time in a translation unit, preferably in a .cpp file.

10.31.2.15 PH_DEFINE_LOG_GROUP

```
#define PH_DEFINE_LOG_GROUP(
                groupName,
                category)
```

Value:

```
    const ::ph::Logger& internal_impl_logger_access_##groupName()\
    {\
        static const std::size_t logGroupIndex = ::ph::detail::core_logging::add_log_group(#groupName,
      #category);\
    \
        return ::ph::detail::core_logging::get_logger();\
    }
```

Defines a logger.

10.31.2.16 PH_LOG

```
#define PH_LOG(
               groupName,
               level,
               formatString,
               ...) PH_LOG_FORMAT_STRING_TO_CORE_LOGGER(groupName, level, formatString, __VA_↩
ARGS__)
```

10.31.2.17 PH_LOG_FORMAT_STRING_TO_CORE_LOGGER

```
#define PH_LOG_FORMAT_STRING_TO_CORE_LOGGER(
               groupName,
               level,
               formatString,
               ...)
```

Value:

```
    PH_LOG_RAW_STRING_TO_CORE_LOGGER(\
        groupName,\
        level,\
        std::format(formatString __VA_OPT__(,) __VA_ARGS__))
```

10.31.2.18 PH_LOG_RAW_STRING_TO_CORE_LOGGER

```
#define PH_LOG_RAW_STRING_TO_CORE_LOGGER(
               groupName,
               level,
               rawStringExpr)
```

Value:

```
    do\
    {\
        constexpr auto logLevel = ::ph::ELogLevel::level;\
        const auto rawString = rawStringExpr;\
        if constexpr(::ph::is_once(logLevel))\
        {\
            static const bool PH_CONCAT_2(dummy, __LINE__) = [&]()\
            {\
                ::ph::detail::core_logging::log_to_logger(\
                    internal_impl_logger_access_##groupName(),\
                    #groupName,\
                    logLevel,\
                    rawString);\
                return true;\
            }();\
        }\
        else\
        {\
            ::ph::detail::core_logging::log_to_logger(\
                internal_impl_logger_access_##groupName(),\
                #groupName,\
                logLevel,\
                rawString);\
        }\
    } while(0)
```

10.31.2.19 PH_LOG_STRING

```
#define PH_LOG_STRING(
               groupName,
               level,
               rawString) PH_LOG_RAW_STRING_TO_CORE_LOGGER(groupName, level, rawString)
```

10.32 logging.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00007 #include "Common/Log/logger_fwd.h"
00008 #include "Common/Log/ELogLevel.h"
00009 #include "Common/config.h"
00010 #include "Common/macro.h"
00011
00012 #include <string>
00013 #include <vector>
00014 #include <cstddef>
00015 #include <format>
00016
00024 namespace ph
00025 {
00026
00027 struct LogGroup final
00028 {
00029 public:
00030     std::string groupName;
00031     std::string category;
00032 };
00033
00034 class LogGroups final
00035 {
00036 public:
00037     inline LogGroups() = default;
00038     inline LogGroups(const LogGroups& other) = default;
00039     inline LogGroups& operator = (const LogGroups& rhs) = default;
00040
00041     std::size_t addGroup(std::string_view groupName, std::string_view category = "");
00042     std::size_t numGroups() const;
00043     const LogGroup& getGroup(std::size_t index) const;
00044
00045 private:
00046     std::vector<LogGroup> m_groups;
00047 };
00048
00049 LogGroups get_core_log_groups();
00050
00051 }// end namespace ph
00052
00053 namespace ph::detail::core_logging
00054 {
00055
00059 void init();
00060
00064 void exit();
00065
00069 Logger& get_logger();
00070
00074 std::size_t add_log_group(std::string_view groupName, std::string_view category = "");
00075
00079 void log_to_logger(const Logger& logger, std::string_view groupName, ELogLevel logLevel,
      std::string_view logMessage);
00080
00081 }// end namespace ph::detail::core_logging
00082
00087 #define PH_DECLARE_LOG_GROUP(groupName)\
00088     const ::ph::Logger& internal_impl_logger_access_##groupName()
00089
00092 #define PH_DEFINE_LOG_GROUP(groupName, category)\
00093     const ::ph::Logger& internal_impl_logger_access_##groupName()\
00094     {\
00095         static const std::size_t logGroupIndex = ::ph::detail::core_logging::add_log_group(#groupName,
      #category);\
00096     \
00097         return ::ph::detail::core_logging::get_logger();\
00098     }
00099
00100 #define PH_DEFINE_INLINE_LOG_GROUP(groupName, category)\
00101     inline const ::ph::Logger& internal_impl_logger_access_##groupName()\
00102     {\
00103         static const std::size_t logGroupIndex = ::ph::detail::core_logging::add_log_group(#groupName,
      #category);\
00104     \
00105         return ::ph::detail::core_logging::get_logger();\
00106     }
00107
00111 #define PH_DEFINE_INTERNAL_LOG_GROUP(groupName, category)\
00112     namespace\
00113     {\
00114         PH_DEFINE_INLINE_LOG_GROUP(groupName, category);\
```

```
00115     }
00116
00120 #define PH_DEFINE_EXTERNAL_LOG_GROUP(groupName, category) PH_DEFINE_INLINE_LOG_GROUP(groupName,
      category)
00121
00122 #define PH_LOG_RAW_STRING_TO_CORE_LOGGER(groupName, level, rawStringExpr)\
00123     do\
00124     {\
00125         constexpr auto logLevel = ::ph::ELogLevel::level;\
00126         const auto rawString = rawStringExpr;\
00127         if constexpr(::ph::is_once(logLevel))\
00128         {\
00129             static const bool PH_CONCAT_2(dummy, __LINE__) = [&]()\
00130             {\
00131                 ::ph::detail::core_logging::log_to_logger(\
00132                     internal_impl_logger_access_##groupName(),\
00133                     #groupName,\
00134                     logLevel,\
00135                     rawString);\
00136                 return true;\
00137             }();\
00138         }\
00139         else\
00140         {\
00141             ::ph::detail::core_logging::log_to_logger(\
00142                 internal_impl_logger_access_##groupName(),\
00143                 #groupName,\
00144                 logLevel,\
00145                 rawString);\
00146         }\
00147     } while(0)
00148
00149 // TODO: it could be beneficial to determine when can we use std::vformat()
00150 // instead of always using std::format() for logging
00151 // PH_LOG_STRING() variant for directly logging a runtime string?
00152
00153 #define PH_LOG_FORMAT_STRING_TO_CORE_LOGGER(groupName, level, formatString, ...)\
00154     PH_LOG_RAW_STRING_TO_CORE_LOGGER(\
00155         groupName,\
00156         level,\
00157         std::format(formatString __VA_OPT__(,) __VA_ARGS__))
00158
00159 #if PH_ENABLE_DEBUG_LOG
00160     #define PH_DEBUG_LOG_STRING(groupName, rawString) PH_LOG_RAW_STRING_TO_CORE_LOGGER(groupName,
      Debug, rawString)
00161     #define PH_DEBUG_LOG_STRING_ONCE(groupName, rawString) PH_LOG_RAW_STRING_TO_CORE_LOGGER(groupName,
      DebugOnce, rawString)
00162     #define PH_DEBUG_LOG(groupName, formatString, ...) PH_LOG_FORMAT_STRING_TO_CORE_LOGGER(groupName,
      Debug, formatString, __VA_ARGS__)
00163     #define PH_DEBUG_LOG_ONCE(groupName, formatString, ...)
      PH_LOG_FORMAT_STRING_TO_CORE_LOGGER(groupName, DebugOnce, formatString, __VA_ARGS__)
00164 #else
00165     #define PH_DEBUG_LOG_STRING(groupName, rawString) PH_NO_OP()
00166     #define PH_DEBUG_LOG_STRING_ONCE(groupName, rawString) PH_NO_OP()
00167     #define PH_DEBUG_LOG(groupName, formatString, ...) PH_NO_OP()
00168     #define PH_DEBUG_LOG_ONCE(groupName, formatString, ...) PH_NO_OP()
00169 #endif
00170
00171 #define PH_LOG_STRING(groupName, level, rawString) PH_LOG_RAW_STRING_TO_CORE_LOGGER(groupName, level,
      rawString)
00172 #define PH_LOG(groupName, level, formatString, ...) PH_LOG_FORMAT_STRING_TO_CORE_LOGGER(groupName,
      level, formatString, __VA_ARGS__)
00173
00174 namespace ph
00175 {
00176
00177 // Photon renderer's default log group
00178 PH_DECLARE_LOG_GROUP(PhotonRenderer);
00179
00185 #if PH_ENABLE_DEBUG_LOG
00186     #define PH_DEFAULT_DEBUG_LOG_STRING(rawString) PH_DEBUG_LOG_STRING(PhotonRenderer, rawString)
00187     #define PH_DEFAULT_DEBUG_LOG_STRING_ONCE(rawString) PH_DEBUG_LOG_STRING_ONCE(PhotonRenderer,
      rawString)
00188     #define PH_DEFAULT_DEBUG_LOG(formatString, ...) PH_DEBUG_LOG(PhotonRenderer, formatString,
      __VA_ARGS__)
00189     #define PH_DEFAULT_DEBUG_LOG_ONCE(formatString, ...) PH_DEBUG_LOG_ONCE(PhotonRenderer,
      formatString, __VA_ARGS__)
00190 #else
00191     #define PH_DEFAULT_DEBUG_LOG_STRING(rawString, ...) PH_NO_OP()
00192     #define PH_DEFAULT_DEBUG_LOG_STRING_ONCE(rawString, ...) PH_NO_OP()
00193     #define PH_DEFAULT_DEBUG_LOG(formatString, ...) PH_NO_OP()
00194     #define PH_DEFAULT_DEBUG_LOG_ONCE(formatString, ...) PH_NO_OP()
00195 #endif
00196
00197 #define PH_DEFAULT_LOG_STRING(level, rawString) PH_LOG_STRING(PhotonRenderer, level, rawString)
00198 #define PH_DEFAULT_LOG(level, formatString, ...) PH_LOG(PhotonRenderer, level, formatString,
      __VA_ARGS__)
```

```
00200
00201 }// end namespace ph
```

10.33 Include/Common/macro.h File Reference

Useful macro definitions for general operations.

Macros

- #define PH_CONCAT_2(a, b) a##b
- #define PH_CONCAT_3(a, b, c) a##b##c
- #define PH_CONCAT_4(a, b, c, d) a##b##c##d
- #define PH_CONCAT_5(a, b, c, d, e) a##b##c##d##e
- #define PH_CONCAT_6(a, b, c, d, e, f) a##b##c##d##e##f
- #define PH_CONCAT_7(a, b, c, d, e, f, g) a##b##c##d##e##f##g
- #define PH_CONCAT_8(a, b, c, d, e, f, g, h) a##b##c##d##e##f##g##h
- #define PH_NO_OP() ((void)0)

  Places an expression that does nothing. Useful in situations where the macro expansion is intended to mimic a function scope expression while doing nothing.

10.33.1 Detailed Description

Useful macro definitions for general operations.

10.33.2 Macro Definition Documentation

10.33.2.1 PH_CONCAT_2

```
#define PH_CONCAT_2(
              a,
              b) a##b
```

10.33.2.2 PH_CONCAT_3

```
#define PH_CONCAT_3(
              a,
              b,
              c) a##b##c
```

10.33.2.3 PH_CONCAT_4

```
#define PH_CONCAT_4(
              a,
              b,
              c,
              d) a##b##c##d
```

10.33.2.4 PH_CONCAT_5

```
#define PH_CONCAT_5(
              a,
              b,
              c,
              d,
              e) a##b##c##d##e
```

10.33.2.5 PH_CONCAT_6

```
#define PH_CONCAT_6(
              a,
              b,
              c,
              d,
              e,
              f) a##b##c##d##e##f
```

10.33.2.6 PH_CONCAT_7

```
#define PH_CONCAT_7(
              a,
              b,
              c,
              d,
              e,
              f,
              g) a##b##c##d##e##f##g
```

10.33.2.7 PH_CONCAT_8

```
#define PH_CONCAT_8(
              a,
              b,
              c,
              d,
              e,
              f,
              g,
              h) a##b##c##d##e##f##g##h
```

10.33.2.8 PH_NO_OP

```
#define PH_NO_OP() ((void)0)
```

Places an expression that does nothing. Useful in situations where the macro expansion is intended to mimic a function scope expression while doing nothing.

10.34 macro.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00008 #define PH_CONCAT_2(a, b) a##b
00009 #define PH_CONCAT_3(a, b, c) a##b##c
00010 #define PH_CONCAT_4(a, b, c, d) a##b##c##d
00011 #define PH_CONCAT_5(a, b, c, d, e) a##b##c##d##e
00012 #define PH_CONCAT_6(a, b, c, d, e, f) a##b##c##d##e##f
00013 #define PH_CONCAT_7(a, b, c, d, e, f, g) a##b##c##d##e##f##g
00014 #define PH_CONCAT_8(a, b, c, d, e, f, g, h) a##b##c##d##e##f##g##h
00015
00020 #define PH_NO_OP() ((void)0)
```

10.35 Include/Common/math_basics.h File Reference

Basic math utilities. For more math functions, see `Engine` project's `math.h`.

```
#include "Common/assertion.h"
#include <type_traits>
#include <bit>
#include <concepts>
#include <limits>
```

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::math

Functions

- template<typename T >
  constexpr bool ph::math::is_power_of_2 (const T value)

  Determines whether `value` is a power of 2 number.
- template<std::integral T>
  T ph::math::ceil_div (const T numerator, const T denominator)

  Divide `numerator` by `denominator` and round up to integer. Both inputs must be positive integer. Specifically, `numerator` >= 0 and `denominator` > 0.
- template<std::integral T>
  T ph::math::next_multiple (const T value, const T multiple)

  *Get the next number that is an integer multiple of `multiple`. Specifically, get the minimum number x = C * `multiple` >= `value` where C is an integer >= 0. Currently supports positive integers only.*
- template<std::integral T>
  T ph::math::next_power_of_2_multiple (const T value, const T multiple)

  Same as next_multiple(T, T) except that `multiple` must be a power of 2 number.

10.35.1 Detailed Description

Basic math utilities. For more math functions, see `Engine` project's `math.h`.

10.36 math_basics.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00008 #include "Common/assertion.h"
00009
00010 #include <type_traits>
00011 #include <bit>
00012 #include <concepts>
00013 #include <limits>
00014
00015 namespace ph::math
00016 {
00017
00020 template<typename T>
00021 inline constexpr bool is_power_of_2(const T value)
00022 {
00023     if constexpr(std::is_unsigned_v<T>)
00024     {
00025         // STL has this function for unsigned types only
00026         return std::has_single_bit(value);
00027     }
00028     else
00029     {
00030         return (value > 0) && !(value & (value - 1));
00031     }
00032 }
00033
00037 template<std::integral T>
00038 inline T ceil_div(const T numerator, const T denominator)
00039 {
00040     PH_ASSERT_GE(numerator, 0);
00041     PH_ASSERT_GT(denominator, 0);
00042
00043     // Check for possible overflow when doing `numerator` + (`denominator` - 1).
00044     // (i.e., MAX >= `numerator` + (`denominator` - 1), rearranged to minimize overflow)
00045     PH_ASSERT_GE(std::numeric_limits<T>::max() - numerator, denominator - 1);
00046
00047     // We group the (`denominator` - 1) together as `numerator` is usually far larger than
00048     // `denominator`, doing (`denominator` - 1) first has better chance to avoid an overflow
00049     return (numerator + (denominator - 1)) / denominator;
00050 }
00051
00056 template<std::integral T>
00057 inline T next_multiple(const T value, const T multiple)
00058 {
00059     PH_ASSERT_GT(multiple, 0);
00060
00061     return ceil_div(value, multiple) * multiple;
00062 }
00063
00066 template<std::integral T>
00067 inline T next_power_of_2_multiple(const T value, const T multiple)
00068 {
00069     PH_ASSERT_GE(value, 0);
00070     PH_ASSERT(is_power_of_2(multiple));
00071
00072     // Reference: https://stackoverflow.com/a/9194117
00073
00074     // Check for possible overflow when doing `value` + (`multiple` - 1).
00075     // (see the implementation of `ceil_div()` for details about this test)
00076     PH_ASSERT_GE(std::numeric_limits<T>::max() - value, multiple - 1);
00077
00078     // `& ~(multiple - 1)` is the same as `& -multiple` for two's complement arithmetic
00079     return (value + (multiple - 1)) & (0 - multiple);
00080
00081     // Note:
00082     // MSVC may emit C4146 warning on applying a unary negate operation on unsigned types. Subtracting
    a
00083     // value from zero is the same as taking its negative, but using the binary subtraction operator
    avoids
00084     // the warning about taking the negative of an unsigned value. Hence the `0 - multiple` instead of
    `-multiple`.
00085     // Reference: https://stackoverflow.com/a/26893482
00086 }
00087
00088 }// end namespace ph::math
```

10.37 Include/Common/memory.h File Reference

Low-level memory allocation routines.

```
#include "Common/assertion.h"
#include <cstddef>
#include <memory>
#include "Common/memory.ipp"
```

Classes

- struct ph::detail::AlignedMemoryDeleter

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::detail

  Implementation detail mainly for internal usages.

Typedefs

- template<typename T >
  using ph::TAlignedMemoryUniquePtr = std::unique_ptr<T, detail::AlignedMemoryDeleter>

Functions

- void * ph::detail::allocate_aligned_memory (std::size_t numBytes, std::size_t alignmentInBytes)
- void ph::detail::free_aligned_memory (void *ptr)
- template<typename T = void>
  auto ph::make_aligned_memory (std::size_t numBytes, std::size_t alignmentInBytes) -> TAlignedMemoryUniquePtr< T >

  Create an aligned memory resource.
- template<typename T >
  void ph::from_bytes (const std::byte *srcBytes, T *out_dstValue)
- template<typename T >
  void ph::to_bytes (const T &srcValue, std::byte *out_dstBytes)
- template<std::size_t N>
  void ph::reverse_bytes (std::byte *bytes)
- template<typename T >
  T * ph::start_implicit_lifetime_as (void *ptr) noexcept

  Wrapper for `std::start_lifetime_as()`. Primarily a fallback when C++23 is not available. This function may touch the storage. For cv overloads or one that does not touch the storage, use `std::start_lifetime_as()` (requires C++23).
- template<typename T >
  T * ph::start_implicit_lifetime_as_array (void *ptr, std::size_t numArrayElements) noexcept

  Wrapper for `std::start_lifetime_as_array()`. Primarily a fallback when C++23 is not available. This function may touch the storage. For cv overloads or one that does not touch the storage, use `std::start_lifetime_as_array()` (requires C++23).

10.37.1 Detailed Description

Low-level memory allocation routines.

10.38 memory.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00007 #include "Common/assertion.h"
00008
00009 #include <cstddef>
00010 #include <memory>
00011
00012 namespace ph
00013 {
00014
00015 namespace detail
00016 {
00017
00026 [[nodiscard]]
00027 void* allocate_aligned_memory(std::size_t numBytes, std::size_t alignmentInBytes);
00028
00034 void free_aligned_memory(void* ptr);
00035
00036 struct AlignedMemoryDeleter
00037 {
00038     inline void operator () (void* const ptr) const
00039     {
00040         free_aligned_memory(ptr);
00041     }
00042
00043     inline void operator () (const void* const ptr) const
00044     {
00045         // `const_cast` to support const overload. This is fine as aligned memory is only returned as
00046         // a non-const pointer (see `allocate_aligned_memory()`), and casting an originally non-const
00047         // type is safe.
00048         free_aligned_memory(const_cast<void*>(ptr));
00049     }
00050 };
00051
00052 }// end namespace detail
00053
00054 template<typename T>
00055 using TAlignedMemoryUniquePtr = std::unique_ptr<T, detail::AlignedMemoryDeleter>;
00056
00057 // Note that `detail::AlignedMemoryDeleter` is for empty base optimization on `std::unique_ptr`,
00058 // see https://stackoverflow.com/questions/42715492/stdunique-ptr-and-custom-deleters.
00059 // This would reduce the size of the resulting `unique_ptr` to the size of a single pointer.
00060 // Reference:
      https://stackoverflow.com/questions/45341371/memory-efficient-custom-deleter-for-stdunique-ptr
00061 //
00062 // The following test will ensure this is true:
00063 static_assert(sizeof(TAlignedMemoryUniquePtr<void>) == sizeof(void*));
00064
00079 template<typename T = void>
00080 inline auto make_aligned_memory(std::size_t numBytes, std::size_t alignmentInBytes)
00081 -> TAlignedMemoryUniquePtr<T>;
00082
00083 template<typename T>
00084 void from_bytes(const std::byte* srcBytes, T* out_dstValue);
00085
00086 template<typename T>
00087 void to_bytes(const T& srcValue, std::byte* out_dstBytes);
00088
00089 template<std::size_t N>
00090 void reverse_bytes(std::byte* bytes);
00091
00098 template<typename T>
00099 T* start_implicit_lifetime_as(void* ptr) noexcept;
00100
00107 template<typename T>
00108 T* start_implicit_lifetime_as_array(void* ptr, std::size_t numArrayElements) noexcept;
00109
00110 }// end namespace ph
00111
00112 #include "Common/memory.ipp"
```

10.39 Include/Common/memory.ipp File Reference

```
#include "Common/memory.h"
#include <cstdint>
```

```
#include <version>
#include <algorithm>
#include <type_traits>
#include <bit>
#include <cstring>
#include <new>
```

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::detail

  Implementation detail mainly for internal usages.

Concepts

- concept ph::detail::CPermissiveImplicitLifetime

Functions

- template<typename T = void>
  auto ph::make_aligned_memory (std::size_t numBytes, std::size_t alignmentInBytes) -> TAlignedMemoryUniquePtr< T >

  Create an aligned memory resource.
- template<typename T >
  void ph::from_bytes (const std::byte *srcBytes, T *out_dstValue)
- template<typename T >
  void ph::to_bytes (const T &srcValue, std::byte *out_dstBytes)
- template<std::size_t N>
  void ph::reverse_bytes (std::byte *bytes)
- template<typename T >
  T * ph::start_implicit_lifetime_as (void *ptr) noexcept

  Wrapper for `std::start_lifetime_as()`. Primarily a fallback when C++23 is not available. This function may touch the storage. For cv overloads or one that does not touch the storage, use `std::start_lifetime_as()` (requires C++23).
- template<typename T >
  T * ph::start_implicit_lifetime_as_array (void *ptr, std::size_t numArrayElements) noexcept

  Wrapper for `std::start_lifetime_as_array()`. Primarily a fallback when C++23 is not available. This function may touch the storage. For cv overloads or one that does not touch the storage, use `std::start_lifetime_as_array()` (requires C++23).

10.40 memory.ipp

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Common/memory.h"
00004
00005 #include <cstdint>
00006 #include <version>
00007 #include <algorithm>
00008 #include <type_traits>
00009 #include <bit>
```

```
00010 #include <cstring>
00011 #include <new>
00012
00013 namespace ph
00014 {
00015
00016 template<typename T>
00017 inline auto make_aligned_memory(const std::size_t numBytes, const std::size_t alignmentInBytes)
00018 -> TAlignedMemoryUniquePtr<T>
00019 {
00020     if constexpr(!std::is_same_v<T, void>)
00021     {
00022         PH_ASSERT_EQ(alignmentInBytes % alignof(T), 0);
00023     }
00024
00025     void* const ptr = detail::allocate_aligned_memory(numBytes, alignmentInBytes);
00026
00027     // `static_cast` to `T*` is fine here: array types have implicit-lifetime, and now `ptr` points
00028     // to an array of `T` and pointer arithmetic is valid. Note that every element in `T*` still has
00029     // not started their lifetime if `T` is not an implicit-lifetime type.
00030     return TAlignedMemoryUniquePtr<T>(static_cast<T*>(ptr));
00031 }
00032
00033 template<typename T>
00034 inline void from_bytes(const std::byte* const srcBytes, T* const out_dstValue)
00035 {
00036     static_assert(std::is_trivially_copyable_v<T>);
00037
00038     PH_ASSERT(srcBytes);
00039     PH_ASSERT(out_dstValue);
00040
00041     std::copy(
00042         srcBytes,
00043         srcBytes + sizeof(T),
00044         reinterpret_cast<std::byte*>(out_dstValue));
00045 }
00046
00047 template<typename T>
00048 inline void to_bytes(const T& srcValue, std::byte* const out_dstBytes)
00049 {
00050     static_assert(std::is_trivially_copyable_v<T>);
00051
00052     PH_ASSERT(out_dstBytes);
00053
00054     std::copy(
00055         reinterpret_cast<const std::byte*>(&srcValue),
00056         reinterpret_cast<const std::byte*>(&srcValue) + sizeof(T),
00057         out_dstBytes);
00058 }
00059
00060 template<std::size_t N>
00061 inline void reverse_bytes(std::byte* const bytes)
00062 {
00063     PH_ASSERT(bytes);
00064
00065     // Cases for each `N`. One may also consider to add an implementation for a specific `N`.
00066
00067     if constexpr(N == 1)
00068     {
00069         // Nothing to do, single byte is already its own reverse
00070         return;
00071     }
00072     else if constexpr(N == 2)
00073     {
00074         std::uint16_t twoBytes;
00075         from_bytes(bytes, &twoBytes);
00076         twoBytes = std::byteswap(twoBytes);
00077         to_bytes(twoBytes, bytes);
00078     }
00079     else if constexpr(N == 4)
00080     {
00081         std::uint32_t fourBytes;
00082         from_bytes(bytes, &fourBytes);
00083         fourBytes = std::byteswap(fourBytes);
00084         to_bytes(fourBytes, bytes);
00085     }
00086     else if constexpr(N == 8)
00087     {
00088         std::uint64_t eightBytes;
00089         from_bytes(bytes, &eightBytes);
00090         eightBytes = std::byteswap(eightBytes);
00091         to_bytes(eightBytes, bytes);
00092     }
00093     else
00094     {
00095         std::reverse(bytes, bytes + N);
00096     }
```

```
00097 }
00098
00099 namespace detail
00100 {
00101
00107 template<typename T>
00108 concept CPermissiveImplicitLifetime = std::disjunction_v<
00109     std::is_scalar<T>,
00110     std::is_array<T>,
00111     std::is_aggregate<T>,
00112     std::conjunction<
00113         std::is_trivially_destructible<T>,
00114         std::disjunction<
00115             std::is_trivially_default_constructible<T>,
00116             std::is_trivially_copy_constructible<T>,
00117             std::is_trivially_move_constructible<T>>>;
00118
00119 }// end namespace detail
00120
00121 template<typename T>
00122 inline T* start_implicit_lifetime_as(void* ptr) noexcept
00123 {
00124 #if __cpp_lib_start_lifetime_as
00125     return std::start_lifetime_as<T>(ptr);
00126 #else
00127     return start_implicit_lifetime_as_array<T>(ptr, 1);
00128 #endif
00129 }
00130
00131 template<typename T>
00132 inline T* start_implicit_lifetime_as_array(void* ptr, std::size_t numArrayElements) noexcept
00133 {
00134     static_assert(
00135         std::is_trivially_copyable_v<T>
00136 #if __cpp_lib_is_implicit_lifetime
00137         && std::is_implicit_lifetime_v<T>
00138 #else
00139         && detail::CPermissiveImplicitLifetime<T>
00140 #endif
00141         );
00142
00143 #if __cpp_lib_start_lifetime_as
00144     return std::start_lifetime_as_array<T>(ptr, numArrayElements);
00145 #else
00146     // `std::memmove()` is one of the "magic" operations that implicitly create objects of
00147     // implicit lifetime type, we can hijack this property to do our work
00148     // (see https://stackoverflow.com/questions/76445860/implementation-of-stdstart-lifetime-as)
00149     // Note that using `new(ptr) std::byte[sizeof(T) * numArrayElements]` as in Robert Leahy's talk in
00150     // CppCon 2022 is an alternative to `std::memmove`, except that the object representation (value)
00151     // will be indeterminate due to placement new.
00152     return std::launder(static_cast<T*>(std::memmove(ptr, ptr, sizeof(T) * numArrayElements)));
00153 #endif
00154 }
00155
00156 }// end namespace ph
```

10.41 Include/Common/os.h File Reference

Operating system detection macros and utilities.

```
#include <cstddef>
#include <filesystem>
```

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::os

Macros

- #define PH_OPERATING_SYSTEM_IS_WINDOWS 0
- #define PH_OPERATING_SYSTEM_IS_LINUX 0
- #define PH_OPERATING_SYSTEM_IS_OSX 0

Enumerations

- enum class ph::os::EWindowsVersion {
  ph::os::Unknown = 0 , ph::os::Windows_2000 , ph::os::Windows_XP , ph::os::Windows_Vista ,
  ph::os::Windows_7 , ph::os::Windows_8 , ph::os::Windows_8_1 , ph::os::Windows_10 }

Functions

- EWindowsVersion ph::os::get_windows_version ()

  Get current Windows version at runtime.
- std::size_t ph::os::get_L1_cache_line_size_in_bytes ()

  Get size of L1 cache at runtime.
- std::filesystem::path ph::os::get_executable_path ()

  Get the path to the currently running executable. Answering the question, "Where am I?".

10.41.1 Detailed Description

Operating system detection macros and utilities.

The following macros will be defined as 1 for each operating system:

Windows: `PH_OPERATING_SYSTEM_IS_WINDOWS` Linux: `PH_OPERATING_SYSTEM_IS_LINUX` mac↩
OS: `PH_OPERATING_SYSTEM_IS_OSX`

10.41.2 Macro Definition Documentation

10.41.2.1 PH_OPERATING_SYSTEM_IS_LINUX

```
#define PH_OPERATING_SYSTEM_IS_LINUX 0
```

10.41.2.2 PH_OPERATING_SYSTEM_IS_OSX

```
#define PH_OPERATING_SYSTEM_IS_OSX 0
```

10.41.2.3 PH_OPERATING_SYSTEM_IS_WINDOWS

```
#define PH_OPERATING_SYSTEM_IS_WINDOWS 0
```

10.42 os.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00015 #include <cstddef>
00016 #include <filesystem>
00017
00018 // Defined on Windows x64 & x86
00019 #if defined(_WIN32)
00020
00021     #define PH_OPERATING_SYSTEM_IS_WINDOWS 1
00022
00023 // Defined on Linux
00024 #elif defined(__linux__)
00025
00026     #define PH_OPERATING_SYSTEM_IS_LINUX 1
00027
00028 // Defined on Apple platforms
00029 // Reference: https://stackoverflow.com/questions/12132933/preprocessor-macro-for-os-x-targets
00030 #elif defined(__APPLE__) || defined(__MACH__)
00031
00032     #include <TargetConditionals.h>
00033
00034     #if TARGET_OS_MAC == 1 && TARGET_OS_OSX == 1
00035         #define PH_OPERATING_SYSTEM_IS_OSX 1
00036     #else
00037         #error "Unsupported Apple operating system."
00038     #endif
00039
00040 #else
00041
00042     #error "Unsupported operating system."
00043
00044 #endif
00045
00046 #ifndef PH_OPERATING_SYSTEM_IS_WINDOWS
00047     #define PH_OPERATING_SYSTEM_IS_WINDOWS 0
00048 #endif
00049
00050 #ifndef PH_OPERATING_SYSTEM_IS_LINUX
00051     #define PH_OPERATING_SYSTEM_IS_LINUX 0
00052 #endif
00053
00054 #ifndef PH_OPERATING_SYSTEM_IS_OSX
00055     #define PH_OPERATING_SYSTEM_IS_OSX 0
00056 #endif
00057
00058 namespace ph::os
00059 {
00060
00061 enum class EWindowsVersion
00062 {
00063     Unknown = 0,
00064
00065     // Later version must have larger value
00066
00067     Windows_2000,
00068     Windows_XP,
00069     Windows_Vista,
00070     Windows_7,
00071     Windows_8,
00072     Windows_8_1,
00073     Windows_10
00074 };
00075
00078 EWindowsVersion get_windows_version();
00079
00083 std::size_t get_L1_cache_line_size_in_bytes();
00084
00089 std::filesystem::path get_executable_path();
00090
00091 }// end namespace ph::os
```

10.43 Include/Common/primitive_type.h File Reference

```
#include "Common/config.h"
#include <cstdint>
#include <climits>
```

Namespaces

- namespace ph

  The root for all renderer implementations.

Typedefs

- using ph::real = float
- using ph::integer = int
- using ph::hiReal = float64
- using ph::hiInteger = int64

- using ph::int8 = std::int8_t

  Fixed-size integer types.
- using ph::uint8 = std::uint8_t
- using ph::int16 = std::int16_t
- using ph::uint16 = std::uint16_t
- using ph::int32 = std::int32_t
- using ph::uint32 = std::uint32_t
- using ph::int64 = std::int64_t
- using ph::uint64 = std::uint64_t

- using ph::int8f = std::int_fast8_t

  Fastest integer types with size guarantee. For example, `uint32f` is an unsigned integer with at least 32 bits.
- using ph::uint8f = std::uint_fast8_t
- using ph::int16f = std::int_fast16_t
- using ph::uint16f = std::uint_fast16_t
- using ph::int32f = std::int_fast32_t
- using ph::uint32f = std::uint_fast32_t
- using ph::int64f = std::int_fast64_t
- using ph::uint64f = std::uint_fast64_t

- using ph::float32 = float

  Fixed-size floating-point types.
- using ph::float64 = double

Functions

- constexpr real ph::operator""_r (const long double cookedValue)

10.44 primitive_type.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Common/config.h"
00004
00005 #include <cstdint>
00006 #include <climits>
00007
00008 namespace ph
00009 {
00010
00014 using int8    = std::int8_t;
00015 using uint8   = std::uint8_t;
00016 using int16   = std::int16_t;
00017 using uint16  = std::uint16_t;
00018 using int32   = std::int32_t;
00019 using uint32  = std::uint32_t;
00020 using int64   = std::int64_t;
00021 using uint64  = std::uint64_t;
00023
00028 using int8f   = std::int_fast8_t;
00029 using uint8f  = std::uint_fast8_t;
00030 using int16f  = std::int_fast16_t;
00031 using uint16f = std::uint_fast16_t;
00032 using int32f  = std::int_fast32_t;
00033 using uint32f = std::uint_fast32_t;
00034 using int64f  = std::int_fast64_t;
00035 using uint64f = std::uint_fast64_t;
00037
00041 using float32 = float;
00042 using float64 = double;
00044
00045 #if PH_STRICT_FLOATING_POINT_SIZES
00046     static_assert(sizeof(float32) * CHAR_BIT == 32);
00047     static_assert(sizeof(float64) * CHAR_BIT == 64);
00048 #endif
00049
00050 #if PH_USE_DOUBLE_REAL
00051     using real = double;
00052 #else
00053     using real = float;
00054 #endif
00055
00056 using integer   = int;
00057 using hiReal    = float64;
00058 using hiInteger = int64;
00059
00060 constexpr inline real operator "" _r(const long double cookedValue)
00061 {
00062     return static_cast<real>(cookedValue);
00063 }
00064
00065 }// end namespace ph
```

10.45 Include/Common/profiling.h File Reference

Profiling functions.

```
#include "Common/config.h"
#include "Common/macro.h"
#include "Common/ThirdParty/lib_tracy.h"
```

Macros

- #define PH_DEFINE_PROFILE_UNIT_NAME(unitName)
- #define PH_PROFILE_LOOP_MARK(unitName) PH_NO_OP()
- #define PH_PROFILE_LOOP_BEGIN(unitName) PH_NO_OP()
- #define PH_PROFILE_LOOP_END(unitName) PH_NO_OP()
- #define PH_PROFILE_SCOPE() PH_NO_OP()
- #define PH_PROFILE_NAMED_SCOPE(nameStr) PH_NO_OP()
- #define PH_PROFILE_NAME_THIS_THREAD(threadNameStr) PH_NO_OP()

10.45.1 Detailed Description

Profiling functions.

10.45.2 Macro Definition Documentation

10.45.2.1 PH_DEFINE_PROFILE_UNIT_NAME

```
#define PH_DEFINE_PROFILE_UNIT_NAME(
              unitName)
```

10.45.2.2 PH_PROFILE_LOOP_BEGIN

```
#define PH_PROFILE_LOOP_BEGIN(
              unitName) PH_NO_OP()
```

10.45.2.3 PH_PROFILE_LOOP_END

```
#define PH_PROFILE_LOOP_END(
              unitName) PH_NO_OP()
```

10.45.2.4 PH_PROFILE_LOOP_MARK

```
#define PH_PROFILE_LOOP_MARK(
              unitName) PH_NO_OP()
```

10.45.2.5 PH_PROFILE_NAME_THIS_THREAD

```
#define PH_PROFILE_NAME_THIS_THREAD(
              threadNameStr) PH_NO_OP()
```

10.45.2.6 PH_PROFILE_NAMED_SCOPE

```
#define PH_PROFILE_NAMED_SCOPE(
              nameStr) PH_NO_OP()
```

10.45.2.7 PH_PROFILE_SCOPE

```
#define PH_PROFILE_SCOPE() PH_NO_OP()
```

10.46 profiling.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00007 #include "Common/config.h"
00008 #include "Common/macro.h"
00009
00010 #include "Common/ThirdParty/lib_tracy.h"
00011
00012 #if PH_PROFILING
00013
00017 #define PH_DEFINE_PROFILE_UNIT_NAME(unitName)\
00018     inline constexpr const char* internal_impl_profile_unit_name_##unitName = #unitName
00019
00020 #else
00021
00022 #define PH_DEFINE_PROFILE_UNIT_NAME(unitName)
00023
00024 #endif
00025
00026 #if PH_PROFILING && PH_THIRD_PARTY_HAS_TRACY
00027
00028 /*
00029 Note on names used by Tracy: Some Tracy macros require unique names to have unique pointer, see
00030 Tracy manual 3.1.2 "Unique pointers". `PH_DEFINE_PROFILE_UNIT_NAME()` meet the requirement.
00031 Macro parameters named `unitName` require unique names. For any other name (macro parameters with a
00032 `Str` suffix), see Tracy manual 3.1 "Handling text strings" more details (TL;DR: use literal for
00033 names without a size parameter; otherwise the string data will be copied).
00034 */
00035
00036 #define PH_PROFILE_LOOP_MARK(unitName)\
00037     FrameMarkNamed(internal_impl_profile_unit_name_##unitName)
00038
00039 #define PH_PROFILE_LOOP_BEGIN(unitName)\
00040     FrameMarkStart(internal_impl_profile_unit_name_##unitName)
00041
00042 #define PH_PROFILE_LOOP_END(unitName)\
00043     FrameMarkEnd(internal_impl_profile_unit_name_##unitName)
00044
00045 #define PH_PROFILE_SCOPE()\
00046     ZoneScoped
00047
00048 #define PH_PROFILE_NAMED_SCOPE(nameStr)\
00049     ZoneScopedN(nameStr)
00050
00051 #define PH_PROFILE_NAME_THIS_THREAD(threadNameStr)\
00052     tracy::SetThreadName(threadNameStr)
00053
00054 #else
00055
00056 #define PH_PROFILE_LOOP_MARK(unitName) PH_NO_OP()
00057 #define PH_PROFILE_LOOP_BEGIN(unitName) PH_NO_OP()
00058 #define PH_PROFILE_LOOP_END(unitName) PH_NO_OP()
00059 #define PH_PROFILE_SCOPE() PH_NO_OP()
00060 #define PH_PROFILE_NAMED_SCOPE(nameStr) PH_NO_OP()
00061 #define PH_PROFILE_NAME_THIS_THREAD(threadNameStr) PH_NO_OP()
00062
00063 #endif
```

10.47 Include/Common/stats.h File Reference

```
#include <atomic>
#include <chrono>
#include <cstdint>
#include <string>
#include <string_view>
#include <vector>
```

Classes

- struct ph::detail::stats::TimeCounter

- struct ph::detail::stats::ScopedTimer
- class ph::TimerStatsReport
- struct ph::TimerStatsReport::TimeRecord
- struct ph::TimerStatsReport::GroupedTimeRecord

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::detail

  Implementation detail mainly for internal usages.
- namespace ph::detail::stats

Macros

- #define PH_DEFINE_INLINE_TIMER_STAT(statName, categoryName)
- #define PH_DEFINE_INTERNAL_TIMER_STAT(statName, categoryName)
- #define PH_DEFINE_EXTERNAL_TIMER_STAT(statName, categoryName) PH_DEFINE_INLINE_TIMER_STAT(statName, categoryName)
- #define PH_SCOPED_TIMER(statName)

10.47.1 Macro Definition Documentation

10.47.1.1 PH_DEFINE_EXTERNAL_TIMER_STAT

```
#define PH_DEFINE_EXTERNAL_TIMER_STAT(
              statName,
              categoryName)  PH_DEFINE_INLINE_TIMER_STAT(statName, categoryName)
```

10.47.1.2 PH_DEFINE_INLINE_TIMER_STAT

```
#define PH_DEFINE_INLINE_TIMER_STAT(
              statName,
              categoryName)
```

Value:

```
inline ::ph::detail::stats::TimeCounter& internal_impl_time_counter_access_##statName()\
{\
    static ::ph::detail::stats::TimeCounter counter(#statName, #categoryName);\
    return counter;\
}
```

10.47.1.3 PH_DEFINE_INTERNAL_TIMER_STAT

```
#define PH_DEFINE_INTERNAL_TIMER_STAT(
              statName,
              categoryName)
```

Value:

```
namespace\
{\
    PH_DEFINE_INLINE_TIMER_STAT(statName, categoryName);\
}
```

10.47.1.4 PH_SCOPED_TIMER

```
#define PH_SCOPED_TIMER(
                statName)
```

Value:

```
    ::ph::detail::stats::ScopedTimer internal_impl_scopedTimer##statName(\
        internal_impl_time_counter_access_##statName())
```

10.48 stats.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <atomic>
00004 #include <chrono>
00005 #include <cstdint>
00006 #include <string>
00007 #include <string_view>
00008 #include <vector>
00009
00010 namespace ph
00011 {
00012
00013 namespace detail::stats
00014 {
00015
00016 struct TimeCounter final
00017 {
00018     const std::string    name;
00019     const std::string    category;
00020     std::atomic_uint64_t totalMicroseconds;
00021     std::atomic_uint64_t count;
00022
00023     TimeCounter(std::string name, std::string category);
00024     void addMicroseconds(std::uint64_t microseconds);
00025 };
00026
00027 struct ScopedTimer final
00028 {
00029     using Clock = std::chrono::steady_clock;
00030
00031     TimeCounter&      counter;
00032     Clock::time_point startTime;
00033
00034     explicit ScopedTimer(TimeCounter& counter);
00035     ~ScopedTimer();
00036 };
00037
00038 }// end namespace detail::stats
00039
00040 class TimerStatsReport final
00041 {
00042 public:
00043     struct TimeRecord final
00044     {
00045         std::string   name;
00046         std::string   category;
00047         std::uint64_t totalMicroseconds;
00048         std::uint64_t count;
00049
00050         TimeRecord();
00051     };
00052
00053     struct GroupedTimeRecord final
00054     {
00055         std::string   groupName;
00056         std::uint64_t totalMicroseconds;
00057         std::uint64_t count;
00058
00059         std::vector<GroupedTimeRecord> subgroups;
00060
00061         GroupedTimeRecord();
00062     };
00063
00064     TimerStatsReport();
00065
00066     GroupedTimeRecord getGroupedTimeRecord() const;
```

```
00067     std::string proportionalReport() const;
00068     std::string averagedReport() const;
00069     std::string detailedReport() const;
00070     std::string rawReport() const;
00071
00072 private:
00073     enum class EGroupedReport
00074     {
00075         ProportionOnly,
00076         AverageOnly,
00077         ProportionWithAverage,
00078         ProportionWithAverageAndTotal
00079     };
00080
00081     std::vector<TimeRecord> m_records;
00082
00083     static GroupedTimeRecord makeGroupedTimeRecordRecursive(const std::vector<TimeRecord>& records);
00084
00085     static std::string makeGroupedReportRecursive(
00086         const GroupedTimeRecord& records,
00087         EGroupedReport           reportType,
00088         const std::string&       linePrefix);
00089 };
00090
00091 }// end namespace ph
00092
00093 #define PH_DEFINE_INLINE_TIMER_STAT(statName, categoryName)\
00094     inline ::ph::detail::stats::TimeCounter& internal_impl_time_counter_access_##statName()\
00095     {\
00096         static ::ph::detail::stats::TimeCounter counter(#statName, #categoryName);\
00097         return counter;\
00098     }
00099
00100 #define PH_DEFINE_INTERNAL_TIMER_STAT(statName, categoryName)\
00101     namespace\
00102     {\
00103         PH_DEFINE_INLINE_TIMER_STAT(statName, categoryName);\
00104     }
00105
00106 #define PH_DEFINE_EXTERNAL_TIMER_STAT(statName, categoryName)\
00107     PH_DEFINE_INLINE_TIMER_STAT(statName, categoryName)
00108
00109 #define PH_SCOPED_TIMER(statName)\
00110     ::ph::detail::stats::ScopedTimer internal_impl_scopedTimer##statName(\
00111         internal_impl_time_counter_access_##statName())
```

10.49 Include/Common/ThirdParty/lib_tracy.h File Reference

10.50 lib_tracy.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #if PH_THIRD_PARTY_HAS_TRACY
00004
00005 #ifndef WIN32_LEAN_AND_MEAN
00006 #define WIN32_LEAN_AND_MEAN
00007 #endif
00008
00009 // This should be done by the build script.
00010 #ifndef TRACY_ENABLE
00011 #error "Tracy profiler is not enabled. Please make sure `TRACY_ENABLE` is defined globally by the
      build script."
00012 #endif
00013
00014 #include <tracy/Tracy.hpp>
00015
00016 #endif
```

10.51 Include/Common/utility.h File Reference

```
#include "Common/config.h"
#include <string>
```

```
#include <type_traits>
#include <array>
#include <cstddef>
#include <concepts>
#include "Common/utility.ipp"
```

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::detail

  Implementation detail mainly for internal usages.

Macros

- #define PH_NOT_IMPLEMENTED_WARNING()

Functions

- void ph::detail::output_not_implemented_warning (const std::string &filename, const std::string &lineNumber)
- template<typename T >
  consteval std::size_t ph::sizeof_in_bits ()

  Calculates number of bits an instance of type `T` occupies.
- template<typename T , std::size_t N>
  constexpr std::array< T, N > ph::make_array (const T &element)

  Creates an `std::array` filled with the same element.
- template<std::integral DstType, std::integral SrcType>
  DstType ph::lossless_integer_cast (const SrcType &src)
- template<std::floating_point DstType, std::floating_point SrcType>
  DstType ph::lossless_float_cast (const SrcType &src)
- template<typename DstType , typename SrcType >
  DstType ph::lossless_cast (const SrcType &src)

  Cast numeric value to another type without any loss of information. If there is any possible overflow or numeric precision loss, exception is thrown.
- template<typename DstType , typename SrcType >
  DstType ph::lossless_cast (const SrcType &src, DstType *const out_dst)

10.51.1 Macro Definition Documentation

10.51.1.1 PH_NOT_IMPLEMENTED_WARNING

```
#define PH_NOT_IMPLEMENTED_WARNING()
```

Value:

```
    do\
    {\
        ::ph::detail::output_not_implemented_warning(\
            std::string(__FILE__),\
            std::to_string(__LINE__));\
    } while(0)
```

10.52 utility.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Common/config.h"
00004
00005 #include <string>
00006 #include <type_traits>
00007 #include <array>
00008 #include <cstddef>
00009 #include <concepts>
00010
00011 namespace ph::detail
00012 {
00013
00014 void output_not_implemented_warning(
00015     const std::string& filename,
00016     const std::string& lineNumber);
00017
00018 }// end namespace ph::detail
00019
00020 #define PH_NOT_IMPLEMENTED_WARNING()\
00021     do\
00022     {\
00023         ::ph::detail::output_not_implemented_warning(\
00024             std::string(__FILE__),\
00025             std::to_string(__LINE__));\
00026     } while(0)
00027
00028 namespace ph
00029 {
00030
00033 template<typename T>
00034 consteval std::size_t sizeof_in_bits();
00035
00039 template<typename T, std::size_t N>
00040 constexpr std::array<T, N> make_array(const T& element);
00041
00042 template<std::integral DstType, std::integral SrcType>
00043 DstType lossless_integer_cast(const SrcType& src);
00044
00045 template<std::floating_point DstType, std::floating_point SrcType>
00046 DstType lossless_float_cast(const SrcType& src);
00047
00053 template<typename DstType, typename SrcType>
00054 DstType lossless_cast(const SrcType& src);
00055
00056 template<typename DstType, typename SrcType>
00057 DstType lossless_cast(const SrcType& src, DstType* const out_dst);
00058
00059 }// end namespace ph
00060
00061 #include "Common/utility.ipp"
```

10.53 Include/Common/utility.ipp File Reference

```
#include "Common/utility.h"
#include "Common/assertion.h"
#include "Common/exceptions.h"
#include <utility>
#include <climits>
#include <limits>
```

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::detail

  Implementation detail mainly for internal usages.

Functions

- template<typename T , std::size_t... Is>
  constexpr std::array< T, sizeof...(Is)> ph::detail::make_array (T element, std::index_sequence< Is... >)
- template<typename T >
  consteval std::size_t ph::sizeof_in_bits ()

  Calculates number of bits an instance of type `T` occupies.
- template<typename T , std::size_t N>
  constexpr std::array< T, N > ph::make_array (const T &element)

  Creates an `std::array` filled with the same element.
- template<std::integral DstType, std::integral SrcType>
  DstType ph::lossless_integer_cast (const SrcType &src)
- template<std::floating_point DstType, std::floating_point SrcType>
  DstType ph::lossless_float_cast (const SrcType &src)
- template<typename DstType , typename SrcType >
  DstType ph::lossless_cast (const SrcType &src)

  Cast numeric value to another type without any loss of information. If there is any possible overflow or numeric precision loss, exception is thrown.
- template<typename DstType , typename SrcType >
  DstType ph::lossless_cast (const SrcType &src, DstType *const out_dst)

10.54 utility.ipp

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "Common/utility.h"
00004 #include "Common/assertion.h"
00005 #include "Common/exceptions.h"
00006
00007 #include <utility>
00008 #include <climits>
00009 #include <limits>
00010
00011 namespace ph
00012 {
00013
00014 namespace detail
00015 {
00016
00017 template<typename T, std::size_t... Is>
00018 inline constexpr std::array<T, sizeof...(Is)> make_array(
00019     T element,
00020     std::index_sequence<Is...>)
00021 {
00022     // Use sequence as pattern to repeat `element', also to avoid unused warnings
00023     return {(static_cast<void>(Is), element)...};
00024 }
00025
00026 }// end namespace detail
00027
00028 template<typename T>
00029 inline consteval std::size_t sizeof_in_bits()
00030 {
00031     return CHAR_BIT * sizeof(T);
00032 }
00033
00034 template<typename T, std::size_t N>
00035 inline constexpr std::array<T, N> make_array(const T& element)
00036 {
00037     return detail::make_array(element, std::make_index_sequence<N>());
00038 }
00039
00040 template<std::integral DstType, std::integral SrcType>
00041 inline DstType lossless_integer_cast(const SrcType& src)
00042 {
00043     using SrcLimits = std::numeric_limits<SrcType>;
00044     using DstLimits = std::numeric_limits<DstType>;
00045
00046     // Note that the use of `std::cmp_<X>' functions are important as the comparisons
```

```
00047     // may be signed <-> unsigned comparisons, which may cause signed limits to overflow
00048
00049     // TODO: we may need to cast src to some integer first to support char and bool types (they are
      not supported by cmp functions)
00050
00051     constexpr bool mayHavePositiveOverflow = std::cmp_greater(SrcLimits::max(), DstLimits::max());
00052     constexpr bool mayHaveNegativeOverflow = std::cmp_less(SrcLimits::lowest(), DstLimits::lowest());
00053
00054     if constexpr(mayHavePositiveOverflow)
00055     {
00056         if(std::cmp_greater(src, DstLimits::max()))
00057         {
00058             throw_formatted<OverflowException>("cast results in positive overflow: {} exceeds the
      limit {}",
00059                 src, DstLimits::max());
00060         }
00061     }
00062
00063     if constexpr(mayHaveNegativeOverflow)
00064     {
00065         if(std::cmp_less(src, DstLimits::lowest()))
00066         {
00067             throw_formatted<OverflowException>("cast results in negative overflow: {} exceeds the
      limit {}",
00068                 src, DstLimits::lowest());
00069         }
00070     }
00071
00072     // All possible integer overflow scenarios are checked so it is safe to cast now
00073     return static_cast<DstType>(src);
00074 }
00075
00076 template<std::floating_point DstType, std::floating_point SrcType>
00077 inline DstType lossless_float_cast(const SrcType& src)
00078 {
00079     // Nothing to do if both types are the same
00080     if constexpr(std::is_same_v<SrcType, DstType>)
00081     {
00082         return src;
00083     }
00084     // If we are converting to a wider floating-point type, generally it will be lossless
00085     else if constexpr(sizeof(DstType) > sizeof(SrcType))
00086     {
00087         // We need both types to be IEEE-754
00088         static_assert(std::numeric_limits<SrcType>::is_iec559);
00089         static_assert(std::numeric_limits<DstType>::is_iec559);
00090
00091         return static_cast<DstType>(src);
00092     }
00093     // Otherwise, cast to `DstType` then back to `SrcType` and see if there is any difference
00094     else
00095     {
00096         const auto dst = static_cast<DstType>(src);
00097         const auto dstBackToSrc = static_cast<SrcType>(dst);
00098         if(src != dstBackToSrc)
00099         {
00100             throw_formatted<NumericException>("cast results in numeric precision loss: {} -> {}",
00101                 src, dstBackToSrc);
00102         }
00103
00104         return dst;
00105     }
00106 }
00107
00108 template<typename DstType, typename SrcType>
00109 inline DstType lossless_cast(const SrcType& src)
00110 {
00111     // Integer -> Integer
00112     if constexpr(std::is_integral_v<SrcType> && std::is_integral_v<DstType>)
00113     {
00114         return lossless_integer_cast<DstType>(src);
00115     }
00116     // Integer -> Floating-point
00117     else if constexpr(std::is_integral_v<SrcType> && std::is_floating_point_v<DstType>)
00118     {
00119         // TODO
00120         PH_ASSERT_UNREACHABLE_SECTION();
00121         return 0;
00122     }
00123     // Floating-point -> Integer
00124     else if constexpr(std::is_floating_point_v<SrcType> && std::is_integral_v<DstType>)
00125     {
00126         // TODO
00127         PH_ASSERT_UNREACHABLE_SECTION();
00128         return 0;
00129     }
00130     // Floating-point -> Floating-point
```

```
00131     else
00132     {
00133         static_assert(std::is_floating_point_v<SrcType> && std::is_floating_point_v<DstType>);
00134 
00135         return lossless_float_cast<DstType>(src);
00136     }
00137 }
00138 
00139 template<typename DstType, typename SrcType>
00140 inline DstType lossless_cast(const SrcType& src, DstType* const out_dst)
00141 {
00142     PH_ASSERT(out_dst);
00143 
00144     *out_dst = lossless_cast<DstType>(src);
00145     return *out_dst;
00146 }
00147 
00148 }// end namespace ph
```

10.55 Include/Common/Utility/CommandLineArguments.h File Reference

```
#include "Common/Utility/string_utils.h"
#include <string>
#include <vector>
#include <optional>
#include <type_traits>
```

Classes

- class ph::CommandLineArguments

  Helper for parsing command line arguments.

Namespaces

- namespace ph

  The root for all renderer implementations.

10.56 CommandLineArguments.h

Go to the documentation of this file.

```
00001 #pragma once
00002 
00003 #include "Common/Utility/string_utils.h"
00004 
00005 #include <string>
00006 #include <vector>
00007 #include <optional>
00008 #include <type_traits>
00009 
00010 namespace ph
00011 {
00012 
00015 class CommandLineArguments final
00016 {
00017 public:
00018     CommandLineArguments(int argc, char* argv[]);
00019 
00023     std::string getProgramName() const;
00024 
00027     bool isEmpty() const;
00028 
00032     std::string retrieveString(const std::string& defaultString = "");
```

```
00033
00037     std::vector<std::string> retrieveStrings(std::size_t numValues);
00038
00049     std::vector<std::string> retrieveOptionArguments(const std::string& optionPrefix);
00050
00057     std::vector<std::string> retrieveStrings(
00058         const std::string& startingPrefix,
00059         const std::string& endingPrefix,
00060         bool shouldIncludeStart = true,
00061         bool shouldIncludeEnd = true);
00062
00066     template<typename T>
00067     T retrieveInt(T defaultInt = 0);
00068
00072     template<typename T>
00073     T retrieveFloat(T defaultFloat = 0.0f);
00074
00075     template<typename T>
00076     std::optional<T> retrieve();
00077
00078 private:
00079     std::string              m_programName;
00080     std::vector<std::string> m_arguments;
00081 };
00082
00083 // In-header Implementations:
00084
00085 inline std::string CommandLineArguments::getProgramName() const
00086 {
00087     return m_programName;
00088 }
00089
00090 inline bool CommandLineArguments::isEmpty() const
00091 {
00092     return m_arguments.empty();
00093 }
00094
00095 template<typename T>
00096 inline T CommandLineArguments::retrieveInt(T defaultInt)
00097 {
00098     static_assert(std::is_integral_v<T>,
00099         "expect argument type to be integer");
00100
00101     auto optInt = retrieve<T>();
00102     return optInt ? *optInt : defaultInt;
00103 }
00104
00105 template<typename T>
00106 inline T CommandLineArguments::retrieveFloat(T defaultFloat)
00107 {
00108     static_assert(std::is_floating_point_v<T>,
00109         "expect argument type to be floating-point");
00110
00111     auto optFloat = retrieve<T>();
00112     return optFloat ? *optFloat : defaultFloat;
00113 }
00114
00115 template<typename T>
00116 inline std::optional<T> CommandLineArguments::retrieve()
00117 {
00118     if(isEmpty())
00119     {
00120         return std::nullopt;
00121     }
00122
00123     std::string argument = m_arguments.front();
00124     m_arguments.erase(m_arguments.begin());
00125
00126     if constexpr(std::is_same_v<T, std::string>)
00127     {
00128         return argument;
00129     }
00130     else
00131     {
00132         static_assert(std::is_integral_v<T> || std::is_floating_point_v<T>,
00133             "expect argument type to be integer or floating-point");
00134
00135         return string_utils::parse_number<T>(argument);
00136     }
00137 }
00138
00139 }// end namespace ph
```

10.57 Include/Common/Utility/string_utils.h File Reference

String manipulation helpers.

```
#include "Common/Utility/string_utils_table.h"
#include "Common/assertion.h"
#include "Common/exceptions.h"
#include <cstddef>
#include <string>
#include <algorithm>
#include <string_view>
#include <stdexcept>
#include <charconv>
#include <limits>
#include <climits>
#include <type_traits>
#include <format>
#include <concepts>
#include <array>
```

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::string_utils

  Contains various string manipulation helpers.
- namespace ph::string_utils::detail_from_to_char

Concepts

- concept ph::string_utils::CHasToString

Macros

- #define PH_DEFINE_INLINE_TO_STRING_FORMATTER_SPECIALIZATION(...)
- #define PH_DEFINE_INLINE_TO_STRING_FORMATTER(...)

  Defines a formatter which calls the `toString()` method. For example, to define a `toString()` formatter for the class `SomeType`, place the macro after class definition:
- #define PH_DEFINE_INLINE_TO_STRING_FORMATTER_TEMPLATE(...) PH_DEFINE_INLINE_TO_STRING_FORMATTER __VA_ARGS__)

  Defines a formatter template which calls the `toString()` method. For example, to define a `toString()` formatter for the class template `TSomeType`, place the macro after class definition:

Enumerations

- enum class ph::string_utils::EWhitespace { ph::string_utils::Common , ph::string_utils::Standard }

Functions

- template<EWhitespace TYPE = EWhitespace::Common>
  std::string_view ph::string_utils::get_whitespaces ()
- template<EWhitespace TYPE = EWhitespace::Common>
  constexpr bool ph::string_utils::is_whitespace (const char ch)
- bool ph::string_utils::has_any_of (const std::string_view srcStr, const std::string_view candidates)
- bool ph::string_utils::has_none_of (const std::string_view srcStr, const std::string_view candidates)
- std::string_view ph::string_utils::cut_head (const std::string_view srcStr, const std::string_view candidates)

  Remove characters from the beginning.
- std::string_view ph::string_utils::cut_tail (const std::string_view srcStr, const std::string_view candidates)

  Remove characters from the end.
- std::string_view ph::string_utils::cut_ends (const std::string_view srcStr, const std::string_view candidates)

  Remove characters from both ends.
- template<EWhitespace TYPE = EWhitespace::Common>
  std::string_view ph::string_utils::trim_head (const std::string_view srcStr)

  Remove white spaces from the beginning.
- template<EWhitespace TYPE = EWhitespace::Common>
  std::string_view ph::string_utils::trim_tail (const std::string_view srcStr)

  Remove white spaces from the end.
- template<EWhitespace TYPE = EWhitespace::Common>
  std::string_view ph::string_utils::trim (const std::string_view srcStr)

  Remove white spaces from both ends.
- std::string_view ph::string_utils::next_token (std::string_view srcStr, std::string_view *const out_remaining↩
  Str=nullptr, const std::string_view tokenSeparators=get_whitespaces<>())

  Retrieve a token from a string.
- char ph::string_utils::az_to_AZ (const char ch)

  Convert lower-case characters to upper-case.
- char ph::string_utils::AZ_to_az (const char ch)

  Convert upper-case characters to lower-case.
- void ph::string_utils::az_to_AZ (std::string &str)

  Convert lower-case characters to upper-case.
- void ph::string_utils::AZ_to_az (std::string &str)

  Convert upper-case characters to lower-case.
- std::string ph::string_utils::repeat (const std::string_view str, const std::size_t n)

  Repeat the input string for N times.
- void ph::string_utils::erase_all (std::string &str, const char ch)

  Remove all occurrence of a character in the string.
- void ph::string_utils::detail_from_to_char::throw_from_std_errc_if_has_error (const std::errc errorCode)
- template<typename T >
  T ph::string_utils::parse_float (const std::string_view floatStr)

  Returns a float by processing its string representation. Supports float, double, and long double.
- template<typename T >
  T ph::string_utils::parse_int (std::string_view intStr)

  Returns an integer by processing its string representation. Supports the following:
- template<typename NumberType >
  NumberType ph::string_utils::parse_number (const std::string_view numberStr)

  Returns a number by processing its string representation. Accepts all types supported by parse_float(std::string_view) and parse_int(std::string_view).
- template<typename T >
  std::size_t ph::string_utils::stringify_float (const T value, char *const out_buffer, const std::size_t bufferSize)

  Converts a float to string.

- template<std::integral T>
  std::size_t ph::string_utils::stringify_int_alphabetic (const T value, char *const out_buffer, const std::size_t bufferSize, const int base)

  Converts an integer to base [2, 62] string.
- template<std::integral T>
  std::size_t ph::string_utils::stringify_int (const T value, char *const out_buffer, const std::size_t bufferSize, const int base=10)

  Converts an integer to string.
- template<typename NumberType >
  std::size_t ph::string_utils::stringify_number (const NumberType value, char *const out_buffer, const std::size_t bufferSize)

  Converts a number to string. Accepts all types supported by stringify_float(T, char, std::size_t) and stringify_int(T, char*, std::size_t). The written string is not null terminated.*
- template<typename NumberType >
  std::string & ph::string_utils::stringify_number (const NumberType value, std::string &out_str, const std::size_t maxChars=64)

  Converts a number to string. Similar to `stringify_number(NumberType, char*, std::size_t)`, except that this variant writes to `std::string` and the resulting string is guaranteed to be null terminated (by calling `std::string::c_str()`).
- template<typename NumberType >
  std::string ph::string_utils::stringify_number (const NumberType value, const std::size_t maxChars=64)

  Converts a number to string. Similar to `stringify_number(NumberType, std::string&, std::size_t)`, except that this variant creates a new string.

10.57.1 Detailed Description

String manipulation helpers.

10.57.2 Macro Definition Documentation

10.57.2.1 PH_DEFINE_INLINE_TO_STRING_FORMATTER

```
#define PH_DEFINE_INLINE_TO_STRING_FORMATTER(
              ...)
```

Value:

```
template<>\
PH_DEFINE_INLINE_TO_STRING_FORMATTER_SPECIALIZATION(__VA_ARGS__)
```

Defines a formatter which calls the `toString()` method. For example, to define a `toString()` formatter for the class `SomeType`, place the macro after class definition:

```
class SomeType { (class definitions) };

PH_DEFINE_INLINE_TO_STRING_FORMATTER(SomeType);
```

Parameters

...	The type to define a formatter for.

10.57.2.2 PH_DEFINE_INLINE_TO_STRING_FORMATTER_SPECIALIZATION

```
#define PH_DEFINE_INLINE_TO_STRING_FORMATTER_SPECIALIZATION(
                ...)
```

Value:

```
    struct ::std::formatter<__VA_ARGS__> : ::std::formatter<::std::string>\
    {\
        static_assert(::ph::string_utils::CHasToString<__VA_ARGS__>,\
            "type " #__VA_ARGS__ " must have a const method toString() and the result should be "\
            "implicitly convertible to std::string"); \
    \
        /* `parse()` is inherited from the base class */\
    \
        /* Define `format()` by calling `std::string`'s implementation with custom type's `toString()`*/\
        inline auto format(const __VA_ARGS__& value, ::std::format_context& ctx) const\
        {\
            return ::std::formatter<::std::string>::format(\
                value.toString(), ctx);\
        }\
    }
```

10.57.2.3 PH_DEFINE_INLINE_TO_STRING_FORMATTER_TEMPLATE

```
#define PH_DEFINE_INLINE_TO_STRING_FORMATTER_TEMPLATE(
                ...)    PH_DEFINE_INLINE_TO_STRING_FORMATTER_SPECIALIZATION(__VA_ARGS__)
```

Defines a formatter template which calls the `toString()` method. For example, to define a `toString()` formatter for the class template `TSomeType`, place the macro after class definition:

```
template<typename T>
class TSomeType { (class definitions) };

template<typename T>
PH_DEFINE_INLINE_TO_STRING_FORMATTER_TEMPLATE(TSomeType<T>);
```

Parameters

...	The type to define a formatter for.

10.58 string_utils.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00007 #include "Common/Utility/string_utils_table.h"
00008 #include "Common/assertion.h"
00009 #include "Common/exceptions.h"
00010
00011 #include <cstddef>
00012 #include <string>
00013 #include <algorithm>
00014 #include <string_view>
00015 #include <stdexcept>
00016 #include <charconv>
00017 #include <limits>
00018 #include <climits>
00019 #include <type_traits>
00020 #include <format>
00021 #include <concepts>
00022 #include <array>
00023
00024 namespace ph::string_utils
00025 {
00026
00027 template<typename ObjType>
00028 concept CHasToString = requires (const ObjType& obj)
00029 {
00030     { obj.toString() } -> std::convertible_to<std::string_view>;
```

```
00031 };
00032
00033 }// end namespace ph::string_utils
00034
00035 #define PH_DEFINE_INLINE_TO_STRING_FORMATTER_SPECIALIZATION(...)\
00036     struct ::std::formatter<__VA_ARGS__> : ::std::formatter<::std::string>\
00037     {\
00038         static_assert(::ph::string_utils::CHasToString<__VA_ARGS__>,\
00039             "type " #__VA_ARGS__ " must have a const method toString() and the result should be "\
00040             "implicitly convertible to std::string"); \
00041     \
00042         /* `parse()` is inherited from the base class */\
00043     \
00044         /* Define `format()` by calling `std::string`'s implementation with custom type's
    `toString()`*/\
00045         inline auto format(const __VA_ARGS__& value, ::std::format_context& ctx) const\
00046         {\
00047             return ::std::formatter<::std::string>::format(\
00048                 value.toString(), ctx);\
00049         }\
00050     }
00051
00064 #define PH_DEFINE_INLINE_TO_STRING_FORMATTER(...)\
00065     template<>\
00066     PH_DEFINE_INLINE_TO_STRING_FORMATTER_SPECIALIZATION(__VA_ARGS__)
00067
00082 #define PH_DEFINE_INLINE_TO_STRING_FORMATTER_TEMPLATE(...)\
00083     PH_DEFINE_INLINE_TO_STRING_FORMATTER_SPECIALIZATION(__VA_ARGS__)
00084
00085 namespace ph::string_utils
00086 {
00087
00088 enum class EWhitespace
00089 {
00091     Common,
00092
00094     Standard
00095 };
00096
00097 template<EWhitespace TYPE = EWhitespace::Common>
00098 inline std::string_view get_whitespaces()
00099 {
00100     if constexpr(TYPE == EWhitespace::Common)
00101     {
00102         return table::common_whitespaces;
00103     }
00104     else if constexpr(TYPE == EWhitespace::Standard)
00105     {
00106         return table::standard_whitespaces;
00107     }
00108     else
00109     {
00110         static_assert(TYPE == EWhitespace::Common || TYPE == EWhitespace::Standard,
00111             "Must include a case for each enum entry; did you forget to add one?");
00112
00113         return "";
00114     }
00115 }
00116
00117 template<EWhitespace TYPE = EWhitespace::Common>
00118 inline constexpr bool is_whitespace(const char ch)
00119 {
00120     return get_whitespaces<TYPE>().find(ch) != std::string_view::npos;
00121 }
00122
00123 inline bool has_any_of(const std::string_view srcStr, const std::string_view candidates)
00124 {
00125     const auto foundPos = srcStr.find_first_of(candidates);
00126     return foundPos != std::string_view::npos;
00127 }
00128
00129 inline bool has_none_of(const std::string_view srcStr, const std::string_view candidates)
00130 {
00131     return !has_any_of(srcStr, candidates);
00132 }
00133
00143 inline std::string_view cut_head(const std::string_view srcStr, const std::string_view candidates)
00144 {
00145     const auto nonCutPos = srcStr.find_first_not_of(candidates);
00146
00147     auto cutStr = srcStr;
00148
00149     // remove_prefix(): behavior is undefined for inputPos > size(), avoid that
00150     // with the ternary operator
00151     cutStr.remove_prefix(
00152         nonCutPos != std::string_view::npos ? nonCutPos : srcStr.size());
00153
```

```
00154     return cutStr;
00155 }
00156
00166 inline std::string_view cut_tail(const std::string_view srcStr, const std::string_view candidates)
00167 {
00168     const auto nonCutPos = srcStr.find_last_not_of(candidates);
00169
00170     auto cutStr = srcStr;
00171
00172     // remove_suffix(): behavior is undefined for inputPos > size(), avoid that
00173     // with the ternary operator;
00174     // also, if <nonCutPos> is not npos, <srcStr> will not be empty
00175     cutStr.remove_suffix(
00176         nonCutPos != std::string_view::npos ? srcStr.size() - 1 - nonCutPos : srcStr.size());
00177
00178     return cutStr;
00179 }
00180
00190 inline std::string_view cut_ends(const std::string_view srcStr, const std::string_view candidates)
00191 {
00192     return cut_head(cut_tail(srcStr, candidates), candidates);
00193 }
00194
00200 template<EWhitespace TYPE = EWhitespace::Common>
00201 inline std::string_view trim_head(const std::string_view srcStr)
00202 {
00203     return cut_head(srcStr, get_whitespaces<TYPE>());
00204 }
00205
00211 template<EWhitespace TYPE = EWhitespace::Common>
00212 inline std::string_view trim_tail(const std::string_view srcStr)
00213 {
00214     return cut_tail(srcStr, get_whitespaces<TYPE>());
00215 }
00216
00222 template<EWhitespace TYPE = EWhitespace::Common>
00223 inline std::string_view trim(const std::string_view srcStr)
00224 {
00225     return trim_head<TYPE>(trim_tail<TYPE>(srcStr));
00226 }
00227
00235 inline std::string_view next_token(
00236     std::string_view        srcStr,
00237     std::string_view* const out_remainingStr = nullptr,
00238     const std::string_view  tokenSeparators = get_whitespaces<>())
00239 {
00240     srcStr = cut_head(srcStr, tokenSeparators);
00241
00242     const auto separatorPos = srcStr.find_first_of(tokenSeparators);
00243     if(separatorPos != std::string_view::npos)
00244     {
00245         const auto nextToken = srcStr.substr(0, separatorPos);
00246         if(out_remainingStr)
00247         {
00248             // `separatorPos + 1` as we do not want to include the separator
00249             *out_remainingStr = srcStr.substr(separatorPos + 1);
00250         }
00251
00252         return nextToken;
00253     }
00254     else
00255     {
00256         return srcStr;
00257     }
00258 }
00259
00265 inline char az_to_AZ(const char ch)
00266 {
00267     static_assert(std::numeric_limits<unsigned char>::max() == table::ASCII_TO_UPPER.size() - 1);
00268
00269     const auto mappedCharIdx = static_cast<unsigned char>(ch);
00270     return static_cast<char>(table::ASCII_TO_UPPER[mappedCharIdx]);
00271 }
00272
00278 inline char AZ_to_az(const char ch)
00279 {
00280     static_assert(std::numeric_limits<unsigned char>::max() == table::ASCII_TO_LOWER.size() - 1);
00281
00282     const auto mappedCharIdx = static_cast<unsigned char>(ch);
00283     return static_cast<char>(table::ASCII_TO_LOWER[mappedCharIdx]);
00284 }
00285
00293 inline void az_to_AZ(std::string& str)
00294 {
00295     for(char& ch : str)
00296     {
00297         ch = az_to_AZ(ch);
```

```
00298     }
00299 }
00300
00308 inline void AZ_to_az(std::string& str)
00309 {
00310     for(char& ch : str)
00311     {
00312         ch = AZ_to_az(ch);
00313     }
00314 }
00315
00318 inline std::string repeat(const std::string_view str, const std::size_t n)
00319 {
00320     const std::size_t totalSize = str.size() * n;
00321
00322     // Valid for the case where <totalSize> is 0
00323     std::string result;
00324     result.reserve(totalSize);
00325     for(std::size_t i = 0; i < n; ++i)
00326     {
00327         result += str;
00328     }
00329
00330     return result;
00331 }
00332
00335 inline void erase_all(std::string& str, const char ch)
00336 {
00337     str.erase(std::remove(str.begin(), str.end(), ch), str.end());
00338 }
00339
00340 namespace detail_from_to_char
00341 {
00342
00343 inline void throw_from_std_errc_if_has_error(const std::errc errorCode)
00344 {
00345     // According to several sources, 0, or zero-initialized std::errc,
00346     // indicates no error.
00347     //
00348     // [1] see the example for std::from_chars
00349     //     https://en.cppreference.com/w/cpp/utility/from_chars
00350     // [2] https://stackoverflow.com/a/63567008
00351     //
00352     constexpr std::errc NO_ERROR_VALUE = std::errc();
00353
00354     switch(errorCode)
00355     {
00356     case NO_ERROR_VALUE:
00357         return;
00358
00359     case std::errc::invalid_argument:
00360         throw InvalidArgumentException(
00361             "input cannot be interpreted as a numeric value");
00362
00363     case std::errc::result_out_of_range:
00364         throw OverflowException(
00365             "result will overflow the arithmetic type");
00366
00367     case std::errc::value_too_large:
00368         throw OutOfRangeException(
00369             "result cannot fit in the output buffer");
00370
00371     default:
00372         throw RuntimeException(
00373             "unknown error: std::errc = " + std::to_string(
00374             static_cast<std::underlying_type_t<std::errc>>(errorCode)));
00375     }
00376 }
00377
00378 }// end namespace detail_from_to_char
00379
00383 template<typename T>
00384 inline T parse_float(const std::string_view floatStr)
00385 {
00386     static_assert(std::is_floating_point_v<T>,
00387         "parse_float() accepts only floating point type.");
00388
00389     // `std::from_chars()` do not ignore leading whitespaces, we need to do it manually
00390     const std::string_view floatStrNoLeadingWS = trim_head(floatStr);
00391
00392     T value;
00393     const std::from_chars_result result = std::from_chars(
00394         floatStrNoLeadingWS.data(),
00395         floatStrNoLeadingWS.data() + floatStrNoLeadingWS.size(),
00396         value);
00397
00398     detail_from_to_char::throw_from_std_errc_if_has_error(result.ec);
```

```
00399
00400     return value;
00401 }
00402
00408 template<typename T>
00409 inline T parse_int(std::string_view intStr)
00410 {
00411     // TODO: option to handle base prefix (e.g., 0x)
00412
00413     static_assert(std::is_integral_v<T>,
00414         "parse_int() accepts only integer type.");
00415
00416     // `std::from_chars()` do not ignore leading whitespaces, we need to do it manually
00417     intStr = trim_head(intStr);
00418
00419     int base = 10;
00420     if(intStr.starts_with("0x"))
00421     {
00422         base = 16;
00423
00424         // Remove "0x" as `std::from_chars()` do not recognize base prefix
00425         intStr.remove_prefix(2);
00426     }
00427
00428     // `std::from_chars()` does not support `bool` so we treat it as unsigned char
00429     using IntType = std::conditional_t<std::is_same_v<T, bool>, unsigned char, T>;
00430
00431     std::remove_const_t<IntType> intValue;
00432     const std::from_chars_result result = std::from_chars(
00433         intStr.data(),
00434         intStr.data() + intStr.size(),
00435         intValue,
00436         base);
00437
00438     detail_from_to_char::throw_from_std_errc_if_has_error(result.ec);
00439
00440     return static_cast<T>(intValue);
00441 }
00442
00446 template<typename NumberType>
00447 inline NumberType parse_number(const std::string_view numberStr)
00448 {
00449     if constexpr(std::is_floating_point_v<NumberType>)
00450     {
00451         return parse_float<NumberType>(numberStr);
00452     }
00453     else
00454     {
00455         static_assert(std::is_integral_v<NumberType>);
00456
00457         return parse_int<NumberType>(numberStr);
00458     }
00459 }
00460
00474 template<typename T>
00475 inline std::size_t stringify_float(const T value, char* const out_buffer, const std::size_t
     bufferSize)
00476 {
00477     // TODO: option to handle base prefix (e.g., 0x)
00478     // TODO: option to handle precision
00479
00480     static_assert(std::is_floating_point_v<T>,
00481         "stringify_float() accepts only floating point type.");
00482
00483     PH_ASSERT(out_buffer);
00484     PH_ASSERT_GE(bufferSize, 1);
00485
00486     const std::to_chars_result result = std::to_chars(
00487         out_buffer,
00488         out_buffer + bufferSize,
00489         value);
00490
00491     detail_from_to_char::throw_from_std_errc_if_has_error(result.ec);
00492
00493     // Must written at least a char, and must not exceed bufferSize
00494     PH_ASSERT(out_buffer < result.ptr && result.ptr <= out_buffer + bufferSize);
00495     return static_cast<std::size_t>(result.ptr - out_buffer);
00496 }
00497
00508 template<std::integral T>
00509 inline std::size_t stringify_int_alphabetic(
00510     const T value,
00511     char* const out_buffer,
00512     const std::size_t bufferSize,
00513     const int base)
00514 {
00515     PH_ASSERT(out_buffer);
```

```
00516     PH_ASSERT_GE(bufferSize, 1);
00517     PH_ASSERT_IN_RANGE_INCLUSIVE(base, 2, 62);
00518
00519     // Treat `bool` as unsigned char (for arithmetics)
00520     using IntType = std::conditional_t<std::is_same_v<T, bool>, unsigned char, T>;
00521     auto intValue = static_cast<std::remove_const_t<IntType>>(value);
00522
00523     std::size_t numCharsWritten = 0;
00524
00525     // Write sign
00526     if constexpr(std::is_signed_v<T>)
00527     {
00528         if(intValue < 0)
00529         {
00530             out_buffer[0] = '-';
00531             ++numCharsWritten;
00532
00533             intValue = -intValue;
00534         }
00535     }
00536
00537     // Use a temporary buffer, enough to hold base 2 output
00538     std::array<unsigned char, sizeof(IntType) * CHAR_BIT> tmpBuffer;
00539     auto tmpBufferEnd = tmpBuffer.end();
00540
00541     PH_ASSERT_GE(intValue, 0);
00542     do
00543     {
00544         *(--tmpBufferEnd) = table::BASE62_DIGITS[intValue % base];
00545         intValue /= base;
00546     } while(intValue > 0);
00547
00548     auto numDigits = tmpBuffer.end() - tmpBufferEnd;
00549     if(numCharsWritten + numDigits > bufferSize)
00550     {
00551         throw_formatted<OutOfRangeException>(
00552             "result cannot fit in the output buffer: need={}, given={}",
00553             numCharsWritten + numDigits, bufferSize);
00554     }
00555     else
00556     {
00557         std::copy(tmpBufferEnd, tmpBuffer.end(), out_buffer + numCharsWritten);
00558         numCharsWritten += numDigits;
00559     }
00560
00561     return numCharsWritten;
00562 }
00563
00574 template<std::integral T>
00575 inline std::size_t stringify_int(
00576     const T value,
00577     char* const out_buffer,
00578     const std::size_t bufferSize,
00579     const int base = 10)
00580 {
00581     PH_ASSERT_IN_RANGE_INCLUSIVE(base, 2, 62);
00582
00583     // Base in [2, 36] is supported by STL via `to_chars()`
00584     if(2 <= base && base <= 36)
00585     {
00586         PH_ASSERT(out_buffer);
00587         PH_ASSERT_GE(bufferSize, 1);
00588
00589         // `std::to_chars()` does not support `bool` so we treat it as unsigned char
00590         using IntType = std::conditional_t<std::is_same_v<T, bool>, unsigned char, T>;
00591         const auto intValue = static_cast<IntType>(value);
00592
00593         std::to_chars_result result = std::to_chars(
00594             out_buffer,
00595             out_buffer + bufferSize,
00596             intValue,
00597             base);
00598
00599         detail_from_to_char::throw_from_std_errc_if_has_error(result.ec);
00600
00601         // Must written at least a char, and must not exceed bufferSize
00602         PH_ASSERT(out_buffer < result.ptr && result.ptr <= out_buffer + bufferSize);
00603         return static_cast<std::size_t>(result.ptr - out_buffer);
00604     }
00605     else
00606     {
00607         return stringify_int_alphabetic(value, out_buffer, bufferSize, base);
00608     }
00609 }
00610
00616 template<typename NumberType>
00617 inline std::size_t stringify_number(
```

```
00618     const NumberType value,
00619     char* const out_buffer,
00620     const std::size_t bufferSize)
00621 {
00622     if constexpr(std::is_floating_point_v<NumberType>)
00623     {
00624         return stringify_float<NumberType>(value, out_buffer, bufferSize);
00625     }
00626     else
00627     {
00628         static_assert(std::is_integral_v<NumberType>);
00629 
00630         return stringify_int<NumberType>(value, out_buffer, bufferSize);
00631     }
00632 }
00633 
00641 template<typename NumberType>
00642 inline std::string& stringify_number(
00643     const NumberType value,
00644     std::string& out_str,
00645     const std::size_t maxChars = 64)
00646 {
00647     const auto originalSize = out_str.size();
00648     out_str.resize(originalSize + maxChars);
00649 
00650     const std::size_t newSize = string_utils::stringify_number<NumberType>(
00651         value, out_str.data() + originalSize, maxChars);
00652 
00653     out_str.resize(originalSize + newSize);
00654     return out_str;
00655 }
00656 
00663 template<typename NumberType>
00664 inline std::string stringify_number(
00665     const NumberType value,
00666     const std::size_t maxChars = 64)
00667 {
00668     std::string str;
00669     stringify_number(value, str, maxChars);
00670     return str;
00671 }
00672 
00673 }// end namespace ph::string_utils
```

10.59 Include/Common/Utility/string_utils_table.h File Reference

```
#include <array>
#include <string_view>
```

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::string_utils

  Contains various string manipulation helpers.
- namespace ph::string_utils::table

Variables

- constexpr std::string_view ph::string_utils::table::common_whitespaces = " \n\r\t"

  Commonly used whitespace characters.
- constexpr std::string_view ph::string_utils::table::standard_whitespaces = " \n\r\t\v\f"

  Standard whitespace characters.
- constexpr std::array< unsigned char, 256 > ph::string_utils::table::ASCII_TO_UPPER
- constexpr std::array< unsigned char, 256 > ph::string_utils::table::ASCII_TO_LOWER
- constexpr std::array< unsigned char, 62 > ph::string_utils::table::BASE62_DIGITS

10.60 string_utils_table.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <array>
00004 #include <string_view>
00005
00006 namespace ph::string_utils::table
00007 {
00008
00013 inline constexpr std::string_view common_whitespaces = " \n\r\t";
00014
00019 inline constexpr std::string_view standard_whitespaces = " \n\r\t\v\f";
00020
00026 inline constexpr std::array<unsigned char, 256> ASCII_TO_UPPER =
00027 {{
00028     0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
00029     0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
00030     0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
00031     0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
00032     0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
00033     0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F,
00034     0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
00035     0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
00036     0x40, 'A', 'B', 'C', 'D', 'E', 'F', 'G',
00037     'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
00038     'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
00039     'X', 'Y', 'Z', 0x5B, 0x5C, 0x5D, 0x5E, 0x5F,
00040     0x60, 'A', 'B', 'C', 'D', 'E', 'F', 'G',
00041     'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
00042     'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
00043     'X', 'Y', 'Z', 0x7B, 0x7C, 0x7D, 0x7E, 0x7F,// 127
00044     0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
00045     0x88, 0x89, 0x8A, 0x8B, 0x8C, 0x8D, 0x8E, 0x8F,
00046     0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97,
00047     0x98, 0x99, 0x9A, 0x9B, 0x9C, 0x9D, 0x9E, 0x9F,
00048     0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7,
00049     0xA8, 0xA9, 0xAA, 0xAB, 0xAC, 0xAD, 0xAE, 0xAF,
00050     0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7,
00051     0xB8, 0xB9, 0xBA, 0xBB, 0xBC, 0xBD, 0xBE, 0xBF,
00052     0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7,
00053     0xC8, 0xC9, 0xCA, 0xCB, 0xCC, 0xCD, 0xCE, 0xCF,
00054     0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7,
00055     0xD8, 0xD9, 0xDA, 0xDB, 0xDC, 0xDD, 0xDE, 0xDF,
00056     0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7,
00057     0xE8, 0xE9, 0xEA, 0xEB, 0xEC, 0xED, 0xEE, 0xEF,
00058     0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7,
00059     0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, 0xFF // 255
00060 }};
00061
00067 inline constexpr std::array<unsigned char, 256> ASCII_TO_LOWER =
00068 {{
00069     0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
00070     0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
00071     0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
00072     0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
00073     0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
00074     0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F,
00075     0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
00076     0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
00077     0x40, 'a', 'b', 'c', 'd', 'e', 'f', 'g',
00078     'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
00079     'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
00080     'x', 'y', 'z', 0x5B, 0x5C, 0x5D, 0x5E, 0x5F,
00081     0x60, 'a', 'b', 'c', 'd', 'e', 'f', 'g',
00082     'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
00083     'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
00084     'x', 'y', 'z', 0x7B, 0x7C, 0x7D, 0x7E, 0x7F, // 127
00085     0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
00086     0x88, 0x89, 0x8A, 0x8B, 0x8C, 0x8D, 0x8E, 0x8F,
00087     0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97,
00088     0x98, 0x99, 0x9A, 0x9B, 0x9C, 0x9D, 0x9E, 0x9F,
00089     0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7,
00090     0xA8, 0xA9, 0xAA, 0xAB, 0xAC, 0xAD, 0xAE, 0xAF,
00091     0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7,
00092     0xB8, 0xB9, 0xBA, 0xBB, 0xBC, 0xBD, 0xBE, 0xBF,
00093     0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7,
00094     0xC8, 0xC9, 0xCA, 0xCB, 0xCC, 0xCD, 0xCE, 0xCF,
00095     0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7,
00096     0xD8, 0xD9, 0xDA, 0xDB, 0xDC, 0xDD, 0xDE, 0xDF,
00097     0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7,
00098     0xE8, 0xE9, 0xEA, 0xEB, 0xEC, 0xED, 0xEE, 0xEF,
00099     0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7,
00100     0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, 0xFF // 255
```

```
00101 }};
00102
00103 inline constexpr std::array<unsigned char, 62> BASE62_DIGITS =
00104 {{
00105     '0', '1', '2', '3', '4', '5', '6', '7',
00106     '8', '9', 'a', 'b', 'c', 'd', 'e', 'f',
00107     'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
00108     'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
00109     'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D',
00110     'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
00111     'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
00112     'U', 'V', 'W', 'X', 'Y', 'Z'
00113 }};
00114
00115 }// end namespace ph::string_utils::table
```

10.61 Include/Common/Utility/Timestamp.h File Reference

```
#include <string>
#include <chrono>
#include <ctime>
```

Classes

- class ph::Timestamp

  Represents a point in time.

Namespaces

- namespace ph

  The root for all renderer implementations.

10.62 Timestamp.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <string>
00004 #include <chrono>
00005 #include <ctime>
00006
00007 namespace ph
00008 {
00009
00015 class Timestamp final
00016 {
00017 public:
00018     Timestamp();
00019
00020     std::string toYMD() const;
00021     std::string toHMS() const;
00022     std::string toHMSMilliseconds() const;
00023     std::string toHMSMicroseconds() const;
00024     std::string toYMDHMS() const;
00025     std::string toYMDHMSMilliseconds() const;
00026     std::string toYMDHMSMicroseconds() const;
00027     std::string toString() const;
00028
00029 private:
00030     std::chrono::system_clock::time_point m_time;
00031
00032     std::time_t toCTime() const;
00033     std::string toYMDWithOldAPI() const;
```

```
00034     std::string toHMSWithOldAPI() const;
00035 };
00036
00037 // In-header Implementations:
00038
00039 inline Timestamp::Timestamp() :
00040     m_time(std::chrono::system_clock::now())
00041 {}
00042
00043 inline std::time_t Timestamp::toCTime() const
00044 {
00045     return std::chrono::system_clock::to_time_t(m_time);
00046 }
00047
00048 }// end namespace ph
```

10.63 README.md File Reference

10.64 Source/Common/assertion.cpp File Reference

```
#include "Common/assertion.h"
#include "Common/os.h"
#include "Common/debug.h"
#include <cstdlib>
#include <iostream>
```

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::detail

  Implementation detail mainly for internal usages.

Functions

- void ph::detail::output_assertion_message (const std::string &filename, const std::string &lineNumber, const std::string &condition, const std::string &message)
- void ph::detail::on_assertion_failed ()

10.65 Source/Common/config.cpp File Reference

```
#include "Common/config.h"
#include "Common/assertion.h"
```

Namespaces

- namespace ph

  The root for all renderer implementations.

10.66 Source/Common/Config/IniFile.cpp File Reference

```
#include "Common/Config/IniFile.h"
#include "Common/io_exceptions.h"
#include "Common/Utility/string_utils.h"
#include <fstream>
```

Namespaces

- namespace ph

  The root for all renderer implementations.

10.67 Source/Common/debug.cpp File Reference

```
#include "Common/debug.h"
#include "Common/config.h"
#include "Common/os.h"
#include <version>
#include <signal.h>
```

Namespaces

- namespace ph

  The root for all renderer implementations.

Macros

- #define psnip_trap() raise(SIGABRT)

Functions

- void ph::debug_break ()
- std::string ph::obtain_stack_trace ()

10.67.1 Macro Definition Documentation

10.67.1.1 psnip_trap

```
#define psnip_trap() raise(SIGABRT)
```

10.68 Source/Common/exception.cpp File Reference

```
#include "Common/exceptions.h"
```

Namespaces

- namespace ph

  The root for all renderer implementations.

10.69 Source/Common/Log/Logger.cpp File Reference

```
#include "Common/Log/Logger.h"
#include "Common/Log/ELogLevel.h"
#include "Common/config.h"
#include "Common/os.h"
#include "Common/Utility/Timestamp.h"
#include <iostream>
#include <string>
#include <utility>
```

Namespaces

- namespace ph

  The root for all renderer implementations.

10.70 Source/Common/logging.cpp File Reference

```
#include "Common/logging.h"
#include "Common/Log/Logger.h"
#include "Common/assertion.h"
#include "Common/Utility/Timestamp.h"
#include <utility>
#include <mutex>
#include <fstream>
#include <iostream>
#include <algorithm>
#include <filesystem>
#include <optional>
```

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::detail

  Implementation detail mainly for internal usages.
- namespace ph::detail::core_logging

  Core logging functionalities. Most logs will output information (logs) via a main logger, which we refer to as "core logger". This namespace contains implementation details for core logging functionalities.

Functions

- void ph::detail::core_logging::init ()

  Initializes core logging functionalities. Any logging is only valid after calling `init()`.
- void ph::detail::core_logging::exit ()

  Terminates core logging functionalities. Cleanup after logging is finished.
- Logger & ph::detail::core_logging::get_logger ()

  Get the core logger.
- std::size_t ph::detail::core_logging::add_log_group (std::string_view groupName, std::string_view category="")

  Add a log group to the core logger.
- void ph::detail::core_logging::log_to_logger (const Logger &logger, std::string_view groupName, ELogLevel logLevel, std::string_view logMessage)

  Log information to the specified logger.
- ph::PH_DEFINE_LOG_GROUP (PhotonRenderer, Core)
- LogGroups ph::get_core_log_groups ()

10.71 Source/Common/memory.cpp File Reference

```
#include "Common/memory.h"
#include "Common/math_basics.h"
#include "Common/os.h"
#include <cstdlib>
```

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::detail

  Implementation detail mainly for internal usages.

Functions

- void * ph::detail::allocate_aligned_memory (std::size_t numBytes, std::size_t alignmentInBytes)
- void ph::detail::free_aligned_memory (void *ptr)

10.72 Source/Common/os.cpp File Reference

```
#include "Common/os.h"
#include "Common/assertion.h"
#include <new>
#include <array>
```

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::os

Functions

- EWindowsVersion ph::os::get_windows_version ()

  Get current Windows version at runtime.
- std::size_t ph::os::get_L1_cache_line_size_in_bytes ()

  Get size of L1 cache at runtime.
- std::filesystem::path ph::os::get_executable_path ()

  Get the path to the currently running executable. Answering the question, "Where am I?".

10.73 Source/Common/profiling.cpp File Reference

```
#include "Common/profiling.h"
```

10.74 Source/Common/stats.cpp File Reference

```
#include "Common/stats.h"
#include "Common/assertion.h"
#include <utility>
#include <mutex>
#include <format>
#include <unordered_map>
```

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::detail

  Implementation detail mainly for internal usages.
- namespace ph::detail::stats

10.75 Source/Common/utility.cpp File Reference

```
#include "Common/utility.h"
#include <iostream>
```

Namespaces

- namespace ph

  The root for all renderer implementations.
- namespace ph::detail

  Implementation detail mainly for internal usages.

Functions

- void ph::detail::output_not_implemented_warning (const std::string &filename, const std::string &lineNumber)

10.76 Source/Common/Utility/CommandLineArguments.cpp File Reference

```
#include "Common/Utility/CommandLineArguments.h"
#include "Common/assertion.h"
#include <algorithm>
```

Namespaces

- namespace ph

  The root for all renderer implementations.

10.77 Source/Common/Utility/Timestamp.cpp File Reference

```
#include "Common/Utility/Timestamp.h"
#include "Common/assertion.h"
#include "Common/os.h"
#include <format>
#include <mutex>
#include <sstream>
```

Namespaces

- namespace ph

  The root for all renderer implementations.

Index

~LogicalException
    ph::LogicalException, 62
~RuntimeException
    ph::RuntimeException, 65
~ScopedTimer
    ph::detail::stats::ScopedTimer, 66

add_log_group
    ph::detail::core_logging, 26
addGroup
    ph::LogGroups, 60
addLogHandler
    ph::Logger, 58
addMicroseconds
    ph::detail::stats::TimeCounter, 68
allocate_aligned_memory
    ph::detail, 24
append
    ph::IniFile, 54
ASCII_TO_LOWER
    ph::string_utils::table, 40
ASCII_TO_UPPER
    ph::string_utils::table, 40
assertion.h
    PH_ASSERT, 77
    PH_ASSERT_EQ, 77
    PH_ASSERT_GE, 77
    PH_ASSERT_GT, 77
    PH_ASSERT_IN_RANGE, 77
    PH_ASSERT_IN_RANGE_EXCLUSIVE, 77
    PH_ASSERT_IN_RANGE_INCLUSIVE, 78
    PH_ASSERT_LE, 78
    PH_ASSERT_LT, 78
    PH_ASSERT_MSG, 78
    PH_ASSERT_NE, 78
    PH_ASSERT_UNREACHABLE_SECTION, 78
    PH_INTERNAL_RANGE_MSG, 79
    PH_STATIC_ASSERT_DEPENDENT_FALSE, 79
averagedReport
    ph::TimerStatsReport, 70
AZ_to_az
    ph::string_utils, 32
az_to_AZ
    ph::string_utils, 33

BASE62_DIGITS
    ph::string_utils::table, 40

category
    ph::detail::stats::TimeCounter, 68
    ph::LogGroup, 59
    ph::TimerStatsReport::TimeRecord, 69
ceil_div
    ph::math, 28
clear
    ph::IniFile, 54
Clock
    ph::detail::stats::ScopedTimer, 66
CommandLineArguments
    ph::CommandLineArguments, 44
Common
    ph::string_utils, 32
common_whitespaces
    ph::string_utils::table, 40
compiler.h
    PH_COMPILER_IS_CLANG, 81
    PH_COMPILER_IS_GCC, 81
    PH_COMPILER_IS_MSVC, 81
config.h
    PH_ABORT_ON_ASSERTION_FAILED, 83
    PH_CONFIG_DIRECTORY, 83
    PH_DEBUG, 83
    PH_ENABLE_DEBUG_LOG, 83
    PH_ENABLE_HIT_EVENT_STATS, 83
    PH_ENGINE_VERSION, 84
    PH_ENSURE_LOCKFREE_ALGORITHMS_ARE_LOCKLESS, 84
    PH_HIT_PROBE_CACHE_BYTES, 84
    PH_HIT_PROBE_DEPTH, 84
    PH_INTERNAL_RESOURCE_DIRECTORY, 84
    PH_LOG_FILE_DIRECTRY, 84
    PH_MEMORY_ARENA_DEFAULT_BLOCK_SIZE_IN_BYTES, 84
    PH_NUMERIC_IMAGE_MAX_ELEMENTS, 84
    PH_PRINT_STACK_TRACE_ON_ASSERTION_FAILED, 85
    PH_PROFILING, 85
    PH_PSDL_VERSION, 85
    PH_RENDER_MODE, 85
    PH_RENDER_MODE_ACES, 85
    PH_RENDER_MODE_FULL_SPECTRAL, 85
    PH_RENDER_MODE_LINEAR_SRGB, 85
    PH_RENDER_MODE_SPECTRAL, 85
    PH_RENDERER_RESOURCE_DIRECTORY, 85
    PH_SCRIPT_DIRECTORY, 86
    PH_SDL_MAX_FIELDS, 86
    PH_SDL_MAX_FUNCTIONS, 86
    PH_SPECTRUM_SAMPLED_MAX_WAVELENGTH_NM, 86

PH_SPECTRUM_SAMPLED_MIN_WAVELENGTH_NM, 86
PH_SPECTRUM_SAMPLED_NUM_SAMPLES, 86
PH_STRICT_ASYMMETRIC_IMPORTANCE_TRANSPORT, 86
PH_STRICT_FLOATING_POINT_SIZES, 86
PH_STRICT_OBJECT_LIFETIME, 87
PH_TFUNCTION_DEFAULT_MIN_SIZE_IN_BYTES, 87
PH_USE_DOUBLE_REAL, 87
count
ph::detail::stats::TimeCounter, 68
ph::TimerStatsReport::GroupedTimeRecord, 50
ph::TimerStatsReport::TimeRecord, 69
counter
ph::detail::stats::ScopedTimer, 67
cut_ends
ph::string_utils, 33
cut_head
ph::string_utils, 33
cut_tail
ph::string_utils, 34

Debug
ph, 20
debug.cpp
psnip_trap, 145
debug.h
PH_DEBUG_BREAK, 93
debug_break
ph, 20
DebugOnce
ph, 20
DEPENDENT_FALSE
ph::detail, 26
detailedReport
ph::TimerStatsReport, 70
Documentation/namespace_ph.dox, 75
Documentation/namespace_ph_detail.dox, 75
Documentation/namespace_ph_detail_core_logging.dox, 75
Documentation/namespace_ph_string_utils.dox, 75

ELogLevel
ph, 19
erase_all
ph::string_utils, 34
Error
ph, 20
ErrorOnce
ph, 20
EWhitespace
ph::string_utils, 32
EWindowsVersion
ph::os, 29
Exception
ph, 16
exit
ph::detail::core_logging, 26

FileIOError
ph::FileIOError, 48
FilesystemError
ph::FilesystemError, 49
findPropertyIndex
ph::IniFile, 54
findSectionIndex
ph::IniFile, 54
float32
ph, 16
float64
ph, 16
free_aligned_memory
ph::detail, 25
from_bytes
ph, 20

get_core_log_groups
ph, 20
get_executable_path
ph::os, 30
get_L1_cache_line_size_in_bytes
ph::os, 30
get_logger
ph::detail::core_logging, 27
get_whitespaces
ph::string_utils, 34
get_windows_version
ph::os, 30
getCurrentSectionName
ph::IniFile, 54
getGroup
ph::LogGroups, 60
getGroupedTimeRecord
ph::TimerStatsReport, 70
getProgramName
ph::CommandLineArguments, 44
getPropertyName
ph::IniFile, 54
getPropertyValue
ph::IniFile, 54
getSectionName
ph::IniFile, 54
GroupedTimeRecord
ph::TimerStatsReport::GroupedTimeRecord, 50
groupName
ph::LogGroup, 59
ph::TimerStatsReport::GroupedTimeRecord, 50

has_any_of
ph::string_utils, 34
has_none_of
ph::string_utils, 34
hiInteger
ph, 16
hiReal
ph, 16

Include/Common/assertion.h, 76, 79
Include/Common/compiler.h, 80, 81
Include/Common/config.h, 82, 87
Include/Common/Config/IniFile.h, 88, 89
Include/Common/Container/detail.h, 90, 91
Include/Common/Container/StdUnorderedStringSet.h, 91, 92
Include/Common/Container/TStdUnorderedStringMap.h, 92, 93
Include/Common/debug.h, 93
Include/Common/exceptions.h, 94, 95
Include/Common/io_exceptions.h, 96
Include/Common/Log/ELogLevel.h, 97, 98
Include/Common/Log/Logger.h, 98, 99
Include/Common/Log/logger_fwd.h, 99, 100
Include/Common/logging.h, 100, 106
Include/Common/macro.h, 108, 110
Include/Common/math_basics.h, 110, 111
Include/Common/memory.h, 111, 113
Include/Common/memory.ipp, 113, 114
Include/Common/os.h, 116, 118
Include/Common/primitive_type.h, 118, 120
Include/Common/profiling.h, 120, 122
Include/Common/stats.h, 122, 124
Include/Common/ThirdParty/lib_tracy.h, 125
Include/Common/utility.h, 125, 127
Include/Common/utility.ipp, 127, 128
Include/Common/Utility/CommandLineArguments.h, 130
Include/Common/Utility/string_utils.h, 132, 135
Include/Common/Utility/string_utils_table.h, 141, 142
Include/Common/Utility/Timestamp.h, 143
IniFile
  ph::IniFile, 53
init
  ph::detail::core_logging, 27
int16
  ph, 16
int16f
  ph, 16
int32
  ph, 16
int32f
  ph, 16
int64
  ph, 17
int64f
  ph, 17
int8
  ph, 17
int8f
  ph, 17
integer
  ph, 17
IOException
  ph::FilesystemError, 49
  ph::IOException, 57
is_once
  ph, 20
is_power_of_2
  ph::math, 28
is_transparent
  ph::detail::HeterogeneousStringHash, 51
is_whitespace
  ph::string_utils, 35
isEmpty
  ph::CommandLineArguments, 44

log
  ph::Logger, 58, 59
log_to_logger
  ph::detail::core_logging, 27
Logger
  ph::Logger, 58
logging.h
  PH_DEBUG_LOG, 102
  PH_DEBUG_LOG_ONCE, 102
  PH_DEBUG_LOG_STRING, 102
  PH_DEBUG_LOG_STRING_ONCE, 102
  PH_DECLARE_LOG_GROUP, 102
  PH_DEFAULT_DEBUG_LOG, 102
  PH_DEFAULT_DEBUG_LOG_ONCE, 103
  PH_DEFAULT_DEBUG_LOG_STRING, 103
  PH_DEFAULT_DEBUG_LOG_STRING_ONCE, 103
  PH_DEFAULT_LOG, 103
  PH_DEFAULT_LOG_STRING, 103
  PH_DEFINE_EXTERNAL_LOG_GROUP, 103
  PH_DEFINE_INLINE_LOG_GROUP, 104
  PH_DEFINE_INTERNAL_LOG_GROUP, 104
  PH_DEFINE_LOG_GROUP, 104
  PH_LOG, 104
  PH_LOG_FORMAT_STRING_TO_CORE_LOGGER, 105
  PH_LOG_RAW_STRING_TO_CORE_LOGGER, 105
  PH_LOG_STRING, 105
LogGroups
  ph::LogGroups, 60
LogHandler
  ph, 17
LogicalException
  ph::IllegalOperationException, 52
  ph::InvalidArgumentException, 56
  ph::LogicalException, 61
  ph::OutOfRangeException, 63
  ph::UninitializedObjectException, 73
lossless_cast
  ph, 20
lossless_float_cast
  ph, 21
lossless_integer_cast
  ph, 21

macro.h
  PH_CONCAT_2, 108
  PH_CONCAT_3, 108

PH_CONCAT_4, 108
PH_CONCAT_5, 108
PH_CONCAT_6, 109
PH_CONCAT_7, 109
PH_CONCAT_8, 109
PH_NO_OP, 109
make_aligned_memory
ph, 21
make_array
ph, 21
ph::detail, 25
makeColoredStdOutLogPrinter
ph::Logger, 59
makeStdOutLogPrinter
ph::Logger, 59

name
ph::detail::stats::TimeCounter, 68
ph::TimerStatsReport::TimeRecord, 69
next_multiple
ph::math, 28
next_power_of_2_multiple
ph::math, 28
next_token
ph::string_utils, 35
Note
ph, 20
NoteOnce
ph, 20
numGroups
ph::LogGroups, 60
numProperties
ph::IniFile, 55
numSections
ph::IniFile, 55

obtain_stack_trace
ph, 22
on_assertion_failed
ph::detail, 25
operator()
ph::detail::AlignedMemoryDeleter, 43
ph::detail::HeterogeneousStringHash, 51
operator=
ph::LogGroups, 61
os.h
PH_OPERATING_SYSTEM_IS_LINUX, 117
PH_OPERATING_SYSTEM_IS_OSX, 117
PH_OPERATING_SYSTEM_IS_WINDOWS, 117
output_assertion_message
ph::detail, 25
output_not_implemented_warning
ph::detail, 25

parse_float
ph::string_utils, 35
parse_int
ph::string_utils, 35
parse_number
ph::string_utils, 35
ph, 13
Debug, 20
debug_break, 20
DebugOnce, 20
ELogLevel, 19
Error, 20
ErrorOnce, 20
Exception, 16
float32, 16
float64, 16
from_bytes, 20
get_core_log_groups, 20
hiInteger, 16
hiReal, 16
int16, 16
int16f, 16
int32, 16
int32f, 16
int64, 17
int64f, 17
int8, 17
int8f, 17
integer, 17
is_once, 20
LogHandler, 17
lossless_cast, 20
lossless_float_cast, 21
lossless_integer_cast, 21
make_aligned_memory, 21
make_array, 21
Note, 20
NoteOnce, 20
obtain_stack_trace, 22
PH_DECLARE_LOG_GROUP, 22
PH_DEFINE_LOG_GROUP, 22
real, 17
reverse_bytes, 22
sizeof_in_bits, 22
start_implicit_lifetime_as, 22
start_implicit_lifetime_as_array, 23
StdUnorderedStringSet, 17
TAlignedMemoryUniquePtr, 17
throw_formatted, 23
to_bytes, 23
TStdUnorderedStringMap, 18
uint16, 18
uint16f, 18
uint32, 18
uint32f, 18
uint64, 18
uint64f, 18
uint8, 18
uint8f, 18
Warning, 20
WarningOnce, 20
ph::CommandLineArguments, 43
CommandLineArguments, 44

getProgramName, 44
isEmpty, 44
retrieve, 45
retrieveFloat, 45
retrieveInt, 45
retrieveOptionArguments, 45
retrieveString, 45
retrieveStrings, 46
ph::Config, 46
RENDERER_RESOURCE_DIRECTORY, 47
ph::CPhotonException, 41
ph::detail, 23
allocate_aligned_memory, 24
DEPENDENT_FALSE, 26
free_aligned_memory, 25
make_array, 25
on_assertion_failed, 25
output_assertion_message, 25
output_not_implemented_warning, 25
ph::detail::AlignedMemoryDeleter, 43
operator(), 43
ph::detail::core_logging, 26
add_log_group, 26
exit, 26
get_logger, 27
init, 27
log_to_logger, 27
ph::detail::CPermissiveImplicitLifetime, 41
ph::detail::HeterogeneousStringHash, 51
is_transparent, 51
operator(), 51
ph::detail::stats, 27
ph::detail::stats::ScopedTimer, 66
∼ScopedTimer, 66
Clock, 66
counter, 67
ScopedTimer, 66
startTime, 67
ph::detail::stats::TimeCounter, 67
addMicroseconds, 68
category, 68
count, 68
name, 68
TimeCounter, 67
totalMicroseconds, 68
ph::FileIOError, 47
FileIOError, 48
whatStr, 48
ph::FilesystemError, 48
FilesystemError, 49
IOException, 49
whatStr, 49
ph::IllegalOperationException, 52
LogicalException, 52
ph::IniFile, 53
append, 54
clear, 54
findPropertyIndex, 54
findSectionIndex, 54
getCurrentSectionName, 54
getPropertyName, 54
getPropertyValue, 54
getSectionName, 54
IniFile, 53
numProperties, 55
numSections, 55
read, 55
save, 55
setCurrentSection, 55
setProperty, 55
ph::InvalidArgumentException, 56
LogicalException, 56
ph::IOException, 57
IOException, 57
ph::Logger, 58
addLogHandler, 58
log, 58, 59
Logger, 58
makeColoredStdOutLogPrinter, 59
makeStdOutLogPrinter, 59
ph::LogGroup, 59
category, 59
groupName, 59
ph::LogGroups, 60
addGroup, 60
getGroup, 60
LogGroups, 60
numGroups, 60
operator=, 61
ph::LogicalException, 61
∼LogicalException, 62
LogicalException, 61
whatStr, 62
ph::math, 28
ceil_div, 28
is_power_of_2, 28
next_multiple, 28
next_power_of_2_multiple, 28
ph::NumericException, 62
RuntimeException, 63
ph::os, 29
EWindowsVersion, 29
get_executable_path, 30
get_L1_cache_line_size_in_bytes, 30
get_windows_version, 30
Unknown, 29
Windows_10, 29
Windows_2000, 29
Windows_7, 29
Windows_8, 29
Windows_8_1, 29
Windows_Vista, 29
Windows_XP, 29
ph::OutOfRangeException, 63
LogicalException, 63
ph::OverflowException, 64

ph::RuntimeException, 65
~RuntimeException, 65
RuntimeException, 65
whatStr, 66
ph::string_utils, 30
AZ_to_az, 32
az_to_AZ, 33
Common, 32
cut_ends, 33
cut_head, 33
cut_tail, 34
erase_all, 34
EWhitespace, 32
get_whitespaces, 34
has_any_of, 34
has_none_of, 34
is_whitespace, 35
next_token, 35
parse_float, 35
parse_int, 35
parse_number, 35
repeat, 36
Standard, 32
stringify_float, 36
stringify_int, 36
stringify_int_alphabetic, 37
stringify_number, 37, 38
trim, 38
trim_head, 38
trim_tail, 39
ph::string_utils::CHasToString, 41
ph::string_utils::detail_from_to_char, 39
throw_from_std_errc_if_has_error, 39
ph::string_utils::table, 40
ASCII_TO_LOWER, 40
ASCII_TO_UPPER, 40
BASE62_DIGITS, 40
common_whitespaces, 40
standard_whitespaces, 40
ph::TimerStatsReport, 69
averagedReport, 70
detailedReport, 70
getGroupedTimeRecord, 70
proportionalReport, 70
rawReport, 70
TimerStatsReport, 70
ph::TimerStatsReport::GroupedTimeRecord, 50
count, 50
GroupedTimeRecord, 50
groupName, 50
subgroups, 50
totalMicroseconds, 50
ph::TimerStatsReport::TimeRecord, 68
category, 69
count, 69
name, 69
TimeRecord, 69
totalMicroseconds, 69
ph::Timestamp, 70
Timestamp, 71
toHMS, 71
toHMSMicroseconds, 71
toHMSMilliseconds, 71
toString, 71
toYMD, 71
toYMDHMS, 71
toYMDHMSMicroseconds, 72
toYMDHMSMilliseconds, 72
ph::UninitializedObjectException, 72
LogicalException, 73
PH_ABORT_ON_ASSERTION_FAILED
config.h, 83
PH_ASSERT
assertion.h, 77
PH_ASSERT_EQ
assertion.h, 77
PH_ASSERT_GE
assertion.h, 77
PH_ASSERT_GT
assertion.h, 77
PH_ASSERT_IN_RANGE
assertion.h, 77
PH_ASSERT_IN_RANGE_EXCLUSIVE
assertion.h, 77
PH_ASSERT_IN_RANGE_INCLUSIVE
assertion.h, 78
PH_ASSERT_LE
assertion.h, 78
PH_ASSERT_LT
assertion.h, 78
PH_ASSERT_MSG
assertion.h, 78
PH_ASSERT_NE
assertion.h, 78
PH_ASSERT_UNREACHABLE_SECTION
assertion.h, 78
PH_COMPILER_IS_CLANG
compiler.h, 81
PH_COMPILER_IS_GCC
compiler.h, 81
PH_COMPILER_IS_MSVC
compiler.h, 81
PH_CONCAT_2
macro.h, 108
PH_CONCAT_3
macro.h, 108
PH_CONCAT_4
macro.h, 108
PH_CONCAT_5
macro.h, 108
PH_CONCAT_6
macro.h, 109
PH_CONCAT_7
macro.h, 109
PH_CONCAT_8
macro.h, 109

PH_CONFIG_DIRECTORY
config.h, 83
PH_DEBUG
config.h, 83
PH_DEBUG_BREAK
debug.h, 93
PH_DEBUG_LOG
logging.h, 102
PH_DEBUG_LOG_ONCE
logging.h, 102
PH_DEBUG_LOG_STRING
logging.h, 102
PH_DEBUG_LOG_STRING_ONCE
logging.h, 102
PH_DECLARE_LOG_GROUP
logging.h, 102
ph, 22
PH_DEFAULT_DEBUG_LOG
logging.h, 102
PH_DEFAULT_DEBUG_LOG_ONCE
logging.h, 103
PH_DEFAULT_DEBUG_LOG_STRING
logging.h, 103
PH_DEFAULT_DEBUG_LOG_STRING_ONCE
logging.h, 103
PH_DEFAULT_LOG
logging.h, 103
PH_DEFAULT_LOG_STRING
logging.h, 103
PH_DEFINE_EXTERNAL_LOG_GROUP
logging.h, 103
PH_DEFINE_EXTERNAL_TIMER_STAT
stats.h, 123
PH_DEFINE_INLINE_LOG_GROUP
logging.h, 104
PH_DEFINE_INLINE_TIMER_STAT
stats.h, 123
PH_DEFINE_INLINE_TO_STRING_FORMATTER
string_utils.h, 134
PH_DEFINE_INLINE_TO_STRING_FORMATTER_SPECIALIZATION
string_utils.h, 134
PH_DEFINE_INLINE_TO_STRING_FORMATTER_TEMPLATE
string_utils.h, 135
PH_DEFINE_INTERNAL_LOG_GROUP
logging.h, 104
PH_DEFINE_INTERNAL_TIMER_STAT
stats.h, 123
PH_DEFINE_LOG_GROUP
logging.h, 104
ph, 22
PH_DEFINE_PROFILE_UNIT_NAME
profiling.h, 121
PH_ENABLE_DEBUG_LOG
config.h, 83
PH_ENABLE_HIT_EVENT_STATS
config.h, 83
PH_ENGINE_VERSION
config.h, 84
PH_ENSURE_LOCKFREE_ALGORITHMS_ARE_LOCKLESS
config.h, 84
PH_HIT_PROBE_CACHE_BYTES
config.h, 84
PH_HIT_PROBE_DEPTH
config.h, 84
PH_INTERNAL_RANGE_MSG
assertion.h, 79
PH_INTERNAL_RESOURCE_DIRECTORY
config.h, 84
PH_LOG
logging.h, 104
PH_LOG_FILE_DIRECTRY
config.h, 84
PH_LOG_FORMAT_STRING_TO_CORE_LOGGER
logging.h, 105
PH_LOG_RAW_STRING_TO_CORE_LOGGER
logging.h, 105
PH_LOG_STRING
logging.h, 105
PH_MEMORY_ARENA_DEFAULT_BLOCK_SIZE_IN_BYTES
config.h, 84
PH_NO_OP
macro.h, 109
PH_NOT_IMPLEMENTED_WARNING
utility.h, 126
PH_NUMERIC_IMAGE_MAX_ELEMENTS
config.h, 84
PH_OPERATING_SYSTEM_IS_LINUX
os.h, 117
PH_OPERATING_SYSTEM_IS_OSX
os.h, 117
PH_OPERATING_SYSTEM_IS_WINDOWS
os.h, 117
PH_PRINT_STACK_TRACE_ON_ASSERTION_FAILED
config.h, 85
PH_PROFILE_LOOP_BEGIN
profiling.h, 121
PH_PROFILE_LOOP_END
profiling.h, 121
PH_PROFILE_LOOP_MARK
profiling.h, 121
PH_PROFILE_NAME_THIS_THREAD
profiling.h, 121
PH_PROFILE_NAMED_SCOPE
profiling.h, 121
PH_PROFILE_SCOPE
profiling.h, 121
PH_PROFILING
config.h, 85
PH_PSDL_VERSION
config.h, 85
PH_RENDER_MODE
config.h, 85
PH_RENDER_MODE_ACES
config.h, 85
PH_RENDER_MODE_FULL_SPECTRAL
config.h, 85

PH_RENDER_MODE_LINEAR_SRGB
config.h, 85
PH_RENDER_MODE_SPECTRAL
config.h, 85
PH_RENDERER_RESOURCE_DIRECTORY
config.h, 85
PH_SCOPED_TIMER
stats.h, 123
PH_SCRIPT_DIRECTORY
config.h, 86
PH_SDL_MAX_FIELDS
config.h, 86
PH_SDL_MAX_FUNCTIONS
config.h, 86
PH_SPECTRUM_SAMPLED_MAX_WAVELENGTH_NM
config.h, 86
PH_SPECTRUM_SAMPLED_MIN_WAVELENGTH_NM
config.h, 86
PH_SPECTRUM_SAMPLED_NUM_SAMPLES
config.h, 86
PH_STATIC_ASSERT_DEPENDENT_FALSE
assertion.h, 79
PH_STRICT_ASYMMETRIC_IMPORTANCE_TRANSPORT
config.h, 86
PH_STRICT_FLOATING_POINT_SIZES
config.h, 86
PH_STRICT_OBJECT_LIFETIME
config.h, 87
PH_TFUNCTION_DEFAULT_MIN_SIZE_IN_BYTES
config.h, 87
PH_USE_DOUBLE_REAL
config.h, 87
Photon Common Library, 1
profiling.h
PH_DEFINE_PROFILE_UNIT_NAME, 121
PH_PROFILE_LOOP_BEGIN, 121
PH_PROFILE_LOOP_END, 121
PH_PROFILE_LOOP_MARK, 121
PH_PROFILE_NAME_THIS_THREAD, 121
PH_PROFILE_NAMED_SCOPE, 121
PH_PROFILE_SCOPE, 121
proportionalReport
ph::TimerStatsReport, 70
psnip_trap
debug.cpp, 145

rawReport
ph::TimerStatsReport, 70
read
ph::IniFile, 55
README.md, 144
real
ph, 17
RENDERER_RESOURCE_DIRECTORY
ph::Config, 47
repeat
ph::string_utils, 36
retrieve
ph::CommandLineArguments, 45
retrieveFloat
ph::CommandLineArguments, 45
retrieveInt
ph::CommandLineArguments, 45
retrieveOptionArguments
ph::CommandLineArguments, 45
retrieveString
ph::CommandLineArguments, 45
retrieveStrings
ph::CommandLineArguments, 46
reverse_bytes
ph, 22
RuntimeException
ph::NumericException, 63
ph::RuntimeException, 65

save
ph::IniFile, 55
ScopedTimer
ph::detail::stats::ScopedTimer, 66
setCurrentSection
ph::IniFile, 55
setProperty
ph::IniFile, 55
sizeof_in_bits
ph, 22
Source/Common/assertion.cpp, 144
Source/Common/config.cpp, 144
Source/Common/Config/IniFile.cpp, 145
Source/Common/debug.cpp, 145
Source/Common/exception.cpp, 145
Source/Common/Log/Logger.cpp, 146
Source/Common/logging.cpp, 146
Source/Common/memory.cpp, 147
Source/Common/os.cpp, 147
Source/Common/profiling.cpp, 148
Source/Common/stats.cpp, 148
Source/Common/utility.cpp, 148
Source/Common/Utility/CommandLineArguments.cpp, 149
Source/Common/Utility/Timestamp.cpp, 149
Standard
ph::string_utils, 32
standard_whitespaces
ph::string_utils::table, 40
start_implicit_lifetime_as
ph, 22
start_implicit_lifetime_as_array
ph, 23
startTime
ph::detail::stats::ScopedTimer, 67
stats.h
PH_DEFINE_EXTERNAL_TIMER_STAT, 123
PH_DEFINE_INLINE_TIMER_STAT, 123
PH_DEFINE_INTERNAL_TIMER_STAT, 123
PH_SCOPED_TIMER, 123
StdUnorderedStringSet
ph, 17
string_utils.h

PH_DEFINE_INLINE_TO_STRING_FORMATTER, 134
PH_DEFINE_INLINE_TO_STRING_FORMATTER_SPECIALIZATION, 134
PH_DEFINE_INLINE_TO_STRING_FORMATTER_TEMPLATE, 135
stringify_float
ph::string_utils, 36
stringify_int
ph::string_utils, 36
stringify_int_alphabetic
ph::string_utils, 37
stringify_number
ph::string_utils, 37, 38
subgroups
ph::TimerStatsReport::GroupedTimeRecord, 50

TAlignedMemoryUniquePtr
ph, 17
throw_formatted
ph, 23
throw_from_std_errc_if_has_error
ph::string_utils::detail_from_to_char, 39
TimeCounter
ph::detail::stats::TimeCounter, 67
TimeRecord
ph::TimerStatsReport::TimeRecord, 69
TimerStatsReport
ph::TimerStatsReport, 70
Timestamp
ph::Timestamp, 71
to_bytes
ph, 23
toHMS
ph::Timestamp, 71
toHMSMicroseconds
ph::Timestamp, 71
toHMSMilliseconds
ph::Timestamp, 71
toString
ph::Timestamp, 71
totalMicroseconds
ph::detail::stats::TimeCounter, 68
ph::TimerStatsReport::GroupedTimeRecord, 50
ph::TimerStatsReport::TimeRecord, 69
toYMD
ph::Timestamp, 71
toYMDHMS
ph::Timestamp, 71
toYMDHMSMicroseconds
ph::Timestamp, 72
toYMDHMSMilliseconds
ph::Timestamp, 72
trim
ph::string_utils, 38
trim_head
ph::string_utils, 38
trim_tail
ph::string_utils, 39

TStdUnorderedStringMap
ph, 18

uint16
ph, 18
uint16f
ph, 18
uint32
ph, 18
uint32f
ph, 18
uint64
ph, 18
uint64f
ph, 18
uint8
ph, 18
uint8f
ph, 18
Unknown
ph::os, 29
utility.h
PH_NOT_IMPLEMENTED_WARNING, 126

Warning
ph, 20
WarningOnce
ph, 20
whatStr
ph::FileIOError, 48
ph::FilesystemError, 49
ph::LogicalException, 62
ph::RuntimeException, 66
Windows_10
ph::os, 29
Windows_2000
ph::os, 29
Windows_7
ph::os, 29
Windows_8
ph::os, 29
Windows_8_1
ph::os, 29
Windows_Vista
ph::os, 29
Windows_XP
ph::os, 29